Android Telephony

Telephony

What is Android Telephony?

- **Android Telephony** framework provides us the functionalities of the mobile.

- It gives us information about functionalities like calls, SMS, MMS, network, data services, IMEI number, voice call, Video call and so on.

Telephony

Architecture of Android Telephony

- Android Telephony architecture works in 4 layers that are : Communication Processor, Radio Interface Layer (RIL), Framework Services , Applications

- Communication Processor : It is an input/output processor to distribute and collect data from a number of remote terminals. It is a specialized processor designed to communicate with the data communication network.

- Radio Interface Layer : It is a bridge between the hardware and Android phone framework services. Rather we say, it is a protocol stack for Telephone. It has two main components that are:
  - **RIL Daemon**– It starts when the android system starts. It reads the system properties to find a library that is to be used for Vendor RIL.
  - **Vendor RIL**– It is also known as RIL Driver. It can be understood as a library that is specific to each modem.

Telephony

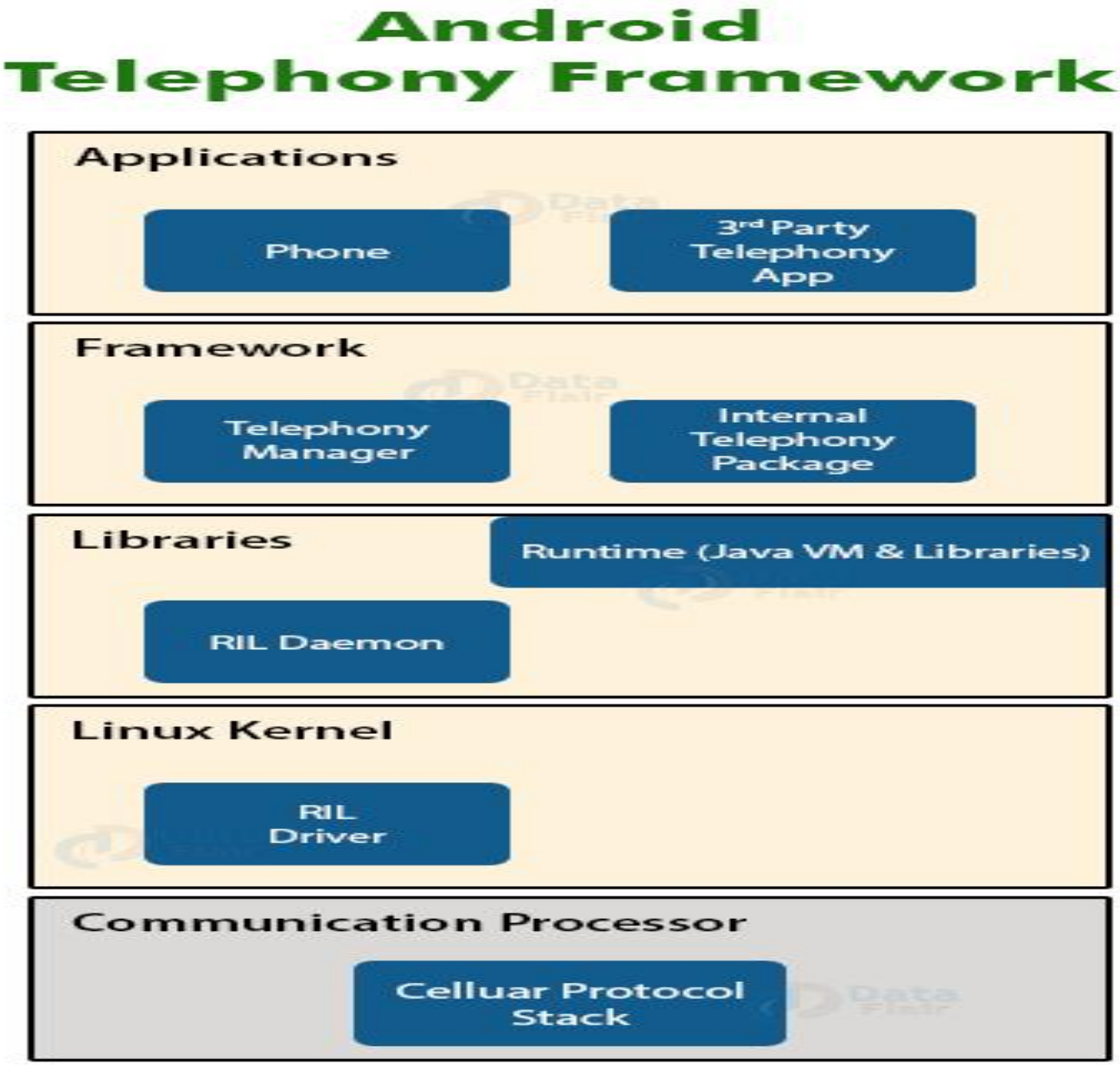
Architecture of Android Telephony

- Framework Services : The telephony Framework starts and initializes along with the system. All the queries by Application API are directed to RIL using these services.

- Application : These are the Application UI related to telephony such as Dialer, SMS, MMS, Call tracker, etc. These applications start with the android system boot up. These are tied with framework services of telephony.

- Android Telephony Framework consists of two types of packages that are:
  - **Internal Telephony Packages:** This is generally the used for default telephony app.apk.
  - **Open Technology Packages:** This is for third-party apps.

Telephony


Android
Telephony Framework
Applications
Phone
3rd Party Telephony App
Framework
Telephony Manager
Internal Telephony Package
Libraries
Runtime (Java VM & Libraries)
RIL Daemon
Linux Kernel
RIL Driver
Communication Processor
Celluar Protocol Stack

Telephony

Telephony Framework

- **Telephony framework** consists of components like
  - Service State Tracker (SST),
  - SIM IO subsystem,
  - GsmCdmaCallTracker,
  - Data Connection Tracker (DCT),
  - SIM toolkit,
  - Support for MMI codes.
  - It also provides us certain services like Voice services (MO, MT , Call forwarding, call hold, 3-way call, Call supplementary),
  - SMS ,
  - MMS,
  - voice mail,
  - Network Acess Services (NAS-Voice, data and bearer registration),
  - SIM Services ,
  - Device Management Services(DMS),
  - Phone book Manager(PBM — Managing Contacts).

Telephony

Telephony Applications

- Dialer
- Contacts
- Messaging — SMS/MMS(BasicSmsReceiver)
- Settings app — a) Airplane mode b) APN (access point name) settings C)Network selection (auto/manual) d) Call settings, such as, call waiting, call forwarding/diverting e) CLIR (caller identity restrictions) f)FDN (fixed dialling Number)
- SIM Application Toolkit (SAT)
- Browser
- CellBroadcastReceiver
- CarrierConfig
- PhoneCommon

Telephony

Hardware Support for Telephony

- With the arrival of Wi-Fi-only Android devices, you can no longer assume that telephony will be supported on all the hardware on which your application may be available.

- **Marking Telephony as a Required Hardware Feature**

- To specify that your application requires telephony support to function, you can add a uses-feature node to your application manifest:
  <uses-feature android:name="android.hardware.telephony"
  android:required="true"/>

Telephony

Hardware Support for Telephony

- **Checking for Telephony Hardware**

- If you use telephony APIs but they aren't strictly necessary for your application to be used, you can check for the existence of telephony hardware before attempting to make use of the related APIs.

- Use the Package Manager's **hasSystemFeature** method, specifying the FEATURE_TELEPHONY feature.

- The Package Manager also includes constants to query the existence of CDMA- and GSM-specific hardware.

Telephony

Hardware Support for Telephony

```
PackageManager pm = getPackageManager();
boolean telephonySupported =
    pm.hasSystemFeature(PackageManager.FEATURE_TELEPHONY);
boolean gsmSupported =
   pm.hasSystemFeature(PackageManager.FEATURE_TELEPHONY_CDMA);
boolean cdmaSupported =
   pm.hasSystemFeature(PackageManager.FEATURE_TELEPHONY_GSM);
```

Telephony

Using Telephony

- The Android telephony APIs let your applications access the underlying telephone hardware stack, making it possible to create your own dialer — or integrate call handling and phone state monitoring into your applications.

- **Initiating Phone Calls**

- Best practice for initiating phone calls is to use an Intent.ACTION_DIAL Intent, specifying the number to dial by setting the Intents data using a tel: schema:

```
Intent whoyougonnacall = new Intent(Intent.ACTION_DIAL,
                                    Uri.parse("tel:555-2368"));
startActivity(whoyougonnacall);
```

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

- This starts a dialer Activity that should be prepopulated with the number you specified. As a result using the ACTION_DIAL Intent action doesn't require any special permissions.

Telephony

Using Telephony

- **Replacing the Native Dialer**
- Replacing the native dialer application involves two steps:
    1. **Intercept Intents serviced by the native dialer.**
    2. **Initiate and manage outgoing calls.**

- The native dialer application responds to Intent actions corresponding to a user pressing the hardware call button, asking to view data using the tel: schema, or making an ACTION_DIAL request using the tel: schema, as shown in the previous section.

- To intercept these requests, include intent-filter tags on the manifest entries for your replacement dialer Activity that listens for the following actions:
    - Intent.ACTION_CALL_BUTTON — This action is broadcast when the device's hardware call button is pressed. Create an Intent Filter that listens for this action as a default action.
    - Intent.ACTION_DIAL — The Intent Filter used to capture this action should be both default and browsable (to support dial requests from the browser) and must specify the tel: schema to replace existing dialer functionality (though it can support additional schemes).

Telephony

Using Telephony

- **Replacing the Native Dialer**
  - Intent.ACTION_VIEW — The view action is used by applications wanting to view a piece of data. Ensure that the Intent Filter specifies the tel: schema to allow your new Activity to be used to view telephone numbers.
- The manifest snippet which shows an Activity with Intent Filters that will capture each of these actions.

```
<activity
    android:name=".MyDialerActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.CALL_BUTTON" />
        <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
     <intent-filter>
         <action android:name="android.intent.action.VIEW" />
          <action android:name="android.intent.action.DIAL" />
          <category android:name="android.intent.category.DEFAULT" />
          <category android:name="android.intent.category.BROWSABLE" />
           <data android:scheme="tel" />
      </intent-filter>
</activity>
```

Telephony

Telephony Manager

- Android **Telephony Manager** provides access to the information about the Telephony Services on the devices.
- Application will be the using the methods in the TelephonyManager class to determine Telephony States, services, subscriber information .
- You do not instantiate this class directly; instead, you retrieve a reference to an instance through Context.getSystemService(Context.TELEPHONY_SERVICE).
- It consist of the android.telephony and android.telephony.gsm Packages.
- Applications will be registering a listener to receive notification of changes in the telephone states.
- Its contents are : Device, Phone type, State of Mobile, SIM card, Uniqueness Features

Telephony

Telephony Manager

- To work with Telephony Manager and to read the phone details we need ***<uses-permission android:name="android.permission.READ_PHONE_STATE"/>*** permission. So add this permission in your manifest file.

- Accessing the Telephony Manager : Have an object of TelephonyMnager

  TelephonyManager tm ;

  tm= (TelephonyManager) getSystemService(TELEPHONY_SERVICE);

- The getDeviceId() method is used to get the IMEI/MEID of the device. If the device is a GSM device then IMEI will be returned and if the device is a CDMA device then MEID will be returned. This device id can be used to uniquely identify the device.

- Get IMEI Number of Phone : String IMEINumber=tm.getDeviceId();

- Get Subscriber ID : String subscriberID=tm.getSubscriberId();

Telephony

Telephony Manager

- Get SIM Serial Number :

  String SIMSerialNumber=tm.getSimSerialNumber();

- Get Network Country ISO Code

  String networkCountryISO=tm.getNetworkCountryIso();

- Get SIM Country ISO Code

  String SIMCountryISO=tm.getSimCountryIso();

- Get the device software version

  String softwareVersion=tm.getDeviceSoftwareVersion()

- Get the Voice mail number

  String voiceMailNumber=tm.getVoiceMailNumber();

Telephony

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;
                                              }
```

Telephony

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;

        }
```

Telephony

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();
```

Telephony

Getting SIM Details

```
        // 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();
}}
```

Telephony

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;
```

Telephony

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";
```

Telephony

Monitoring Changes in Phone State Using the Phone State Listener

- The Android telephony APIs lets you monitor changes to phone state and associated details such as incoming phone numbers.

- Changes to the phone state are monitored using the PhoneStateListener class, with some state changes also broadcast as Intents.

- To monitor and manage phone state, your application must specify the READ_PHONE_STATE uses-permission:

<uses-permission android:name=”android.permission.READ_PHONE_STATE”/>

- Create a new class that implements the Phone State Listener to monitor, and respond to, phone state change events, including call state (ringing, off hook, and so on), cell location changes, voice-mail and call-forwarding status, phone service changes, and changes in mobile signal strength.

- Within your Phone State Listener implementation, override the event handlers of the events you want to react to.

Telephony

Monitoring Changes in Phone State Using the Phone State Listener

- After creating your own Phone State Listener, register it with the Telephony Manager using a bitmask to indicate the events you want to listen for:

```
telephonyManager.listen(phoneStateListener,
PhoneStateListener.LISTEN_CALL_FORWARDING_INDICATOR |
PhoneStateListener.LISTEN_CALL_STATE |
PhoneStateListener.LISTEN_CELL_LOCATION |
PhoneStateListener.LISTEN_DATA_ACTIVITY |
PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |
PhoneStateListener.LISTEN_MESSAGE_WAITING_INDICATOR |
PhoneStateListener.LISTEN_SERVICE_STATE |
PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
```

- To unregister a listener, call listen and pass in PhoneStateListener.LISTEN_NONE as the bitmask parameter:

```
telephonyManager.listen(phoneStateListener,
PhoneStateListener.LISTEN_NONE);
```

Telephony

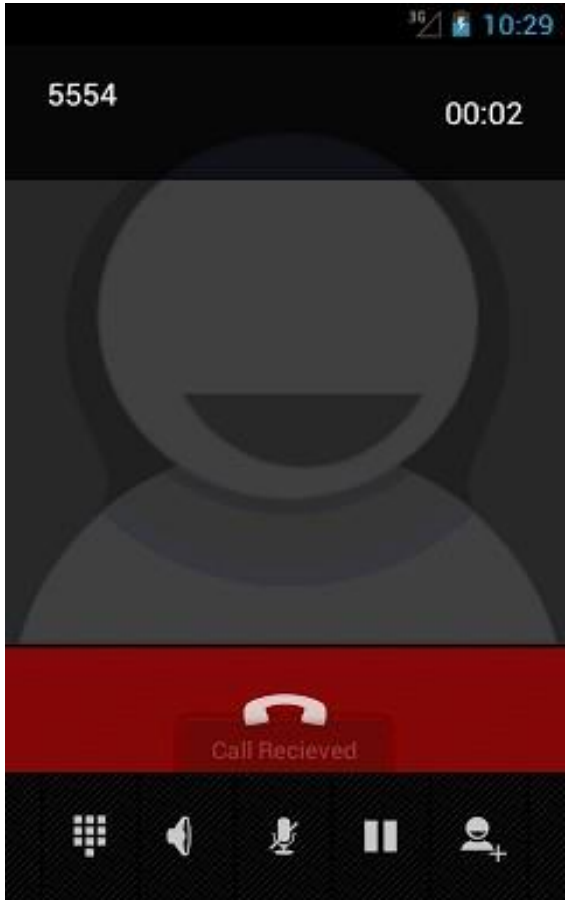
Monitoring Incoming Phone Calls

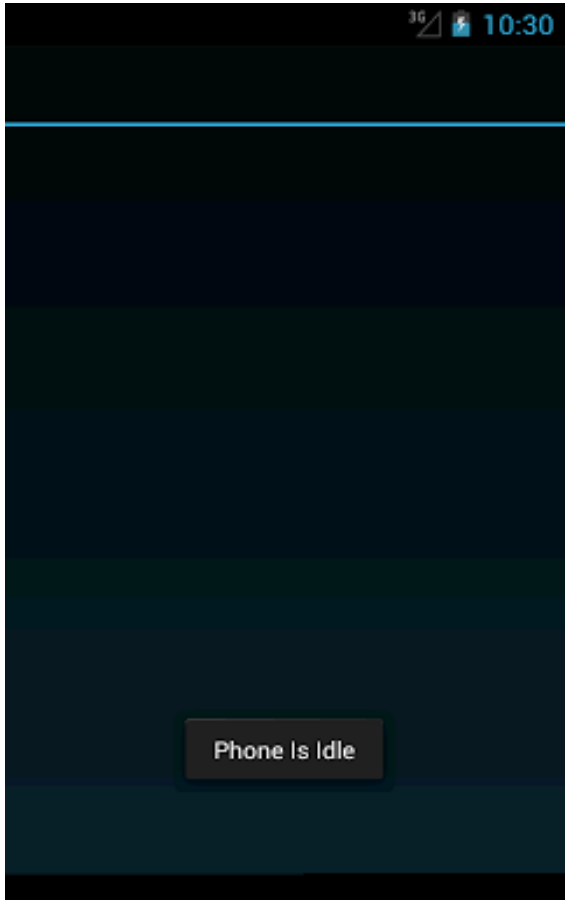
- If your application should respond to incoming phone calls only while it is running, you can override the onCallStateChanged method in your Phone State Listener implementation, and register it to receive notifications when the call state changes:

```
PhoneStateListener callStateListener = new PhoneStateListener() {
public void onCallStateChanged(int state, String incomingNumber) {
String callStateStr = “Unknown”;
switch (state) {
    case TelephonyManager.CALL_STATE_IDLE :
      callStateStr = “idle”; break;
    case TelephonyManager.CALL_STATE_OFFHOOK :
        callStateStr = “offhook”; break;
     case TelephonyManager.CALL_STATE_RINGING :
        callStateStr = “ringing. Incoming number is: “+ incomingNumber;
        break;
     default : break;
}
Toast.makeText(MyActivity.this,callStateStr, Toast.LENGTH_LONG).show();}};
```

Telephony

Incoming Call Receiver In Android Running Screens





Telephony

Monitoring Incoming Phone Calls

```
telephonyManager.listen(callStateListener,
        PhoneStateListener.LISTEN_CALL_STATE);
```

- The onCallStateChanged handler receives the phone number associated with incoming calls, and the state parameter represents the current call state as one of the following three values:
  - TelephonyManager.CALL_STATE_IDLE — When the phone is neither ringing nor in a call
  - TelephonyManager.CALL_STATE_RINGING — When the phone is ringing
  - TelephonyManager.CALL_STATE_OFFHOOK — When the phone is currently in a call

- Note that as soon as the state changes to CALL_STATE_RINGING, the system will display the incoming call screen, asking users if they want to answer the call.

- Your application must be running to receive this callback.

Telephony

Tracking Cell Location Changes

- You can get notifications whenever the current cell location changes by overriding onCellLocationChanged on a Phone State Listener implementation.

- Before you can register to listen for cell location changes, you need to add the ACCESS_COARSE_LOCATION permission to your application manifest:

```
<uses-permission
android:name=”android.permission.ACCESS_COARSE_LOCATION”/>
```

- The onCellLocationChanged handler receives a CellLocation object that includes methods for extracting different location information based on the type of phone network.

- In the case of a GSM network, the cell ID (getCid) and the current location area code (getLac) are available.

- For CDMA networks, you can obtain the current base station ID (getBaseStationId) and the latitude (getBaseStationLatitude) and longitude (getBaseStationLongitude) of that base station.

Telephony

Tracking Cell Location Changes

- The following code snippet shows how to implement a Phone State Listener to monitor cell location changes, displaying a Toast that includes the received network location details.

```
PhoneStateListener cellLocationListener = new PhoneStateListener() {
    public void onCellLocationChanged(CellLocation location) {
      if (location instanceof GsmCellLocation) {
         GsmCellLocation gsmLocation = (GsmCellLocation)location;
          Toast.makeText(getApplicationContext(),
           String.valueOf(gsmLocation.getCid()),Toast.LENGTH_LONG).show();
      }
      else if (location instanceof CdmaCellLocation) {
            CdmaCellLocation cdmaLocation = (CdmaCellLocation)location;
            StringBuilder sb = new StringBuilder();
            sb.append(cdmaLocation.getBaseStationId());
            sb.append("\n@");
           sb.append(cdmaLocation.getBaseStationLatitude());
           sb.append(cdmaLocation.getBaseStationLongitude());
```

Telephony

Tracking Cell Location Changes

```
        Toast.makeText(getApplicationContext(),sb.toString(),
            Toast.LENGTH_LONG).show();
          }
      }
};

telephonyManager.listen(cellLocationListener,
                                PhoneStateListener.LISTEN_CELL_LOCATION);
```

Telephony

Tracking Service Changes

- The onServiceStateChanged handler tracks the service details for the device's cell service.

- Use the ServiceState parameter to find details of the current service state.

- The getState method on the Service State object returns the current service state as one of the following ServiceState constants:
  - STATE_IN_SERVICE — Normal phone service is available.
  - STATE_EMERGENCY_ONLY — Phone service is available but only for emergency calls.
  - STATE_OUT_OF_SERVICE — No cell phone service is currently available.
  - STATE_POWER_OFF — The phone radio is turned off (usually when airplane mode is enabled).

- A series of getOperator* methods is available to retrieve details on the operator supplying the cell phone service, whereas getRoaming tells you if the device is currently using a roaming profile:

Telephony

Tracking Service Changes

```
PhoneStateListener serviceStateListener = new PhoneStateListener() {
    public void onServiceStateChanged(ServiceState serviceState) {
        if (serviceState.getState() == ServiceState.STATE_IN_SERVICE) {
            String toastText = “Operator: “ + serviceState.getOperatorAlphaLong();
             Toast.makeText(MyActivity.this, toastText,
             Toast.LENGTH_SHORT).show();
         }
     }
};

telephonyManager.listen(serviceStateListener,
                                PhoneStateListener.LISTEN_SERVICE_STATE);
```

Telephony

Monitoring Data Connectivity and Data Transfer Status Changes

- You can use a Phone State Listener to monitor changes in mobile data connectivity and mobile data transfer.

- Note that this does not include data transferred using Wi-Fi.

- The Phone State Listener includes two event handlers for monitoring the device's data connection.
  - Override **onDataActivity** to track data transfer activity, and
  - **onDataConnectionStateChanged** to request notifications for data connection state changes:

Telephony

Monitoring Data Connectivity and Data Transfer Status Changes

```
PhoneStateListener dataStateListener = new PhoneStateListener() {
    public void onDataActivity(int direction) {
        String dataActivityStr = “None”;
        switch (direction) {
            case TelephonyManager.DATA_ACTIVITY_IN :
                dataActivityStr = “Downloading”; break;
            case TelephonyManager.DATA_ACTIVITY_OUT :
                dataActivityStr = “Uploading”; break;
            case TelephonyManager.DATA_ACTIVITY_INOUT :
                dataActivityStr = “Uploading/Downloading”; break;
            case TelephonyManager.DATA_ACTIVITY_NONE :
                dataActivityStr = “No Activity”; break;
        }
        Toast.makeText(MyActivity.this,“Data Activity is “ + dataActivityStr,
        Toast.LENGTH_LONG).show();
}
```

Telephony

Monitoring Data Connectivity and Data Transfer Status Changes

```
public void onDataConnectionStateChanged(int state) {
        String dataStateStr = “Unknown”;
         switch (state) {
               case TelephonyManager.DATA_CONNECTED :
                     dataStateStr = “Connected”; break;
               case TelephonyManager.DATA_CONNECTING :
                     dataStateStr = “Connecting”; break;
                case TelephonyManager.DATA_DISCONNECTED :
                      dataStateStr = “Disconnected”; break;
                case TelephonyManager.DATA_SUSPENDED :
                       dataStateStr = “Suspended”; break;
           }
           Toast.makeText(MyActivity.this,“Data Connectivity is “ + dataStateStr,
            Toast.LENGTH_LONG).show();
       }
};
telephonyManager.listen(dataStateListener,
                                PhoneStateListener.LISTEN_DATA_ACTIVITY |
                                PhoneStateListener.LISTEN_DATA_CONNECTION_STATE);
```


THANKS