

ALMA MATER STUDIORUM - UNIVERSITY OF BOLOGNA

# Android Sensors Support

Carlo Antenucci

[carlo.antenucci@gmail.com](mailto:carlo.antenucci@gmail.com)

[carlo.antenucci@studio.unibo.it](mailto:carlo.antenucci@studio.unibo.it)



## Contents

<b>1. Android Sensor Framework</b>	5
1.1. Introduction to sensors	5
1.2. Android sensor framework	6
1.3. Sensor availability	6
1.4. Sensor coordinate system	8
1.5. Identifying sensors and sensor capabilities	8
1.6. Monitoring sensor events	13
1.7. Best practices for accessing and using sensors	18
1.8. How Android Sensor Framework works	19
<b>2. AndroidSensorSupport</b>	20
2.1. How AndroidSensorSupport works	20
2.2. AndroidSensorData package	20
2.3. AndroidSensorSupport package	31
2.4. How to use AndroidSensorSupport	44
<b>3. Conclusions</b>	51
<b>References</b>	52

### Abstract

Most Android-powered device have several built-in sensors that measure motion, orientation and other various environmental condition, provide raw data with high precision and accuracy, and are useful to monitor three-dimensional device movement and positioning or monitor changes in the ambient environment near a device.

Android platform supports three category of sensors:

**Motion sensors:** measure acceleration and rotational forces along three axis. This category includes *accelerometer*, *gravity sensor*, *gyroscope* and *rotational vector sensor*.

**Environmental sensors:** sensors included in this category (*barometer*, *photometer* and *thermometer*) measure various environmental parameters such as temperature, pressure, illumination, humidity.

**Position sensors:** measure physical position of a device using *orientation sensor* and *magnetometer*.

The access to a sensor and the raw sensor data acquisition are simplified by using Android Sensor Framework that provides several classes and interfaces that helps developer to perform a large number of sensor-related task.

The first part of this paper is based on the Android Developers sensors documentation[1]: introduces the Android sensor framework and explains how to use sensors.

In the second part is introduced a sensor and sensor data model that realizes a new layer between Android Sensor Framework and the final application. The model has been developed as project activity related to the Engineering of Software System course. This layer simplifies the sensor usage and allow a developer to ask to the sensor his data value (Android Sensor Framework only notify a change, but developers cannot request information to the sensor without save the last update in a variable).

All source code and informations are available at <http://carloantenucci.github.io/AndroidSensorSupport/>

## 1. Android Sensor Framework

### 1.1. Introduction to sensors

The Android Sensor Framework lets access to access many type of sensors, some of these are hardware-based (real sensors) and some are software-based (virtual sensors).

Hardware-based sensors are built into the device and they derive data by directly measuring specific environmental properties, while software-based sensors are not physical devices despite they mimic an hardware-based sensor. This second group derive their data from one or more of the hardware-based sensors.

Table 1: Sensor types supported by the Android platform

Sensor	Type	Description	Sensor fusion
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z).	If it is software uses Accelerometer to derive data
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in $rad/s$ that is applied to a device on all three physical axes (x, y, and z).	
TYPE_LINEAR_ACCELERATION	Software	Measures the acceleration force in $m/s^2$ that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	— Accelerometer — Gravity sensor
TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <code>getRotationMatrix()</code> method.	— Accelerometer — Geomagnetic field
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	If it is software uses Gyroscope to derive data
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius ( $^{\circ}C$ ).	
TYPE_LIGHT	Hardware	Measures the ambient illumination in lx.	
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in $hPa$ or $mbar$ .	
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the ambient humidity in percent (%).	
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degree Celsius ( $^{\circ}C$ ).	
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in $\mu T$ .	
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device.	

## 1.2. Android sensor framework

Android sensor framework is part of the `android.hardware` package. This subsystem includes the interface, named sensor Hardware Abstraction Layer (sensor HAL), between the hardware driver and the other framework classes and interfaces which allows developers to

- Identify sensors and sensor capabilities  
useful for application with features that needs a specific sensor type or capabilities (identify all sensors that are present on a device and disable features that rely on sensors not present)
- Monitor sensor events  
raw sensors data acquisition. Every time a sensor detects a change (normally every x nanoseconds, with x defined by one of `SENSOR_DELAY_*` value) in the parameter that is measuring, notify this change using a sensor event that provides four different informations:
  - Name of the sensor that triggered the event
  - Timestamp of the event in nanoseconds<sup>1</sup>
  - Accuracy of the event
  - Raw data that triggered the event

This tasks can be performed using the sensor-related APIs introduced by classes and interfaces included in Android sensor framework:

**SensorManager** This class creates an instance of the sensor service and provides methods to access and listens sensors, register and un register sensor listeners, acquire device orientation informations and also defines several sensors constants useful to report sensor accuracy, set data acquisition rates, and calibrate sensors.

**Sensor** This class is useful to create an instance of a specific sensor and provides various methods that determine sensor's capabilities.

**SensorEvent** Android uses this class to create a sensor event object which provides sensor event's informations such as raw sensor data, sensor type that generated the event, event accuracy and time stamp.

**SensorEventListener** This interface is useful to create two callback methods that receive notification (a sensor event) when sensor values or sensor accuracy change.

## 1.3. Sensor availability

Sensors availability is different within devices and is different too among Android versions because the Android sensors have been introduced in different platmorm releases.

Many sensors have been introduced by Android 1.5 Cupcake (API Level 3), but some were not implemented and not available until Android 2.3 Gingerbread (API Level 9) that introduces too new sensors. Other sensors were introduced by Android 4.0 Ice Cream Sandwich (API Level 14) that also deprecates two sensors, replaced by newer and better sensors.

The following table summarize the availability of each sensor in each Android release.

---

<sup>1</sup> An Android Project Member says: "[...]The timestamps are not defined as being the Unix time; they're just "a time" that's only valid for a given sensor. [...]"[2]

Table 2: Sensor types supported by the Android platform

Sensor	Android 4 Ice Cream Sandwich (API Level 14)	Android 2.3 Gingerbread (API Level 9)	Android 2.2 Froyo (API Level 8)	Android 1.5 Cupcake (API Level 3)
TYPE_ACCELEROMETER	YES	YES	YES	YES
TYPE_GRAVITY	YES	YES	n/a	n/a
TYPE_GYROSCOPE	YES	YES	n/a <sup>a</sup>	n/a <sup>a</sup>
TYPE_LINEAR_ACCELERATION	YES	YES	n/a	n/a
TYPE_ORIENTATION	YES <sup>b</sup>	YES <sup>b</sup>	YES <sup>b</sup>	YES
TYPE_ROTATION_VECTOR	YES	YES	n/a	n/a
TYPE_AMBIENT_TEMPERATURE	YES	n/a	n/a	n/a
TYPE_LIGHT	YES	YES	YES	YES
TYPE_PRESSURE	YES	YES	n/a <sup>a</sup>	n/a <sup>a</sup>
TYPE_RELATIVE_HUMIDITY	YES	n/a	n/a	n/a
TYPE_TEMPERATURE	YES <sup>b</sup>	YES	YES	YES
TYPE_MAGNETIC_FIELD	YES	YES	YES	YES
TYPE_PROXIMITY	YES	YES	YES	YES

<sup>a</sup> Added in Android 1.5 (API Level 3), but not available until Android 2.3 (API Level 9).<sup>b</sup> Sensor available but deprecated

### 1.4. Sensor coordinate system

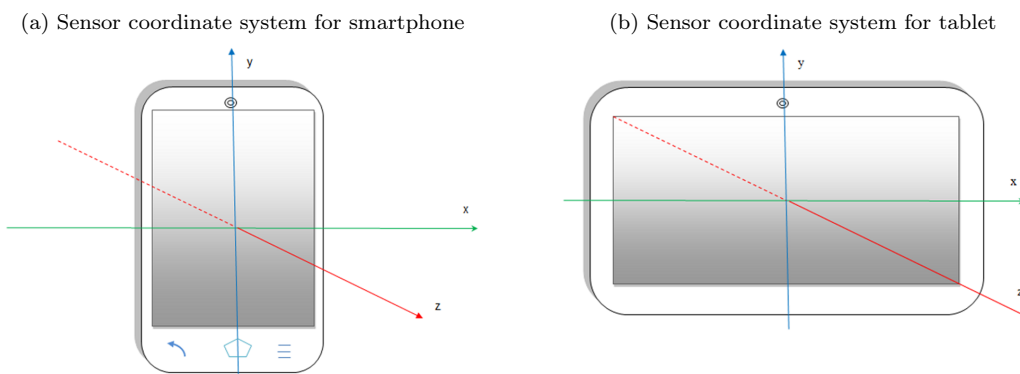
Generally the sensor framework uses a standard 3-axis coordinate system to express data values.

X, Y and Z values are represented respectively by `values[0]`, `values[1]` and `values[2]` of `SensorEvent` object. Some sensors, such as proximity sensor, light sensor, pressure sensor and temperature sensor, provides single values represented by the only `values[0]`.

For `TYPE_ACCELEROMETER`, `TYPE_GRAVITY`, `TYPE_GYROSCOPE`, `TYPE_LINEAR_ACCELERATION` and `TYPE_MAGNETIC_FIELD` the sensors the coordinate system is defined relatively to the device's screen when the device is held in its default orientation (portrait for smartphones, landscape for many tablet). When the device is in its default orientation the X axis is horizontal and points to the right, the Y axis is vertical and points up and the Z axis points toward the outside of the screen face.

The most important thing to understand is that the axes are not swapped when the device screen orientation changes.

Figure 1.1: Sensor coordinate system[?]



### 1.5. Identifying sensors and sensor capabilities

The Android sensor framework provides several methods that make it easy to determine at runtime which sensors are on a device and the capabilities of each sensor, such as maximum range, resolution, power requirements, minimum delay and vendor.

First of all, to identify the sensors on a device, it is necessary to obtain a reference to the sensor service by creating an instance of the `SensorManager` class by calling the `getSystemService()` method using the `SENSOR_SERVICE` as parameter:

```
private SensorManager mSensorManager;
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Now, calling the `getSensorList()` method and using `TYPE_ALL` constant as parameter, `SensorManager` returns the list of every sensor on the device:

```
List<Sensor> deviceSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
```

Using, instead of `TYPE_ALL` constant, another constant provided by `Sensor` class such as `TYPE_GYROSCOPE`, `TYPE_LINEAR_ACCELERATION`, `TYPE_GRAVITY` etc. `SensorManager` class returns a list of all sensors of the given type.

It is also possible to determine if a specific type of sensor exists on a device by using the `getDefaultSensor()` method with the sensor type constant as parameter. If a device has more than one sensor for the given type, one of them must be designed as the default sensor and if the default sensor does not exist, the method returns null (which means that the device does not have that type of sensor). The following code checks if there is an accelerometer on the device:



```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if ( mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null ) {
    // Success! There's an accelerometer.
}
else{
    // Failure! No accelerometer.
}
```

In addition to listing the sensors on a device, in possible, using public methods of `Sensor` class, determine capabilities and attributes of each sensor. This is useful if an application can have different behavior depending on sensors, or sensor capabilities, available on a device. With this methods is possible to obtain sensor's resolution and maximum range of measurement (using `getResolution()` and `getMaximumRange()`) or sensor's power requirement with `getPower()` method; other two methods particularly useful to optimize an application for different manufacturers' sensors or different sensors' version are `getVendor()` -for the manufacturer- and `getVersion()` -to obtain sensor's version-.

The following sample code shows how to use `getVendor()` and `getVersion()` methods to optimize an application using gravity sensor if its vendor is *Google Inc.* and its version is 3, if that particular sensor is not present on device the application try to use the accelerometer:

Another useful methods is the `getMinDelay()` which returns the minimum time interval (in microseconds) between two data sensed by a sensor. Any sensor that returns a *non-zero* value is a streaming sensor -this type of sensors sense data at regular intervals and were introduced by Android 2.3 Gingerbread (API Level 9)- while if a sensor returns zero, it means that the sensor is not streaming sensor, and it reports data only when there is a change in the parameter it is sensing. This method is useful because using it is possible to determine the maximum rate at which a sensor can acquire data.

**Sensors identification code**<sup>2</sup> The following code realize an Android application that lists all sensors in a device and its own properties. Some methods were not introduced until API Level 9, so it works on Android 2.3 (Gingerbread) or latest.

The layout file (`res/layout/activity_about_sensors.xml`) defines the Android widgets used, their ID and their properties (position, dimensions, alignment, etc.). In this application is needed a **Spinner** to select which is the sensor to be inspected and a **TableLayout** that contains a row for each property and every row contains two **TextView** (a label and a value field).

Listing 1: `activity_about_sensors.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Spinner
        android:id="@+id/spinner_sensors"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true" />

    <TableLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/spinner_sensors" >

        <TableRow
            android:id="@+id/tableRowVersion"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <TextView
                android:id="@+id/labelVersion"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Version:_"
                android:textAppearance="?android:attr/textAppearanceMedium" />

            <TextView
                android:id="@+id/valueVersion"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text=""
                android:textAppearance="?android:attr/textAppearanceMedium" />

        </TableRow>

        <TableRow
            android:id="@+id/tableRowVendor"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >

            <TextView
                android:id="@+id/labelVendor"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Vendor:_"
                android:textAppearance="?android:attr/textAppearanceMedium" />

            <TextView
                android:id="@+id/valueVendor"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text=""
                android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableLayout>
```

<sup>2</sup> This app is available on github in AboutSensors project

```

</TableRow>

<TableRow
    android:id="@+id/tableRowPower"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/labelPower"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Power:␣"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/valuePower"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceMedium" />

</TableRow>

<TableRow
    android:id="@+id/tableRowMaxRange"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/labelMaxRange"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Maximum␣Range:␣"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/valueMaxRange"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceMedium" />

</TableRow>

<TableRow
    android:id="@+id/tableRowResolution"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/labelResolution"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Resolution:␣"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/valueResolution"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:textAppearance="?android:attr/textAppearanceMedium" />

</TableRow>

<TableRow
    android:id="@+id/tableRowMinDelay"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/labelMinDelay"
        android:layout_width="wrap_content"

```

```

        android:layout_height="wrap_content"
        android:text="Minimum Delay: "
        android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/valueMinDelay"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text=""
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </TableRow>
</TableLayout>

</RelativeLayout>

```

The activity code is really simple: when Android system creates the activity (`onCreate()` method), the first thing to do is the initialization of resources. The `initializeResources()` method assigns to each field its resource referenced in layout file, creates the `Spinner`'s `ArrayAdapter` with sensors' names and call the `addSpinnerListener()` method that assigns to the `Spinner` a new `onItemSelectedListener` that changes the shown values with the sensor selected values.

Listing 2: AboutSensors.java

```

package it.unibo.android.aboutSensors;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class AboutSensors extends Activity {

    //fields definition
    private Spinner spinnerList;
    private SensorManager manager;
    private List<Sensor> sensors;
    private Sensor sensorSelected;
    private ArrayAdapter<String> adapter;
    private TextView textVersion;
    private TextView textVendor;
    private TextView textPower;
    private TextView textMaxRange;
    private TextView textResolution;
    private TextView textMinDelay;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //set the default layout
        setContentView(R.layout.activity_about_sensors);
        //initialize the resources
        initializeResources();
    }

    private void initializeResources() {
        //obtain the SensorManager instance and get the sensors list
        manager = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensors = manager.getSensorList(Sensor.TYPE_ALL);
        //get the TextView where put the sensor information
        textVersion = (TextView) findViewById(R.id.valueVersion);
        textVendor = (TextView) findViewById(R.id.valueVendor);
        textPower = (TextView) findViewById(R.id.valuePower);
        textMaxRange = (TextView) findViewById(R.id.valueMaxRange);
        textResolution = (TextView) findViewById(R.id.valueResolution);
        textMinDelay = (TextView) findViewById(R.id.valueMinDelay);
        //get the spinner
        spinnerList = (Spinner) findViewById(R.id.spinner_sensors);
        //set up the spinner adapter with sensors names
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, getSensorsNames());
        adapter.setDropDownViewResource(android.R.layout.simple_dropdown_item_1line);
    }
}

```

```

        spinnerList.setAdapter(adapter);
        //add the spinner listener
        addSpinnerListener();
    }

    private void addSpinnerListener() {
        spinnerList.setOnItemClickListener(new OnItemSelectedListener(){

            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                                    int position, long id) {
                //when an item is selected update the text view
                //with the selected sensor's informations
                Sensor s = sensors.get(position);
                if(s != sensorSelected){
                    System.out.println(s.toString());
                    textViewVersion.setText(""+s.getVersion());
                    textViewVendor.setText(""+s.getVendor());
                    textViewPower.setText(s.getPower()+"µmA");
                    textViewMaxRange.setText(""+s.getMaximumRange());
                    textViewResolution.setText(""+s.getResolution());
                    textViewMinDelay.setText(s.getMinDelay()/1000+"ms");
                    sensorSelected = s;
                }
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
                textViewVersion.setText("");
                textViewVendor.setText("");
            }

        });
    }

    private List<String> getSensorsNames() {
        //returns the list that contains the names of all available sensors
        List<String> sensorNames = new ArrayList<String>();
        for (Sensor s: sensors)
            sensorNames.add(s.getName());
        return sensorNames;
    }
}

```

## 1.6. Monitoring sensor events

SensorEventListener interface introduces two callback methods which must be implemented to monitor raw sensor data. These methods are `onAccuracyChanged()` and `onSensorChanged()`.

These two methods are invoked by Android system, the first whenever the sensor's accuracy changes (this method provides the reference to the `Sensor` object that changed and its new accuracy, whose state is represented by one of four constants defined in `SensorManager` class:

- `SENSOR_STATUS_ACCURACY_LOW`
- `SENSOR_STATUS_ACCURACY_MEDIUM`
- `SENSOR_STATUS_ACCURACY_HIGH`
- `SENSOR_STATUS_ACCURACY_UNRELIABLE`

The `onSensorChanged()` method, instead, is invoked by system when a sensor reports a new value. This method provides a new `SensorEvent` object that contains informations about the new sensor data (the new data recorded by the sensor, its accuracy, the sensor which generates the new data and the relative timestamp).

**Sensor events code**<sup>3</sup> The following code is based on AboutSensors project. In this application is shown, with the previous informations, the raw sensor data, received by the selected sensor.

In layout file (`res/layout/activity_about_sensor_events.xml`) above the sensor informations table is defined a new table that contains 3 rows with 3 labels and 3 `TextView`, one for each axes value (X, Y and Z). The following code shown only the difference between the previous and the new layout.

<sup>3</sup> This app is available on github in the project AboutSensorEvents

This part of code begins after the `</TableLayout>` tag.

Listing 3: activity\_about\_sensor\_events.xml

```
<TextView
    android:id="@+id/LabelSensorData"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="false"
    android:layout_alignParentRight="false"
    android:layout_below="@+id/tableLayout1"
    android:layout_centerHorizontal="true"
    android:text="Sensor Data:"
    android:textAppearance="?android:attr/textAppearanceLarge" />

<TableLayout
    android:id="@+id/SensorData"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/LabelSensorData" >

    <TableRow
        android:id="@+id/SensorDataX"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/LabelDataX"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="SensorDataX:"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/ValueDataX"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium" />
    </TableRow>

    <TableRow
        android:id="@+id/SensorDataY"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/LabelDataY"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="SensorDataY:"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/ValueDataY"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceMedium" />
    </TableRow>

    <TableRow
        android:id="@+id/SensorDataZ"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/LabelDataZ"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="SensorDataZ:"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
```

```
        android:id="@+id/ValueDataZ"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:textAppearance="?android:attr/textAppearanceMedium" />  
    </TableRow>  
</TableLayout>  
</RelativeLayout>
```

The activity, like the `AboutSensor` application, changes sensors information when a new sensor is selected from `Spinner` and shows, in addition to sensor informations, the sensor row data updated by the sensor event listener, in fact, the `SpinnerListener`, by `onItemSelected` method, changes the `SensorEventListener` calling the `updateSensorListener` method, which removes old listener from previous selected sensor and attaches to the new selected sensor a new listener. This method is called too when the application is resumed (`onResume()` method), and, when the application is paused (`onPause()` method) the sensor listener is detached from sensor.

The `SensorEventListener` class extends `SensorEventListener` interface the defines

- `onSensorChanged` method that updates the sensor data's text views
- `onAccuracyChanged` that, in this case, does nothing

Listing 4: `AboutSensorEvents.java`

```
package it.unibo.android.aboutSensorEvents;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class AboutSensorEvents extends Activity {

    //fields definition
    private Spinner spinnerList;
    private SensorManager manager;
    private List<Sensor> sensors;
    private Sensor sensorSelected;
    private ArrayAdapter<String> adapter;
    private TextView textVersion;
    private TextView textVendor;
    private TextView textPower;
    private TextView textMaxRange;
    private TextView textResolution;
    private TextView textMinDelay;
    private SensorEventListener listener;
    private TextView textX;
    private TextView textY;
    private TextView textZ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //set the default layout
        setContentView(R.layout.activity_about_sensor_events);
        //initialize the resources
        initializeResources();
    }

    @Override
    protected void onResume() {
        super.onResume();
        manager.registerListener(listener, sensorSelected, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onPause() {
        super.onPause();
        manager.unregisterListener(listener, sensorSelected);
    }

    private void initializeResources() {
        //obtain the SensorManager instance and get the sensors list
        manager = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensors = manager.getSensorList(Sensor.TYPE_ALL);
        //get the TextView where put the sensor information
        textVersion = (TextView) findViewById(R.id.valueVersion);
        textVendor = (TextView) findViewById(R.id.valueVendor);
        textPower = (TextView) findViewById(R.id.valuePower);
        textMaxRange = (TextView) findViewById(R.id.valueMaxRange);
        textResolution = (TextView) findViewById(R.id.valueResolution);
        textMinDelay = (TextView) findViewById(R.id.valueMinDelay);
    }
}
```



```
//get the TextView where put sensor raw data received
textX = (TextView)findViewById(R.id.ValueDataX);
textY = (TextView)findViewById(R.id.ValueDataY);
textZ = (TextView)findViewById(R.id.ValueDataZ);
//get the spinner
spinnerList = (Spinner)findViewById(R.id.spinner_sensors);
//set up the spinner adapter with sensors names
adapter = new ArrayAdapter<String>(this, android.R.layout.simple_spinner_item, getSensorsNames());
adapter.setDropDownViewResource(android.R.layout.simple_dropdown_item_1line);
spinnerList.setAdapter(adapter);
//add the spinner listener
addSpinnerListener();
}

private void updateSensorListener() {
    //if exist a selected element remove old listener
    if(listener!=null)
        manager.unregisterListener(listener, sensorSelected);
    //register new listener
    listener = new SensorEventsListener();
    manager.registerListener(listener, sensors.get(spinnerList.getSelectedItemPosition()), SensorManager.SENSOR_DELAY_NORMAL);
    //POJO observe with loop: DISASTER!

manager.registerListener(new SensorEventListener() {
    //
    //
    //
    @Override
    public void onSensorChanged(SensorEvent event) {
        System.out.println("EventDetected");
        while(true){
            ;
        }
    }
    //
    //
    //
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // TODO Auto-generated method stub
    }
}, sensors.get(spinnerList.getSelectedItemPosition()), SensorManager.SENSOR_DELAY_NORMAL);
}

private void addSpinnerListener() {
    spinnerList.setOnItemClickListener(new OnItemSelectedListener(){
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
                                int position, long id) {
            //when an item is selected update the text view
            //with the selected sensor's informations
            Sensor s = sensors.get(position);
            if(s != sensorSelected){
                System.out.println(s.toString());
                textViewVersion.setText(""+s.getVersion());
                textViewVendor.setText(""+s.getVendor());
                textViewPower.setText(s.getPower()+"mA");
                textViewMaxRange.setText(""+s.getMaximumRange());
                textViewResolution.setText(""+s.getResolution());
                textViewMinDelay.setText(s.getMinDelay()/1000+"ms");
                updateSensorListener();
                sensorSelected = s;
            }
        }
        @Override
        public void onNothingSelected(AdapterView<?> parent) {
            textViewVersion.setText("");
            textViewVendor.setText("");
        }
    });
}

private List<String> getSensorsNames() {
    //returns the list that contains the names of all available sensors
    List<String> sensorNames = new ArrayList<String>();
    for (Sensor s: sensors)
        sensorNames.add(s.getName());
    return sensorNames;
}

private class SensorEventsListener implements SensorEventListener {
    @Override
    public void onSensorChanged(SensorEvent event) {
        //show values
        textViewX.setText(""+event.values[0]);
    }
}
```

```

        textY.setText(""+event.values[1]);
        textZ.setText(""+event.values[2]);
        if(event.sensor.getType() == Sensor.TYPE_SIGNIFICANT_MOTION){
            textX.setText(""+event.timestamp);
        }
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        //do something when accuracy changes
    }
}
}

```

### 1.7. Best practices for accessing and using sensors

In this section are discussed the guidelines to design an optimized sensor implementation.

These guidelines are recommended best practices for anyone who is using the sensor framework to access sensors and acquire sensor data.

#### Verify sensors' availability

Android platform does not require to the manufacturer that a device includes particular sensors, then, before using a specific sensor it is necessary to check if this sensor exists.

Assume the existence of a sensor simply because is frequently used is a bad practice.

#### Unregister sensor listeners

If a sensor listener is registered and its activity is paused, the sensor will continue to acquire data and use battery resources unless the listener is unregistered. The better way to optimize resources usage is to unregister the listener each time the application is paused, or the sensor is no longer needed, and then register it again when the app is resumed. This is possible using two methods provided by the **Activity** class:

**onPause()** called when the application is going down (or lost the screen)

**onResume()** invoked when the application comes back to the screen

The following code is an example that show how to use this two methods in an **Activity** that implements **SensorEventListener** interface:

```

private SensorManager mSensorManager;
private Sensor mSensor
...

@Override protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this, mSensor);
}

@Override protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_NORMAL);
}

```

#### Don't test your code on the emulator

The Android emulator cannot emulate sensors, thus, currently, is not possible to test the sensor code on Android Virtual Devices. The best way to run a test on sensor code is by using a physical device. There are, however, sensor simulators that can be used to simulate sensor output.

#### Don't block the **onSensorChanged()** method

Android system may call the **onSensorChanged(SensorEvent)** method quite often because sensor data can changes at high rate, hence is recommended, as the best practice, do as little is possible into **onSensorChanged(SensorEvent)** so as not to block his execution.

If an application requires any data filtering or reduction of sensor data, it would be better perform that work outside of the **onSensorChanged(SensorEvent)** method.

*Remark.* If the **onSensorChange(SensorEvent)** contains a loop the application is stopped by the system.

### Avoid using deprecated methods or sensor types

Several methods and constants, such as `TYPE_ORIENTATION` or `TYPE_TEMPERATURE`, have been deprecated, the first has been replaced by `getOrientation()` method and the second, in devices that are running Android 4 Ice Cream Sandwich, by `TYPE_AMBIENT_TEMPERATURE` constant.

### Choose sensor delay carefully

The `registerListener()` method requires, in addition to the listener and the sensor, the minimum delay between two notifications. The choice of the delivery rate must be suitable for the application or use-case: sensors can provide data at very high rates and allowing the system to send extra data which is not needed wastes system resources and uses battery power. Like the accuracy, Android provides, in `SensorManager` class, four different constants that defines four different delivery rate:

`SENSOR_DELAY_NORMAL` delivery rate of 200000 $\mu$ s

`SENSOR_DELAY_UI` delivery rate of 60000 $\mu$ s

`SENSOR_DELAY_GAME` delivery rate of 20000 $\mu$ s

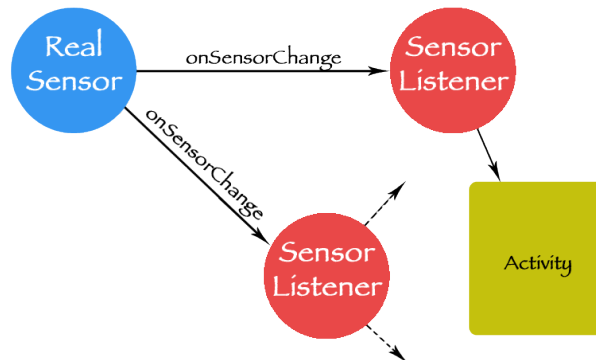
`SENSOR_DELAY_FASTEST` delivery rate of 0 $\mu$ s (enable the real delay time of the sensor, obtained by `getMinDelay()` method)

## 1.8. How Android Sensor Framework works

As mentioned, to access sensor information, developer must register one (or more) listener to a sensor through the sensor manager, after that Android System, according with the delay selected, calls the call-back method (`onSensorChanged()`) passing a `SensorEvent` object, as parameter. This interaction can be seen like a sort of observer pattern in which the sensor represent the observable object; when it changes its state the Android System notifies this change to the observers (each class that implements `SensorEventListener`) registered to the sensor through the `SensorManager` register method.

The following image tries to represent this interaction.

Figure 1.2: How Android Sensor Framework works



## 2. AndroidSensorSupport

**AndroidSensorSupport** models Android sensors and extends the Android Sensor Framework and simplifies the interaction with Android sensors: each **AndroidSensor**, defined in this layer contains a specific **AndroidSensorData** that contains all informations relative to the last sensor update and it can be always requested with a get method and not only when the data changes. For example, using this layer, we can request the accelerometer data when the magnetic field changes or vice-versa.

All **AndroidSensors** defined are, at the same time observer (of the phisic sensor because implements **SensorEventListener**) and observable: the developer can attach to the **AndroidSensor** a listener that does something when the data changes. The only operation that the **AndroidSensor** does when the system calls **onSensorChanged** is to update his sensor data and notifies this change to all his observers.

Another advantage is the **SensorFactory** that the developer must use to obtain an **AndroidSensor**: passing to the factory the **SensorManager** instance, the sensor type and the delay (one of delays defined in **SensorManager** class) and it returns an instance of **AndroidSensor** of the specific type requested.

Last, but not least, **AndroidSensorData** contains methods to convert it-self in strings: there are two possible representations: JSON representation and Prolog-like representation that have a functor for the data type and contains other functors for each data (values, accuracy, timestamp, ...). Is defined also an utility class that reconstruct an **AndroidSensorData** from JSON string.

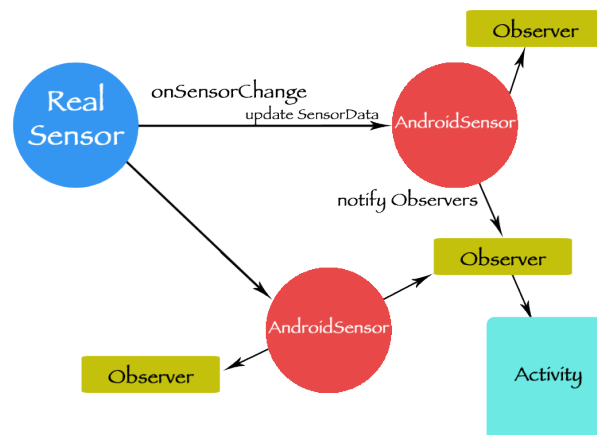
In the following sub-sections is analyzed the project and described how to use this layer with a final example.

### 2.1. How AndroidSensorSupport works

**AndroidSensorSupport** provides a new layer between the application and the Android Sensor Framework: now developers can insert the business logic tha needs sensors informations directly into an observer which is connected to the real sensor through the **AndroidSensor** class (which implements the **SensorEventListener** and extends the **Observable** class).

The following figure shows how the **AndroidSensorSupport** works:

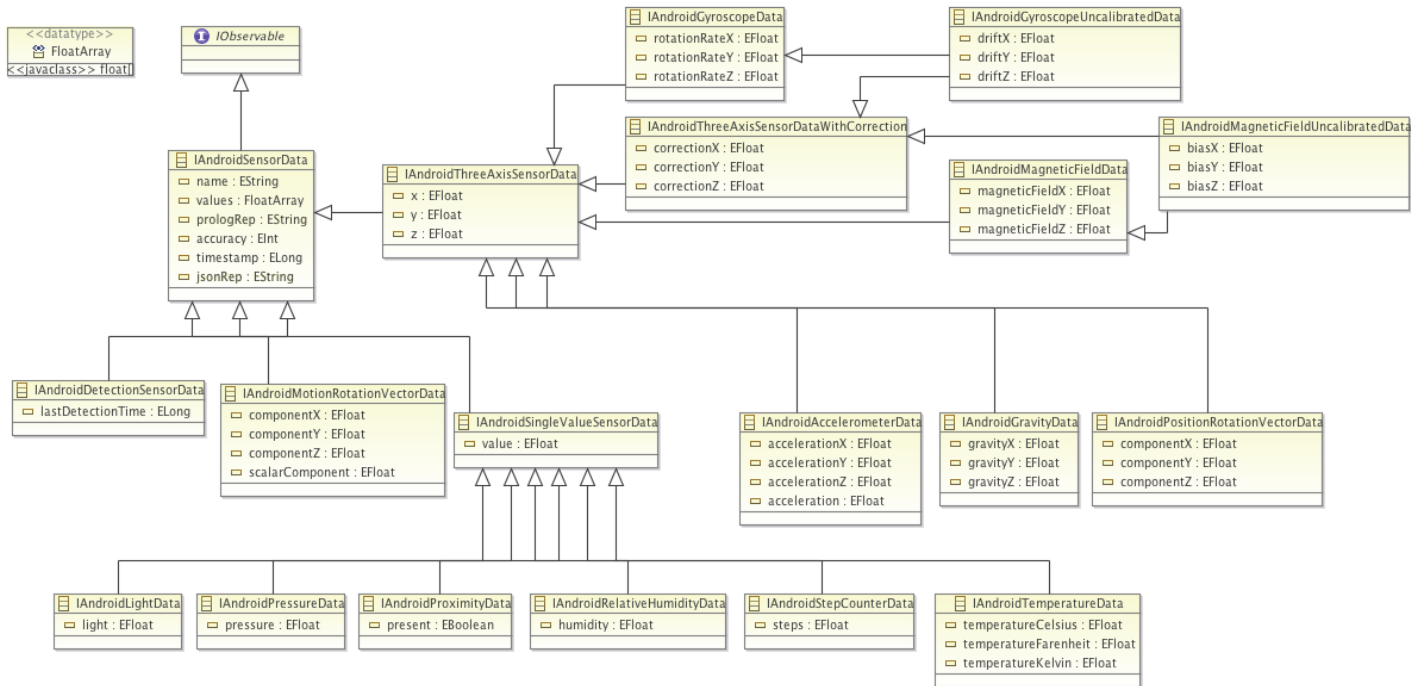
Figure 2.1: How Android Sensor Support works



### 2.2. AndroidSensorData package

The `it.unibo.android.sensorData` package contains all interfaces and implementations that realizes the **SensorData** model. Each sensor has a particular instance of **AndroidSensorData** that extends the superclass as shown in the image below:

Figure 2.2: SensorData class diagram



### 2.2.1. AndroidSensorData

This class provides the standard methods and variables of each subclass. It only implements the `IObserver` interface defined in the `uniboInterfaces.jar` library and extends the `Observable` java class.

The following codes shows the `AndroidSensorData` interface:

Listing 5: IAndroidSensorData.java

```
/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces;
/**

public interface IAndroidSensorData {
    /**
     * Get method that
     * @return the SensorData's name
     */
    public String getName();
    /**
     * Get method that
     * @return the float array of sensor's values
     */
    public float[] giveMeValues();
    /**
     * Get method that
     * @return the SensorData prolog representation (a string)
     */
    public String getPrologRep();
    /**
     * Get method that
     * @return measurement accuracy of the sensor
     */
    public int getAccuracy();
    /**
     * Get method that
     * @return timestamp of measurement (relative to the sensor)
     */
    public long getTimestamp();
    /**
     * Get method that
     * @return the SensorData Json representation (a string)

```

```

    */
    public String getJsonRep();
}

```

This methods are useful to obtain all standard informations such as values as array of float, accuracy, timestamp, name and the two string representations. In the following code, is shown the implementation of a generic `AndroidSensorData`.

Listing 6: `AndroidSensorData.java`

```

package it.unibo.android.sensorData.implementation;

import it.unibo.android.sensorData.interfaces.IAndroidSensorData;
import com.google.gson.Gson;

public class AndroidSensorData implements IAndroidSensorData{

    private String name;

    protected transient float[] values ;
    private long timestamp;
    private int accuracy;

    public AndroidSensorData(float[] values, int accuracy, long timestamp){
        this.name = this.getClass().getSimpleName();
        this.values = values;
        this.accuracy = accuracy;
        this.timestamp = timestamp;
    }

    @Override
    public String getName() {
        return name;
    }

    public float[] giveMeValues() {
        return values;
    }

    @Override
    public String getPrologRep() {
        String rep = getName()+"(values(";
        for(int i=0;i<values.length;i++){
            rep += "v"+i+"("+values[i]+")";
            if(i!=values.length)
                rep += ",";
        }
        rep += "),timestamp("+getTimestamp()+"),accuracy("+getAccuracy()+"))";
        return rep;
    }

    @Override
    public int getAccuracy() {
        return accuracy;
    }

    @Override
    public long getTimestamp() {
        return timestamp;
    }

    @Override
    public String getJsonRep(){
        return (new Gson()).toJson(this);
    }
}

```

Each `AndroidSensorData` extends this class and provides specific features for his particular case.

### 2.2.2. AndroidSingleValueSensorData

This class adds to the `AndroidSensorData` only one method that returns the single value measured by the sensor:

Listing 7: `IAndroidSingleValueSensorData.java`

```
/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces;

public interface IAndroidSingleValueSensorData extends IAndroidSensorData {
    /**
     * Get method that
     * @return the significant value of a SingleValueSensor measurement
     */
    public float getValue();
}
```

In the implementation is extended the constructor that saves in a variable the measured value (`value[0]` passed as argument) and overrides the `getPrologRep()` method:

Listing 8: `AndroidSingleValueSensorData.java`

```
package it.unibo.android.sensorData.implementation;

import it.unibo.android.sensorData.interfaces.IAndroidSingleValueSensorData;

public class AndroidSingleValueSensorData extends AndroidSensorData implements
    IAndroidSingleValueSensorData {

    protected float value;

    public AndroidSingleValueSensorData(float[] values, int accuracy, long timestamp) {
        super(values, accuracy, timestamp);
        value = values[0];
    }

    @Override
    public float getValue() {
        return value;
    }

    @Override
    public String getPrologRep() {
        String rep = getName()+"(value("+getValue()+"), ";
        rep += "timestamp("+getTimestamp()+"), accuracy("+getAccuracy()+"))";
        return rep;
    }
}
```

From this class are extended other subclasses that provides specific methods that calls the super `getValue()` and in some cases had a conversion:

`IAndroidLightData`

— `getLight()`; returns the light in lx.

`IAndroidPressureData`

— `getPressure()`; returns the pressure in hPa or mbar.

`IAndroidProximityData`

— `isPresent()`; this method returns a boolean: if the device detects something returns true.

`IAndroidRelativeHumidity`

— `getHumidity()`; returns the relative humidity in %.

`IAndroidTemperatureData`

— `getTemperatureCelsius()`; returns the temperature in Celsius degrees.

— `getTemperatureFahrenheit()`; returns the temperature in Fahrenheit degrees.

— `getTemperatureKelvin()`; returns the temperature in Kelvin degrees.

### 2.2.3. AndroidThreeAxisSensorData

This class defines the standard `SensorData` for each three axis sensor (accelerometer, gyroscope, etc.). It adds to the `AndroidSensorData` three methods (one for each axis' values)

Listing 9: `IAndroidThreeAxisSensorData.java`

```
/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces;

public interface IAndroidThreeAxisSensorData extends IAndroidSensorData {
    /**
     * Get method that
     * @return the measurement value on the X axis of a three axis sensor
     */
    public float getX();
    /**
     * Get method that
     * @return the measurement value on the Y axis of a three axis sensor
     */
    public float getY();
    /**
     * Get method that
     * @return the measurement value on the Z axis of a three axis sensor
     */
    public float getZ();
}
```

Like the `AndroidSingleValueSensorData` it defines a variable for each axis' value, extends the constructor and overrides the `getPrologRep()` method:

Listing 10: `AndroidThreeAxisSensorData.java`

```
package it.unibo.android.sensorData.implementation;

import it.unibo.android.sensorData.interfaces.IAndroidThreeAxisSensorData;

public class AndroidThreeAxisSensorData extends AndroidSensorData implements
    IAndroidThreeAxisSensorData {

    float x, y, z;

    public AndroidThreeAxisSensorData(float[] values, int accuracy, long timestamp) {
        super(values, accuracy, timestamp);
        x = values[0];
        y = values[1];
        z = values[2];
    }

    @Override
    public float getX() {
        return x;
    }

    @Override
    public float getY() {
        return y;
    }

    @Override
    public float getZ() {
        return z;
    }

    @Override
    public String getPrologRep(){
        String rep = getName()+"(x("+getX()+"),y("+getY()+"),z("+getZ()+"),";
        rep += "timestamp("+getTimestamp()+"),accuracy("+getAccuracy()+"))";
        return rep;
    }
}
```

and, still other `SensorData`, each class that extends this have specific methods:

#### `IAndroidAccelerometerData`

- `getAccelerationX()`; returns the acceleration value along the X axis;
- `getAccelerationY()`; returns the acceleration value along the Y axis;
- `getAccelerationZ()`; returns the acceleration value along the Z axis;



- `getAcceleration()`; returns the device's acceleration calculating as  $\sqrt{a_x^2 + a_y^2 + a_z^2}$

**IAndroidGravityData**

- `getGravityX()`; returns the gravity value along the X axis;
- `getGravityY()`; returns the gravity value along the Y axis;
- `getGravityZ()`; returns the gravity value along the Z axis

**IAndroidGyroscopeData**

- `getRotationRateX()`; returns the rotation rate value around the X axis;
- `getRotationRateY()`; returns the rotation rate value around the Y axis;
- `getRotationRateZ()`; returns the rotation rate value around the Z axis

**IAndroidMagneticFieldData**

- `getMagneticFieldX()`; returns the magnetic field value along the X axis;
- `getMagneticFieldY()`; returns the magnetic field value along the Y axis;
- `getMagneticFieldZ()`; returns the magnetic field value along the Z axis

**IAndroidPositionRotationVectorData**

- `getComponentX()`; returns the rotation component value around the X axis;
- `getComponentY()`; returns the rotation component value around the Y axis;
- `getComponentZ()`; returns the rotation component value around the Z axis

#### 2.2.4. AndroidThreeAxisSensorDataWithCorrection

This class extends the `ThreeAxisSensorData` and adds to it other three values that represents the correction applied by the system according to the sensor informations:

Listing 11: IAndroidThreeAxisSensorDataWithCorrection.java

```
/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces;

public interface IAndroidThreeAxisSensorDataWithCorrection extends IAndroidThreeAxisSensorData {
    /**
     * Get method that
     * @return the correction value of measurement on the X axis of a three axis sensor
     */
    public float getCorrectionX();
    /**
     * Get method that
     * @return the correction value of measurement on the Y axis of a three axis sensor
     */
    public float getCorrectionY();
    /**
     * Get method that
     * @return the correction value of measurement on the Z axis of a three axis sensor
     */
    public float getCorrectionZ();
}
```

Like the other subclasses it defines a variable for each axis' correction values, extends the constructor and overrides the `getPrologRep()` method:

Listing 12: AndroidThreeAxisSensorDataWithCorrection.java

```
package it.unibo.android.sensorData.implementation;

import it.unibo.android.sensorData.interfaces.IAndroidThreeAxisSensorDataWithCorrection;

public class AndroidThreeAxisSensorDataWithCorrection extends AndroidThreeAxisSensorData implements
    IAndroidThreeAxisSensorDataWithCorrection {

    float correctionX, correctionY, correctionZ;

    public AndroidThreeAxisSensorDataWithCorrection(float[] values, int accuracy, long timestamp) {
        super(values, accuracy, timestamp);
        correctionX = values[3];
        correctionY = values[4];
        correctionZ = values[5];
    }

    @Override
    public float getCorrectionX() {
        return correctionX;
    }

    @Override
```

```

    public float getCorrectionY() {
        return correctionY;
    }

    @Override
    public float getCorrectionZ() {
        return z;
    }

    @Override
    public String getPrologRep() {
        String rep = getName()+"(x("+getX()+"),y("+getY()+"),z("+getZ()+"),";
        rep += "correctionX("+getCorrectionX()+"),correctionY("+getCorrectionY()+"),";
        rep += "correctionZ("+getCorrectionZ()+"),timestamp("+getTimestamp()+"),";
        rep += "accuracy("+getAccuracy()+")";
        return rep;
    }
}

```

The only two sensors that have this type of data representation are the `GyroscopeUncalibrated` and the `MagneticFieldUncalibrated` and their data representation extends this class but implements also the `GyroscopeData` or `MagneticFieldData` representation. In the following code is show the `AndroidGyroscopeUncalibratedData` for the `AndroidMagneticFieldUncalibratedData` the implementation is similar, the only difference is the name of get methods for correction (the gyroscope have `getDrift` and the magnetic field have `getBias`):

Listing 13: `IAndroidGyroscopeUncalibratedData.java`

```

/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces.motionSensors;

import it.unibo.android.sensorData.interfaces.IAndroidThreeAxisSensorDataWithCorrection;

public interface IAndroidGyroscopeUncalibratedData extends
    IAndroidThreeAxisSensorDataWithCorrection, IAndroidGyroscopeData {

    /**
     * Get method that
     * @return the drift on the X axis
     */
    public float getDriftX();

    /**
     * Get method that
     * @return the drift on the Y axis
     */
    public float getDriftY();

    /**
     * Get method that
     * @return the drift on the Z axis
     */
    public float getDriftZ();
}

```

Listing 14: `AndroidGyroscopeUncalibratedData.java`

```

package it.unibo.android.sensorData.implementation.motionSensors;

import it.unibo.android.sensorData.implementation.AndroidThreeAxisSensorDataWithCorrection;
import it.unibo.android.sensorData.interfaces.motionSensors.IAndroidGyroscopeUncalibratedData;

public class AndroidGyroscopeUncalibratedData extends AndroidThreeAxisSensorDataWithCorrection implements
    IAndroidGyroscopeUncalibratedData {

    public AndroidGyroscopeUncalibratedData(float[] values, int accuracy, long timestamp) {
        super(values, accuracy, timestamp);
    }

    @Override
    public float getRotationRateX() {
        return getX();
    }

    @Override
    public float getRotationRateY() {
        return getY();
    }

    @Override
    public float getRotationRateZ() {

```

```

        return getZ();
    }

    @Override
    public float getDriftX() {
        return getCorrectionX();
    }

    @Override
    public float getDriftY() {
        return getCorrectionY();
    }

    @Override
    public float getDriftZ() {
        return getCorrectionZ();
    }
}

```

### 2.2.5. AndroidMotionRotationVectorData

This class realizes the data representation for a motion rotation vector sensor. It is different from a `PositionRotationVectorData` because he can have 4 values: the same of position rotation vector and a scalar component:

Listing 15: IAndroidMotionVectorRotationData.java

```

/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces.motionSensors;
/**

import it.unibo.android.sensorData.interfaces.IAndroidSensorData;

public interface IAndroidMotionRotationVectorData extends IAndroidSensorData {
    /**
     * Get method that
     * @return the rotation component around X axis
     */
    public float getComponentX();
    /**
     * Get method that
     * @return the rotation component around Y axis
     */
    public float getComponentY();
    /**
     * Get method that
     * @return the rotation component around Z axis
     */
    public float getComponentZ();
    /**
     * Get method that
     * @return the rotation's scalar component (can be null)
     */
    public float getScalarComponent();
}

```

Listing 16: AndroidMotionVectorRotationData.java

```

package it.unibo.android.sensorData.implementation.motionSensors;

import it.unibo.android.sensorData.implementation.AndroidSensorData;
import it.unibo.android.sensorData.interfaces.motionSensors.IAndroidMotionRotationVectorData;

public class AndroidMotionRotationVectorData extends AndroidSensorData implements
    IAndroidMotionRotationVectorData {

    protected float componentX;
    protected float componentY;
    protected float componentZ;
    protected float componentS;

    public AndroidMotionRotationVectorData(float[] values, int accuracy, long timestamp) {
        super(values, accuracy, timestamp);
        componentX = values[0];
        componentY = values[1];
        componentZ = values[2];
        if (values.length == 4)
            componentS = values[3];
    }
}

```

```

    }

    @Override
    public float getComponentX() {
        return componentX;
    }

    @Override
    public float getComponentY() {
        return componentY;
    }

    @Override
    public float getComponentZ() {
        return componentZ;
    }

    @Override
    public float getScalarComponent() {
        return componentS;
    }

    @Override
    public String getPrologRep() {
        String rep = getName()+"(x("+getComponentX()+"),y("+getComponentY()+"),z("+getComponentZ()+"),";
        rep += "s("+getScalarComponent()+"),timestamp("+getTimestamp()+"),accuracy("+getAccuracy()+"))";
        return rep;
    }
}

```

### 2.2.6. *AndroidDetectionSensorData*

This kind of sensor data representation is different from the others: it don't have any value and the only thing that it save is the real time of the detection (when the constructor is called):

Listing 17: *IAndroidDetectionSensorData.java*

```

/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.interfaces;

public interface IAndroidDetectionSensorData extends IAndroidSensorData {
    /**
     * Get method that
     * @return the timestamp of last detection
     */
    public long getLastDetectionTime();
}

```

Listing 18: *AndroidDetectionSensorData.java*

```

package it.unibo.android.sensorData.implementation;

import it.unibo.android.sensorData.interfaces.IAndroidDetectionSensorData;

/**
 * This class is used for each detection sensors: SIGNIFICANT_MOTION and STEP_DETECTOR
 */
public class AndroidDetectionSensorData extends AndroidSensorData implements
    IAndroidDetectionSensorData {

    private long detectionTime;

    public AndroidDetectionSensorData(float[] values, int accuracy, long timestamp) {
        super(values, accuracy, timestamp);
        detectionTime = System.currentTimeMillis();
    }

    //for json reconstruction
    public AndroidDetectionSensorData(float[] values, int accuracy, long timestamp, long detectionTime) {
        super(values, accuracy, timestamp);
        this.detectionTime = detectionTime;
    }

    @Override
    /**
     * This method, IN THIS CLASS, returns a null value because this type of sensors
     * doesn't have values but only notifies a detection, so may return an empty array.
     */
    public float[] giveMeValues() {
        return null;
    }
}

```

```

    }

    @Override
    public String getPrologRep(){
        String rep = getName()+"(timestamp("+getTimestamp()+"),accuracy("+getAccuracy()+"))";
        return rep;
    }

    @Override
    public long getLastDetectionTime() {
        return detectionTime;
    }

}

```

Is needed an override of `getValues()` method because he don't have values, so is correct that this method returns null and is also necessary to add another constructor that have the detection time as parameter for the JSON reconstruction.

### 2.2.7. AndroidSensorDataUtils

This is a static class that provides a reconstruction of `SensorData` object from a JSON string. It only contains a static method that obtain information using a JSON parser and then, according with the name, calls the specific constructor of `SensorData`:

Listing 19: AndroidSensorDataUtils.java

```

/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorData.utils;

import it.unibo.android.sensorData.implementation.*;
import it.unibo.android.sensorData.implementation.motionSensors.*;
import it.unibo.android.sensorData.implementation.positionSensors.*;
import it.unibo.android.sensorData.implementation.environmentSensors.*;
import it.unibo.android.sensorData.interfaces.IAndroidSensorData;

import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

public class AndroidSensorDataUtils {
    /**
     * This class reconstruct an AndroidSensorData and returns an instance of
     * specific AndroidSensorData subclasses.
     * @param json string, received from the device
     * @return an instance of a specific AndroidSensorData subclass
     */
    public static IAndroidSensorData buildDataFromJson(String json){
        JsonParser parser = new JsonParser();
        JsonObject object = parser.parse(json).getAsJsonObject();
        String name = object.get("name").getString();
        long timestamp = object.get("timestamp").getAsLong();
        int accuracy = object.get("accuracy").getAsInt();
        /*
         * MOTION SENSORS
         */
        if(name.equals("AndroidAccelerometerData")){
            return new AndroidAccelerometerData(obtainValues("ThreeAxis", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidGravityData")){
            return new AndroidGravityData(obtainValues("ThreeAxis", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidGyroscopeData")){
            return new AndroidGyroscopeData(obtainValues("ThreeAxis", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidGyroscopeUncalibratedData")){
            return new AndroidGyroscopeUncalibratedData(
                obtainValues("ThreeAxisWithCorrection", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidRotationVectorData")){
            return new AndroidMotionRotationVectorData(obtainValues("RotationVectorData", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidDetectionSensorData")){
            return new AndroidDetectionSensorData(obtainValues("Detection", object),
                accuracy, timestamp, object.get("detectionTime").getAsLong());
        }
    }
}

```

```

        if(name.equals("AndroidStepCounterData")){
            return new AndroidStepCounterData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        /*
         * POSITION SENSORS
         */
        if(name.equals("AndroidMagneticFieldData")){
            return new AndroidMagneticFieldData(obtainValues("ThreeAxis", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidMagneticFieldUncalibratedData")){
            return new AndroidMagneticFieldUncalibratedData(
                obtainValues("ThreeAxisWithCorrection", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidGeomagneticRotationVectorData")){
            return new AndroidGeomagneticRotationVectorData(obtainValues("ThreeAxis", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidGameRotationVectorData")){
            return new AndroidGameRotationVectorData(obtainValues("ThreeAxis", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidProximityData")){
            return new AndroidProximityData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        //Only for retro-compatibility
        if(name.equals("AndroidOrientationData")){
            return new AndroidOrientationData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidTemperatureData")){
            return new AndroidTemperatureData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidLightData")){
            return new AndroidLightData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidPressureData")){
            return new AndroidPressureData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        if(name.equals("AndroidRelativeHumidityData")){
            return new AndroidRelativeHumidityData(obtainValues("SingleValue", object),
                accuracy, timestamp);
        }
        return null;
    }

    private static float[] obtainValues(String type, JsonObject obj) {
        float [] values;
        switch(type){
            case "ThreeAxis":
                values = new float [3];
                values[0] = obj.get("x").getAsFloat();
                values[1] = obj.get("y").getAsFloat();
                values[2] = obj.get("z").getAsFloat();
                return values;
            case "RotationVectorData":
                values = new float [4];
                values[0] = obj.get("componentX").getAsFloat();
                values[1] = obj.get("componentY").getAsFloat();
                values[2] = obj.get("componentZ").getAsFloat();
                try{
                    values[3] = obj.get("componentS").getAsFloat();
                }catch(Exception e){ System.out.println("componentS does not exist");}
                return values;
            case "ThreeAxisWithCorrection":
                values = new float [6];
                values[0] = obj.get("x").getAsFloat();
                values[1] = obj.get("y").getAsFloat();
                values[2] = obj.get("z").getAsFloat();
                values[4] = obj.get("correctionX").getAsFloat();
                values[5] = obj.get("correctionY").getAsFloat();
                values[6] = obj.get("correctionZ").getAsFloat();
                return values;
            case "SingleValue":
                values = new float [1];
                values[0] = obj.get("value").getAsFloat();
                return values;
            default: return new float [3];
        }
    }
}

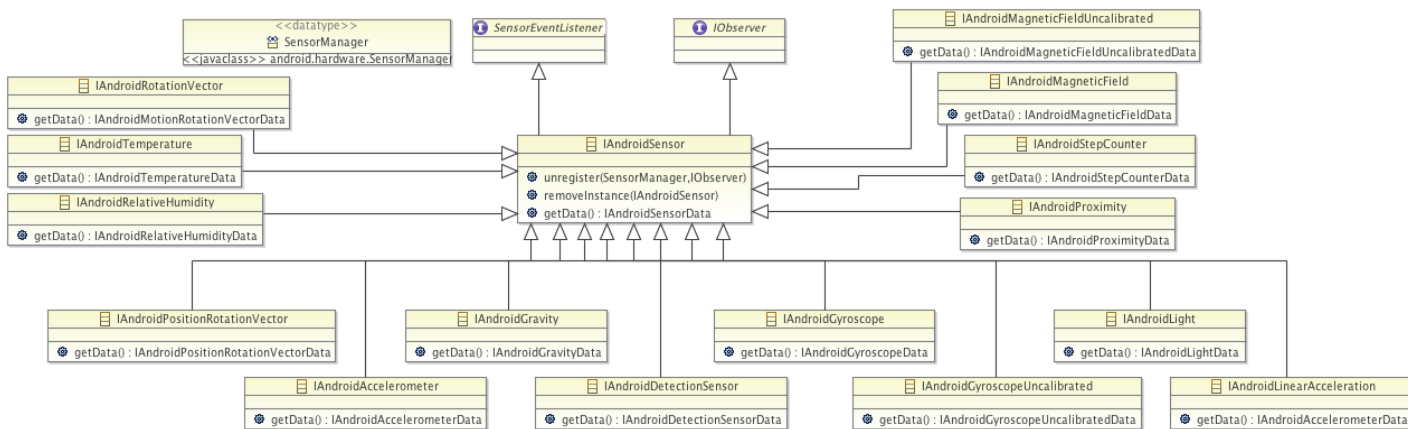
```

### 2.3. *AndroidSensorSupport* package

The *AndroidSensorSupport* package contains all classes that describe a sensor, extending the Android's idea of sensor. As mentioned for Android a sensor data cannot be asked to the sensor and sensors doesn't have a variable that contains last update values. These classes implements the Android's *SensorEventListener* interface and also *IObserver* and *IObservable* interfaces defined in *uniboInterfaces.jar* contained in *it.unibo.iss.libs*. Each sensor extends the *Observable* class, so the developer can attach observer(s) eachone can do different tasks, without interferences with the sensor update (this is the only operation that the *onSensorChange()* method does, followed by the update of observers).

The following image represent the diagram class of the *AndroidSensorSupport* package:

Figure 2.3: *SensorSupport* class diagram



### 2.3.1. AndroidSensor

`AndroidSensor` is the superclass that defines all methods that each sensor must provide to the developer. The interface shows only methods usefull for the developer :

Listing 20: `IAndroidSensor.java`

```
/**
 * @author Carlo Antenucci
 */
package it.unibo.android.sensorSupport.interfaces;

import it.unibo.is.interfaces.IObservable;
import it.unibo.is.interfaces.IObserver;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import it.unibo.android.sensorData.interfaces.IAndroidSensorData;

public interface IAndroidSensor extends SensorEventListener, IObserver, IObservable{
    /**
     * This method deletes observers and unregister the AndroidSensor.
     * @param manager the SensorManager instance
     */
    public void unregister();
    /**
     * Get method that
     * @return an instance of AndroidSensorData (according to the sensor)
     */
    public IAndroidSensorData getData();
}
```

Listing 21: `AndroidSensor.java`

```
package it.unibo.android.sensorSupport.implementations;

import it.unibo.android.sensorSupport.interfaces.IAndroidSensor;
import it.unibo.is.interfaces.IObserver;

import java.util.Observable;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import it.unibo.android.sensorData.implementation.AndroidSensorData;
import it.unibo.android.sensorData.interfaces.IAndroidSensorData;

public abstract class AndroidSensor extends Observable implements IAndroidSensor {

    protected static IAndroidSensorData sensorData;
    protected SensorManager myManager;

    public AndroidSensor(SensorManager manager){
        myManager = manager;
    }

    @Override
    /**
     * @inheritDoc
     *
     * This implementation realize the update of sensorData when the new measurement is notified to this list
     */
    public void onSensorChanged(SensorEvent event) {
        sensorData = new AndroidSensorData(event.values, event.accuracy, event.timestamp);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }

    @Override
    /**
     * When this method is called a notification is sent to all observers
     */
    public void update(Observable arg0, Object arg1) {
        this.setChanged();
        this.notifyObservers( arg1 );
    }

    @Override
    public void addObserver(IObserver arg0) {
        super.addObserver(arg0);
    }

    @Override
    public void unregister(){
        this.deleteObservers();
    }
}
```



```

        myManager.unregisterListener(this);
    }

    @Override
    public IAndroidSensorData getData(){
        return sensorData;
    }
}

```

The interfaces describes all methods useful for the developer:

**unregister()** removes all observers and unregister the **AndroidSensor** from the **SensorManager**. Usefull when the application goes **onPause()**

**getData()** returns the last updated **SensorData**. In this class returns a generic **SensorData**, but in subclasses this method is overwritten by another **getData()** that returns the specific **SensorData**.

while the implementation defines also all “background” method:

**onSensorChanged()** inherited from **SensorEventListener**, that in this case only updates the **SensorData**. In subclassess also call the update method.

**update()** inherited from the **Observable** class. Set his value as changed and then send a notification to all observers with the update data.

**addObserver()** inherited from the **Observable** class. Only call the super method.

As mentioned, all subclasses

- overrides the **onSensorChange()** method updating the specific **SensorData** and then calls the update method passing as paramenter the new **SensorData**.
- overrides the **unregister()** method: deletes the instance from the instances list and then calls the super method.
- overwrite the **getData()** method with a specific method that returns a specific type of **SensorData**.
- defines a **getInstance()** method that, if exist, returns an instance of the specific **AndroidSensor** requested, else calls the constructor and then return this instance.
- defines a private constructor that register the **AndroidSensor** to the **SensorManager** and initialize a new **SensorData** as null (until the first update is not received).

The following code shows an **AndroidAccelerometer**. In the interface the only method introduced is the new **getData()** that returns an **IAndroidAccelerometerData**, while in the implementation are defined the methods described above:

Listing 22: **AndroidAccelerometer.java**

```

package it.unibo.android.sensorSupport.implementations.motionSensors;

import it.unibo.android.sensorSupport.implementations.AndroidSensor;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidAccelerometer;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import it.unibo.android.sensorData.implementation.motionSensors.AndroidAccelerometerData;
import it.unibo.android.sensorData.interfaces.motionSensors.IAndroidAccelerometerData;

public class AndroidAccelerometer extends AndroidSensor implements
    IAndroidAccelerometer {

    private static AndroidAccelerometer[] instances = new AndroidAccelerometer[4];
    protected IAndroidAccelerometerData sensorData;

    /**
     * This method returns, if exist, an instance of AndroidAccelerometer, else
     * the AndroidAccelerometer is created and returned.
     *
     * @param manager the instance of SensorManager
     * @param delay the registration delay defined by one of <b>SensorManager.SENSOR_DELAY_*</b>
     *
     * @return the instance of AndroidAccelerometer
     */
    public static AndroidAccelerometer getInstance(SensorManager manager, int delay){
        Sensor accelerometer = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        if(accelerometer != null){
            int index;
            switch (delay){
                case SensorManager.SENSOR_DELAY_NORMAL:
                    index = 0;
                    break;
                case SensorManager.SENSOR_DELAY_UI:
                    index = 1;
                    break;
                case SensorManager.SENSOR_DELAY_GAME:
                    index = 2;
                    break;
            }
        }
    }
}

```

```

        case SensorManager.SENSOR_DELAY_FASTEST:
            index = 3;
            break;
        default: return null;
    }
    if (instances[index] == null)
        instances[index] = new AndroidAccelerometer(manager, accelerometer, delay);
    return instances[index];
}
else
    return null;
}

private AndroidAccelerometer(SensorManager manager, Sensor accelerometer, int delay){
    super(manager);
    manager.registerListener(this, accelerometer, delay);
}

@Override
public void onSensorChanged(SensorEvent event) {
    sensorData = new AndroidAccelerometerData(event.values, event.accuracy, event.timestamp);
    this.update(this, sensorData);
}

@Override
public IAndroidAccelerometerData getData(){
    return sensorData;
}

@Override
public void unregister(){
    for(int i = 0; i < instances.length; i++)
        if(this.equals(instances[i]))
            instances[i] = null;
    super.unregister();
}
}

```

For each other sensor the implementation is the same, the only difference is the type of **SensorData** and the type of the instances list (obviously is the same of the **AndroidSensor**). For the **SensorData** type is possible to reference with the following table that realizes a match between the sensor type defined by Android, and the new **AndroidSensor** classess with the **AndroidSensorData**:

Table 3: Relation between Sensor Type, AndroidSensors and AndroidSensorData

	Sensor type	AndroidSensor	AndroidSensorData
Env. Sensors	TYPE_AMBIENT_TEMPERATURE	AndroidTemperature	AndroidTemperatureData
	TYPE_LIGHT	AndroidLight	AndroidLightData
	TYPE_PRESSURE	AndroidPressure	AndroidPressureData
	TYPE_RELATIVE_HUMIDITY	AndroidRelativeHumidity	AndroidRelativeHumidityData
	TYPE_TEMPERATURE <sup>a</sup>	AndroidTemperature	AndroidTemperatureData
Motion Sensors	TYPE_ACCELEROMETER	AndroidAccelerometer	AndroidAccelerometerData
	TYPE_GRAVITY	AndroidGravity	AndroidGravityData
	TYPE_GYROSCOPE	AndroidGyroscope	AndroidGyroscopeData
	TYPE_GYROSCOPE_UNCALIBRATED	AndroidGyroscopeUncalibrated	AndroidGyroscopeUncalibratedData
	TYPE_LINEAR_ACCELERATION	AndroidLinearAcceleration	AndroidAccelerometerData
	TYPE_ROTATION_VECTOR	AndroidRotationVector	AndroidMotionRotationVectorData
	TYPE_SIGNIFICANT_MOTION	AndroidSignificantMotion	AndroidDetectionSensorData
	TYPE_STEP_COUNTER	SensorStepCounter	AndroidStepCounterData
	TYPE_STEP_DETECTOR	AndroidStepDetector	AndroidDetectionSensorData
Position Sensors	TYPE_GAME_ROTATION_VECTOR	AndroidGameRotationVector	AndroidPositionRotationVectorData
	TYPE_GEOMAGNETIC_ROTATION_VECTOR	AndroidGeomagneticRotationVector	AndroidPositionRotationVectorData
	TYPE_MAGNETIC_FIELD	AndroidMagneticField	AndroidMagneticFieldData
	TYPE_MAGNETIC_FIELD_UNCALIBRATED	AndroidMagneticFieldUncalibrated	AndroidMagneticFieldUncalibratedData
	TYPE_ORIENTATION <sup>b</sup>	AndroidOrientation	AndroidOrientationData
	TYPE_PROXIMITY	AndroidProximity	AndroidProximityData

<sup>a</sup> This sensor type is replaced in Android 4.0 (API Level 14) with TYPE\_AMBIENT\_TEMPERATURE.

<sup>b</sup> This sensor was deprecated in Android 2.2 (API Level 8). The sensor framework provides alternate methods for acquiring device orientation.

### 2.3.2. AndroidSensorFactory

This class is defined in the main implementation package (`it.unibo.android.sensorSupport.implementation`) and provides methods to obtain an instance of a particular `AndroidSensor`. Developers can ask to the factory a specific sensor in two methods:

`getInstance()` passing the type of sensor as a parameter (with the delay, the observer and the `SensorManager`).  
`get[SENSOR]()` where `[SENSOR]` is the name of the sensor. In this case developer must pass to the factory only the `SensorManager` and the delay; there are two possibilities: developer can pass also the observer or not (the observer can be not necessary, for example if the application not constantly need notification of the update but only want to request `SensorData` in specific times).

AndroidSensorFactory code is shown below:

Listing 23: AndroidSensorFactory.java

```
package it.unibo.android.sensorSupport.implementations;

import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidAmbientTemperature;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidLight;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidPressure;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidRelativeHumidity;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidTemperature;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidAccelerometer;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidGravity;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidGyroscope;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidGyroscopeUncalibrated;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidLinearAcceleration;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidRotationVector;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidSignificantMotion;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidStepCounter;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidStepDetector;
import it.unibo.android.sensorSupport.implementations.positionSensors.AndroidGameRotationVector;
import it.unibo.android.sensorSupport.implementations.positionSensors.AndroidMagneticField;
```

```

import it.unibo.android.sensorSupport.implementations.positionSensors.AndroidMagneticFieldUncalibrated;
import it.unibo.android.sensorSupport.implementations.positionSensors.AndroidOrientation;
import it.unibo.android.sensorSupport.implementations.positionSensors.AndroidProximity;
import it.unibo.android.sensorSupport.interfaces.IAndroidSensor;
import it.unibo.android.sensorSupport.interfaces.environmentSensors.IAndroidLight;
import it.unibo.android.sensorSupport.interfaces.environmentSensors.IAndroidPressure;
import it.unibo.android.sensorSupport.interfaces.environmentSensors.IAndroidRelativeHumidity;
import it.unibo.android.sensorSupport.interfaces.environmentSensors.IAndroidTemperature;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidAccelerometer;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidDetectionSensor;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidGravity;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidGyroscope;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidGyroscopeUncalibrated;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidLinearAcceleration;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidRotationVector;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidStepCounter;
import it.unibo.android.sensorSupport.interfaces.positionSensors.IAndroidMagneticField;
import it.unibo.android.sensorSupport.interfaces.positionSensors.IAndroidMagneticFieldUncalibrated;
import it.unibo.android.sensorSupport.interfaces.positionSensors.IAndroidOrientation;
import it.unibo.android.sensorSupport.interfaces.positionSensors.IAndroidPositionRotationVector;
import it.unibo.android.sensorSupport.interfaces.positionSensors.IAndroidProximity;
import it.unibo.is.interfaces.IObserver;
import android.hardware.Sensor;
import android.hardware.SensorManager;

public class AndroidSensorFactory {
    /**
     * This method return an instance of an AndroidSensor.
     *
     * @param manager the sensor manager
     * @param type the type of sensor you need
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidSensor
     */
    public static IAndroidSensor getSensor(SensorManager manager, int type, int delay, IObserver observer){
        IAndroidSensor sensor = null;
        switch(type){

            /*
             * MOTION SENSORS
             */

            case Sensor.TYPE_ACCELEROMETER:
                sensor = AndroidAccelerometer.getInstance(manager, delay);
                break;
            case Sensor.TYPE_LINEAR_ACCELERATION:
                sensor = AndroidLinearAcceleration.getInstance(manager, delay);
                break;
            case Sensor.TYPE_GYROSCOPE:
                sensor = AndroidGyroscope.getInstance(manager, delay);
                break;
            case Sensor.TYPE_GYROSCOPE_UNCALIBRATED:
                sensor = AndroidGyroscopeUncalibrated.getInstance(manager, delay);
                break;
            case Sensor.TYPE_GRAVITY:
                sensor = AndroidGravity.getInstance(manager, delay);
                break;
            case Sensor.TYPE_ROTATION_VECTOR:
                sensor = AndroidRotationVector.getInstance(manager, delay);
                break;
            case Sensor.TYPE_SIGNIFICANT_MOTION:
                sensor = AndroidSignificantMotion.getInstance(manager, delay);
                break;
            case Sensor.TYPE_STEP_DETECTOR:
                sensor = AndroidStepDetector.getInstance(manager, delay);
                break;
            case Sensor.TYPE_STEP_COUNTER:
                sensor = AndroidStepCounter.getInstance(manager, delay);
                break;

            /*
             * POSITION SENSORS
             */

            case Sensor.TYPE_MAGNETIC_FIELD:
                sensor = AndroidMagneticField.getInstance(manager, delay);
                break;
            case Sensor.TYPE_MAGNETIC_FIELD_UNCALIBRATED:
                sensor = AndroidMagneticFieldUncalibrated.getInstance(manager, delay);
                break;
            case Sensor.TYPE_GAME_ROTATION_VECTOR:
                sensor = AndroidGameRotationVector.getInstance(manager, delay);
                break;
            case Sensor.TYPE_GEOMAGNETIC_ROTATION_VECTOR:
                sensor = AndroidGeomagneticRotationVector.getInstance(manager, delay);

```

```

        break;
    case Sensor.TYPE_PROXIMITY:
        sensor = AndroidProximity.getInstance(manager, delay);
        break;
    //Only for retro-compatibility
    case Sensor.TYPE_ORIENTATION:
        sensor = AndroidOrientation.getInstance(manager, delay);
        break;

    /*
     * ENVIRONMENT SENSORS
     */

    case Sensor.TYPE_AMBIENT_TEMPERATURE:
        sensor = AndroidAmbientTemperature.getInstance(manager, delay);
        break;
    case Sensor.TYPE_LIGHT:
        sensor = AndroidLight.getInstance(manager, delay);
        break;
    case Sensor.TYPE_PRESSURE:
        sensor = AndroidPressure.getInstance(manager, delay);
        break;
    case Sensor.TYPE_RELATIVE_HUMIDITY:
        sensor = AndroidRelativeHumidity.getInstance(manager, delay);
        break;
    //Only for retro-compatibility
    case Sensor.TYPE_TEMPERATURE:
        sensor = AndroidTemperature.getInstance(manager, delay);
        break;
    default: return null;
}
if(sensor != null)
    sensor.addObserver(observer);
return sensor;
}

/*
 * MOTION SENSORS
 */

/**
 * This method return an instance of an AndroidAccelerometer
 * @param manager the sensor manager
 * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
 * @return an instance of an AndroidAccelerometer
 */
public static IAndroidAccelerometer getAccelerometer(SensorManager manager, int delay){
    return AndroidAccelerometer.getInstance(manager, delay);
}

/**
 * This method return an instance of an AndroidAccelerometer and registers to it the observer
 *
 * @param manager the sensor manager
 * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
 * @param observer the observer of the sensor
 *
 * @return an instance of an AndroidAccelerometer
 */
public static IAndroidAccelerometer getAccelerometer(SensorManager manager,
    int delay, IObserver observer) {
    IAndroidAccelerometer a = getAccelerometer(manager, delay);
    if (a != null)
        a.addObserver(observer);
    return a;
}

/**
 * This method return an instance of an AndroidLinearAcceleration
 * @param manager the sensor manager
 * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
 * @return an instance of an AndroidLinearAcceleration
 */
public static IAndroidLinearAcceleration getLinearAcceleration(SensorManager manager, int delay){
    return AndroidLinearAcceleration.getInstance(manager, delay);
}

/**
 * This method return an instance of an AndroidLinearAcceleration and registers to it the observer
 *
 * @param manager the sensor manager
 * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
 * @param observer the observer of the sensor
 *
 * @return an instance of an AndroidLinearAcceleration

```

```

    */
    public static IAndroidLinearAcceleration getLinearAcceleration(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidLinearAcceleration a = getLinearAcceleration(manager, delay);
        if (a != null)
            a.addObserver(observer);
        return a;
    }

    /**
     * This method return an instance of an AndroidGyroscope
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidGyroscope
     */
    public static IAndroidGyroscope getGyroscope(SensorManager manager, int delay){
        return AndroidGyroscope.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidGyroscope and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidGyroscope
     */
    public static IAndroidGyroscope getGyroscope(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidGyroscope g = getGyroscope(manager, delay);
        if (g != null)
            g.addObserver(observer);
        return g;
    }

    /**
     * This method return an instance of an AndroidGyroscopeUncalibrated
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidGyroscopeUncalibrated
     */
    public static IAndroidGyroscopeUncalibrated getGyroscopeUncalibrated(SensorManager manager, int delay){
        return AndroidGyroscopeUncalibrated.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidGyroscopeUncalibrated and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidGyroscopeUncalibrated
     */
    public static IAndroidGyroscopeUncalibrated getGyroscopeUncalibrated(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidGyroscopeUncalibrated g = getGyroscopeUncalibrated(manager, delay);
        if (g != null)
            g.addObserver(observer);
        return g;
    }

    /**
     * This method return an instance of an AndroidGravity
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidGravity
     */
    public static IAndroidGravity getGravity(SensorManager manager, int delay){
        return AndroidGravity.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidGravity and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidGravity
     */
    public static IAndroidGravity getGravity(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidGravity g = getGravity(manager, delay);
        if (g != null)
            g.addObserver(observer);
    }

```

```

        return g;
    }

    /**
     * This method return an instance of an AndroidRotationVector
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidRotationVector
     */
    public static IAndroidRotationVector getRotationVector(SensorManager manager, int delay){
        return AndroidRotationVector.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidRotationVector and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidRotationVector
     */
    public static IAndroidRotationVector getRotationVector(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidRotationVector r = getRotationVector(manager, delay);
        if (r != null)
            r.addObserver(observer);
        return r;
    }

    /**
     * This method return an instance of an AndroidStepDetector
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidStepDetector
     */
    public static IAndroidDetectionSensor getStepDetector(SensorManager manager, int delay){
        return AndroidStepDetector.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidStepDetector and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidStepDetector
     */
    public static IAndroidDetectionSensor getStepDetector(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidDetectionSensor s = getStepDetector(manager, delay);
        if (s != null)
            s.addObserver(observer);
        return s;
    }

    /**
     * This method return an instance of an AndroidSignificantMotion
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidSignificantMotion
     */
    public static IAndroidDetectionSensor getSignificantMotion(SensorManager manager, int delay){
        return AndroidSignificantMotion.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidSignificantMotion and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidSignificantMotion
     */
    public static IAndroidDetectionSensor getSignificantMotion(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidDetectionSensor s = getSignificantMotion(manager, delay);
        if (s != null)
            s.addObserver(observer);
        return s;
    }

    /**
     * This method return an instance of an AndroidStepCounter
     * @param manager the sensor manager

```

```

    * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
    * @return an instance of an AndroidStepCounter
    */
    public static IAndroidStepCounter getStepCounter(SensorManager manager, int delay){
        return AndroidStepCounter.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidStepCounter and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidStepCounter
     */
    public static IAndroidStepCounter getStepCounter(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidStepCounter s = getStepCounter(manager, delay);
        if (s != null)
            s.addObserver(observer);
        return s;
    }

    /**
     * POSITION SENSORS
     */

    /**
     * This method return an instance of an AndroidProximity
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidProximity
     */
    public static IAndroidProximity getProximity(SensorManager manager, int delay){
        return AndroidProximity.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidProximity and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidProximity
     */
    public static IAndroidProximity getProximity(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidProximity p = getProximity(manager, delay);
        if (p != null)
            p.addObserver(observer);
        return p;
    }

    /**
     * This method return an instance of an AndroidMagneticField
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidMagneticField
     */
    public static IAndroidMagneticField getMagnetometer(SensorManager manager, int delay){
        return AndroidMagneticField.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidMagneticField and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidMagneticField
     */
    public static IAndroidMagneticField getMagnetometer(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidMagneticField m = getMagnetometer(manager, delay);
        if (m != null)
            m.addObserver(observer);
        return m;
    }

    /**
     * This method return an instance of an AndroidMagneticFieldUncalibrated
     * @param manager the sensor manager

```



```

    * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
    * @return an instance of an AndroidMagneticFieldUncalibrated
    */
    public static IAndroidMagneticFieldUncalibrated getMagneticFieldUncalibrated(SensorManager manager, int delay) {
        return AndroidMagneticFieldUncalibrated.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidMagneticFieldUncalibrated and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidMagneticFieldUncalibrated
     */
    public static IAndroidMagneticFieldUncalibrated getMagneticFieldUncalibrated(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidMagneticFieldUncalibrated m = getMagneticFieldUncalibrated(manager, delay);
        if (m != null)
            m.addObserver(observer);
        return m;
    }

    /**
     * This method return an instance of an AndroidGameRotationVector
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidGameRotationVector
     */
    public static IAndroidPositionRotationVector getGameRotationVector(SensorManager manager, int delay){
        return AndroidGameRotationVector.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidGameRotationVector and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidGameRotationVector
     */
    public static IAndroidPositionRotationVector getGameRotationVector(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidPositionRotationVector g = getGameRotationVector(manager, delay);
        if (g != null)
            g.addObserver(observer);
        return g;
    }

    /**
     * This method return an instance of an AndroidGeomagneticRotationVector
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidGeomagneticRotationVector
     */
    public static IAndroidPositionRotationVector getGeomagneticRotationVector(SensorManager manager, int delay){
        return AndroidGeomagneticRotationVector.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidGeomagneticRotationVector and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidGeomagneticRotationVector
     */
    public static IAndroidPositionRotationVector getGeomagneticRotationVector(SensorManager manager,
        int delay, IObserver observer) {
        IAndroidPositionRotationVector g = getGeomagneticRotationVector(manager, delay);
        if (g != null)
            g.addObserver(observer);
        return g;
    }
}

@Deprecated
/**
 * This method is deprecated. Is defined only for retro-compatibility.
 * It return an instance of an AndroidOrientation
 *
 * @param manager the sensor manager
 * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
 * @return an instance of an AndroidOrientation
 */
public static IAndroidOrientation getOrientation(SensorManager manager, int delay){

```

```

        return AndroidOrientation.getInstance(manager, delay);
    }

    @Deprecated
    /**
     * This method is deprecated. Is defined only for retro-compatibility.
     * It return an instance of an AndroidOrientation and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidOrientation
     */
    public static IAndroidOrientation getOrientation(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidOrientation o = getOrientation(manager, delay);
        if (o != null)
            o.addObserver(observer);
        return o;
    }

    /**
     *
     * ENVIRONMENT SENSORS
     *
     */

    /**
     * This method return an instance of an AndroidAmbientTemperature
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidAmbientTemperature
     */
    public static IAndroidTemperature getAmbientTemperatureSensor(SensorManager manager, int delay){
        return AndroidAmbientTemperature.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidAmbientTemperature and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidAmbientTemperature
     */
    public static IAndroidTemperature getAmbientTemperatureSensor(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidTemperature t = getAmbientTemperatureSensor(manager, delay);
        if (t != null)
            t.addObserver(observer);
        return t;
    }

    /**
     * This method return an instance of an AndroidLight
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidLight
     */
    public static IAndroidLight getLightSensor(SensorManager manager, int delay){
        return AndroidLight.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidLight and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidLight
     */
    public static IAndroidLight getLightSensor(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidLight l = getLightSensor(manager, delay);
        if (l != null)
            l.addObserver(observer);
        return l;
    }

    /**
     * This method return an instance of an AndroidPressure
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidPressure
     */

```

```

    */
    public static IAndroidPressure getPressureSensor(SensorManager manager, int delay){
        return AndroidPressure.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidPressure and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidPressure
     */
    public static IAndroidPressure getPressureSensor(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidPressure p = getPressureSensor(manager, delay);
        if (p != null)
            p.addObserver(observer);
        return p;
    }

    /**
     * This method return an instance of an AndroidRelativeHumidity
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidRelativeHumidity
     */
    public static IAndroidRelativeHumidity getRelativeHumiditySensor(SensorManager manager, int delay){
        return AndroidRelativeHumidity.getInstance(manager, delay);
    }

    /**
     * This method return an instance of an AndroidRelativeHumidity and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidRelativeHumidity
     */
    public static IAndroidRelativeHumidity getRelativeHumiditySensor(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidRelativeHumidity h = getRelativeHumiditySensor(manager, delay);
        if (h != null)
            h.addObserver(observer);
        return h;
    }

    @Deprecated
    /**
     * This method is deprecated. Is defined only for retro-compatibility.
     * It return an instance of an AndroidOrientation
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @return an instance of an AndroidTemperature
     */
    public static IAndroidTemperature getTemperature(SensorManager manager, int delay){
        return AndroidTemperature.getInstance(manager, delay);
    }

    @Deprecated
    /**
     * This method is deprecated. Is defined only for retro-compatibility.
     * It return an instance of an AndroidOrientation and registers to it the observer
     *
     * @param manager the sensor manager
     * @param delay the sensor delay that you can choose from SensorManager.SENSOR_DELAY_*
     * @param observer the observer of the sensor
     *
     * @return an instance of an AndroidTemperature
     */
    public static IAndroidTemperature getTemperature(SensorManager manager,
        int delay, IObservable observer) {
        IAndroidTemperature t = getTemperature(manager, delay);
        if (t != null)
            t.addObserver(observer);
        return t;
    }
}

```

Last two methods are defined also for each other sensors.

## 2.4. How to use AndroidSensorSupport

The `AndroidSensorSupport` layer, as mentioned, simplifies the sensors utilization, and provides to the developer methods to obtain the last updated `SensorData` overcoming the constraint of wait the next sensor update or save values each time (it does this operation automatically and transparently).

In this section is shown how to use this layer also using an example.

### 2.4.1. Request a Sensor

To obtain an instance of a sensor developer have to ask it to the `AndroidSensorFactory` that provides to discriminate the type of sensor and then forwards the request to the specific `AndroidSensor`. There are three methods to obtain an `AndroidSensor`:

#### 1. `getSensor(SensorManager, SensorType, Delay, Observer)`

This method needs the sensor manager, the type of sensor that the developer wants the delay (one of each defined in `SensorManager` class) and the observer that receives a notification each time the `SensorData` changes. The observer can be useful to do some operations with `SensorData` or apply threshold.

In this case is necessary apply a cast to the `AndroidSensor` type because the `getSensor` method returns a generic `IAAndroidSensor`.

```
IAAndroidAccelerometer accelerometer = (IAAndroidAccelerometer)
    AndroidSensorFactory.getSensor( manager,
                                   SensorManager.SENSOR_TYPE_ACCELEROMETER,
                                   SensorManager.SENSOR_DELAY_NORMAL,
                                   observer);
```

#### 2. `get[SENSOR_TYPE](SensorManager, Delay, Observer)`

This method returns a specific type of `AndroidSensor` (specified in the method name e.g. `getAccelerometer(...)` or `getLightSensor(...)`), then the cast is not necessary. Like the `getSensor(...)` method needs the sensor manager instance, the delay and the observer.

```
AndroidLight light = AndroidSensorFactory.getLightSensor(manager, delay, observer);
```

#### 3. `get[SENSOR_TYPE](SensorManager, Delay, Observer)`

Like the previous method, this returns a specific type of `AndroidSensor` too. This method doesn't need the observer, then the developer request a sensor that doesn't notifies changes but his data can be obtained calling the `getData()` method.

```
AndroidProximity proximity = AndroidSensorFactory.getProximitySensor(manager, delay);
```

### 2.4.2. Unregister sensor

To unregister a sensor, for example when the application goes `onPause()` and is no longer necessary the sensor usage, developer can call the `unregister(...)` method that detaches the sensor from the sensor manager (that must be passed as parameter).

```
gyroscope.unregister(manager);
```

### 2.4.3. Operation with SensorData

There are two ways to do operations whit `SensorData`:

#### 1. Using the `getData()` method

`getData()` returns the `SensorData` which contains all informations about the last update. After obtaining data, developers, can do what they wants with this:

```
AndroidLight light = AndroidSensorFactory.getLightSensor(manager, delay);
AndroidLightData data = light.getData();
if(data.getValue < 50)
    System.out.println("turn_on_the_flash_light");
else
    System.out.println("sunlight_is_precious");
```

## 2. Doing something in the observer

the observer registered to the sensor receives a notification each time the sensor updates his data. Developer can do in the observer filtering operations. Can be defined only one observer for each sensor (in this case developer must check which sensor updates data) or an observer for only one sensor. In the following code is shown how discriminate which sensor updates the observer:

```
Observer o = new Observer(){
    @Override
    public void update(Observable observable, Object data) {
        if(arg0 instanceof AndroidAccelerometer){
            AndroidAccelerometerData d = (AndroidAccelerometerData)arg1;
            if(d.getAcceleration()>THRESHOLD)
                send(d.getJsonRep());
        }
        else if (arg0 instanceof AndroidProximity){
            AndroidProximityData d = (AndroidProximityData)arg1;
            if (d.isPresent)
                System.out.println("Obstacle_detected");
        }
    }
    private void send(String msg){
        ...
    }
};
```

#### 2.4.4. Example<sup>4</sup>

In this example are shown all previous methods used in an Android application that shows `SensorData` obtained from four different sensors (Accelerometer, Proximity, Light and Temperature), two of which have the same observer and the other do not have observers.

Next code shows the layout used for the application:

Listing 24: `sensorsupport_example.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/white"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:paddingTop="20dip"
        android:text="@string/title"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:textStyle="bold" />

    <TableLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingTop="10dip"
        android:stretchColumns="*" >

        <TableRow>

            <TextView
                android:id="@+id/sen1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="@string/sensor1"
                android:textColor="@color/dark_grey"
                android:textSize="14sp" />

            <TextView
                android:id="@+id/sen2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="@string/sensor2"
                android:textColor="@color/dark_grey"
                android:textSize="14sp" />

            <TextView
                android:id="@+id/sen3"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="@string/sensor3"
                android:textColor="@color/dark_grey"
                android:textSize="14sp" />

            <TextView
                android:id="@+id/sen4"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="@string/sensor4"
                android:textColor="@color/dark_grey"
                android:textSize="14sp" />

        </TableRow>

    </TableLayout>

</LinearLayout>
```

---

<sup>4</sup> This app is available on github in `sensorSupport.example` project

```

</TableRow>

<TableRow>

    <TextView
        android:id="@+id/s1x"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView
        android:id="@+id/s2x"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView
        android:id="@+id/s3x"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView
        android:id="@+id/s4x"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />
</TableRow>
<TableRow>

    <TextView
        android:id="@+id/s1y"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView
        android:id="@+id/s2y"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView
        android:id="@+id/s3y"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView
        android:id="@+id/s4y"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />
</TableRow>
<TableRow>

    <TextView
        android:id="@+id/s1z"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

    <TextView

```

```

        android:id="@+id/s2z"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="@color/dark_grey" />

<TextView
    android:id="@+id/s3z"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:textColor="@color/dark_grey" />

<TextView
    android:id="@+id/s4z"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:textColor="@color/dark_grey" />

</TableRow>
</TableLayout>

<TextView
    android:id="@+id/other"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:paddingTop="20dip"
    android:text="@string/jsonRep"
    android:textColor="@color/dark_grey"
    android:textSize="16sp"
    android:textStyle="bold" />

</LinearLayout>

```

In this project are presents, in the src directory two packagers: the first one contains the `SensorObserver`, while the second contains the `ExampleActivity`.

**SensorObserver** The `SensorObserver` implements the `IObserver` interface and realize the only one observer for each sensor. The constructor save the instance of the `Activity` as `IOutputView` then, when the `AndroidSensor` notifies the update use this reference to update the GUI.

This is the code:

Listing 25: `SensorObserver.java`

```

package it.unibo.android.sensorSupport.observer;

import java.util.Observable;

import it.unibo.android.sensorData.implementation.environmentSensors.AndroidLightData;
import it.unibo.android.sensorData.implementation.environmentSensors.AndroidTemperatureData;
import it.unibo.android.sensorData.implementation.motionSensors.AndroidAccelerometerData;
import it.unibo.android.sensorData.implementation.positionSensors.AndroidProximityData;
import it.unibo.android.sensorData.interfaces.IAndroidSensorData;
import it.unibo.android.sensorSupport.example.ExampleActivity;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidAmbientTemperature;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidLight;
import it.unibo.android.sensorSupport.implementations.environmentSensors.AndroidTemperature;
import it.unibo.android.sensorSupport.implementations.motionSensors.AndroidAccelerometer;
import it.unibo.android.sensorSupport.implementations.positionSensors.AndroidProximity;
import it.unibo.is.interfaces.IObserver;

public class SensorObserver implements IObserver {

    ExampleActivity activity;

    public SensorObserver(ExampleActivity v){
        activity = v;
    }

    @Override
    public void update(Observable arg0, Object arg1) {
        if(arg0 instanceof AndroidAccelerometer){
            AndroidAccelerometerData d = (AndroidAccelerometerData) arg1;

```



```

        activity.setData(d.getJsonRep());
    }
    else if(arg0 instanceof AndroidProximity){
        AndroidProximityData d = (AndroidProximityData) arg1;
        activity.setData(d.getJsonRep());
    }
    else if(arg0 instanceof AndroidLight){
        AndroidLightData d = (AndroidLightData) arg1;
        activity.setData(d.getJsonRep());
    }
    else if((arg0 instanceof AndroidTemperature) || (arg0 instanceof AndroidAmbientTemperature)){
        AndroidTemperatureData d = (AndroidTemperatureData) arg1;
        activity.setData(d.getJsonRep());
    }
    activity.sensorUpdated(((IAAndroidSensorData) arg1).getName());
}
}

```

**ExampleActivity** The **ExampleActivity** implements **IOutputView**. This interface provides methods to update the GUI, the only one used in this application is **setOutput(msg)**. In this method is used the **AndroidSensorDataUtils** to rebuild the **SensorData** from a JSON string, then, according with the **SensorData** type the application updates the interested text field.

Listing 26: ExampleActivity.java

```

/**
 * @author Carlo Antenucci
 */

package it.unibo.android.sensorSupport.example;

import it.unibo.android.sensorSupport.implementations.AndroidSensorFactory;
import it.unibo.android.sensorSupport.interfaces.environmentSensors.IAndroidLight;
import it.unibo.android.sensorSupport.interfaces.environmentSensors.IAndroidTemperature;
import it.unibo.android.sensorSupport.interfaces.motionSensors.IAndroidAccelerometer;
import it.unibo.android.sensorSupport.interfaces.positionSensors.IAndroidProximity;
import it.unibo.android.sensorSupport.observer.SensorObserver;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorManager;

public class ExampleActivity extends GuiActivity {

    private SensorManager mSensorManager;
    private IAndroidAccelerometer sensor_1;
    private IAndroidProximity sensor_2;
    private IAndroidLight sensor_3;
    private IAndroidTemperature sensor_4;

    private SensorObserver observer;

    @Override
    protected void onStart() {
        super.onStart();
        //obtain sensor manager
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        //ask an instances of sensors
        initSensors();
        //complete the GUI
        String[] names = {
            "Accelerometer",
            "Proximity",
            "Light",
            "Temperatures"
        };
        super.initGui(names);
    }

    @Override
    protected void onResume() {
        super.onResume();
        if((sensor_1 == null) && (sensor_2 == null) && (sensor_3 == null) && (sensor_4 == null))
            initSensors();
    }

    @Override
    protected void onPause() {
        super.onPause();
    }
}

```

```

        if(sensor_1 != null)
            sensor_1.unregister();
        if(sensor_2 != null)
            sensor_2.unregister();
        if(sensor_3 != null)
            sensor_3.unregister();
        if(sensor_4 != null)
            sensor_4.unregister();
        sensor_1 = null;
        sensor_2 = null;
        sensor_3 = null;
        sensor_4 = null;
    }

    private void initSensors(){
        observer = new SensorObserver(this);

        sensor_1 = (IAAndroidAccelerometer) AndroidSensorFactory.getSensor(mSensorManager,
            Sensor.TYPE_ACCELEROMETER,
            SensorManager.SENSOR_DELAY_NORMAL,
            observer);
        sensor_2 = AndroidSensorFactory.getProximity(mSensorManager,
            SensorManager.SENSOR_DELAY_FASTEST,
            observer);
        sensor_3 = AndroidSensorFactory.getLightSensor(mSensorManager,
            SensorManager.SENSOR_DELAY_NORMAL);
        sensor_4 = AndroidSensorFactory.getAmbientTemperatureSensor(mSensorManager,
            SensorManager.SENSOR_DELAY_NORMAL);
        if(sensor_4 == null)
            sensor_4 = AndroidSensorFactory.getTemperature(mSensorManager,
                SensorManager.SENSOR_DELAY_NORMAL);
    }

    public void sensorUpdated(String name){
        switch(name){
            case "AndroidAccelerometerData":
                if(sensor_4 != null)
                    setJSONRep(sensor_4.getData().getJsonRep());
                else
                    setJSONRep("sensor_4␣not␣present\n\n" + sensor_1.getData().getJsonRep());
                break;
            case "AndroidProximityData":
                if(sensor_3 != null)
                    setJSONRep(sensor_3.getData().getJsonRep());
                else
                    setJSONRep("sensor_3␣not␣present\n\n" + sensor_2.getData().getJsonRep());
                break;
            case "AndroidLightData":
                if(sensor_4 != null)
                    setJSONRep(sensor_2.getData().getJsonRep());
                else
                    setJSONRep("sensor_2␣not␣present\n\n" + sensor_3.getData().getJsonRep());
                break;
            case "AndroidTemperatureData":
                if(sensor_4 != null)
                    setJSONRep(sensor_1.getData().getJsonRep());
                else
                    setJSONRep("sensor_1␣not␣present\n\n" + sensor_4.getData().getJsonRep());
                break;
        }
    }
}

```

### 3. Conclusions

In this paper is defined an Android sensor model that offers a layer between Android Sensor Framework and the Android application.

This layer provides some useful features:

- the new sensor classes are not only observer of the real sensor but also observable
- the `onSensorChange()` method updates transparently the `SensorData` and notifies this update to all observers
- in introduced a `SensorData` model that contains all sensor data informations relative to the last update and provide a JSON and a Prolog representation
- is defined an `AndroidSensorFactory` that returns a specific sensor and provides to registrate the `AndroidSensor` to the `SensorManager`
- is defined an `SensorDataUtils` class that rebuild the `SensorData` from a JSON string representation

## References

- [1] Android Developer, "Sensors Overview",  
[http://developer.android.com/guide/topics/sensors/sensors\\_overview.htm](http://developer.android.com/guide/topics/sensors/sensors_overview.htm)
- [2] Android Open Source Project (AOSP) Issue Tracker, "SensorEvent timestamp field incorrectly populated on Nexus 4 devices", <https://code.google.com/p/android/issues/detail?id=56561#c2>