U LISBOA | UNIVERSIDADE DE LISBOA

Android Sensors Tutorial

Tiago Guerreiro
tjvg@di.fc.ul.pt

This tutorial provides a tutored first approach to the Android Sensor Framework. It is a hands-on tutorial to Motion, Environmental and Position Sensors. Basic examples are given and explained when needed along with brief overviews of the framework and each sensor.

The tutorial covers:

- **Sensor Framework Overview;**
- **Types of Sensors**
- **Sensing the Environment: Proximity Sensor**
- **Sensing Motion: Accelerometer**
- **Basic Exercises**

This tutorial was prepared for the following setup:

- Android 2.3+

Sensors Overview

Current devices gather a wide set of sensors spanning from motion to environment and position. These sensors are able to provide raw data with high precision and accuracy. This information can be easily accessed within custom-made applications and, as such, provide behavior dependent on the state of those sensors. The Android platform supports three broad categories of sensors: Motion, Environmental and Position Sensors. **Motion Sensors** measure acceleration forces and rotational forces along three axes; this category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors. **Environmental Sensors** measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity; this category includes barometers, photometers, and thermometers. **Position sensors** measure the physical position of a device. This category includes orientation sensors and magnetometers. Not all Android devices or platforms present all these sensors.

Sensor	Android 4.0	Android 2.3	Android 2.2	Android 1.5
TYPE_ACCELEROMETER	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_AMBIENT_TEMPERATURE	**Yes**	n/a	n/a	n/a
TYPE_GRAVITY	**Yes**	**Yes**	n/a	n/a
TYPE_GYROSCOPE	**Yes**	**Yes**	n/a[1]	n/a[1]
TYPE_LIGHT	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_LINEAR_ACCELERATION	**Yes**	**Yes**	n/a	n/a
TYPE_MAGNETIC_FIELD	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_ORIENTATION	**Yes**[2]	**Yes**[2]	**Yes**[2]	**Yes**
TYPE_PRESSURE	**Yes**	**Yes**	n/a[1]	n/a[1]
TYPE_PROXIMITY	**Yes**	**Yes**	**Yes**	**Yes**
TYPE_RELATIVE_HUMIDITY	**Yes**	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	**Yes**	**Yes**	n/a	n/a
TYPE_TEMPERATURE	**Yes**[2]	**Yes**	**Yes**	**Yes**

Sensor Framework

Data can be accessed from the sensors by using the Android Sensor Framework. This framework provides classes and interfaces along with a standard workflow to easily register access to a sensor and access its data. Moreover, you can also use the framework to query about the sensors available on the device, understand the specification of each device (e.g., maximum range, resolution).

The Framework provides two types of sensors: hardware and software. The first are built into the mobile device and provide their raw measures. Examples are acceleration and geomagnetic field changes sensors. The latter are mimics of hardware sensors but on their turn they derive data from other sensors, providing a processed value. These are called virtual sensors. Examples are the linear acceleration sensor or the gravity one.

The Android Sensor Framework (located in the *android.hardware* package) is composed of the following classes and interfaces:

- **SensorManager:** This class gathers methods to access the capabilities of the device sensor-wise and the ability to register and unregister listeners to those sensors. It also allows to set data acquisition rates and calibrate sensors; this class is used to create an instance of the Sensor Service
- **Sensor:** This is a generic class used to create an instance of a sensor. It also provides methods to query for sensor capabilities;
- **SensorEvent:** Android uses this class to create an event related to a certain sensor. 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:** This interface is used to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

Similarly to user interface (UI) events, sensor events are handled by the registered callbacks.

Typically, in an Android application, you perform two tasks:

- Identify sensors and sensor capabilities;
- Monitor sensor events

A First Example: Proximity Sensor (Sensing the Environment)

Our first example will depict how to use the main functionalities of the Android Sensor Framework. To do so, we will use a simple sensor, the proximity one. This sensor is the one responsible for turning off the screen when the mobile device is near the user's ear. By doing so, Android is trying to reduce battery waste.

Create a new Android application. Add a text view to the activity and add an *id* to it so we can later find it and edit it dynamically.

```
        <TextView
          android:id="@+id/prox"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_below="@+id/textView1"
          android:text="PROX: 0.0" />
```

Then, in the main activity class, declare this TextView and two other object instances, one of a SensorManager and one of a Sensor.

```
TextView prox;
SensorManager sensorManager;
Sensor proxSensor;
```

In the *OnCreate* method of the activity, initialize the aforementioned variables, and register the activity as a listener:

```
prox = (TextView) findViewById(R.id.prox);
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
proxSensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);

sensorManager.registerListener(this,
proxSensor,SensorManager.SENSOR_DELAY_NORMAL);
```

Notice that these listeners should be registered and unregistered every time they are in use/not in use. This means that you should override onPause/OnStop/OnResume methods to register and unregister the listener.

```
sensorManager.unregisterListener(this);
```

To be able to listen to a *SensorEvent*, the main activity class will have to implement *SensorEventListener:*

```
MainActivity extends Activity implements SensorEventListener
```

Accept the IDE's suggestion to add unimplemented methods: *OnSensorChanged* and *OnAccuracyChanged.* If they are missing and not suggested, add them to the class. Proceed by editing the *OnSensorChanged* callback as follows to process a new value from the sensor:

```
public void onSensorChanged(SensorEvent arg0) {

       if (arg0.sensor.getType() == Sensor.TYPE_PROXIMITY) {
              prox.setText("PROX: " + String.valueOf(arg0.values[0]));
       }
}
```

Run the application and explore covering the proximity sensor and analyze the resulting values.

Accelerometer (Sensing Motion)

Create a new Android Application Project. Add three TextView elements to the activity and identify them as X, Y and Z. In the *OnCreate* method, find the TextView objects based on their IDs. Proceed similarly as in the last example but this time by declaring an Accelerometer sensor:

```
x = (TextView) findViewById(R.id.x);
y = (TextView) findViewById(R.id.y);
z = (TextView) findViewById(R.id.z);

accSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
sensorManager.registerListener(this,
accSensor,SensorManager.SENSOR_DELAY_NORMAL);
```

Then, edit the *OnSensorChanged* method to process data from the accelerometer:

```
if (arg0.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
       x.setText("X: " + String.valueOf(arg0.values[0]));
       y.setText("Y: " + String.valueOf(arg0.values[1]));
       z.setText("Z: " + String.valueOf(arg0.values[2]));
}
```

Run the Application as an Android Application.

Exercises

1. Using the accelerometer, create an application that recognizes the orientation of the device (horizontal, vertical and diagonal). Change the background color depending on the orientation;
2. Using the accelerometer, create an application that recognizes shaking the device above a pre-determined threshold. Change the background color randomly when shaking is detected.
3. Using the accelerometer, create an application that recognizes three different “complex” gestures, where a complex gesture is defined as a composition of more than one translation (ex: moving the phone back and forth, making a circle,..).

References

1. http://developer.android.com/guide/topics/sensors/sensors_overview.html