# Sensors in Android Apps

Ausgewählte Kapitel Mobile Applications

Prof. Dr.-Ing. V.Brovkov

# Sensors in Android

| Sensor | Type | Description | Common Uses |
|---|---|---|---|
| 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. | Motion detection (shake, tilt, etc.). |
| TYPE_AMBIENT_TEMPERATURE | Hardware | Measures the ambient room temperature in degrees Celsius (°C). See note below. | Monitoring air temperatures. |
| 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, z). | Motion detection (shake, tilt, etc.). |
| TYPE_GYROSCOPE | Hardware | Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z). | Rotation detection (spin, turn, etc.). |
| TYPE_LIGHT | Hardware | Measures the ambient light level (illumination) in lx. | Controlling screen brightness. |
| TYPE_LINEAR_ACCELERATION | Software or Hardware | 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. | Monitoring acceleration along a single axis. |

# Sensors in Android

| | | | |
|---|---|---|---|
| TYPE_MAGNETIC_FIELD | Hardware | Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT. | Creating a compass. |
| 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 getRotationMatrix() method. | Determining device position. |
| TYPE_PRESSURE | Hardware | Measures the ambient air pressure in hPa or mbar. | Monitoring air pressure changes. |
| TYPE_PROXIMITY | Hardware | Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. | Phone position during a call. |
| TYPE_RELATIVE_HUMIDITY | Hardware | Measures the relative ambient humidity in percent (%). | Monitoring dewpoint, absolute, and relative humidity. |
| TYPE_ROTATION_VECTOR | Software or Hardware | Measures the orientation of a device by providing the three elements of the device's rotation vector. | Motion detection and rotation detection. |
| TYPE_TEMPERATURE | Hardware | Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14 | Monitoring temperatures. |

# Sensors in Android

## Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the `android.hardware` package and includes the following classes and interfaces:

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

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

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

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

# Sensors in Android

- **Identifying sensors and sensor capabilities**

  Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. For example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.

- **Monitor sensor events**

  Monitoring sensor events is how you acquire raw sensor data. A sensor event occurs every time a sensor detects a change in the parameters it is measuring. A sensor event provides you with four pieces of information: the name of the sensor that triggered the event, the timestamp for the event, the accuracy of the event, and the raw sensor data that triggered the event.

# Sensors in Android

**Table 2.** Sensor availability by platform.

| Sensor | Android 4.0 (API Level 14) | Android 2.3 (API Level 9) | Android 2.2 (API Level 8) | Android 1.5 (API Level 3) |
|---|---|---|---|---|
| TYPE_ACCELEROMETER | **Yes** | **Yes** | **Yes** | **Yes** |
| TYPE_AMBIENT_TEMPERATURE | **Yes** | n/a | n/a | n/a |
| TYPE_GRAVITY | **Yes** | **Yes** | n/a | n/a |
| TYPE_GYROSCOPE | **Yes** | **Yes** | n/a[1] | n/a[1] |
| TYPE_LIGHT | **Yes** | **Yes** | **Yes** | **Yes** |
| TYPE_LINEAR_ACCELERATION | **Yes** | **Yes** | n/a | n/a |
| TYPE_MAGNETIC_FIELD | **Yes** | **Yes** | **Yes** | **Yes** |
| TYPE_ORIENTATION | **Yes**[2] | **Yes**[2] | **Yes**[2] | **Yes** |
| TYPE_PRESSURE | **Yes** | **Yes** | n/a[1] | n/a[1] |
| TYPE_PROXIMITY | **Yes** | **Yes** | **Yes** | **Yes** |
| TYPE_RELATIVE_HUMIDITY | **Yes** | n/a | n/a | n/a |
| TYPE_ROTATION_VECTOR | **Yes** | **Yes** | n/a | n/a |
| TYPE_TEMPERATURE | **Yes**[2] | **Yes** | **Yes** | **Yes** |

# Sensors List
# Light Sensor

http://startandroid.ru/ru/uroki/vse-uroki-spiskom/287-urok-137-sensory-uskorenie-orientatsija.html

Projekt: P1371_Sensors

# P1371_Sensors 1

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:id="@+id/btnSensList"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClickSensList"
        android:text="@string/list" >
    </Button>
    <Button
        android:id="@+id/btnSensLight"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/btnSensList"
        android:onClick="onClickSensLight"
        android:text="@string/light" >
    </Button>
    <ScrollView
        android:id="@+id/scroll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/btnSensList" >
        <TextView
            android:id="@+id/tvText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >
        </TextView>
    </ScrollView>
</RelativeLayout>
```

# P1371_Sensors 2

```
14  public class MainActivity extends Activity {
15
16      TextView tvText;
17      SensorManager sensorManager;
18      List<Sensor> sensors;
19      Sensor sensorLight;
20
21      @Override
22      protected void onCreate(Bundle savedInstanceState) {
23          super.onCreate(savedInstanceState);
24          setContentView(R.layout.main);
25          tvText = (TextView) findViewById(R.id.tvText);
26          sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
27          sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
28          sensorLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
29      }
30
```

# P1371_Sensors 3

- **SensorManager**
  - lets you access the device's sensors.
  - Get an instance of this class by calling
    Context.getSystemService();
    with the argument SENSOR_SERVICE.
- **public Sensor getDefaultSensor (int type);**
  - Use this method to get the default sensor for a given type.
    - Note that the returned sensor could be a composite sensor, and its data could be averaged or filtered.
    - If you need to access the raw sensors use getSensorList.
  - Parameters
    - type of sensors requested
  - Returns
    - the default sensors matching the asked type

# P1371_Sensors 4

```
31    public void onClickSensList(View v) {
32
33        sensorManager.unregisterListener(listenerLight, sensorLight);
34        StringBuilder sb = new StringBuilder();
35
36        for (Sensor sensor : sensors) {
37            sb.append("name = ").append(sensor.getName()).append(", type = ")
38                    .append(sensor.getType()).append("\nvendor = ")
39                    .append(sensor.getVendor()).append(" ,version = ")
40                    .append(sensor.getVersion()).append("\nmax = ")
41                    .append(sensor.getMaximumRange()).append(", resolution = ")
42                    .append(sensor.getResolution())
43                    .append("\n-------------------------------------\n");
44        }
45        tvText.setText(sb);
46    }
47
48    public void onClickSensLight(View v) {
49        sensorManager.registerListener(listenerLight, sensorLight,
50                SensorManager.SENSOR_DELAY_NORMAL);
51    }
```

# P1371_Sensors 5

- **public boolean registerListener (SensorEventListener listener, Sensor sensor, int rateUs)**
  - Registers a SensorEventListener for the given sensor.
    - Note: Don't use this method with a one shot trigger sensor such as TYPE_SIGNIFICANT_MOTION.
    - Use requestTriggerSensor(TriggerEventListener, Sensor) instead.
- Parameters
  - **listener** A SensorEventListener object.
  - **sensor** The Sensor to register to.
  - **rateUs** The rate sensor events are delivered at.
    - This is only a hint to the system. Events may be received faster or slower than the specified rate. Usually events are received faster.
    - The value must be one of SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, or SENSOR_DELAY_FASTEST or, the desired delay between events in microseconds
- Returns
  - **true** if the sensor is supported and successfully enabled.

# P1371_Sensors 6

Energie sparen!

```
53      @Override
54      protected void onPause() {
55          super.onPause();
56          sensorManager.unregisterListener(listenerLight, sensorLight);
57      }
58
59      SensorEventListener listenerLight = new SensorEventListener() {
60
61          @Override
62          public void onAccuracyChanged(Sensor sensor, int accuracy) {
63          }
64
65          @Override
66          public void onSensorChanged(SensorEvent event) {
67              tvText.setText(String.valueOf(event.values[0]));
68          }
69      };
70
71  }
```

# P1371_Sensors 7

- public abstract void onAccuracyChanged (Sensor sensor, int accuracy)
  - Called when the accuracy of a sensor has changed.
- Parameters
  - accuracy  The new accuracy of this sensor
    - SENSOR_STATUS_ACCURACY_HIGH
    - SENSOR_STATUS_ACCURACY_MEDIUM
    - SENSOR_STATUS_ACCURACY_LOW
    - SENSOR_STATUS_UNRELIABLE

# P1371_Sensors 8

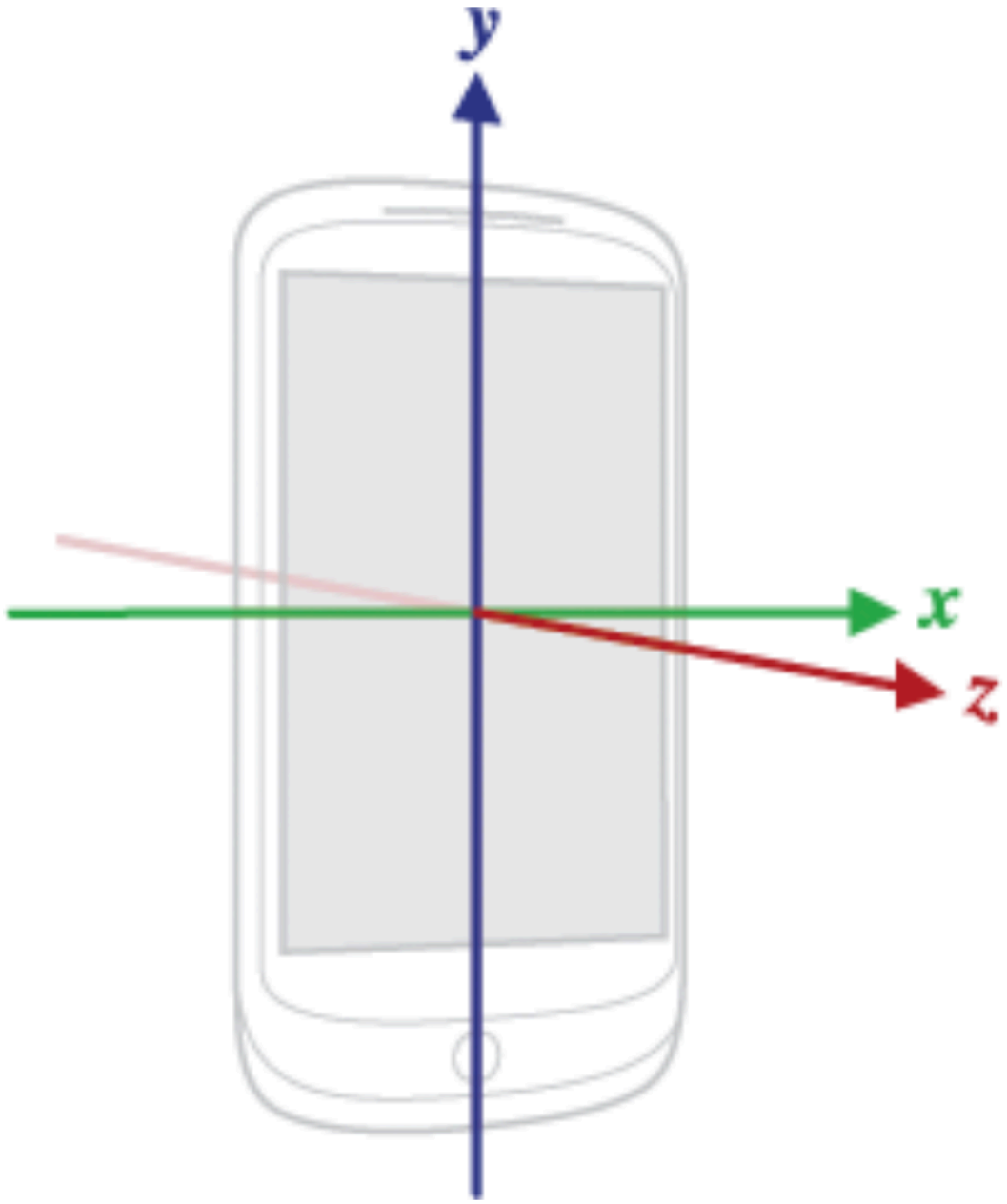
- public abstract void onSensorChanged (SensorEvent event)
  - Called when sensor values have changed.
  - NOTE: The application doesn't own the event object passed as a parameter and therefore cannot hold on to it. The object may be part of an internal pool and may be reused by the framework.
- Parameters
  - event  the SensorEvent.

# Acceleration

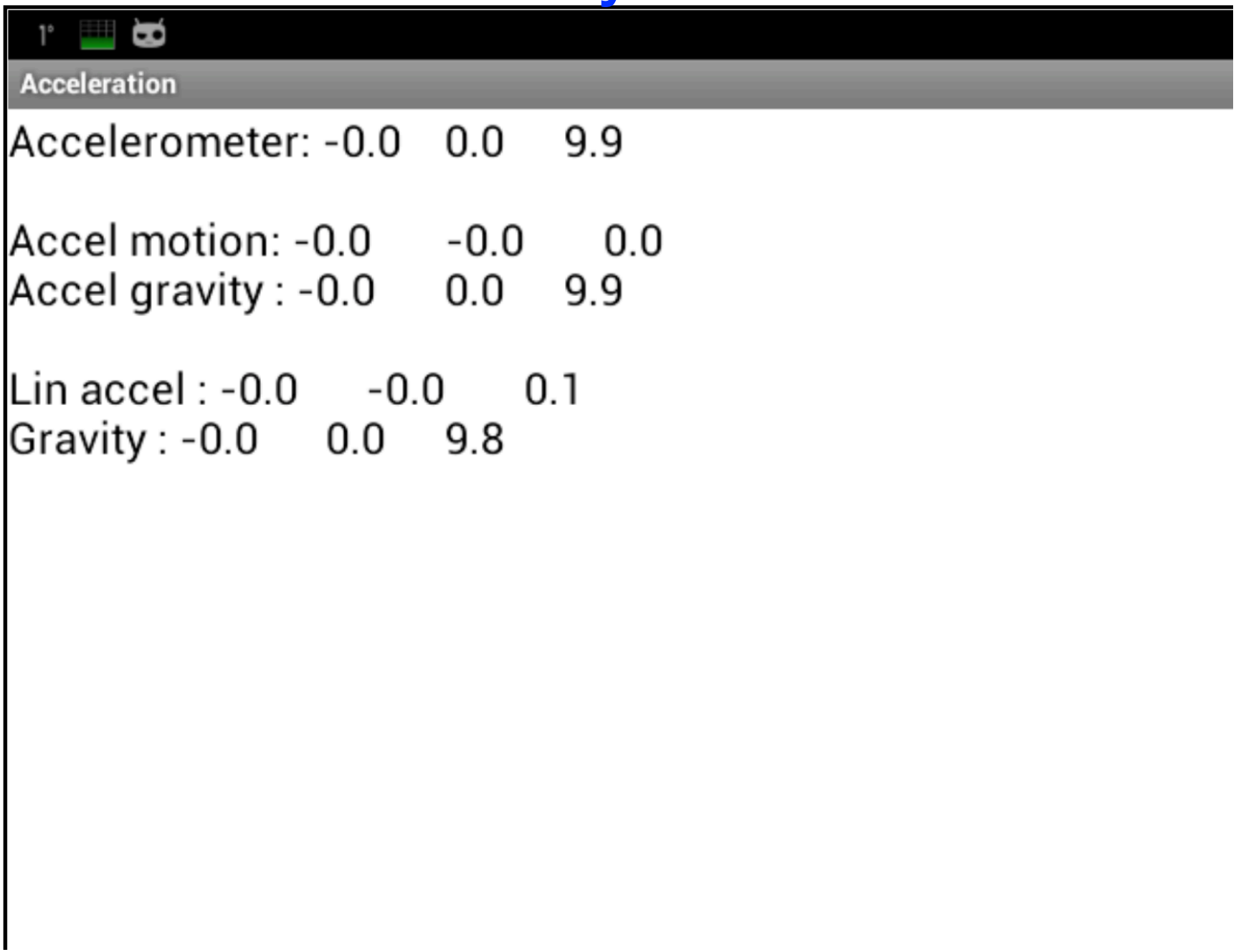
http://startandroid.ru/ru/uroki/vse-uroki-spiskom/287-urok-137-sensory-uskorenie-orientatsija.html

Projekt: P1372_Acceleration

# Koordinatensystem des Gerätes



# Layout:


Acceleration
Accelerometer: -0.0 0.0 9.9
Accel motion: -0.0 -0.0 0.0
Accel gravity : -0.0 0.0 9.9
Lin accel : -0.0 -0.0 0.1
Gravity : -0.0 0.0 9.8

```
14  public class MainActivity extends Activity {
15      TextView tvText;
16      SensorManager sensorManager;
17      Sensor sensorAccel;
18      Sensor sensorLinAccel;
19      Sensor sensorGravity;
20      StringBuilder sb = new StringBuilder();
21      Timer timer;
23      protected void onCreate(Bundle savedInstanceState) {..
35      protected void onResume() {..
58      protected void onPause() {..
63      String format(float values[]) {..
67      void showInfo() {..
76      float[] valuesAccel = new float[3];
77      float[] valuesAccelMotion = new float[3];
78      float[] valuesAccelGravity = new float[3];
79      float[] valuesLinAccel = new float[3];
80      float[] valuesGravity = new float[3];
81      SensorEventListener listener = new SensorEventListener() {
83          public void onAccuracyChanged(Sensor sensor, int accuracy) {..
86          public void onSensorChanged(SensorEvent event) {..
108     };
109 }
```

```
14  public class MainActivity extends Activity {
15      TextView tvText;
16      SensorManager sensorManager;
17      Sensor sensorAccel;
18      Sensor sensorLinAccel;
19      Sensor sensorGravity;
20      StringBuilder sb = new StringBuilder();
21      Timer timer;
22      @Override
23      protected void onCreate(Bundle savedInstanceState) {
24          super.onCreate(savedInstanceState);
25          setContentView(R.layout.main);
26          tvText = (TextView) findViewById(R.id.tvText);
27          sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
28          sensorAccel = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
29          sensorLinAccel = sensorManager
30                  .getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
31          sensorGravity = sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
32
33      }
```

```
34      @Override
35      protected void onResume() {
36          super.onResume();
37          sensorManager.registerListener(listener, sensorAccel,
38                  SensorManager.SENSOR_DELAY_NORMAL);
39          sensorManager.registerListener(listener, sensorLinAccel,
40                  SensorManager.SENSOR_DELAY_NORMAL);
41          sensorManager.registerListener(listener, sensorGravity,
42                  SensorManager.SENSOR_DELAY_NORMAL);
43          timer = new Timer();
44          TimerTask task = new TimerTask() {
45              @Override
46              public void run() {
47                  runOnUiThread(new Runnable() {
48                      @Override
49                      public void run() {
50                          showInfo();
51                      }
52                  });
53              }
54          };
55          timer.schedule(task, 0, 400);
56      }
57      @Override
58      protected void onPause() {
59          super.onPause();
60          sensorManager.unregisterListener(listener);
61          timer.cancel();
62      }
```

```
63      String format(float values[]) {
64          return String.format("%1$.1f\t\t%2$.1f\t\t%3$.1f", values[0],
65                  values[1], values[2]);
66      }
67      void showInfo() {
68          sb.setLength(0);
69          sb.append("Accelerometer: " + format(valuesAccel))
70                  .append("\n\nAccel motion: " + format(valuesAccelMotion))
71                  .append("\nAccel gravity : " + format(valuesAccelGravity))
72                  .append("\n\nLin accel : " + format(valuesLinAccel))
73                  .append("\nGravity : " + format(valuesGravity));
74          tvText.setText(sb);
75      }
76      float[] valuesAccel = new float[3];
77      float[] valuesAccelMotion = new float[3];
78      float[] valuesAccelGravity = new float[3];
79      float[] valuesLinAccel = new float[3];
80      float[] valuesGravity = new float[3];
```

```
81    SensorEventListener listener = new SensorEventListener() {
82        @Override
83        public void onAccuracyChanged(Sensor sensor, int accuracy) {
84        }
85        @Override
86        public void onSensorChanged(SensorEvent event) {
87            switch (event.sensor.getType()) {
88            case Sensor.TYPE_ACCELEROMETER:
89                for (int i = 0; i < 3; i++) {
90                    valuesAccel[i] = event.values[i];
91                    valuesAccelGravity[i] = (float) (0.1 * event.values[i] + 0.9 * valuesAccelGravity[i]);
92                    valuesAccelMotion[i] = event.values[i]
93                            - valuesAccelGravity[i];
94                }
95                break;
96            case Sensor.TYPE_LINEAR_ACCELERATION:
97                for (int i = 0; i < 3; i++) {
98                    valuesLinAccel[i] = event.values[i];
99                }
100               break;
101           case Sensor.TYPE_GRAVITY:
102               for (int i = 0; i < 3; i++) {
103                   valuesGravity[i] = event.values[i];
104               }
105               break;
106           }
107       }
108   };
```

Sensor data

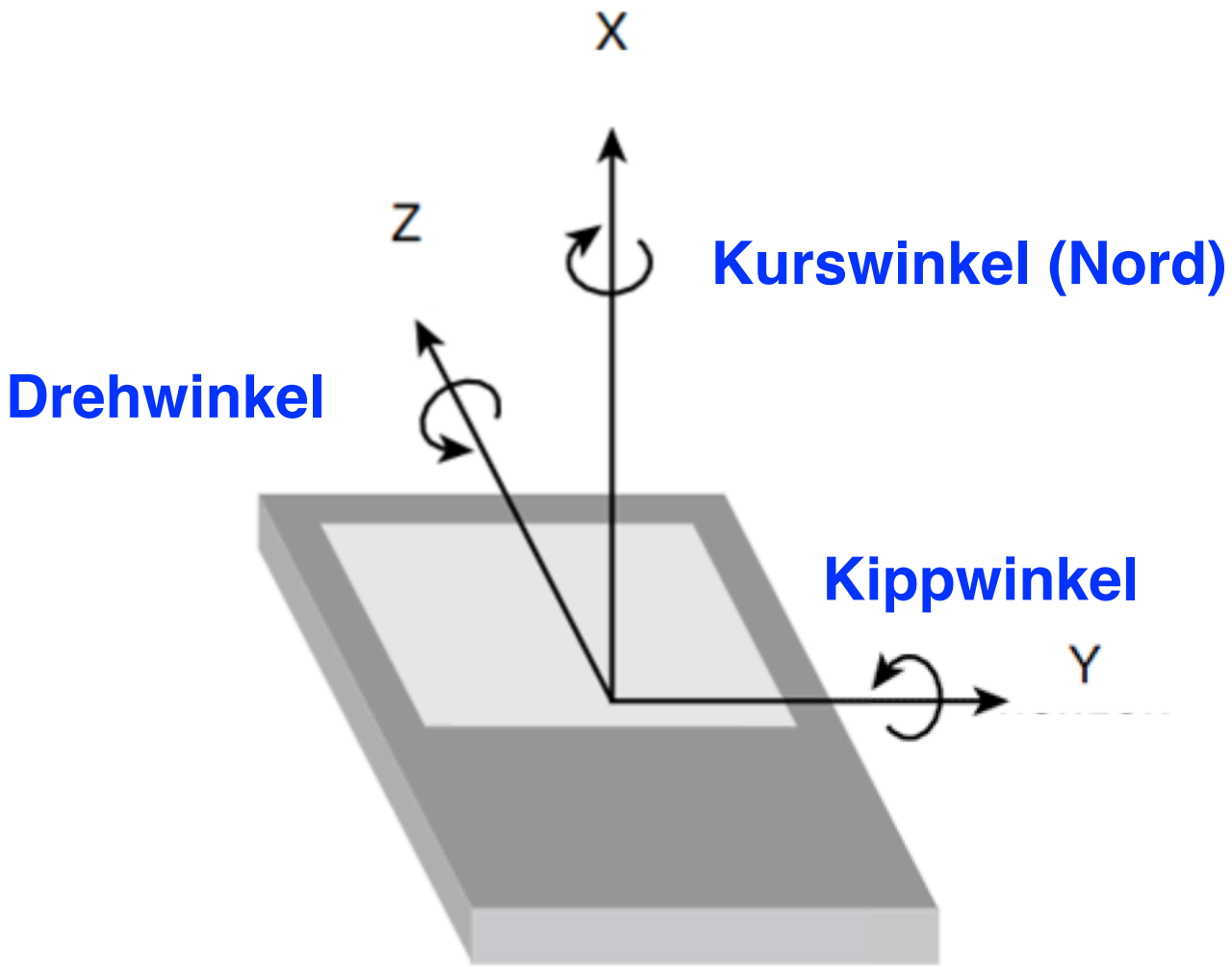
Calculated value!

Calculated value!

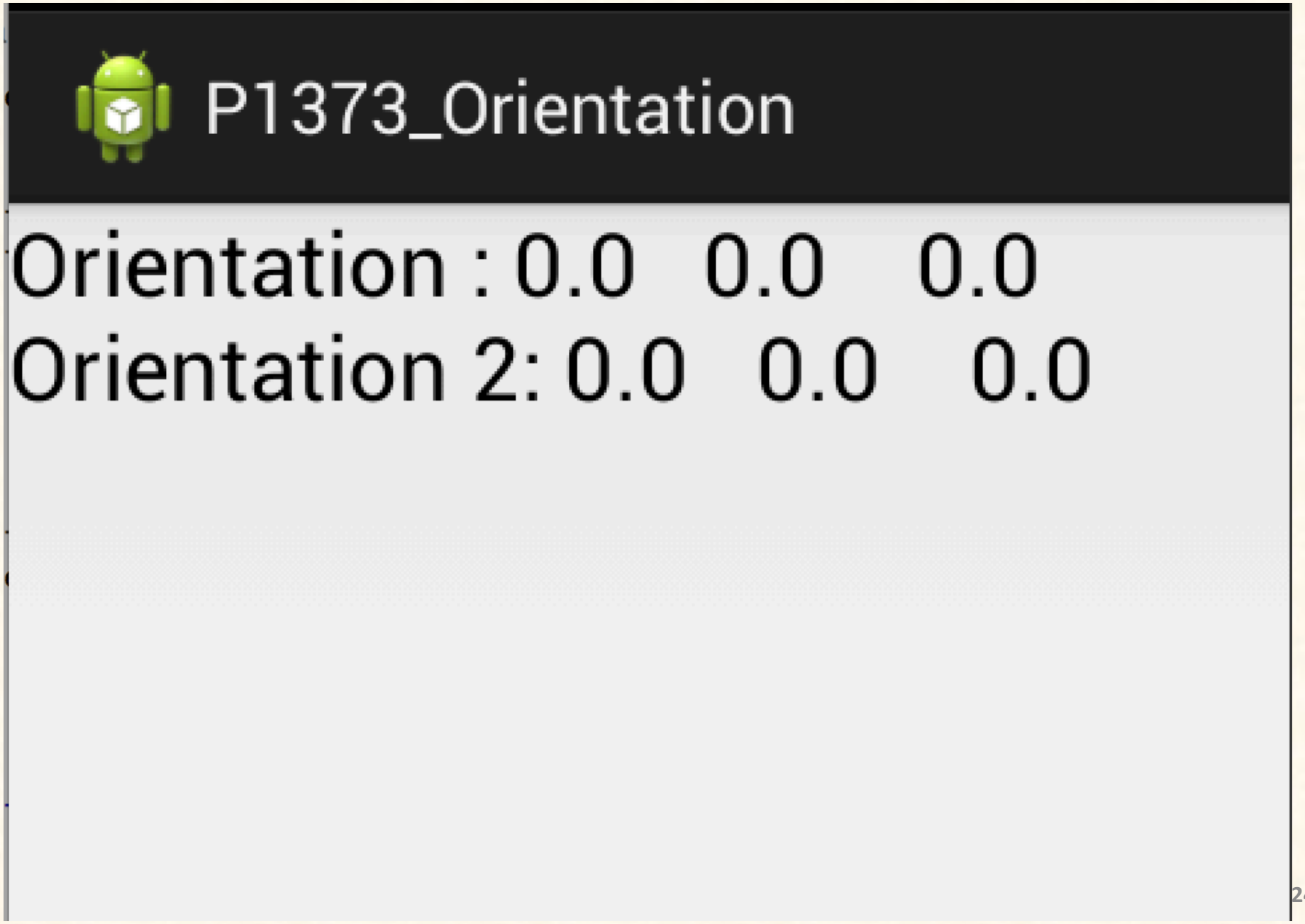
# Orientation

http://startandroid.ru/ru/uroki/vse-uroki-spiskom/287-urok-137-sensory-uskorenie-orientatsija.html

Projekt: P1373_Orientation


X
Z
Kurswinkel (Nord)
Drehwinkel
Kippwinkel
Y

# P1373_Orientation 1



# P1373_Orientation 2

```
 3⊕ import java.util.Timer;..
17
18  public class MainActivity extends Activity {
19      TextView tvText;
20      SensorManager sensorManager;
21      Sensor sensorAccel;
22      Sensor sensorMagnet;
23      StringBuilder sb = new StringBuilder();
24      Timer timer;
25      int rotation;
27⊕     protected void onCreate(Bundle savedInstanceState) {..
37⊕     protected void onResume() {..
63⊕     protected void onPause() {..
68⊕     String format(float values[]) {..
72⊕     void showInfo() {..
78      float[] r = new float[9];
79⊕     void getDeviceOrientation() {..
87      float[] inR = new float[9];
88      float[] outR = new float[9];
89⊕     void getActualDeviceOrientation() {..
117     float[] valuesAccel = new float[3];
118     float[] valuesMagnet = new float[3];
119     float[] valuesResult = new float[3];
120     float[] valuesResult2 = new float[3];
121⊖     SensorEventListener listener = new SensorEventListener() {..
141 }
```

# P1373_Orientation 3

```
18 public class MainActivity extends Activity {
19     TextView tvText;
20     SensorManager sensorManager;
21     Sensor sensorAccel;
22     Sensor sensorMagnet;
23     StringBuilder sb = new StringBuilder();
24     Timer timer;
25     int rotation;
26     @Override
27     protected void onCreate(Bundle savedInstanceState) {
28         super.onCreate(savedInstanceState);
29         setContentView(R.layout.main);
30         tvText = (TextView) findViewById(R.id.tvText);
31         sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
32         sensorAccel = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
33         sensorMagnet = sensorManager
34                 .getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
35     }
```

# P1373_Orientation 4

```
36    @Override
37    protected void onResume() {
38        super.onResume();
39        sensorManager.registerListener(listener, sensorAccel,
40                SensorManager.SENSOR_DELAY_NORMAL);
41        sensorManager.registerListener(listener, sensorMagnet,
42                SensorManager.SENSOR_DELAY_NORMAL);
43        timer = new Timer();
44        TimerTask task = new TimerTask() {
45            @Override
46            public void run() {
47                runOnUiThread(new Runnable() {
48                    @Override
49                    public void run() {
50                        getDeviceOrientation();
51                        getActualDeviceOrientation();
52                        showInfo();
53                    }
54                });
55            }
56        };
57        timer.schedule(task, 0, 400);
58        WindowManager windowManager = ((WindowManager) getSystemService(Context.WINDOW_SERVICE));
59        Display display = windowManager.getDefaultDisplay();
60        rotation = display.getRotation();
61    }
```

# P1373_Orientation 5

```
62      @Override
63      protected void onPause() {
64          super.onPause();
65          sensorManager.unregisterListener(listener);
66          timer.cancel();
67      }
68      String format(float values[]) {
69          return String.format("%1$.1f\t\t%2$.1f\t\t%3$.1f", values[0],
70                  values[1], values[2]);
71      }
72      void showInfo() {
73          sb.setLength(0);
74          sb.append("Orientation : " + format(valuesResult)).append(
75                  "\nOrientation 2: " + format(valuesResult2));
76          tvText.setText(sb);
77      }
78      float[] r = new float[9];
79      void getDeviceOrientation() {..
87      float[] inR = new float[9];
88      float[] outR = new float[9];
89      void getActualDeviceOrientation() {..
117     float[] valuesAccel = new float[3];
118     float[] valuesMagnet = new float[3];
119     float[] valuesResult = new float[3];
120     float[] valuesResult2 = new float[3];
```

# P1373_Orientation 6

```
79      void getDeviceOrientation() {
80          SensorManager.getRotationMatrix(r, null, valuesAccel, valuesMagnet);
81          SensorManager.getOrientation(r, valuesResult);
82          valuesResult[0] = (float) Math.toDegrees(valuesResult[0]);
83          valuesResult[1] = (float) Math.toDegrees(valuesResult[1]);
84          valuesResult[2] = (float) Math.toDegrees(valuesResult[2]);
85          return;
86      }
87      float[] inR = new float[9];
88      float[] outR = new float[9];
```

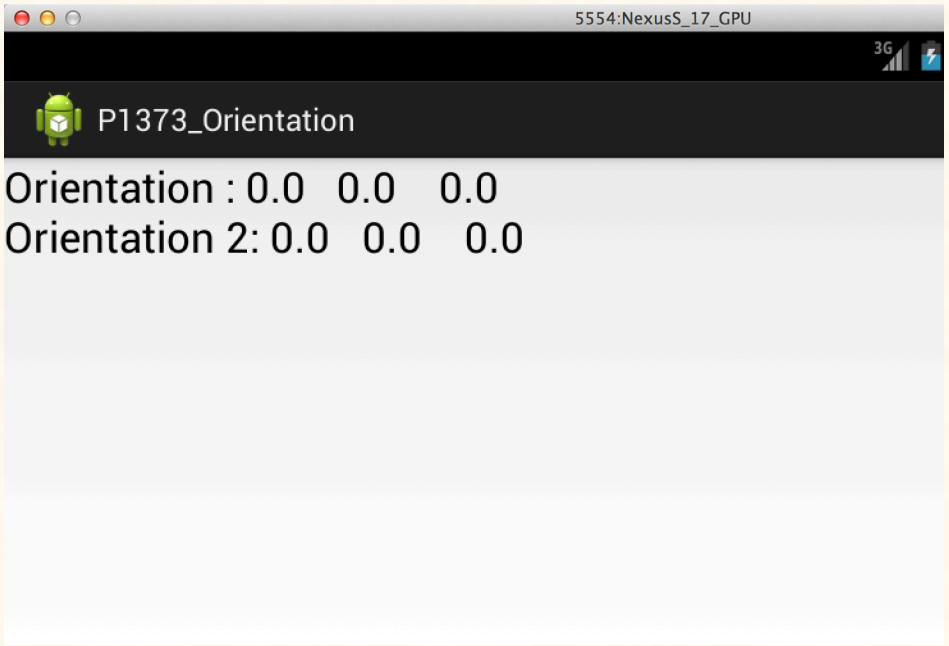
# P1373_Orientation 7

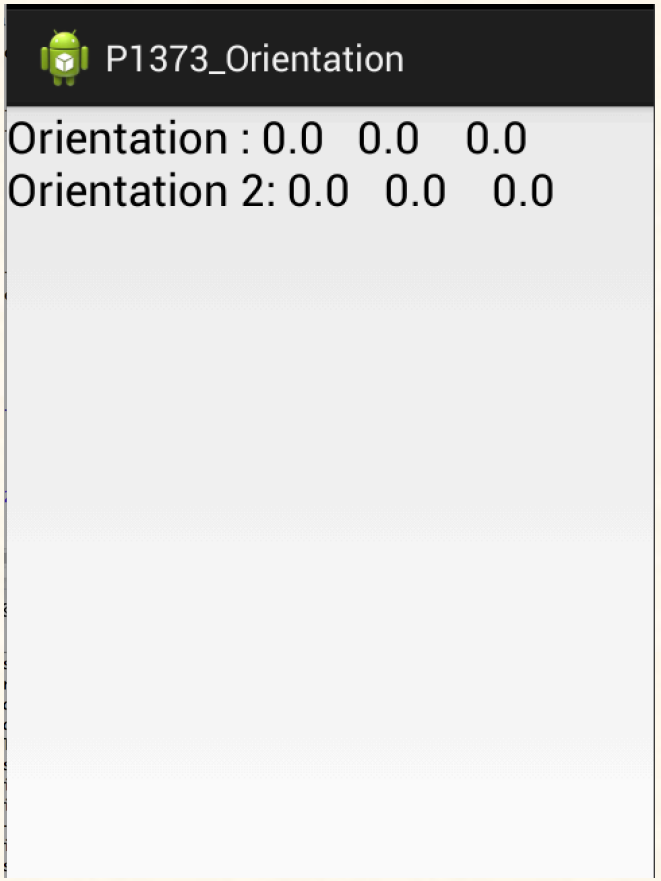
```
89      void getActualDeviceOrientation() {
90          SensorManager.getRotationMatrix(inR, null, valuesAccel, valuesMagnet);
91          int x_axis = SensorManager.AXIS_X;
92          int y_axis = SensorManager.AXIS_Y;
93          switch (rotation) {
94          case (Surface.ROTATION_0):
95              break;
96          case (Surface.ROTATION_90):
97              x_axis = SensorManager.AXIS_Y;
98              y_axis = SensorManager.AXIS_MINUS_X;
99              break;
100         case (Surface.ROTATION_180):
101             y_axis = SensorManager.AXIS_MINUS_Y;
102             break;
103         case (Surface.ROTATION_270):
104             x_axis = SensorManager.AXIS_MINUS_Y;
105             y_axis = SensorManager.AXIS_X;
106             break;
107         default:
108             break;
109         }
110         SensorManager.remapCoordinateSystem(inR, x_axis, y_axis, outR);
111         SensorManager.getOrientation(outR, valuesResult2);
112         valuesResult2[0] = (float) Math.toDegrees(valuesResult2[0]);
113         valuesResult2[1] = (float) Math.toDegrees(valuesResult2[1]);
114         valuesResult2[2] = (float) Math.toDegrees(valuesResult2[2]);
115         return;
116     }
```

# P1373_Orientation 8

```
78      float[] r = new float[9];
79      void getDeviceOrientation() {..
87      float[] inR = new float[9];
88      float[] outR = new float[9];
89      void getActualDeviceOrientation() {..
117     float[] valuesAccel = new float[3];
118     float[] valuesMagnet = new float[3];
119     float[] valuesResult = new float[3];
120     float[] valuesResult2 = new float[3];
121     SensorEventListener listener = new SensorEventListener() {
122         @Override
123         public void onAccuracyChanged(Sensor sensor, int accuracy) {
124         }
125         @Override
126         public void onSensorChanged(SensorEvent event) {
127             switch (event.sensor.getType()) {
128             case Sensor.TYPE_ACCELEROMETER:
129                 for (int i = 0; i < 3; i++) {
130                     valuesAccel[i] = event.values[i];
131                 }
132                 break;
133             case Sensor.TYPE_MAGNETIC_FIELD:
134                 for (int i = 0; i < 3; i++) {
135                     valuesMagnet[i] = event.values[i];
136                 }
137                 break;
138             }
139         }
140     };
```

# P1373_Orientation 9



# Notfalls in AndroidManifest.xml:

**(wenn ein Sensor für Ihre App lebenswichtig ist)**

*<uses-feature*
*android:name="android.hardware.sensor.accelerometer" android:required="true" />*

# Fragen?