Internal Sensor

Sensor Based Mobile Applications

Patrick Ausderau, Ulla Sederlöf, Jarkko Vuori

Helsinki Metropolia University of Applied Science

2022

Metropolia

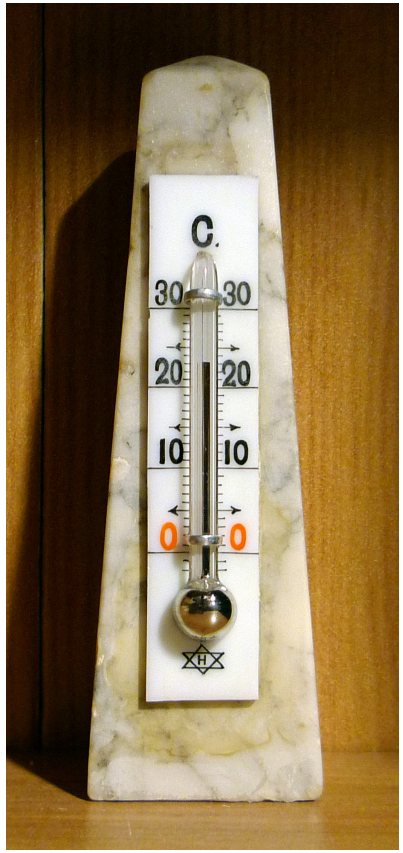
Outline

For more info:
`https://developer.android.com/guide/topics/sensors/sensors_overview`

Sensor

A sensor is a device that

- responds to a physical stimulus such as
  - heat
  - light
  - sound
  - pressure
  - magnetism
  - motion
  - ...
- and transmits a resulting impulse, typically
  - electrical signals
  - optical signals
  - ...



By Anonimski CC BY-SA 3.0, via Wikimedia Commons

http://www.merriam-webster.com/dictionary/sensor
https://en.wikipedia.org/wiki/Sensor

Metropolia

Sensor – Example

- Acoustic, sound, vibration
  - Microphone,...
- Chemical
  - Smoke detector, carbon monoxide detector,...
- Electric current, electric potential, magnetic, radio
  - Galvanometer, magnetometer, metal detector,...
- Flow, fluid velocity
  - Anemometer, water meter,...
- Ionizing radiation, subatomic particles
  - Geiger counter,...
- Position, angle, displacement, distance, speed, acceleration
  - Gravimeter, gyroscope, shock detector,...

Sensor – Example

- Optical, light, imaging, photon
  - Infra-red sensor, Photodetector, ...
- Pressure
  - Barometer, tactile sensor,...
- Force, density, level
  - Alcoholometer, viscometer,...
- Thermal, heat, temperature
  - Microwave radiometer, thermometer,...
- Proximity, presence
  - Doppler radar, motion detector,...
- ...

`https://en.wikipedia.org/wiki/List_of_sensors`

Internal Sensor

The Android platform supports three main categories of sensors

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

Internal Sensor Hack

Using a broad definition, almost any I/O capabilities of an Android device can be transformed into a sensor

- Audio Capture
  - e.g. use the `AudioRecord` to measure the decibel value of ambient noise.
- Camera
  - e.g. use the `MediaRecorder` to get the RGB color of an object
- Touch Motion
  - e.g. use the `MotionEvent.getPressure()` to know how strong the user press on the screen
- Location Services
  - e.g. use the `Location.getSpeed()` to transform the phone into speedometer
- ...
  - your imagination is the limit

Sensor Framework

The sensor framework includes the following four main classes and interfaces

- `SensorManager` class
  - provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information
  - provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors
- `Sensor` class
  - provides various methods that let you determine a sensor's capabilities

Sensor Framework

...Continued

- `SensorEvent` class
  - provides information about a sensor event
  - 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` interface
  - callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes

Sensor Framework

For an app that uses sensor, you usually

- Identify sensors and capabilities
  - runtime check if sensors exist. E.g. allow you to disable any application features that rely on sensors that are not present.
  - and their capabilities. E.g. identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.
- Sensor availability varies from device to device, it can also vary between Android versions[1]

```
1 private lateinit var sm: SensorManager
2 //...
3 sm = getSystemService(Context.SENSOR_SERVICE) as SensorManager
4 sm.getSensorList(Sensor.TYPE_ALL).forEach { Log.d(TAG, it.name) }
```

[1] https://developer.android.com/guide/topics/sensors/sensors_overview.html#table2

Sensor Framework

- **Always** verify that a sensor exists on a device before you attempt to acquire data from it.
- And/or declare in manifest. E.g. if you need magnetometer:

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

```
1  //...
2  if(sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null) {
3    // There is a magnetometer.
4    // You can register listener to get data and use them.
5  } else {
6    // No magnetometer.
7    // Disable related features or provide an alternative if
         possible?
8  }
```

Sensor Framework

Monitor sensor events (if the sensor you need is available)

- `onSensorChanged()` allows you to acquire raw sensor data 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
  - the raw sensor data that triggered the event

Sensor Framework

Monitor sensor events continued...

- `onAccuracyChanged()` provides you with a reference to the Sensor object that changed and the new accuracy of the sensor
- Accuracy is represented by one of four status constants
  - `SENSOR_STATUS_ACCURACY_LOW`
  - `SENSOR_STATUS_ACCURACY_MEDIUM`
  - `SENSOR_STATUS_ACCURACY_HIGH`
  - `SENSOR_STATUS_UNRELIABLE`

Sensor Framework

Monitor sensor events, best practice

- Don't block the `onSensorChanged()` method
  - the sensor data can change at a high rate which may result in many calls to that method
  - if you need to perform long lasting operations ⇒ delegate to e.g. a background task
- To avoid to waste system resources and battery power, choose a delivery rate that is suitable for your application or use-case when registering the event listener with `registerListener()` method.

Example - Initialize

```
1  class MainActivity : ComponentActivity(), SensorEventListener {
2
3    companion object{
4      private lateinit var sm: SensorManager
5      private var sMagnetic: Sensor? = null
6      private val sensorViewModel = SensorViewModel()
7    }
8
9    override fun onCreate(savedInstanceState: Bundle?) {
10     super.onCreate(savedInstanceState)
11
12     sm = getSystemService(Context.SENSOR_SERVICE) as
           SensorManager
13     sMagnetic = sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
14     setContent {
15       ShowSenorData(sensorViewModel)
16     }
17   }
```

Example - Sensor events listener

```
18      override fun onSensorChanged(p0: SensorEvent?) {
19        p0 ?: return
20        if (p0.sensor == sMagnetic) {
21          sensorViewModel.updateValue(
22              getString(
23                  R.string.sensor_val,
24                  p0.values[0],
25                  p0.values[1],
26                  p0.values[2]
27              )
28          )
29        }
30      }
31
32      override fun onAccuracyChanged(p0: Sensor?, p1: Int) {
33        Log.d(TAG, "onAccuracyChanged ${p0?.name}: $p1")
34      }
```

Example - Register/unregister event listener

```
35      override fun onResume() {
36        super.onResume()
37        sMagnetic?.also {
38          sm.registerListener(this, it,
                SensorManager.SENSOR_DELAY_UI)
39        }
40      }
41
42      override fun onPause() {
43        super.onPause()
44        sm.unregisterListener(this)
45      }
46    }
```

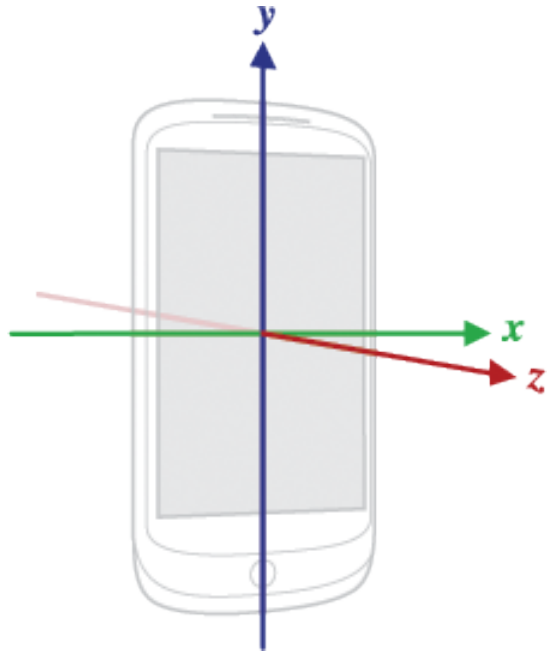
Example - LiveData with Compose[2]

```
47  class SensorViewModel : ViewModel() {
48    private val _value: MutableLiveData<String> = MutableLiveData()
49    val value: LiveData<String> = _value
50
51    fun updateValue(value: String) {
52      _value.value = value
53    }
54  }
55
56  @Composable
57  fun ShowSenorData(sensorViewModel: SensorViewModel) {
58    val value by sensorViewModel.value.observeAsState()
59    Text(value ?: "", Modifier.padding(8.dp))
60  }
```

[2] in module build gradle:
`implementation "androidx.compose.runtime:runtime-livedata:1.2.1"`

Coordinate System

- Some sensors such as
  - Acceleration sensor
  - Gravity sensor
  - Gyroscope
  - ...
- return data based on the standard 3-axis coordinate system
  - the coordinate system is defined relative to the device's screen in portrait mode
  - the X axis is horizontal and points right
  - the Y axis is vertical and points up
  - the Z axis points toward the outside of the screen face
  - the axes are not swapped when the device's screen orientation changes!



Android Open Source Project - Creative Commons Attribution 2.5
https://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords

Motion Sensors

- Two of the Motion Sensors are always hardware-based
  - accelerometer
  - gyroscope
- three of the Motion Sensors can be either hardware-based or software-based
  - gravity
  - linear acceleration
  - rotation vector

Details and usage: `https://developer.android.com/guide/topics/sensors/sensors_motion.html`

Motion Sensors

Motion sensors that are supported on the Android platform

- `TYPE_ACCELEROMETER` – Acceleration force along the 3 axis (including gravity) in $\mathrm{m/s^2}$.
- `TYPE_ACCELEROMETER_UNCALIBRATED` – Same as Acceleration but without any bias correction & estimated bias around the 3 axis in $\mathrm{m/s^2}$.
- `TYPE_GRAVITY` – Force of gravity along the 3 axis in $\mathrm{m/s^2}$.
- `TYPE_GYROSCOPE` – Rate of rotation around the 3 axis in $\mathrm{rad/s}$.
- `TYPE_GYROSCOPE_UNCALIBRATED` – Rate of rotation (without drift compensation) around the 3 axis & estimated drift around the 3 axis in $\mathrm{rad/s}$.

Motion Sensors

Motion sensors that are supported on the Android platform continued...

- `TYPE_LINEAR_ACCELERATION` – Acceleration force along the 3 axis (excluding gravity) in $\mathrm{m/s^2}$.
- `TYPE_ROTATION_VECTOR` – Rotation vector component along the 3 axis ($\mathit{axis} \cdot \sin(\theta/2)$) & scalar component of the rotation vector ($\cos(\theta/2)$) unit-less.
- `TYPE_STEP_COUNTER` – Number of steps taken by the user since the last reboot while the sensor was activated[3].
- `TYPE_STEP_DETECTOR` – Trigger the event when feet hit the ground (no data produced, always returns 1.0).
- (`TYPE_SIGNIFICANT_MOTION` – Special case, it's a a wake up sensor and is of type `TriggerEvent`[4]).

[3] Can steps be a unit?

[4] `https://developer.android.com/reference/android/hardware/TriggerEvent.html`

Position Sensors

- Two of the Position Sensors are always hardware-based
  - geomagnetic field sensor
  - proximity sensor
- The accelerometer is also used as one of the Position Sensors.

Details and usage: `https://developer.android.com/guide/topics/sensors/sensors_position.html`

Position Sensors

Position sensors that are supported on the Android platform

- `TYPE_GAME_ROTATION_VECTOR` – Rotation vector component along the 3 axis ($axis \cdot \sin(\theta/2)$) unit-less. Similar to Motion `TYPE_ROTATION_VECTOR`; but does not use the geomagnetic field. Therefore the Y axis does not point north but instead to some other reference.
- `TYPE_GEOMAGNETIC_ROTATION_VECTOR` – Rotation vector component along the 3 axis ($axis \cdot \sin(\theta/2)$) unit-less. Similar to Motion `TYPE_ROTATION_VECTOR`; but uses magnetometer instead of a gyroscope. Has lower accuracy but reduced power consumption.

Position Sensors

Position sensors that are supported on the Android platform continued...

- `TYPE_MAGNETIC_FIELD` – Geomagnetic field strength along the 3 axis in $\mu\mathrm{T}$.
- `TYPE_MAGNETIC_FIELD_UNCALIBRATED` – Geomagnetic field strength without hard iron calibration along the 3 axis & Iron bias estimation along the 3 axis in $\mu\mathrm{T}$.
- `TYPE_ORIENTATION` – deprecated! There is other way to compute a device's orientation[5] (e.g. azimuth from magnetic north pole).
- `TYPE_PROXIMITY` – Distance from object in cm. Note, some proximity sensors provide only binary values representing near and far.

[5] `https://developer.android.com/guide/topics/sensors/sensors_position.html#sensors-pos-orient`

Environment Sensors

- All four Environment Sensors are always hardware-based
  - relative ambient humidity
  - illuminance
  - ambient pressure
  - ambient temperature

Details and usage: `https://developer.android.com/guide/topics/sensors/sensors_environment.html`

Environment Sensors

Environment sensors that are supported on the Android platform

- `TYPE_AMBIENT_TEMPERATURE` – Ambient air temperature in $^{\circ}\mathrm{C}$.
- `TYPE_LIGHT` – Total luminous flux incident on a surface, per unit area in lx
- `TYPE_PRESSURE` – Ambient air pressure in hPa or mbar.
- `TYPE_RELATIVE_HUMIDITY` – Ambient relative humidity in $\%$.
- `TYPE_TEMPERATURE` – deprecated! Device temperature in $^{\circ}\mathrm{C}$.

Lab_w2_d2 Internal Sensor

- Pick one of the sensor from this list: `https://docs.google.com/spreadsheets/d/182AskpnDolw3an3PMwn4kCxGJCUB3X7KN_aSANRbBsc/` (use your metropolia account). The idea is to share your code in oma so all your classmates can benefit from it. So try to avoid to all choose the same sensor.
- Create an app that will show the data from that sensor. If possible try to show something meaningful. E.g.
  - how many steps does it take for walking 100m?
  - where is the north (south, east, west)?
  - can you detect a device shake?
  - can you detect a device (half) flip/pitch? roll? spin?
  - where the user is based on illumination level[6]
  - ...
- Test your app on a real device[7].

---

[6] `https://en.wikipedia.org/wiki/Lux#Illuminance`

[7] testing on emulator is not optimal. If you really have no choice try `https://developer.android.com/guide/topics/sensors/sensors_overview.html#test-with-the-android-emulator`