Android Sensors

Table of Contents

[Types of Sensors 2](#_Toc100977371)

[Android Sensor Framework 5](#_Toc100977372)

[SensorManager 6](#_Toc100977373)

[Sensor 7](#_Toc100977374)

[Monitoring Events 7](#_Toc100977375)

[Sensor Coordinate System 10](#_Toc100977376)

[Best Practices 10](#_Toc100977377)

Most Android-powered devices have built-in **sensors** that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device.

For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing, and a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

## Types of Sensors

There are mainly three categories of sensors:

* **Motion Sensors** - These measure acceleration forces and rotational forces along the three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.
* **Environmental Sensors** - These measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.
* **Position Sensors** - These measure the physical position of a device. This category includes orientation sensors and magnetometers.

The sensors that fall into each of these categories can either by hardware-based or software-based.

**Hardware-based sensors** are physical components built into the device. They derive their data by directly measuring specific environmental properties.

**Software-based sensors**, also called virtual sensors or synthetic sensors, mimic hardware-based ones. They derive their data from one or more the hardware-based sensors.

A complete list of the different types of sensors available to the Android platform is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sensor** | **Type** | **Description** | **Common Uses** |
| TYPE\_ACCELEROMETER | Hardware | Measures the acceleration force in m/s2 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/s2 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/s2 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. |
| 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. |

Obviously, a single device is unlikely to have every single sensor. It might also be the case that the same device has multiple sensors of the same type. For example, there could be two gravity sensors, each working at a different range.

More details about the various types of sensors can be found [here](https://developer.android.com/guide/topics/sensors/sensors_overview).

## Android Sensor Framework

The **Android Sensor Framework** can be used to access the sensors and acquire their data. The framework consists of a variety of classes and interfaces that allow you to:

* Determine which sensors are available on the device.
* Determine a specific sensor’s properties, such as the maximum range, manufacturer, power requirements, resolution, etc.
* Acquire raw sensor data and define the minimum rate at which it must be acquired.
* Register and unregister sensor event listeners that monitor sensor changes.

We will be looking into four of the classes from the Android Sensor Framework, SensorManager, Sensor, SensorEvent and *SensorEventListener*.

### SensorManager

The SensorManager class can be used to create an instance of the sensor service. Using this class, we can access and list sensors, register and unregister sensor event listeners and acquire orientation information. There are also properties which report the sensor’s accuracy, set data acquisition rates and calibrate sensors.

We can identify which sensors are available like this:

SensorManager sensorManager;  
sensorManager = (SensorManager) getSystemService(Context.*SENSOR\_SERVICE*);  
*List*<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.*TYPE\_ALL*);

JAVA

If instead we want a specific type of sensor, instead of *TYPE\_ALL*, we can use *TYPE\_GYROSCOPE*, *TYPE\_LINEAR\_ACCELERATION*, *TYPE\_GRAVITY*, etc. Another way to do this is to use the getDefaultSensor method.

Sensor magnetometer = sensorManager.getDefaultSensor(Sensor.*TYPE\_MAGNETIC\_FIELD*);

JAVA

If a device has multiple sensors of a given type, one of them will be assigned the **default sensor**, which is the one we will get here. If a specific sensor does not exist in the device, the method will return null.

### Sensor

The Sensor class is used to create an instance of a specific sensor. Various methods then allow us to determine the sensor’s capabilities.

* getMinDelay() – This is the minimum time required by the sensor to read data.
* getPower() – The sensor’s power requirements in microamperes.
* getMaximumRange() – The sensor’s maximum range of measurement.
* getResolution() – The sensor’s resolution.

For the getMinDelay() method, if the sensor returns a non-zero value, it is a **streaming sensor**, which sense data at regular intervals. If the returned value is zero, then the sensor reports data only when there is a change in the parameter it is sensing.

### Monitoring Events

To monitor sensor events, we use the SensorEvent class and the *SensorEventListener* interface. A SensorEvent object includes the following information:

* The raw sensor data.
* The type of sensor that generated the event.
* The accuracy of the data.
* Timestamp of the event.

The *SensorEventListener* interface allows us to create two callback methods that receive SensorEvent objects, one when the sensor value changes (onSensorChanged()) and the other when the sensor accuracy changes (onAccuracyChanged()). Accuracy is represented by one of four status constants, *SENSOR\_STATUS\_ACCURACY\_LOW*, *SENSOR\_STATUS\_ACCURACY\_MEDIUM*, *SENSOR\_STATUS\_ACCURACY\_HIGH*, or *SENSOR\_STATUS\_UNRELIABLE*.

A class which uses sensor event listeners might look like this:

public class SensorActivity extends Activity implements *SensorEventListener* {  
 private SensorManager manager;  
 private Sensor light;  
  
 @Override  
 public final void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 manager = (SensorManager) getSystemService(Context.*SENSOR\_SERVICE*);  
 light = manager.getDefaultSensor(Sensor.*TYPE\_LIGHT*);  
 }  
  
 @Override  
 public void onAccuracyChanged(Sensor sensor, int accuracy) {  
 // the sensor accuracy has changed  
 }  
  
 @Override  
 public void onSensorChanged(SensorEvent event) {  
 float lux = event.values[0];  
 // the light sensor returns 1 value, but other sensors can return more  
 }

@Override  
 protected void onResume() {  
 super.onResume();

int delay = SensorManager.*SENSOR\_DELAY\_NORMAL*;

manager.registerListener(this, light, delay);  
 // registering event listener  
 }

@Override  
 protected void onPause() {  
 super.onPause();  
 manager.unregisterListener(this);  
 // unregistering event listener  
 }  
}

JAVA

The delay parameter controls how often sensor events are sent to the application. The default value, used here, is 200,000 microseconds. Other possible values include *SENSOR\_DELAY\_UI*, at 60,000 microseconds, *SENSOR\_DELAY\_GAME*, at 20,000 microseconds and *SENSOR\_DELAY\_FASTEST*, at 0 microseconds. It is also possible to specify a value in microseconds.

The delay value is set based on how long background actions in our application take. The background actions that process the results of one sensor event must complete before we get the results of the next sensor event in order to keep things running smoothly. How long this gap is depends on the type of application.

## Sensor Coordinate System

The sensor framework uses a **3-axis coordinate system**. For most sensors, the system is defined related to the device’s screen when it is held facing the screen, i.e. the -axis goes horizontally, the -axis goes vertically and the -axis comes out of the screen.

The coordinate system is used by:

* The acceleration sensor
* The gravity sensor
* The gyroscope
* The linear acceleration sensor
* The geomagnetic field sensor

## Best Practices

* Only gather sensor data in the foreground
* Unregister sensor event listeners
* Test with the Android emulator
* Don’t block the onSensorChanged() method
* Avoid using deprecated methods or sensor types
* Verify sensors before using them
* Choose sensor delays carefully