Mobile solutions

Kocsis Gergely

2019.09.16.

Sensor Types

Motion sensors

TYPE_ACCELEROMETER, TYPE_GRAVITY, TYPE_GYROSCOPE, TYPE_ROTATION_VECTOR

Environmental sensors

TYPE_TEMPERATURE/TYPE_AMBIENT_TEMPERATURE, TYPE_PRESSURE, TYPE_LIGHT, TYPE_RELATIVE_HUMIDITY

Position sensors

TYPE_MAGNETIC_FIELD, TYPE_ORIENTATION

Basic classes

SensorManager
You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

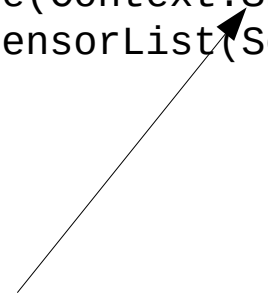
Sensor
You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

SensorEvent
The system uses this class to create a sensor event object, which provides information about a sensor event. A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

SensorEventListener
You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

List sensors

```
private SensorManager sensorManager;
...
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);

tv.setText(…);
```



Here we get a service. Take a look at the possible other services.

Sample class for ServiceEvent-s

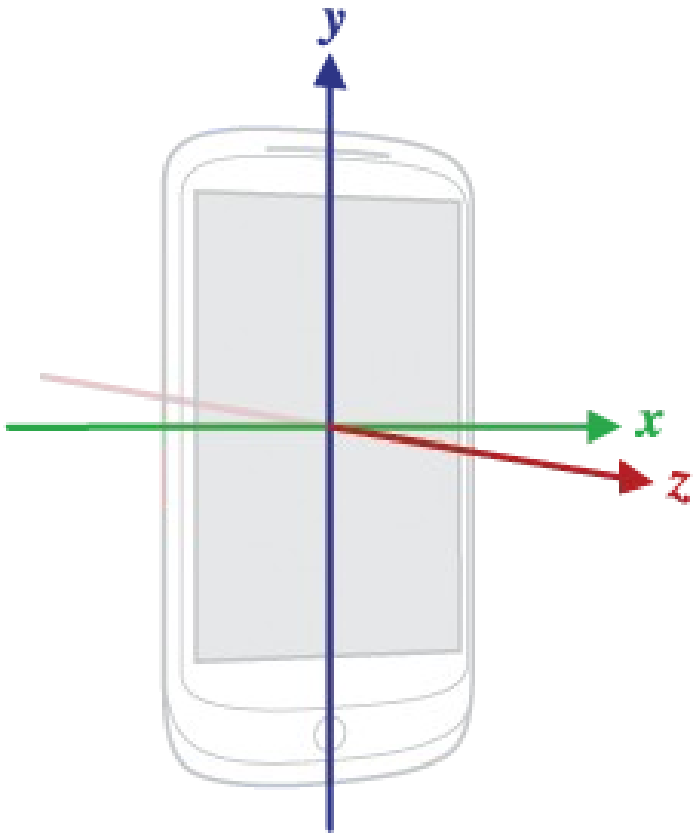
```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor mLight;

    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {    }
    public final void onSensorChanged(SensorEvent event) {
        float lux = event.values[0];
    }
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
}
```

ServiceEvent-s

```
public class MainActivity extends AppCompatActivity {

    private SensorManager sensorManager;
    private LightSensorEventListener sel = new LightSensorEventListener();
    private TextView tv;
    private Sensor mLight;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        sensorManager = (SensorManager)  getSystemService(Context.SENSOR_SERVICE);

        tv = findViewById(R.id.sensorList);
        tv.setMovementMethod(new ScrollingMovementMethod());
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
        sensorManager.registerListener(sel, mLight, SensorManager.SENSOR_DELAY_NORMAL);
        sel.setTv(tv);
    }
    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(sel, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }
    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(sel);
    }
}
```

```
public class LightSensorEventListener implements SensorEventListener
{
    private TextView tv;
    public void setTv(TextView tv) {
        this.tv = tv;
    }
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        float lux = sensorEvent.values[0];
        tv.setText(""+lux);
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {
        //TODO
    }
}
```

3 axis sensors



Detect shake

To the Activity class

```
// variables for shake detection
private static final float SHAKE_THRESHOLD = 3.25f; // m/S**2
private static final int MIN_TIME_BETWEEN_SHAKES_MILLISECS = 1000;
private long mLastShakeTime;
private SensorManager mSensorMgr;
```

To the onCreate method

```
// Get a sensor manager to listen for shakes
mSensorMgr = (SensorManager) getSystemService(SENSOR_SERVICE);

// Listen for shakes
Sensor accelerometer = mSensorMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
if (accelerometer != null) {
    mSensorMgr.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
}
```

Detect shake

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        long curTime = System.currentTimeMillis();
        if ((curTime - mLastShakeTime) > MIN_TIME_BETWEEN_SHAKES_MILLISECS) {

            float x = event.values[0];
            float y = event.values[1];
            float z = event.values[2];

            double acceleration = Math.sqrt(Math.pow(x, 2) +
                    Math.pow(y, 2) +
                    Math.pow(z, 2)) - SensorManager.GRAVITY_EARTH;
            Log.d(APP_NAME, "Acceleration is " + acceleration + "m/s^2");

            if (acceleration > SHAKE_THRESHOLD) {
                mLastShakeTime = curTime;
                Log.d(APP_NAME, "Shake, Rattle, and Roll");
            }
        }
    }
}
```