CHAPTER 16
Accelerometer

Accelerometers are one of the more interesting bits of hardware in smartphones. Earlier devices such as the Openmoko Neo FreeRunner smartphone and the original Apple iPhone included them. Before Android was released I was advocating for Openmoko at open source conferences.

One of my favorite imaginary applications was private key generation. Adhering to the theory that "When privacy is outlawed, only outlaws will have privacy," several people were talking about this as early as 2008 (when I presented the idea at the Ontario Linux Fest). The idea is: if you can't or don't want to exchange private keys over a public channel, you meet on a street corner and shake hands—with each hand having a cell phone concealed in the palm. The devices are touching each other, thus their sensors should record exactly the same somewhat random motions. With a bit of mathematics to filter out the leading and trailing motion of the hands moving together, both devices should have quite a few bits' worth of identical, random data that nobody else has—just what you need for crypto key exchange. I've yet to see anybody implement this, but I must admit I still hope somebody will come through.

Meanwhile, we have many other recipes on accelerometers and other sensors in this chapter…

16.1 Checking for the Presence or Absence of a Sensor

Rachee Singh

Problem

You want to use a given sensor. Before using an Android device for a sensor-based application, you should ensure that the device supports the required sensor.

Solution

Check for the availability of the sensor on the Android device.

Discussion

The `SensorManager` class is used to manage the sensors available on an Android device. So we require an object of this class:

```
SensorManager deviceSensorManager =
    (SensorManager) getSystemService(SOME_SENSOR_SERVICE);
```

Then, using the `getSensorList()` method, we check for the presence of sensors of any type (accelerometer, gyroscope, pressure, etc.). If the list returned contains any elements, this implies that the sensor is present. A `TextView` is used to show the result: either "`Sensor present!`" or "`Sensor absent.`" Example 16-1 shows the code.

Example 16-1. Checking for the accelerometer

```
List<Sensor> sensorList =
        deviceSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);

if (sensorList.size() > 0) {
    sensorPresent = true;
    sensor = sensorList.get(0);

}
else
    sensorPresent = false;

/* Set the face TextView to display sensor presence */
face = (TextView) findViewById(R.id.face);

if (sensorPresent)
    face.setText("Sensor present!");
else
    face.setText("Sensor absent.");
```

16.2 Using the Accelerometer to Detect Shaking

Thomas Manthey

Problem

Sometimes it makes sense to evaluate not only onscreen input, but also gestures like tilting or shaking the phone. You need to use the accelerometer to detect whether the phone has been shaken.

Solution

Register with the accelerometer and compare the current acceleration values on all three axes to the previous ones. If the values have repeatedly changed on at least two axes and those changes exceed a high enough threshold, you can clearly determine shaking.

Discussion

Let's first define *shaking* as a fairly rapid movement of the device in one direction followed by a further movement in another direction, typically (but not necessarily) the opposite one. If we want to detect such a shake motion in an Activity, we need a connection to the hardware sensors; those are exposed by the class `SensorManager`. Furthermore, we need to define a `SensorEventListener` and register it with the `SensorManager`. So the source of our Activity starts as shown in Example 16-2.

Example 16-2. ShakeActivity—getting accelerometer data

```
public class ShakeActivity extends Activity {
    /* The connection to the hardware */
    private SensorManager mySensorManager;

    /* The SensorEventListener lets us wire up to the real hardware events */
    private final SensorEventListener mySensorEventListener =
            new SensorEventListener() {

        public void onSensorChanged(SensorEvent se) {
            /* We will fill this one later */
        }

        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            /* Can be ignored in this example */
        }
    };

    ....
```

In order to implement `SensorEventListener`, we have to implement two methods: `onSensorChanged(SensorEvent se)` and `onAccuracyChanged(Sensor sensor, int accuracy)`. The first one gets called whenever new sensor data is available, and the second one whenever the accuracy of the measurement changes—for example, when the location service switches from GPS to network-based. In our example, we just need to cover `onSensorChanged()`.

Before we continue, let's define some more variables, which will store the information about values of acceleration and some state (see Example 16-3).

Example 16-3. Variables for acceleration

```
/* Here we store the current values of acceleration, one for each axis */
private float xAccel;
private float yAccel;
private float zAccel;

/* And here the previous ones */
private float xPreviousAccel;
private float yPreviousAccel;
private float zPreviousAccel;

/* Used to suppress the first shaking */
private boolean firstUpdate = true;

/* What acceleration difference would we assume as a rapid movement? */
private final float shakeThreshold = 1.5f;

/* Has a shaking motion been started (one direction)? */
private boolean shakeInitiated = false;
```

I hope that the names and comments explain enough about what is stored in these variables; if not, it will become clearer in the next steps.

Now let's connect to the hardware sensors and wire up their events. `onCreate()` is the perfect place to do so (see Example 16-4).

Example 16-4. Initializing for accelerometer data

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mySensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE); ❶
    mySensorManager.registerListener(mySensorEventListener, mySensorManager
            .getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_NORMAL); ❷
}
```

❶ We get a reference to Android's sensor service.

❷ We register the previously defined `SensorEventListener` with the service. More precisely, we register only for events of the accelerometer and for a normal update rate; this could be changed if we needed to be more precise.

Now let's define what we want to do when new sensor data arrives. We have defined a stub for the `SensorEventListener`'s method `onSensorChanged()`, so now we will fill it with some life (see Example 16-5).

Example 16-5. Using the sensor data

```
public void onSensorChanged(SensorEvent se) {
    updateAccelParameters(se.values[0], se.values[1], se.values[2]); ❶
    if ((!shakeInitiated) && isAccelerationChanged()) {  ❷
    shakeInitiated = true;
    } else if ((shakeInitiated) && isAccelerationChanged()) { ❸
        executeShakeAction();
    } else if ((shakeInitiated) && (!isAccelerationChanged())) { ❹
        shakeInitiated = false;
}
}
```

❶ We copy the values of acceleration that we received from the `SensorEvent` into our state variables. The corresponding method is declared like this:

```
/* Store acceleration values from sensor */
private void updateAccelParameters(float xNewAccel, float yNewAccel,
        float zNewAccel) {
    /* We have to suppress the first change of acceleration,
     * it results from first values being initialized with 0 */
    if (firstUpdate) {
        xPreviousAccel = xNewAccel;
        yPreviousAccel = yNewAccel;
        zPreviousAccel = zNewAccel;
        firstUpdate = false;
    } else {
        xPreviousAccel = xAccel;
        yPreviousAccel = yAccel;
        zPreviousAccel = zAccel;
    }
    xAccel = xNewAccel;
    yAccel = yNewAccel;
    zAccel = zNewAccel;
}
```

❷ We test for a rapid change of acceleration and whether any has happened before; if not, we store the information that now has been gathered.

❸ We test again for a rapid change of acceleration, this time with the information from before. If this test is `true`, we can assume a shaking movement according to our definition and commence action.

❹ Finally, we reset the shake status if we detected shaking before but don't get a rapid change of acceleration anymore.

To complete the code, we add the last two methods. The first is the `isAccelerationChanged()` method (see Example 16-6).

Example 16-6. The isAccelerationChanged() method

```
/* If the values of acceleration have changed on at least two axes,
   we are probably in a shake motion */
private boolean isAccelerationChanged() {
    float deltaX = Math.abs(xPreviousAccel - xAccel);
    float deltaY = Math.abs(yPreviousAccel - yAccel);
    float deltaZ = Math.abs(zPreviousAccel - zAccel);
    return (deltaX > shakeThreshold && deltaY > shakeThreshold)
            || (deltaX > shakeThreshold && deltaZ > shakeThreshold)
            || (deltaY > shakeThreshold && deltaZ > shakeThreshold);
}
```

Here we compare the current values of acceleration with the previous ones, and if at least two of them have changed above our threshold, we return `true`.

The last method is `executeShakeAction()`, which does whatever we wish to do when the phone is being shaken:

```
private void executeShakeAction() {
    /* Save the cheerleader, save the world
       or do something more sensible... */
}
```

Source Download URL

The source code for this project is in the Android Cookbook repository, in the subdirectory *SensorShakeDetection* (see "Getting and Using the Code Examples" on page 18).

16.3 Checking Whether a Device Is Facing Up or Down

Rachee Singh

Problem

You want to check for the orientation (facing up/facing down) of the Android device.

Solution

Use a `SensorEventListener` to check for appropriate accelerometer values.

Discussion

To implement a `SensorEventListener`, the `onSensorChanged()` method is called when sensor values change. Within this method, we check to see if the values lie within a particular range for the device to be facing down or facing up. Here is the code to obtain the sensor object for an accelerometer:

```
    List<android.hardware.Sensor> sensorList =
        deviceSensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
    sensor = sensorList.get(0);
```

Example 16-7 shows the `SensorEventListener` implementation.

Example 16-7. The SensorEventListener implementation

```
private SensorEventListener accelerometerListener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        float z = event.values[2];
        if (z >9 && z < 10)
            face.setText("FACE UP");
        else if (z > -10 && z < -9)
            face.setText("FACE DOWN");
    }

    @Override
    public void onAccuracyChanged(Sensor arg0, int arg1) {

    }

};
```

After implementing the listener along with the methods required, we need to register the listener for a particular sensor (which in our case is the accelerometer). `sensor` is an object of the `Sensor` class; it represents the sensor being used in the application (the accelerometer):

```
deviceSensorManager.registerListener(accelerometerListener, sensor, 0, null);
```

Source Download URL

The source code for this project is in the Android Cookbook repository, in the subdirectory *SensorUpOrDown* (see "Getting and Using the Code Examples" on page 18).

16.4 Reading the Temperature Sensor

Rachee Singh

Problem

You need to get temperature values using the temperature sensor.

Solution

Use `SensorManager` and `SensorEventListener` to track changes in temperature values detected by the temperature sensor.

Discussion

We need to create an object of the `SensorManager` class to use sensors in an application. Then we register a listener with the type of sensor we require. To register the listener we provide the name of the listener, a `Sensor` object, and the type of delay (in this case, `SENSOR_DELAY_FASTEST`) to the `registerListener()` method. In this listener, within the overridden `onSensorChanged()` method, we can print the temperature value into a `TextView` named `tempVal`:

```
    SensorManager sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
    sensorManager.registerListener(temperatureListener,
        sensorManager.getDefaultSensor(Sensor.TYPE_TEMPERATURE),
        SensorManager.SENSOR_DELAY_FASTEST);
```

Example 16-8 shows the `SensorEventListener` implementation.

Example 16-8. The SensorEventListener implementation

```
private final SensorEventListener temperatureListener = new SensorEventListener() {
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
    @Override
    public void onSensorChanged(SensorEvent event) {

        tempVal.setText("Temperature is:"+event.values[0]);

    }
};
```

See Also

Recipe 16.1.