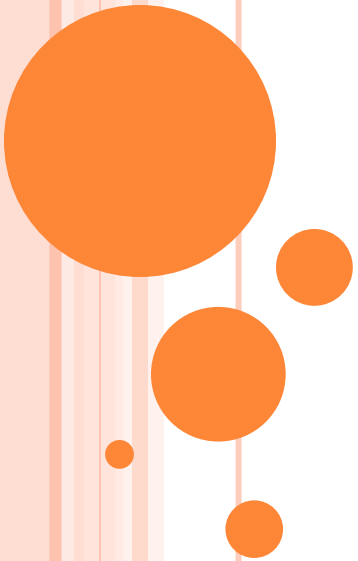

Messaging

Messaging

- Messaging - capability to communicate with the outside world
- Enable you to send sms message to another phone when an event happens
- Events like
  - Geographical location is reached
  - Access a webservice such as currency exchange, weather

Contents

- How to send SMS messages
  - programmatically within your application
  - Using the built-in Messaging application
- How to receive incoming SMS messages
- How to send Email messages from your application

Sending SMS Messages using INTENTS

- To activate the built-in Messaging application from within your application , use an Intent object together with the MIME type ”**vnd.android-dir/mms-sms**”

```
Intent i = new
        Intent(android.content.intent.ACTION_VIEW);
i.putExtra(“address”,”5556;5558;5560”)
i.putExtra(“sms_body”,”hello my friends”)
i.setType(“vnd.android-dir/mms-sms”)
startActivity(i);
```

- You can send SMS to multiple recipients by separating each phone number with semi-colon in the putExtra()
- No permission in AndroidManifest.xml is needed, because your application is ultimately no t the one sending the message

Sending SMS messages Programmatically

- Android has a built-in SMS application that enables to send and receive SMS messages.

- For Example
- Application automatically sends SMS at regular time intervals
- Track location of kids- app sending SMS msg containing the geographical location on every 30 minutes

- So need to programmatically send and receive SMS messages in your Android application

SMS messaging Programmatically

- To send SMS message, use SMS Manager class

  SMSManager sms=SMSManager.getDefault();

- Automatically send an SMS message to a recipient without user intervention (without involving built-in messaging application)

- Need not instantiate this class, call the getDefault() static method to obtain an SMSManager object

- Provide SMS permission in *AndroidManifest .xml*
- *<uses-permission android:name="android.permission.RECEIVE_SMS>*

SMS messaging Programmatically

```
sms.sendTextmessage(phonenumber,null,
                                    message,null,null);
```

- Five arguments to **sendtextmessage()** method

  - **destinationAddress** - phone no of recipient
  - **scAddress** - Service Center Address
  - **Text** – content of the text message
  - **sentIntent** –Pending intent to invoke when the message is sent
  - **deliveryIntent** –Pending intent to invoke when the message has been delivered

Broadcast Receivers

- **Broadcast Receivers** simply respond to broadcast messages from other applications or from the system itself.
- These messages are sometime called events or intents.
- For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use.
- Broadcast receiver is the one who will intercept this communication and will initiate appropriate action.

Broadcast Receivers

- Two important steps to make BroadcastReceiver works for the system broadcasted intents
  - Creating the Broadcast Receiver.
  - Registering Broadcast Receiver

Creating the Broadcast Receiver

- A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the **onReceive()** method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver {
 @Override
    public void onReceive(Context context, Intent intent)
    {
    Toast.makeText(context, "Intent Detected.",
    Toast.LENGTH_LONG).show();
    }
}
```

Registering Broadcast Receiver

- An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file.
- Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.

```
<application android:icon="@drawable/ic_launcher"
   android:label="@string/app_name" android:theme="@style/AppTheme“>
 <receiver android:name="MyReceiver">
 <intent-filter>
<action
   android:name="android.intent.action.BOOT_COMPLETED">
   </action>
</intent-filter>
</receiver>
 </application>
```

whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver* and implemented logic inside *onReceive()* will be executed

Broadcasting Custom Intents

- If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()*method inside your activity class.

```
public void broadcastIntent(View view) {
Intent intent = new Intent();
 intent.setAction("com.example.CUSTOM_INTENT");
 sendBroadcast(intent); }
```

- This intent *com.example.CUSTOM_INTENT* can also be registered in similar way as we have regsitered system generated intent.

Broadcasting Custom Intents

```
<application android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
<receiver android:name="MyReceiver">
<intent-filter>
<action android:name="example.CUSTOM_INTENT">
   </action>
</intent-filter>
</receiver>
</application>
```

Pending Intent

- A Pending Intent specifies an action to take in the future.

- It lets you pass a future Intent to another application and allow that application to execute that Intent as if it had the same permissions as your application, whether or not your application is still around when the Intent is eventually invoked.
- A PendingIntent provides a means for applications to work, even after their process exits. Its important to note that even after the application that created the PendingIntent has been killed, that Intent can still run.

Pending Intent

- To perform a broadcast via a pending intent so get a PendingIntent via PendingIntent.getBroadcast().
- To perform an activity via an pending intent you receive the activity via PendingIntent.getActivity().
- A description of an Intent and target action to perform with it.
- Instances of this class are created with
- getActivity(Context, int, Intent, int),
- getBroadcast(Context, int, Intent, int),
- getService (Context, int, Intent, int);
- Returned object can be handed to other applications so that they can perform the action you described on your behalf at a later time.

Getting feedback after sending a message

- Create two pending intent objects to monitor the status of the SMS message-sending process.
- Pending Intent objects in the onCreate ()

```
sentPI = PendingIntent.getBroadcast(this, 0,
                    new Intent(SENT), 0);
deliveredPI = PendingIntent.getBroadcast(this, 0,
                    new Intent(DELIVERED), 0);
```

- PI objects will be used to send broadcasts later when an SMS message has been sent and delivered

Getting feedback after sending a message

- Pending Intent objects are passed to the last two arguments of the **sendTextMessage()** method
- When a msg is sent correctly or failed to be delivered, then it will be notified of its status via two Pending Intent objects

```
sms.sendTextmessage(phonenumber,null,

                    message, sentPI, deliveredPI);
```

Getting feedback after sending a message

onResume()

- Create and register two Broadcast Receivers
- Intents are fired by the SMSManager when the message has been sent and delivered.
- Broadcast Receivers listen for intents that match SMS_SENT and DELIVERED
- Within each BroadcastReceiver override the OnReceive() method and get the current result code.

onPause ()

- You can unregister the two Broadcastreceivers objects

Receiving SMS Messages

- Receive incoming SMS messages within the application is useful when the application to perform need to perform an action when the a certain SMS message is received.

- Suppose you want to track the location of the phone in case stolen or lost.
  - Create an application that automatically listens for SMS messages containing secret code. Once that SMS is received then send an SMS containing the location's coordinates back to the sender

Receiving SMS Messages

- To listen for the incoming SMS msgs create a **BroadcastReceiver** class,
  - Enable the app to receive intents sent by other applications using the **sendBroadcast()** method.
  - Enable the app to handle events raised by other application
- When an incoming msg is received the OnReceive() method is fired
- If device receives 5 sms msgs , then onReceive () method will be called five times.

Receiving SMS Messages

- To extract the content of each msg , use createFromPDU() method from the SMSMessage class
- Phone no obtained via getOrginatingAddress()-to send autoreply
- Body of the msg via getMessageBody()
- The application will continue to listen to the incoming msgs even if the application is not running, as long as application installed in the device.

Preventing from receiving a message

- To prevent an incoming msg from being handled by the built-in Message application, the app you created needs to handle the msg before the Message app has a chance to do so

- To do this android:priority attribute to the <intent-filter> element like this

  <intent-filter android:priority="100">

- Set this attribute to a high no such as 100. The higher the no the earlier Android executes our application.

- To prevent other applications from seeing the message, call **abortBroadcast()** method of the **BroadcastReceiver** class.

Updating Activity from Broadcast receiver

- To send the SMS message back to the main activity of your application

- For example, you might wish to display the message in a TextView
- When SMSReceiver class receives an SMS message, it will broadcast another Intent object so that any applications listening for this intent can be notified

```
//send a broadcast intent to update the SMS received in the activity---
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("SMS_RECEIVED_ACTION");
broadcastIntent.putExtra("sms", str);
context.sendBroadcast(broadcastIntent);
```

Updating Activity from Broadcast receiver

- Create a BroadcastReceiver object to listen for broadcast intents

```
private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
public void onReceive(Context context, Intent intent) {

//—-display the SMS received in the TextView—-

    TextView SMSes = (TextView) findViewById(R.id.textView1);
    SMSes.setText(intent.getExtras().getString("sms"));
 } };
```

Updating Activity from Broadcast receiver

- When a broadcast intent is received, you update the SMS message in the TextView.
- TextView will display the SMS message only when the message is received while the activity is visible on the screen.
- If the SMS message is received when the activity is not in the foreground, the TextView will not be updated.

Invoking an Activity from a BroadcastReceiver

- The previous example shows how you can pass the SMS message received to be displayed in the activity.

- Situations like your activity may be in the background when the SMS message is received.
- In this case, it would be useful to be able to bring the activity to the foreground when a message is received.

- In the SMSActivity class,
  - register the BroadcastReceiver in the activity's onCreate() event, instead of the onResume() event
  - instead of unregistering it in the onPause() event, unregister it in the onDestroy() event.

- This ensures that even if the activity is in the background, it will still be able to listen for the broadcast intent.

Invoking an Activity from a BroadcastReceiver

- modify the onReceive() event in the SMSReceiver class by using an intent to bring the activity to the foreground before broadcasting another intent

```
Intent mainActivityIntent = new Intent(context, SMSActivity.class);
mainActivityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(mainActivityIntent);
```

- The startActivity() method launches the activity and brings it to the foreground.
- Set the Intent.FLAG_ACTIVITY_NEW_TASK flag because calling startActivity() from outside of an activity context requires the FLAG_ACTIVITY_NEW_TASK flag.

Invoking an Activity from a BroadcastReceiver

- Set the launchMode attribute of the <activity> element in the *AndroidManifest.xml* file to singleTask

<activity android:name=*".MainActivity"*
android:label=*"@string/app_name"*
android:launchMode=*"singleTask"* >

- If you don't set this, multiple instances of the activity will be launched as your application receives SMS messages.

Thank You