Telephony and SMS
Objective

• Telephony
  ➢ Initiating phone calls
  ➢ Reading the phone, network, data connectivity, and SIM states
  ➢ Monitoring changes to the phone, network, data connectivity, and

• SMS
  ➢ Using Intents to send SMS and MMS messages
  ➢ Using the SMS Manager to send SMS Messages
  ➢ Handling incoming SMS messages
Telephony
Overview

• The Android telephony APIs allows:
  ➤ Access the underlying telephone hardware stack
  ➤ Create your own dialer
  ➤ Integrate call handling and phone state monitoring

• For security, you can’t create your own “in call” Activity
  ➤ The screen that is displayed when an incoming call is received or an outgoing call has been placed.
Launching the Dialer

- Use Intent `Intent.ACTION_DIAL` to launch dialer activity.
  - Specify the number to dial using the `tel:` schema as the data component of the Intent.
  - Allows you to manage the call initialization (the default dialer asks the user to explicitly initiate the call).
  - Doesn’t require any permissions
  - The standard way applications should initiate calls.

```java
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:1234567"));
startActivity(intent);
```
Telephony Manager

• Access to the telephony APIs is managed by the Telephony Manager

```java
String svcName = Context.TELEPHONY_SERVICE;
TelephonyManager telephonyManager = (TelephonyManager) getSystemService(svcName);
```

• Thru Telephony Manager you can obtain:
  ➤ the phone type (GSM or CDMA),
  ➤ unique ID (IMEI or MEID),
  ➤ software version,
  ➤ number.

• Requires the **READ_PHONE_STATE** uses-permission be included in the application manifest.

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

Telephony Manager Reference:
Telephony Manager
// Read the phone’s type
int phoneType = telephonyManager.getPhoneType();
switch (phoneType) {
    case (TelephonyManager.PHONE_TYPE_CDMA): //do something
        break;
    case (TelephonyManager.PHONE_TYPE_GSM) : //do something
        break;
    case (TelephonyManager.PHONE_TYPE_NONE): //do something
        break;
    default:
        break;
}

// -- These require READ_PHONE_STATE uses-permission --
// Read the IMEI for GSM or MEID for CDMA
String deviceId = telephonyManager.getDeviceId();

// Read the software version on the phone (note -- not the SDK version)
String softwareVersion = telephonyManager.getDeviceSoftwareVersion();

// Get the phone’s number
String phoneNumber = telephonyManager.getLine1Number();
Reading Data Connection Status

```java
int dataActivity = telephonyManager.getDataActivity();
int dataState = telephonyManager.getDataState();
switch (dataActivity) {
    case TelephonyManager.DATA_ACTIVITY_IN: //Currently receiving IP PPP traffic.
        break;
    case TelephonyManager.DATA_ACTIVITY_OUT: //Currently sending IP PPP traffic.
        break;
    case TelephonyManager.DATA_ACTIVITY_INOUT: //Currently both IN & OUT
        break;
    case TelephonyManager.DATA_ACTIVITY_NONE: //No traffic.
        break;
}
switch (dataState) {
    case TelephonyManager.DATA_CONNECTED: //Connected.
        break;
    case TelephonyManager.DATA_CONNECTING: //Currently setting up data connection
        break;
    case TelephonyManager.DATA_DISCONNECTED: //Disconnected
        break;
    case TelephonyManager.DATA_SUSPENDED: //Suspended
        break;
}```
Reading 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 to
int networkType = telephonyManager.getNetworkType();
switch (networkType) {
    case (TelephonyManager.NETWORK_TYPE_1xRTT): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_CDMA): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_EDGE): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_EVDO_0): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_EVDO_A): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_GPRS): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_HSDPA): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_HSPA): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_HSUPA): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_UMTS): /* ... */ break;
    case (TelephonyManager.NETWORK_TYPE_UNKNOWN): /* ... */ break;
    default: break;
}

Info about Service Providers in USA:
http://en.wikipedia.org/wiki/List_of_United_States_wireless_communications_service_providers
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();

        break;
    }
    default: break;
}
Monitoring Phone Status

- Android lets you:
  - monitor phone state,
  - retrieve incoming phone numbers,
  - observe changes to data connections, signal strength, and network connectivity.

- Must specify the **READ_PHONE_STATE** uses-permission in its manifest.

- Extend **PhoneStateListener** class to listen and respond to:
  - Phone state change events including call state (ringing, off hook, etc.),
  - Cell location changes,
  - Voice-mail and call-forwarding status,
  - Phone service changes,
  - Changes in mobile signal strength.

**PhoneStateListener Reference:**
Monitoring Phone Status

• Phone State Listener skeleton class

```java
PhoneStateListener phoneStateListener = new PhoneStateListener() {
    public void onCallForwardingIndicatorChanged(boolean cfi) {}
    public void onCallStateChanged(int state, String incomingNumber) {}
    public void onCellLocationChanged(CellLocation location) {}
    public void onDataActivity(int direction) {}  
    public void onDataConnectionStateChanged(int state) {}  
    public void onMessageWaitingIndicatorChanged(boolean mwi) {}  
    public void onServiceStateChanged(ServiceState serviceState) {}  
    public void onSignalStrengthChanged(int asu) {}
};
```

• Registering a Phone State Listener

```java
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_STRENGTH);
```
Monitoring Phone Calls

• The `onCallStateChanged` handler receives the phone number associated with incoming calls, and the state parameter represents the current call state:
  ➤ `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

```java
PhoneStateListener callStateListener = new PhoneStateListener() {
    public void onCallStateChanged(int state, String incomingNumber) {
        // TODO React to incoming call.
    }
};

telephonyManager.listen(callStateListener, PhoneStateListener.LISTEN_CALL_STATE);
```
Override `onCellLocationChanged` to listen for cell location changes.

Add the `ACCESS_COARSE_LOCATION` permission to your application manifest:

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

Handler receives a `CellLocation` object that includes methods for extracting the cell ID (`getCid`) and the current LAC (`getLac`).

```java
PhoneStateListener cellLocationListener = new PhoneStateListener() {
    public void onCellLocationChanged(CellLocation location) {
        GsmCellLocation gsmLocation = (GsmCellLocation)location;
        Toast.makeText(getApplicationContext(),
                String.valueOf(gsmLocation.getCid()),
                Toast.LENGTH_LONG).show();
    }
};

telephonyManager.listen(cellLocationListener, PhoneStateListener.LISTEN_CELL_LOCATION);
```
Tracking Service Changes

• The `onServiceStateChanged` handler tracks the service
• Use the `ServiceState` parameter with `getState` method to find details of the current service state.
  ➤ `STATE_IN_SERVICE` Normal phone service is available.
  ➤ `STATE_EMERGENCY_ONLY` Phone service is available only for emergency calls.
  ➤ `STATE_OUT_OF_SERVICE` No cell phone service is currently available.
  ➤ `STATE_POWER_OFF` The phone radio is turned off

• `getOperator*` methods to retrieve details on the operator while `getRoaming` tells you if the device is using a roaming profile.

```java
PhoneStateListener serviceStateListener = new PhoneStateListener() {
    public void onServiceStateChanged(ServiceState serviceState) {
        if (serviceState.getState() == ServiceState.STATE_IN_SERVICE) {
            String toastText = serviceState.getOperatorAlphaLong();
            Toast.makeText(getApplicationContext(), toastText, Toast.LENGTH_SHORT);
        }
    }
};
TelephonyManager.listen(serviceStateListener, PhoneStateListener.LISTEN_SERVICE_STATE);
```

**ServiceState Reference:**
Monitor Data Connection/Activity

- Override `onDataActivity` to track data transfer activity, and `onDataConnectionStateChanged` to request notifications for data connection state changes.

```java
PhoneStateListener dataStateListener = new PhoneStateListener() {
    public void onDataActivity(int direction) {
        switch (direction) {
            case TelephonyManager.DATA_ACTIVITY_IN : break;
            case TelephonyManager.DATA_ACTIVITY_OUT : break;
            case TelephonyManager.DATA_ACTIVITY_INOUT : break;
            case TelephonyManager.DATA_ACTIVITY_NONE : break;
        }
    }
    public void onDataConnectionStateChanged(int state) {
        switch (state) {
            case TelephonyManager.DATA_CONNECTED : break;
            case TelephonyManager.DATA_CONNECTING : break;
            case TelephonyManager.DATA_DISCONNECTED : break;
            case TelephonyManager.DATA_SUSPENDED : break;
        }
    }
};
telephonyManager.listen(dataStateListener, PhoneStateListener.LISTEN_DATA_ACTIVITY | PhoneStateListener.LISTEN_DATA_CONNECTION_STATE);
```
SMS and MMS
Overview

• SMS sends short text messages between mobile phones.
  ➢ Supports sending both text messages and data messages

• MMS (multimedia messaging service) messages have allowed users to send and receive messages that include multimedia attachments such as photos, videos, and audio.

• Using the SMSManager, you can replace the native SMS application to send text messages, react to incoming texts, or use SMS as a data transport layer.

• Use the SEND and SEND_TO actions in Intents to send both SMS and MMS messages using a messaging application installed on the device.
Sending SMS/MMS thru Native App

- Use Intent with `Intent.ACTION_SENDTO` action:
  - Specify a target number using `sms: schema` notation as the Intent data.
  - Include the message you want to send within the Intent payload using an `sms_body extra`.

```java
Intent smsIntent = new Intent(Intent.ACTION_SENDTO, Uri.parse("sms:55512345"));
smsIntent.putExtra("sms_body", "Press send to send me");
startActivity(smsIntent);
```
Sending SMS/MMS thru Native App

- You can also attach files (effectively creating an MMS message) to your messages
  
  ➤ Add an `Intent.EXTRA_STREAM` with the URI of the resource to attach.
  ➤ Set the Intent `type` to the `mime-type` of the attached resource.
  ➤ Use `ACTION_SEND` and include the target phone number as an address extra

```java
// Get the URI of a piece of media to attach.
Uri attached_Uri = Uri.parse("content://media/external/images/media/1");

// Create a new MMS intent
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attached_Uri);
mmsIntent.putExtra("sms_body", "Please see the attached image");
mmsIntent.putExtra("address", "07912355432");
mmsIntent.putExtra(Intent.EXTRA_STREAM, attached_Uri);
mmsIntent.setType("image/png");
startActivity(mmsIntent);
```
Sending SMS Manually

- SMS messaging in Android is handled by the `SmsManager`.
  
  ```java
  SmsManager smsManager = SmsManager.getDefault();
  ```

- Specify the `SEND_SMS` uses-permission.
  
  ```xml
  <uses-permission android:name="android.permission.SEND_SMS"/>
  ```

- Use `sendTextMessage` from the SMS Manager, passing in the address (phone number) of your recipient and the text message you want to send,
  
  ```java
  String sendTo = "5551234";
  String myMessage = "Android supports programmatic SMS messaging!";
  smsManager.sendTextMessage(sendTo, null, myMessage, null, null);
  ```

SMS Manager Reference:

The final two parameters in `sendTextMessage` let you specify Intents to track the transmission and delivery.

Implement and register corresponding Broadcast Receivers that listen for the actions you specify when creating the Pending Intents you pass in `sendTextMessage`.

Intent parameter, `sentIntent`, is fired when the message either is successfully sent or fails to send.

- `Activity.RESULT_OK`
- `SmsManager.RESULT_ERROR_GENERIC_FAILURE`
- `SmsManager.RESULT_ERROR_RADIO_OFF`
- `SmsManager.RESULT_ERROR_NULL_PDU`

The second Intent parameter, `deliveryIntent`, is fired only after the destination recipient receives your SMS message.
String SENT_SMS_ACTION = "SENT_SMS_ACTION";
String DELIVERED_SMS_ACTION = "DELIVERED_SMS_ACTION";

// Create the sentIntent parameter
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI = PendingIntent.getBroadcast(getApplicationContext(),
0, sentIntent, 0);

// Create the deliveryIntent parameter
Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI = PendingIntent.getBroadcast(getApplicationContext(),
0, deliveryIntent, 0);

// Register the Broadcast Receivers
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context _context, Intent _intent) {
        switch (getResultCode()) {
            case Activity.RESULT_OK: [. . . send success actions . . . ]; break;
            case SmsManager.RESULT_ERROR_GENERIC_FAILURE: [. . . generic failure actions . . . ]; break;

            // Other cases...
        }
    }
});
SMS delivery monitoring pattern

case SmsManager.RESULT_ERROR_RADIO_OFF:
    [...] radio off failure actions [...] break;
case SmsManager.RESULT_ERROR_NULL_PDU:
    [...] null PDU failure actions [...] break;
}

registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context _context, Intent _intent) {
        [...] SMS delivered actions [...] 
    }
},
    new IntentFilter(SENT_SMS_ACTION));

// Send the message
smsManager.sendTextMessage(sendTo, null, myMessage, sentPI, deliverPI);
Large SMS Messages

- SMS text messages are normally limited to 160 characters.
- Longer messages need to be broken into a series of smaller parts.

  - `divideMessage` method accepts a string as an input and breaks it into an Array List of messages
  - use the `sendMultipartTextMessage` method on the SMS Manager to transmit the array of messages
  - The `sentIntent` and `deliveryIntent` parameters in the `sendMultipartTextMessage` method are Array Lists that is used to specify different Pending Intents to fire for each message part.

```java
ArrayList<String> messageArray = smsManager.divideMessage(myMessage);
ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();
for (int i = 0; i < messageArray.size(); i++)
    sentIntents.add(sentPI);

csmsManager.sendMultipartTextMessage(sendTo, null,
         messageArray, sentIntents, null);
```
Handling Incoming SMS Messages

• With received SMS, new broadcast Intent is fired with the “android.provider.Telephony.SMS_RECEIVED” action.

• Specify the RECEIVE_SMS manifest permission.

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

• Use the pdu extras key to extract an array of SMS PDUs each of which represents an SMS message

• Call SmsMessage.createFromPdu to convert each PDU byte array into an SMS Message object

  Bundle bundle = intent.getExtras();
  if (bundle != null) {
      Object[] pdus = (Object[]) bundle.get("pdus");
      SmsMessage[] messages = new SmsMessage[pdus.length];
      for (int i = 0; i < pdus.length; i++)
          messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
  }
Example of Incoming SMS Messages

• Register the Broadcast Receiver using an Intent Filter that listens for the android.provider.Telephony.SMS_RECEIVED action string.

```java
final String SMS_RECEIVED = "android.provider.Telephony.SMS_RECEIVED";
IntentFilter filter = new IntentFilter(SMS_RECEIVED);
BroadcastReceiver receiver = new IncomingSMSReceiver(); //defined below
registerReceiver(receiver, filter);
```

• Broadcast Receiver implementation whose onReceive handler checks incoming SMS texts that start with the string @echo, and then sends the same text back to the number that sent it.

```java
public class IncomingSMSReceiver extends BroadcastReceiver {
    private static final String queryString = "@echo";
    private static final String SMS_RECEIVED = "android.provider.Telephony.SMS_RECEIVED";
    ```
public void onReceive(Context _context, Intent _intent) {
    if (_intent.getAction().equals(SMS_RECEIVED)) {
        SmsManager sms = SmsManager.getDefault();
        Bundle bundle = _intent.getExtras();
        if (bundle != null) {
            Object[] pdus = (Object[]) bundle.get("pdus");
            SmsMessage[] messages = new SmsMessage[pdus.length];
            for (int i = 0; i < pdus.length; i++)
                messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);

            for (SmsMessage message : messages) {
                String msg = message.getMessageBody();
                String to = message.getOriginatingAddress();

                if (msg.toLowerCase().startsWith(queryString)) {
                    String out = msg.substring(queryString.length());
                    sms.sendTextMessage(to, null, out, null, null);
                }
            }
        }
    }
}
Simulating Incoming SMS Messages/Calls

- Use the Android debug tools to simulate incoming SMS messages or calls from arbitrary numbers.
Questions?