Module: Mobile Application development (Android)

Session 52: SMS and Accelerometer Monitoring Assignment Practice

### Intro

### For this lab you will be filling in a new Application that simply demonstrates a few key features of the Android framework. In particular, the application demonstrates how to send SMS text messages and how to monitor motion of the device through the Accelerometer. The purpose of the application is simply to demonstrate functionality. When the application starts up, the user is presented with an Activity that allows them to choose which feature they would like to demo by pressing one of two buttons, either SMS or Accelerometer monitoring. When the user hits one of the buttons, it launches the Activity for the selected demo.

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1268252334346/labs/lab-7/Lab7.png?height=400&width=267](https://sites.google.com/site/androidcoursearchive/labs/lab-7/Lab7.png?attredirects=0)

### Objectives

### At the end of this lab you will be expected to know:

### How to send SMS text messages.

### How to listen for changes in the status of a sent SMS text message.

### How to register to receive information from a device's available sensors.

### How to monitor the motion of a physical device.

### Activities

### For this lab you will be working with a brand new application, completely independent of the previous labs. Once you have a general understanding of the components of the application, you will begin implementing it.

### *IMPORTANT:*

### *You will be given a Skeleton Project to work with. This project contains all of the java and resource files you will need to complete the lab. Some method stubs, member variables, and resource values and ids have been added as well. It is important that you not change the names of these methods, variables, and resource values and ids. These are given to you because there are unit tests included in this project as well that depend on these items being declared exactly as they are. These units test will be used to evaluate the correctness of your lab. You have complete access to these test cases during development, which gives you the ability to run these tests yourself. In fact, you are encouraged to run these tests to ensure that your application is functioning properly.*

### 1. Setting Up...

### 1.1 Creating the Project

### To begin, you will need to download and extract the skeleton project for the Lab 7 application.

### [Click Here](https://sites.google.com/site/androidcoursearchive/resources/lab-skeletons/Lab7_v2_partial_stub.zip?attredirects=0) to download the skeleton project.

### Extract the project, making sure to preserve the directory structure.

### Take note of the path to the root folder of the skeleton project.

### Next you will need to setup an Android project for this app.  Use the settings listed below for the remaining fields:

### Project Name:  lab7<userid>

### Build Target:  Android 1.6

### Application Name:  Lab7

### 1.2 Familiarize Yourself with the Project

### The project contains three java files and three XML layout files. Lab7.java defines a simple Activity that allows you to choose which demo you want to start. The Lab7 class makes use of a very simple XML layout file called main.xml. Both Lab7.java and main.xml have been completed for you.

### SMSActivity.java defines a simple SMS text messaging Activity. It allows the user to enter a phone number, a textual message, and send the an SMS text message. It also listens to the status of the sent SMS text message and updates the UI to display changes to the status. The SMSActivity class makes use of an XML layout file called sms.xml which has been completed for you. SMSActivity.java has been partially completed. The UI components have been hooked up and initialized. It will be your job to implement functionality for sending the text message and responding to status changes.

### When the device moves, the Accelerometer sensor in the device measures that movement. AccelActivity.java defines a simple Accelerometer monitoring Activity. It only displays measurements from the device's Accelerometer to the user. It makes use of a simple XML layout file called accel.xml which has been completed for you. AccelActivity.java has been partially completed for you. It is your job to fill in the code necessary to listen for measurements from the Accelerometer and update the display.

### 2. SMS Activity

### In this section of the lab, you will be learning how to send an SMS text message and listen for changes in its delivery status. All the work done in this section will be performed in the SMSActivity.java file. Access to SMS is done through the android.telephony.SmsManager class. It provides an interface for sending both text and data messages, as well as listening for status changes of sent messages and delivery of incoming messages. For this lab, we will only be concerned with sending text messages and monitoring the status of those messages. These exercises should give you enough information to start working with SMS on your own. Data messages are sent in almost the same way as text messages, and monitoring of incoming messages is done through BroadcastReceivers. Both should be fairly straightforward to figure out on your own.

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1268252356692/labs/lab-7/SMSTextMessage.png?height=297&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-7/SMSTextMessage.png?attredirects=0)

### 2.1 Sending an SMS Text Message

### For this subsection, you must simply send an SMS text message.

### 2.1.1 Fill in sendSMS()

### When the send button is pressed, the sendSMS() method is called. This method should send an SMS text message to the phone number contained in the EditText with the Id of R.id.messageNumber. The text of the message should be retrieved from the EditText with the Id of R.id.messageText.

### You need to retrieve an instance of SMSManager by calling its static getDefault() method.

### You can then use one of its send methods to send a text message.

### Don't worry about the PendingIntent arguments just yet, you can pass in null. In the next section you will set these up so that you can respond to status changes.

### Read the Android Documentation on [SMSManager](http://developer.android.com/reference/android/telephony/SmsManager.html)for more details on these methods.

### 

### 2.1.2 Testing SMS with the Emulator

### If you don't have a SIM card for your device, or if you don't have a device you can test sending a text message on the Emulator. You will need to start two separate emulator instances. From there, you can specify the port number of the recipient Emulator as the phone number. The ADB will then deliver the text message to the other Emulator:

### For more details on this, see the Android Developer Guide on [Sending SMS to Another Emulator](http://developer.android.com/guide/developing/tools/emulator.html#calling) and [Emulating SMS](http://developer.android.com/guide/developing/tools/emulator.html#sms)

### You should be able to run your application, and verify that it can send a text message.

### 

### 2.2 Listening for SMS Status Changes

### For this subsection, you will implement the functionality necessary to monitor and respond to changes to the status of your sent text message.

### 2.2.1 Fill in BroadcastReceivers

Status changes of a sent SMS message are reported through Intent broadcasts. Therefore, in order to respond to these changes you will need a BroadcastReceiver, two in fact. There are two separate status changes that get broadcasted. The first status change gets broadcast once the message has been sent. The second is broadcast once the message has been delivered.

### Fill in the MessageSentReceiver.onReceive(...) method so that it updates the text of the TextView with the Id of R.id.statusText:

### Check the result code.

### If it represents success then set the text to R.string.sent.

### If it represents failure then set the text to R.string.error.

### Read the documentation on [SMSManager.sendTextMessage(...)](http://developer.android.com/reference/android/telephony/SmsManager.html" \l "sendTextMessage(java.lang.String,%20java.lang.String,%20java.lang.String,%20android.app.PendingIntent,%20android.app.PendingIntent)) for details on how to interpret the result code.

### Fill in the MessageDeliveredReceiver.onReceive(...) method so that it sets the text of the R.id.statusText TextView to R.string.delivered.

### 2.2.2 Fill in initStatusReceivers()

### The initStatusReceivers() method is called from onCreate(...). You need to register your new receivers to listen for status change broadcasts in this method.

### You will have to create IntentFilters to listen for SMSActivity.SMS\_SENT and SMSActivity.SMS\_DELIVERED action strings.

### Read the Android Documenation on [Context.registerReceiver(...)](http://developer.android.com/reference/android/content/Context.html" \l "registerReceiver(android.content.BroadcastReceiver,%20android.content.IntentFilter))if you forgot how to do this from the last lab.

### 

### 2.2.3 Setup PendingIntents

### The following is to be performed in the sendSMS() method. You will need to create two PendingIntent objects and pass them into your SMSManager.sendTextMessage(...) method call. PendingIntents are a way of creating an Intent to execute at a later time. They can either start a service, start an activity, or send a broadcast.

### Read the static PendingIntent.getBroadcast(...) method for details on how to obtain a PendingIntent.

### You will need to initialize both SMSActivity.pendingSent and SMSActivity.pendingDelivered.

### pendingSent should be initialized with an Intent containing the SMSActivity.SMS\_SENT action string.

### pendingDelivered should be initialized with an Intent containing the SMSActivity.SMS\_DELIVERED action string.

### 

### 2.2.4 Cancel PendingIntents

### When the user hits the reset button, the UI will be reset and the SMSActivity should no longer receive udpates from the message it has previously sent. When the reset button is pressed the cancelPendingIntents() method is called. Fill in this method so that it cancels the SMSActivity's PendingIntents.

### You should now be able to run your application, send a text message and verify that the TextView displaying the status gets updated properly.

### Note: You may not get notified when the message is delivered if you're sending texts through the emulator, don't worry about this.

### 

### 3. Accelerometer

### In this section of the lab, you will be learning how to monitor changes to the Accelerometer sensor. All work in this section will be performed in the AccelActivity.java file. As is usually the case, you will not be interacting with the Accelerometer sensor directly, but will instead be using the android.hardware.SensorManager class. This class provides you with a set of API's for retrieving Sensor objects, querying Sensors for measurements, and registering listeners with Sensors. The Sensor objects themselves do not provide you with measurements. Sensor objects only provide hardware information like vendor, resolution, and range.

### There are a number of different types of Sensors from which data can be obtained. Examples include Magnetic, Light, and Temperature. The ones that you can access depend entirely on which Sensors the device actually has. For this lab you will only be interacting with the Accelerometer Sensor. Interaction with other sensors is performed in the same manner and this will give you a good starting point to learn the rest on your own.

### [https://sites.google.com/site/androidcoursearchive/_/rsrc/1268252375228/labs/lab-7/AccelerometerValues.png?height=400&width=266](https://sites.google.com/site/androidcoursearchive/labs/lab-7/AccelerometerValues.png?attredirects=0)

### 3.1 Monitoring Device Motion

### In the subsections that follow, you register the AccelActivity to be notified of any changes to the Accelerometer. When notified of changes you will simply display the raw data to the screen. There Android SDK does not currently provide a method for simulating the Accelerometer on the Emulator, so the following section must be tested on a physical device to ensure that it is working properly.

### 3.1.1 Fill in registerWithSensor()

### Retrieve an instance of the SensorManager and use it to register this instance of AccelActivity as a SensorEventListener for the Accelerometer Sensor.

### The SensorManager can be obtained by calling getSystemService(Context.SENSOR\_SERVICE) and casting.

### Read the Android Documentation on [SensorManager.registerListener(...)](http://developer.android.com/reference/android/hardware/SensorManager.html" \l "registerListener(android.hardware.SensorEventListener,%20android.hardware.Sensor,%20int)) for information on how to register a SensorEventListener.

### You will need to obtain an instance of the Sensor you want to register with. You will need to use one of the SensorManger's get methods to do this as you cannot instantiate one directly.

### 

### 3.1.2 Fill In onSensorChanged(...)

### 

### When a sensor changes, and thus has a new measurement, the SensorManager will call SensorChangeListener.onSensorChanged(...) on all listeners that have registered with Sensor. It will pass in a SensorEvent object which contains data about the change. You need to fill in the onSensorChanged(...) method so that the new data is displayed on the screen:

### Read the AndroidDocumentation on the [SensorEvent.values](http://developer.android.com/reference/android/hardware/SensorEvent.html" \l "values) field to learn how to get the new data.

### Update the R.id.accelerationX, R.id.accelerationY, and R.id.accelerationZ TextFields to display the new data.

### Run your application to ensure that the Accelerometer values are properly updated on the screen. Remember, you have to do this on a physical device.