

CSE 162 - Lab 2

Sensor Programming

Goal

- Display the gravity and magnetic sensor measurements in real time

Sensor Programming

- Determine which sensors are available on a device.
- Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.
- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.
- Register and unregister sensor event listeners that monitor sensor changes.

Getting the Relevant System Service

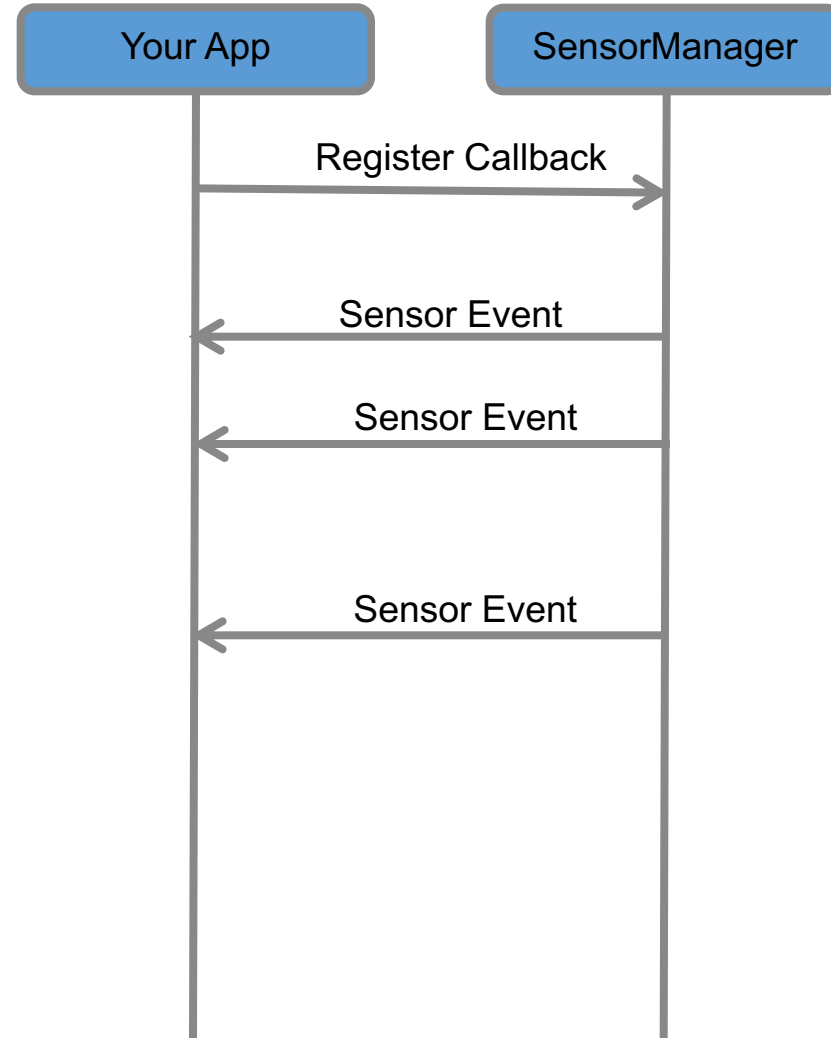
- The non-media (e.g. not camera) sensors are managed by a variety of XXXXManager classes:
 - **SensorManager (accelerometer, gyro, proximity, light, temp)**
- The first step in registering is to obtain a reference to the relevant manager
- Every Activity has a getSystemService() method that can be used to obtain a reference to the needed manager

```
public class MyActivity ... {  
  
    private SensorManager sensorManager_;  
  
    public void onCreate() {  
        ...  
  
        sensorManager_ = (SensorManager) getSystemService(SENSOR_SERVICE);  
    }  
  
}
```

Async Callbacks

Android's sensors are controlled by external services and only send events when they choose to

- An app must register a callback to be notified of a sensor event
- Each sensor has a related XXXListener interface that your callback must implement



Registering for Sensor Updates

- The SensorManager handles registrations for
 - Accelerometer, Temp, Light, Gyro
- In order for an object to receive updates from a sensor, it must implement the SensorEventListener interface
- Once the SensorManager is obtained, you must obtain a reference to the specific sensor you are interested in updates from
- The arguments passed into the registerListener method determine the sensor that you are connected to and the rate at which it will send you updates

How to register for sensor updates?

```
23
24     sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
25     gravity = sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
26     sensorManager.registerListener( listener: this, gravity, SensorManager.SENSOR_DELAY_NORMAL);
27
```

The SensorEventListener Interface

- How to obtain the sensor measurements, implement logics to respond to sensor changes?
- onSensorChanged()
- onAccuracyChanged()

```
28
29      @Override
30      public final void onAccuracyChanged(Sensor sensor, int accuracy){
31          //do something here if sensor accuracy changes
32      }
33
34      @Override
35      public final void onSensorChanged(SensorEvent sensorEvent){
36          final DecimalFormat df = new DecimalFormat( pattern: "0.00");
37          if (sensorEvent.sensor.getType() == Sensor.TYPE_GRAVITY) {
38              float xaccel = sensorEvent.values[0];
39              float yaccel = sensorEvent.values[1];
40              float zaccel = sensorEvent.values[2];
41
42              EditText grav_x = findViewById(R.id.gravValue_x);
43              grav_x.setText( df.format(xaccel)+ "m/s\u00B2");
44
45              EditText grav_y = findViewById(R.id.gravValue_y);
46              grav_y.setText( df.format(yaccel)+ "m/s\u00B2");
47
48              EditText grav_z = findViewById(R.id.gravValue_z);
49              grav_z.setText( df.format(zaccel)+ "m/s\u00B2");
50          }
51      }
52
53  }
```

- onSensorChanged is called when a registered sensor changes value
- SensorEvent contains key information including the sensor type and the actual sensor measurement values.

```
33
34      @Override
35      public final void onSensorChanged(SensorEvent sensorEvent){
36          final DecimalFormat df = new DecimalFormat( pattern: "0.00");
37          if (sensorEvent.sensor.getType() == Sensor.TYPE_GRAVITY) {
38              float xaccel = sensorEvent.values[0];
39              float yaccel = sensorEvent.values[1];
40              float zaccel = sensorEvent.values[2];
41
42              EditText grav_x = findViewById(R.id.gravValue_x);
43              grav_x.setText( df.format(xaccel)+ "m/s\u00B2");
44
45              EditText grav_y = findViewById(R.id.gravValue_y);
46              grav_y.setText( df.format(yaccel)+ "m/s\u00B2");
47
48              EditText grav_z = findViewById(R.id.gravValue_z);
49              grav_z.setText( df.format(zaccel)+ "m/s\u00B2");
50          }
51
52
53      }
```

How to register for multiple sensors?

Here we get two sensors from the sensorManager (gravity & Light)

```
20  @Override
21  public final void onCreate(Bundle savedInstanceState) {
22      super.onCreate(savedInstanceState);
23      setContentView(R.layout.activity_main);
24
25      sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
26      gravity = sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
27      light = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
28
29  }
```

Here we register two sensors using sensorManager.registerListener (gravity & Light)

```
64  }
65
66  @Override
67  protected void onResume() {
68      //register a listener for the sensor
69      super.onResume();
70      sensorManager.registerListener(listener: this, gravity, SensorManager.SENSOR_DELAY_NORMAL);
71      sensorManager.registerListener(listener: this, light, SensorManager.SENSOR_DELAY_NORMAL);
72  }
73
```

Like light sensor & gravity sensor, magnetic sensor is denoted by
“ Sensor.TYPE_MAGNETIC_FIELD ”

onSensorChanged() implements how to collect the values and to display them in the activity xml

```
36 @Override
37 public final void onSensorChanged(SensorEvent sensorEvent){
38     final DecimalFormat df = new DecimalFormat( pattern: "0.00");
39     if (sensorEvent.sensor.getType() == Sensor.TYPE_GRAVITY) {
40         float xaccel = sensorEvent.values[0];
41         float yaccel = sensorEvent.values[1];
42         float zaccel = sensorEvent.values[2];
43
44         EditText grav_x = findViewById(R.id.gravValue_x);
45         grav_x.setText( df.format(xaccel)+ " m/s\u00B2");
46
47         EditText grav_y = findViewById(R.id.gravValue_y);
48         grav_y.setText( df.format(yaccel)+ " m/s\u00B2");
49
50         EditText grav_z = findViewById(R.id.gravValue_z);
51         grav_z.setText( df.format(zaccel)+ " m/s\u00B2");
52     }
53
54     if (sensorEvent.sensor.getType() == Sensor.TYPE_LIGHT) {
55         float light = sensorEvent.values[0];
56
57
58         EditText grav_x = findViewById(R.id.lightValue);
59         grav_x.setText( df.format(light)+ " lux");
60     }
61 }
62
```

onPause() simply unregisters the sensors if there is no change in the readings.

```
74 @Override
75 protected void onPause() {
76     //be sure to unregister the sensor when the activity pauses
77     super.onPause();
78     sensorManager.unregisterListener(this);
79 }
80
```

How to Update the GUI with Sensor Data

- findViewById()

```
41  
42     EditText grav_x = findViewById(R.id.gravValue_x);  
43     grav_x.setText(df.format(xaccel)+ "m/s\u00B2");  
44
```

How to Update the GUI with Sensor Data

- Remember to write the corresponding text widget
- in activity_main.xml, include

```
21      <EditText
22          android:id="@+id/gravValue_x"
23          android:layout_width="wrap_content"
24          android:layout_height="wrap_content"
25          android:layout_below="@+id/textView"
26          android:layout_centerHorizontal="true"
27          android:layout_marginTop="16dp"
28          android:inputType="textMultiLine"
29          android:minWidth="48dp"
30          android:minHeight="48dp"
31          app:layout_constraintBottom_toBottomOf="parent"
32          app:layout_constraintHorizontal_bias="0.17"
33          app:layout_constraintLeft_toLeftOf="parent"
34          app:layout_constraintRight_toRightOf="parent"
35          app:layout_constraintTop_toTopOf="parent"
36          app:layout_constraintVertical_bias="0.071" />
```

Sensor collector app

