Introduction

Modern mobile phones are much more than simple communications devices with a connection to the Internet. With microphones, cameras, accelerometers, compasses, temperature gauges, and brightness detectors, smartphones have become extra-sensory devices, able to augment your own perceptions.

Supported Android Sensors

The following is a list of the sensor-types currently available; note that the hardware on the host device determines which of these sensors are actually available to your application.

- **Sensor.TYPE_ACCELEROMETER** A three-axis accelerometer sensor that returns the current acceleration along three axes in m/s².

- **Sensor.TYPE_GYROSCOPE** A gyroscopic sensor that returns the current device orientation on three axes in degrees.

- **Sensor.TYPE_LIGHT** An ambient light sensor that returns a single value describing the ambient illumination in lux. A light sensor is commonly used to dynamically control the screen brightness.

- **Sensor.TYPE_MAGNETIC_FIELD** A magnetic field sensor that finds the current magnetic field in microteslas along three axes.

- **Sensor.TYPE_ORIENTATION** An orientation sensor that returns the device orientation on three axes in degrees.

- **Sensor.TYPE_PRESSURE** A pressure sensor that returns a single value, the current pressure exerted on the device in kilopascals.
- **Sensor.TYPE_PROXIMITY** A proximity sensor that indicates the distance between the device and the target object in meters.

- **Sensor.TYPE_TEMPERATURE** A thermometer that returns temperature in degrees Celsius.

### Finding Available Sensors

Not all devices have all the sensors listed above so you need to check if the device has the sensor you want or not.

This code can list all sensors on your device:

```java
TextView tv = (TextView) findViewById(R.id.textView1);
SensorManager manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> allSensors = manager.getSensorList(Sensor.TYPE_ALL);

for (Iterator<Sensor> i = allSensors.iterator(); i.hasNext();)
{
    Sensor sensor = i.next();
    tv.append(sensor.getName() + "\n");
}
```

### Using Specific Sensor

1. **Get the Sensor manager Service ..**

```java
SensorManager manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

2. **Use method getDefaultSensor to get the default sensor :D (there can be multiple implementation to a specific sensors)**

```java
Sensor acc = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```
3- Now you have to implement a sensor handler to be triggered

    SensorEventListener myHandler = new SensorEventListener() {

        public void onSensorChanged(SensorEvent event) {
            // here we will read sensor values
        }

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

4- Register the Handler with you sensor and specify the sensor reading frequently

    manager.registerListener(myHandler, acc, SensorManager.SENSOR_DELAY_NORMAL);

    To receive Sensor events, register your Sensor Event Listener with the Sensor Manager. Specify the Sensor object to observe, and the rate at which you want to receive updates.

    ➤ SensorManager.SENSOR_DELAY_FASTEST Specifies the fastest possible Sensor update rate
    ➤ SensorManager.SENSOR_DELAY_GAME Selects an update rate suitable for use in controlling games
    ➤ SensorManager.SENSOR_DELAY_NORMAL Specifies the default update rate
    ➤ SensorManager.SENSOR_DELAY_UI Specifies a rate suitable for updating UI features

5- It's a good idea to unregister the handler when your activity is destroyed

    @Override
    protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();
        manager.unregisterListener(myHandler);
    }
So, what values are returned in the `event` and how we can use it, look at the following table:

**Timestamp**

```java
event.timestamp
```

- `timestamp` The time (in nanoseconds) at which the Sensor event occurred.

**Accuracy**

```java
event.accuracy
```

- `accuracy` The accuracy of the Sensor when the event occurred (low, medium, high, or unreliable, as described in the next list).

- `SensorManager.SENSOR_STATUS_ACCURACY_LOW` Indicates that the Sensor is reporting with low accuracy.

- `SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM` Indicates that the Sensor data is of average accuracy.

- `SensorManager.SENSOR_STATUS_ACCURACY_HIGH` Indicates that the Sensor is reporting with the highest possible accuracy.

- `SensorManager.SENSOR_STATUS_UNRELIABLE` Indicates that the Sensor data is unreliable.

**Values**

```java
event.values[0]
event.values[1]
event.values[2]
```

- `values` A float array that contains the new value(s) detected.
<table>
<thead>
<tr>
<th>SENSOR-TYPE</th>
<th>VALUE COUNT</th>
<th>VALUE COMPOSITION</th>
<th>COMMENTARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_ACCELEROMETER</td>
<td>3</td>
<td>value[0]: Lateral</td>
<td>Acceleration along three axes in m/s². The Sensor Manager includes a set of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[1]: Longitudinal</td>
<td>gravity constants of the form SensorManager.GRAVITY_*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[2]: Vertical</td>
<td></td>
</tr>
<tr>
<td>TYPE_GYROSCOPE</td>
<td>3</td>
<td>value[0]: Azimuth</td>
<td>Device orientation in degrees along three axes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[1]: Pitch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[2]: Roll</td>
<td></td>
</tr>
<tr>
<td>TYPE_LIGHT</td>
<td>1</td>
<td>value[0]: Illumination</td>
<td>Measured in lux. The Sensor Manager includes a set of constants representing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>different standard illuminations of the form SensorManager.LIGHT_*</td>
</tr>
<tr>
<td>TYPE_MAGNETIC_FIELD</td>
<td>3</td>
<td>value[0]: Lateral</td>
<td>Ambient magnetic field measured in microteslas (μT).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[1]: Longitudinal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[2]: Vertical</td>
<td></td>
</tr>
<tr>
<td>TYPE_ORIENTATION</td>
<td>3</td>
<td>value[0]: Azimuth</td>
<td>Device orientation in degrees along three axes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[1]: Roll</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>value[2]: Pitch</td>
<td></td>
</tr>
<tr>
<td>TYPE_PRESSURE</td>
<td>1</td>
<td>value[0]: Pressure</td>
<td>Measured in kilopascals (KP).</td>
</tr>
<tr>
<td>TYPE_PROXIMITY</td>
<td>1</td>
<td>value[0]: Distance</td>
<td>Measured in meters.</td>
</tr>
<tr>
<td>TYPE_TEMPERATURE</td>
<td>1</td>
<td>value[0]: Temperature</td>
<td>Measured in degrees Celsius.</td>
</tr>
</tbody>
</table>
Sensor.TYPE_PROXIMITY

The proximity sensor is usually visible on the face of the device only in bright sunlight. It typically looks like a dark hole underneath the blackened part of the glass, usually at the top of the front face of a smartphone. It consists of a weak infrared LED (light-emitting diode) next to a photodetector. When something (such as the ear of a person making a phone call) comes close enough to the sensor, the photodetector detects the reflected infrared light.

The main application of a proximity sensor is detecting the ear of the user in order to shut down or lock the screen during calls.

```
SensorManager manager = (SensorManager) getSystemService(Service.SENSOR_SERVICE);
final TextView tv = (TextView) findViewById(R.id.textView1);
Sensor mylightSen = manager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
SensorEventListener l = new SensorEventListener() {
    public void onSensorChanged(SensorEvent event) {
        tv.setText("distance in metters :") + event.values[0];
    }
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
    }
};
manager.registerListener(l, mylightSen, SensorManager.SENSOR_DELAY_NORMAL);
```
public class OrientationActivity extends Activity {
    static long lastUpdate=0;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SensorManager manager = (SensorManager)
        getSystemService(Service.SENSOR_SERVICE);

        final TextView tv = (TextView) findViewById(R.id.textView1);

        Sensor mylightSen = manager.getDefaultSensor(Sensor.TYPE_ORIENTATION);

        SensorEventListener l = new SensorEventListener() {
            public void onSensorChanged(SensorEvent event) {
                long actualTime = System.currentTimeMillis();
                if (actualTime - lastUpdate > 500) {
                    lastUpdate = actualTime;
                    // update values for app
                    float headingAngle = event.values[0];
                    float pitchAngle = event.values[1];
                    float rollAngle = event.values[2];
                    tv.setText("Heading: "+headingAngle+
                    "Pitch: "+pitchAngle+
                    "Roll: "+rollAngle);
                }
            }
            public void onAccuracyChanged(Sensor sensor, int accuracy) {
                // handle accuracy changes
            }
        };

        manager.registerListener(l, mylightSen,SensorManager.SENSOR_DELAY_NORMAL);
    }
}
Controlling Device Vibration

```java
String vibratorService = Context.VIBRATOR_SERVICE;
Vibrator vibrator = (Vibrator) getSystemService(vibratorService);

long[] pattern = {1000, 2000, 4000, 8000, 16000};
vibrator.vibrate(pattern, 0); // Execute vibration pattern.
vibrator.vibrate(1000); // Vibrate for 1 second.
```

Need permission

```xml
<uses-permission android:name="android.permission.VIBRATE"/>
```