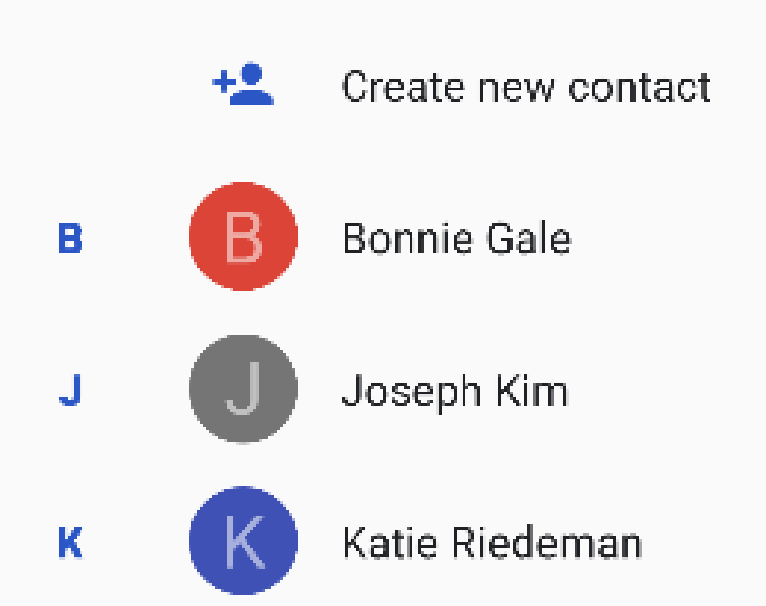
Reading the SMS

Dr. Charles Yu

- Reading the SMS

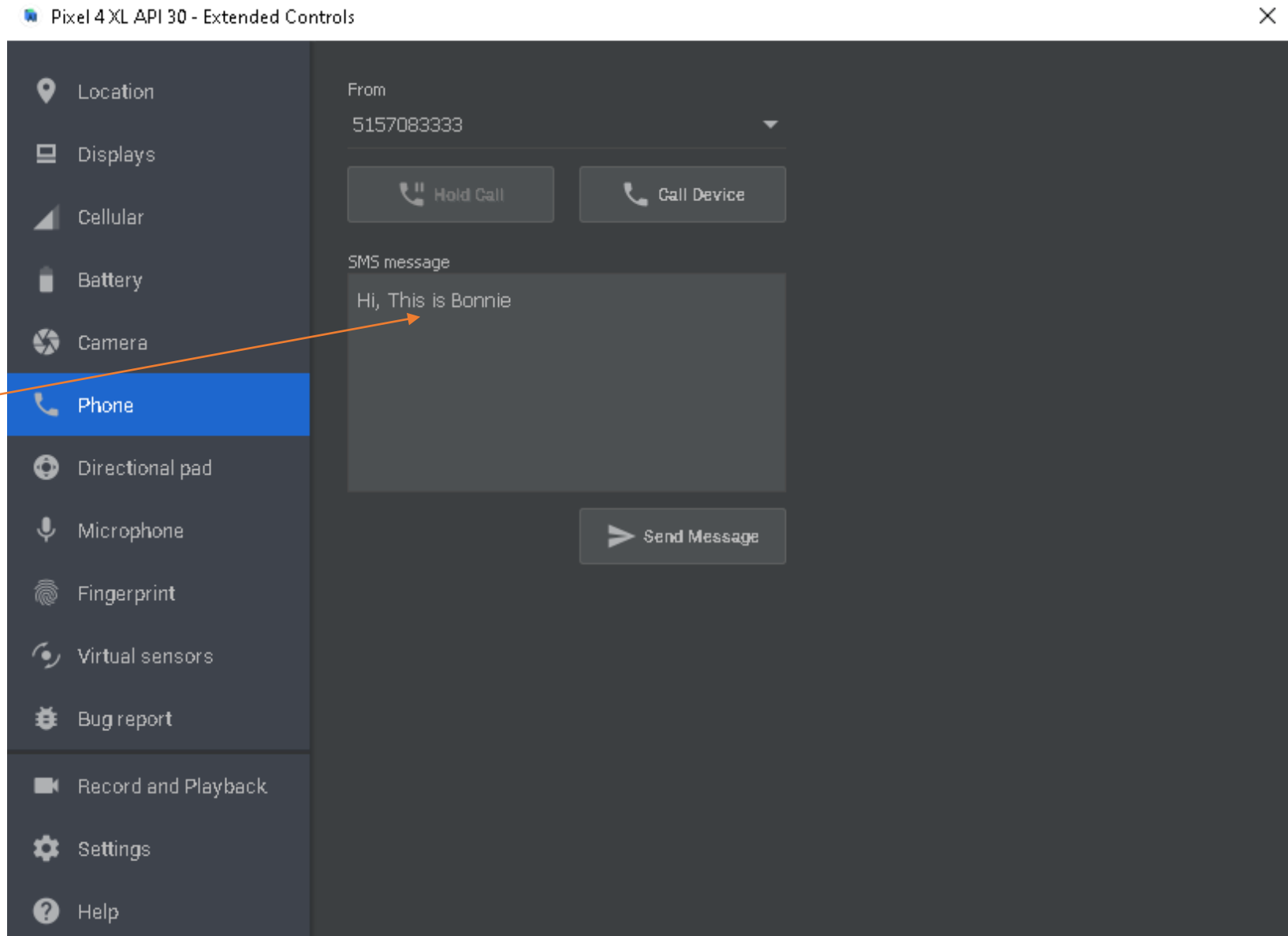
Reading the SMS

- [Demo1]
  - Reading the SMS
- Background
  - So far, I input only 3 person in my contacts



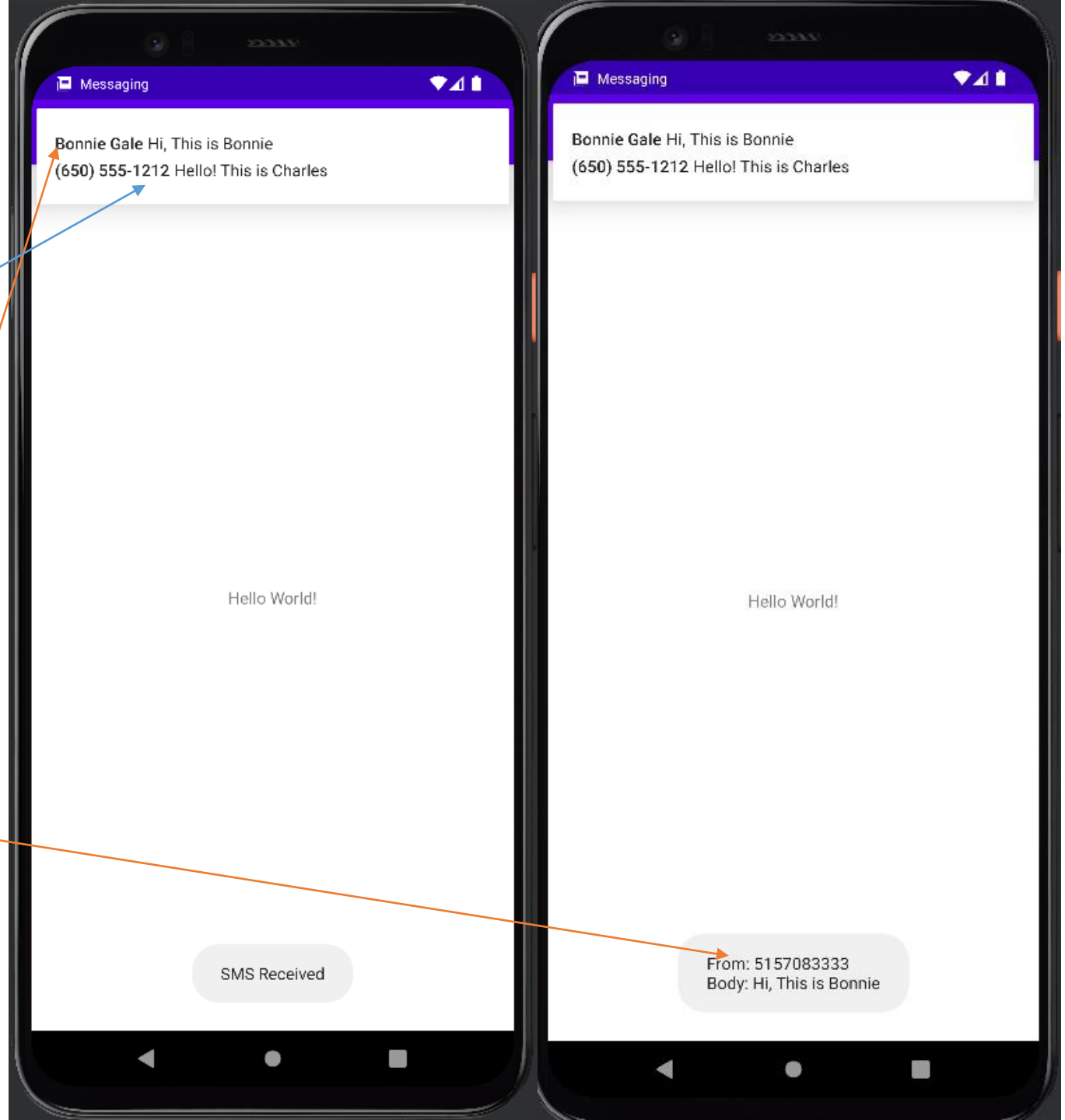
Reading the SMS

- If I use my friend's phone number as "From",
  - and put the message as follows
- Click the "Send Message"



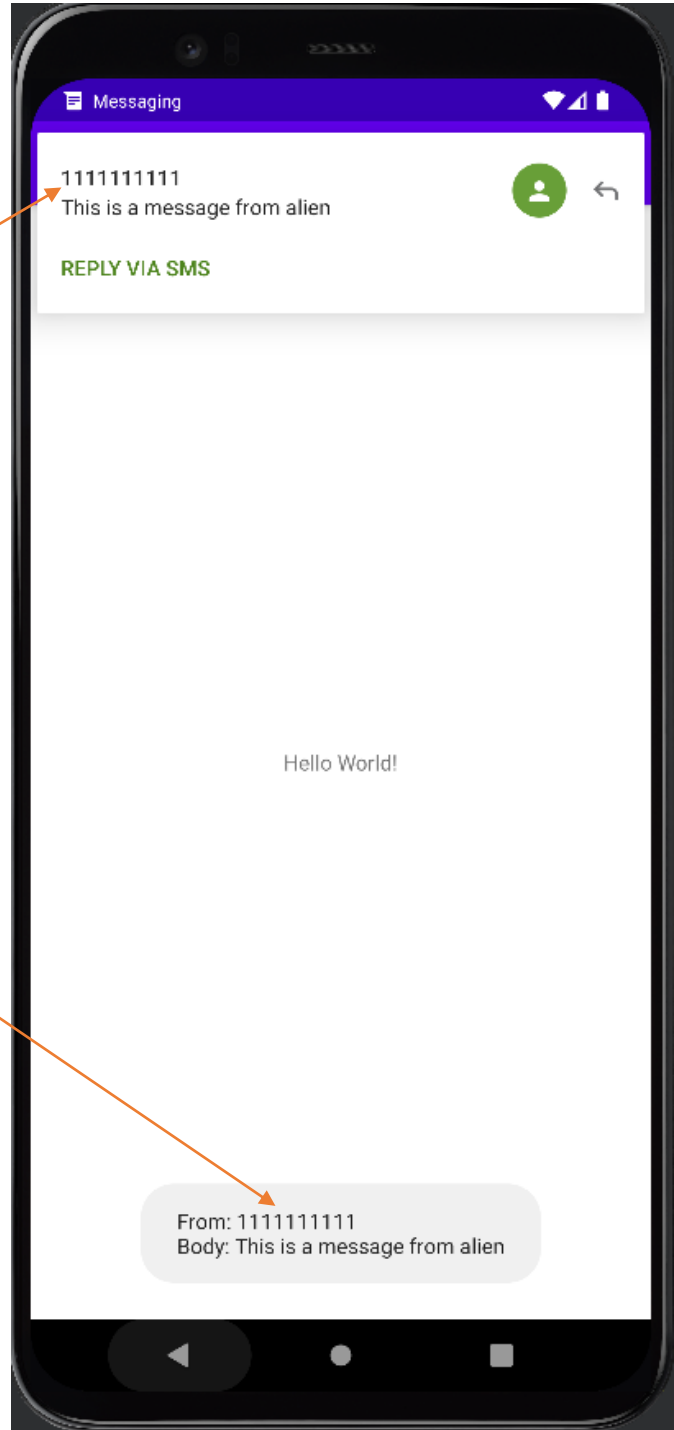
Reading the SMS

- The one in the **bottom** is my **previous** test message.
- Once the mapping of the phone number and the name of the contact **matches**, it can show the name saying **who** send this SMS.
- This is showing the sender, who is in my contact list (Bonnie). Her phone number



Reading the SMS

- What if someone send me a weird message?
  - The person is not in my contact list
  - It is just showing its phone number
- Let’s take a look at the code



Reading the SMS

- Again, we use BroadcastReceiver
  - **AndroidManifest.xml**
- Permission
- This is the **name** of "**action**", we will use that later in our BroadcastReceiver --- SmsReceiver

activity_main.xml | MainActivity.java | AndroidManifest.xml | SmsReceiver.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <!-- <uses-permission android:name="android.permission.READ_SMS" /> -->
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="ReadSMS"
        android:supportsRtl="true"
        android:theme="@style/Theme.ReadSMS"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <receiver android:name=".SmsReceiver"
            android:permission="android.permission.BROADCAST_SMS"
            android:exported="true">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>

    </application>

</manifest>
```

Reading the SMS

- The beginning of the MainActivity
- I use “this.getApplicationContext()” to setup mContext object
- mContext will be used in requesting for runtime permissions

```
2 usages
public class MainActivity extends AppCompatActivity {
    2 usages
    private static final int  MY_PERMISSIONS_REQUEST_RECEIVE_SMS = 0; // This is the request code
    3 usages
    Context mContext;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mContext = this.getApplicationContext();
        grantSMSAccessRights();
    }
```

- grantSMSAccessRights() is the function to ask for run-time SMS receiving access rights
- Request code, which is used in the ActivityCompact.requestPermissions()

Reading the SMS

- MainActivity
- Run-time Permission
- This time, I added a call-back function to **check the status** of permission **after** the **request has been made**.

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mContext = this.getApplicationContext();
    grantSMSAccessRights();
}

1 usage
public void grantSMSAccessRights() {

    if (ContextCompat.checkSelfPermission(mContext, android.Manifest.permission.READ_SMS) != PackageManager.PERMISSION_GRANTED ||
            ContextCompat.checkSelfPermission(mContext, android.Manifest.permission.RECEIVE_SMS) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this,
                new String[]{ android.Manifest.permission.RECEIVE_SMS},
                MY_PERMISSIONS_REQUEST_RECEIVE_SMS);
    }
}

14 usages
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_RECEIVE_SMS:
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText( context: this,  text: "Permission Granted", Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText( context: this,  text: "Permission Not Granted!", Toast.LENGTH_LONG).show();
                this.finish();
            }
    }
}
```

Reading the SMS

- In this callback function, I use the case-switch to **match** my previously defined **request code**. See the slide #8
- If the permission is denied by the user, this is the only one thing I do.
  - Finish this App. Close it up!

```java
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_RECEIVE_SMS:
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                Toast.makeText( context: this,  text: "Permission Granted", Toast.LENGTH_LONG).show();
            } else {
                Toast.makeText( context: this,  text: "Permission Not Granted!", Toast.LENGTH_LONG).show();
                this.finish();
            }
    }
}
```

Reading the SMS

- The way to ask the permission is the same.
  - In this demo, we only need to ask for this:
    - android.Manifest.permission.RECEIVE_SMS
  - In the AndroidManifest.xml, there is a mapping to
    - android.permission.RECEIVE_SMS

```
public void grantSMSAccessRights() {

    if (ContextCompat.checkSelfPermission(mContext, android.Manifest.permission.READ_SMS) != PackageManager.PERMISSION_GRANTED ||
            ContextCompat.checkSelfPermission(mContext, android.Manifest.permission.RECEIVE_SMS) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this,
                new String[]{ android.Manifest.permission.RECEIVE_SMS},
                MY_PERMISSIONS_REQUEST_RECEIVE_SMS);
    }
}
```

Reading the SMS

- It's time to take a look at BroadcastReceiver --- SmsReceiver
- I do not specifically register and unregister the broadcast receiver, **it still can work.**
- Register/Unregister is a good convention.

```java
public class SmsReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, text: "SMS Received", Toast.LENGTH_LONG).show();

        if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")){
            Bundle bundle = intent.getExtras();
            SmsMessage[] msgs;
            String msg_from;
            if (bundle != null) {
                try {

                    /*
                    * Check the SMS PDU
                    * https://developer.android.com/reference/android/telephony/SmsMessage
                    */

                    Object[] pdus = (Object[]) bundle.get("pdus");
                    msgs = new SmsMessage[pdus.length];
                    for (int i = 0; i < msgs.length; i++) {
                        msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                        msg_from = msgs[i].getOriginatingAddress();
                        String msgBody = msgs[i].getMessageBody();

                        Toast.makeText(context, text: "From: " + msg_from + "\n" + "Body: " + msgBody, Toast.LENGTH_LONG).show();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

    }
}
```

Reading the SMS

- Now, if there is an intent is received from the system broadcast, we can **parse** its "**action**" by calling --- getAction()
- Check the slide #7, this **action** is well-defined in the "**intent-filter**", specifying **what kind of actions, this broadcast receiver wants to intercept?**

```
<receiver android:name=".SmsReceiver"
    android:permission="android.permission.BROADCAST_SMS"
    android:exported="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

```java
public class SmsReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, text: "SMS Received", Toast.LENGTH_LONG).show();

        if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")){
            Bundle bundle = intent.getExtras();
            SmsMessage[] msgs;
            String msg_from;
            if (bundle != null) {
                try {

                    /*
                    * Check the SMS PDU
                    * https://developer.android.com/reference/android/telephony/SmsMessage
                    */

                    Object[] pdus = (Object[]) bundle.get("pdus");
                    msgs = new SmsMessage[pdus.length];
                    for (int i = 0; i < msgs.length; i++) {
                        msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                        msg_from = msgs[i].getOriginatingAddress();
                        String msgBody = msgs[i].getMessageBody();

                        Toast.makeText(context, text: "From: " + msg_from + "\n" + "Body: " + msgBody, Toast.LENGTH_LONG).show();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

    }
}
```

Reading the SMS

- If the **action** extracted from the **intent** matches the one in the AndroidManifest.xml, we can use the Bundle to get the "**pdus**" and save it into Object array.
- PDU stands for **Protocol Data Unit**.
- For most of the SMS,

they are supporting 2 modes

  - Text mode
  - PDU mode

- Please check appendix for details
  - About their differences

```java
public class SmsReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, text: "SMS Received", Toast.LENGTH_LONG).show();

        if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")){
            Bundle bundle = intent.getExtras();
            SmsMessage[] msgs;
            String msg_from;
            if (bundle != null) {
                try {

                    /*
                    * Check the SMS PDU
                    * https://developer.android.com/reference/android/telephony/SmsMessage
                    */

                    Object[] pdus = (Object[]) bundle.get("pdus");
                    msgs = new SmsMessage[pdus.length];
                    for (int i = 0; i < msgs.length; i++) {
                        msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                        msg_from = msgs[i].getOriginatingAddress();
                        String msgBody = msgs[i].getMessageBody();

                        Toast.makeText(context, text: "From: " + msg_from + "\n" + "Body: " + msgBody, Toast.LENGTH_LONG).show();
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

    }
}
```

Reading the SMS

- Now, we allocate a SmsMessage object array called msgs.
- msgs is used to populate the data, **parsed** from the Object array, pdus.
- msgs[i] is populated by calling createFromPdv()
- **msg_from** is a **string**, and it is actually the sender's **phone number**. It is obtained by calling **getOriginatingAddress()**
- **msgBody** is another **string**, which is obtained by calling **getMessageBody()**

```
if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")){
    Bundle bundle = intent.getExtras();
    SmsMessage[] msgs;
    String msg_from;
    if (bundle != null) {
        try {

            /*
            * Check the SMS PDU
            * https://developer.android.com/reference/android/telephony/SmsMessage
            */

            Object[] pdus = (Object[]) bundle.get("pdus");
            msgs = new SmsMessage[pdus.length];
            for (int i = 0; i < msgs.length; i++) {
                msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
                msg_from = msgs[i].getOriginatingAddress();
                String msgBody = msgs[i].getMessageBody();

                Toast.makeText(context, text: "From: " + msg_from + "\n" + "Body: " + msgBody, Toast.LENGTH_LONG).show();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Reading the SMS

- You might wonder why we need to do this?
  - When we use the bundle to get the data from “pdus” column, the returned value is actually an Object array.
  - Seriously? Object array is a very low level stuff! (bits and bytes)
    - Raw data
    - Yes, the 1st hand information in your phone, SMS, is raw data! (non-human readable)
  - We need to use the **SmsMessage** class provided by the Android to **construct** the **message**
  - That is why we have these 3 lines

```
msgs[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);
msg_from = msgs[i].getOriginatingAddress();
String msgBody = msgs[i].getMessageBody();
```

Reading the SMS

- Finally, since we only test “a single text message” (SMS), we only have msgs[0]
  - Because there is just one time of a “Toast” about sender’s phone number and message body
- Since the incoming SMS is arriving sporadically, I only need to have a Toast would be good enough.
  - No need for UI displays. i.e. TextView

Appendix

- **SMS Text Mode vs. SMS PDU Mode**
  - https://www.developershome.com/sms/operatingMode.asp
  - https://www.gsmfavorites.com/documents/sms/pdutext/