The Android Telephony API

As a software developer for mobile platforms, you may be interested in incorporating telephony features in your app. For instance, your interactive app might need to notify the user when it starts accessing the Internet while roaming; or, your high-bandwidth video chat application might need to alter its stream quality, based on the type of connection and its latency. These are situations where mobile apps need an interface to the basic phone features and its state, which is when the Telephony APIs come in.

Telephony Manager provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information. Applications can also register a listener to receive notification of telephony state changes.

You do not instantiate this class directly; instead, you retrieve a reference to an instance through Context.getSystemService(Context.TELEPHONY_SERVICE).

Programmers will need ways to not just retrieve telephony data, but also to dial a phone number, intercept outgoing phone calls, or send/receive SMS messages. This is achieved with a combination of classes provided in the android. Telephony package, along with inbuilt Intents and phone services.

The most important of these is the TelephonyManager class, which can be used to retrieve metadata about the SIM card, the physical device and the network provider. Programmers will need to use the Android Context, through the getSystemService() method, with a constant as shown below, to obtain an instance of the TelephonyManager class.

```
// ..creating TelephonyManager from Context
final TelephonyManager mytelMgr =
(TelephonyManager)this.getSystemService(Context.TELEPHONY_SE
RVICE);


public String getCurrentCallState(final TelephonyManager mytelMgr) {
      int callState = mytelMgr.getCallState();
      String callStateString = "NOTKNOWN";
      switch (callState) {
         case TelephonyManager.CALL_STATE_IDLE:
            callStateString = "IDLE";
            break;
         case TelephonyManager.CALL_STATE_OFFHOOK:
            callStateString = "OFFHOOK";
            break;
         case TelephonyManager.CALL_STATE_RINGING:
            callStateString = "RINGING";
            break;
       }
}
```

Additionally, it is often important to know the change in the call or service state of the phone, in an app. For example, you may want to mute your application when a call arrives. This is done by attaching a listener, called PhoneStateListener to the TelephonyManager.

Finally, it is important to note that information retrieval via TelephonyManager is permission-protected. For the application to access this information, the android.permission.READ_PHONE_STATE permission has to be set in the app's manifest.xml.

The steps to sending a simple text message:

- ❖ The app incorporates the android.telephony package.

1) Permissions must be set in the manifest file to send and receive SMS messages, as shown below:

- Send SMS — android.permission.SEND_SMS
- Receive SMS — android.permission.RECEIVE_SMS

2) The app uses an instance of SmsManager, retrieved using the static method getDefault():
final SmsManager sms = SmsManager.getDefault() ;

3) A PendingIntent is created in order for the application to track message delivery status. (A PendingIntent provides a means for an application to work beyond its life, for a particular Intent. Even after the owning application dies, a PendingIntent created by it can be run later.) This pending intent is invoked when the sending handset receives an acknowledgement from the network that the destination handset has received the message.

inal PendingIntent sentIntent = PendingIntent.getActivity(this, 0, new Intent(this,SmsSendCheck.class), 0);

To send a simple text message, the sample code would be like what's shown below:
sms.sendTextMessage("98867xxxxx", null, "Test message from RPBC!", sentIntent, null);

Get the Phone Type CDMA/GSM/NONE

/ /Get the type of network you are connected with

```
int phoneType=tm.getPhoneType();
switch (phoneType)
{
      case (TelephonyManager.PHONE_TYPE_CDMA):
            // your code
               break;
      case (TelephonyManager.PHONE_TYPE_GSM)
            // your code
               break;
      case (TelephonyManager.PHONE_TYPE_NONE):
            // your code
                break;
 }
```

Find whether the Phone is in Roaming, returns true if in roaming

```
boolean isRoaming=tm.isNetworkRoaming();
 if(isRoaming)
       phoneDetails+="\nIs In Roaming : "+"YES";
 else
      phoneDetails+="\nIs In Roaming : "+"NO";
```

Get the SIM state/Details

```
int SIMState=tm.getSimState();
switch(SIMState)
{
    case TelephonyManager.SIM_STATE_ABSENT :
      // your code
      break;
    case
TelephonyManager.SIM_STATE_NETWORK_LOCKED :
      // your code
      break;
    case TelephonyManager.SIM_STATE_PIN_REQUIRED :
      // your code
      break;
    case TelephonyManager.SIM_STATE_PUK_REQUIRED :
      // your code
      break;
    case TelephonyManager.SIM_STATE_READY :
      // your code
      break;
    case TelephonyManager.SIM_STATE_UNKNOWN :
      // your code
      break;

}
```

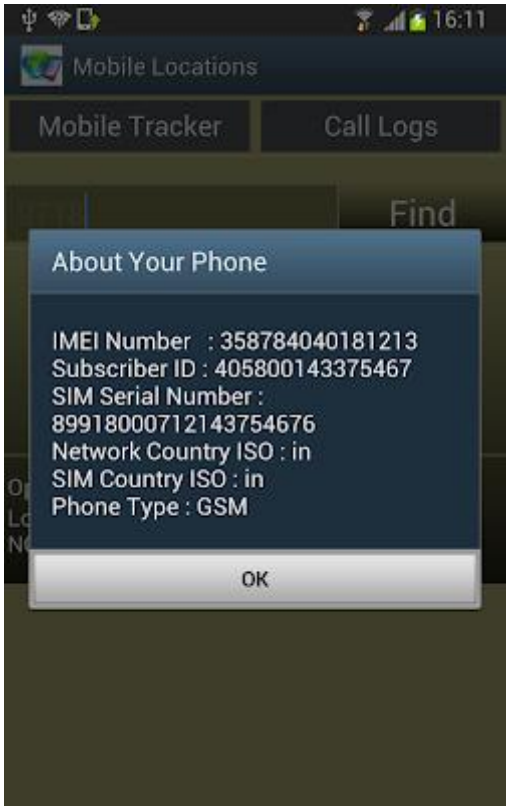
Getting Network Details

```
// Get connected network country ISO code
String networkCountry = telephonyManager.getNetworkCountryIso();

// Get the connected network operator ID (MCC + MNC)
String networkOperatorId = telephonyManager.getNetworkOperator();

// Get the connected network operator name
String networkName = telephonyManager.getNetworkOperatorName();

// Get the type of network you are connected with
int networkType = telephonyManager.getNetworkType();
switch (networkType) {
case (TelephonyManager.NETWORK_TYPE_1xRTT) :"  Your Code ":
break;
case (TelephonyManager.NETWORK_TYPE_CDMA) :"  Your Code ":
break;
case (TelephonyManager.NETWORK_TYPE_EDGE) : "  Your Code ":
break;
case (TelephonyManager.NETWORK_TYPE_EVDO_0) :"  Your Code
":
break;
```



Getting SIM Details :

Using the Object of Telephony Manager class we can get the details like SIM Serial number, Country Code, Network Provider code and other Details.

```
int simState = telephonyManager.getSimState();
switch (simState)
{
        case (TelephonyManager.SIM_STATE_ABSENT): break;
        case
(TelephonyManager.SIM_STATE_NETWORK_LOCKED): break;
        case (TelephonyManager.SIM_STATE_PIN_REQUIRED):
break;
       case (TelephonyManager.SIM_STATE_PUK_REQUIRED):
```

```
break;
        case (TelephonyManager.SIM_STATE_UNKNOWN): break;
        case (TelephonyManager.SIM_STATE_READY):
         {
                  // Get the SIM country ISO code
                 String simCountry =
telephonyManager.getSimCountryIso();
                  // Get the operator code of the active SIM (MCC +
MNC)
                String simOperatorCode =
telephonyManager.getSimOperator();
                 // Get the name of the SIM operator
                String simOperatorName =
telephonyManager.getSimOperatorName();
                 // -- Requires READ_PHONE_STATE uses-
permission --
                 // Get the SIM's serial number
                String simSerial =
telephonyManager.getSimSerialNumber();
        }
}
```