17. Lesson 15/05/23

Receiving and Using Values from Sensors

- How to obtain the list of sensors available in the device.
- How to read the values provided by a sensor.
- How to plot charts of the provided values.

Sensors in Android

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. (Such data can be exploited for a variety of applications).

The Android platform supports **three broad categories of sensors**:

1. **Motion sensors:** measure **acceleration** forces and **rotational** forces along three axes.
   (this category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors).
2. **Environmental sensors:** measure various **environmental parameters**, such as ambient air temperature and pressure, illumination, and humidity.
   (This category includes barometers, photometers and thermometers).
3. **Position sensors:** measure the **physical position** of a device.
   (This category includes orientation sensors).

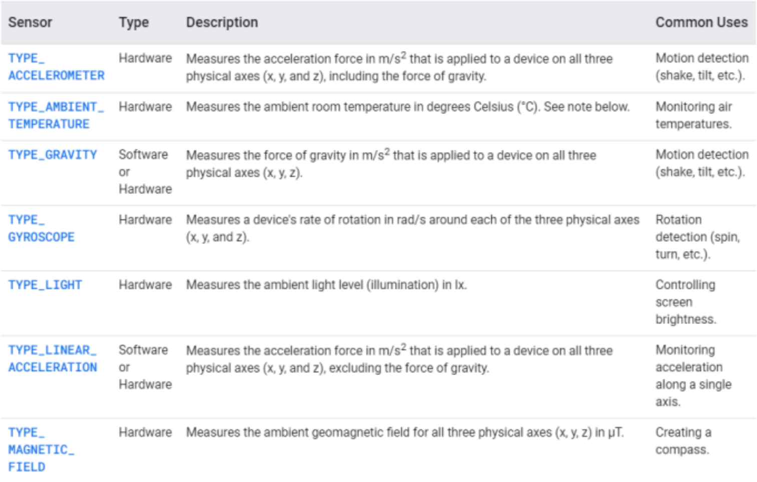
Android Sensor Framework

Android sensor framework allows to **access sensors available on the device and acquir**e raw sensor **data**.

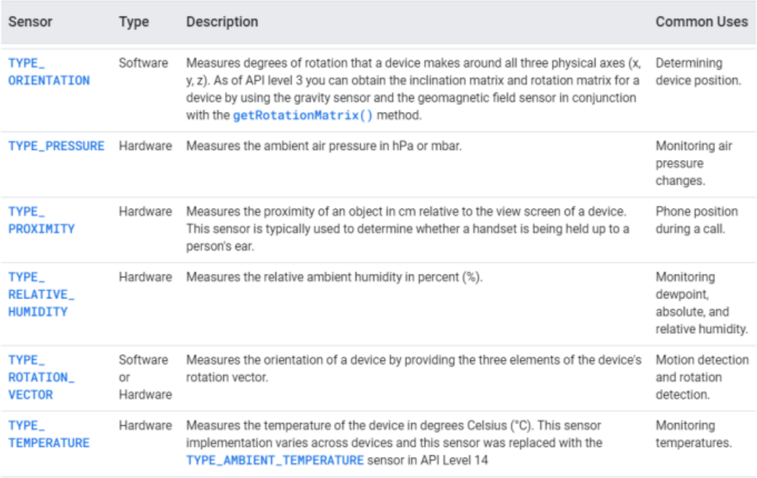
For example you can use the sensor framework to do the following:

- Determine which **sensors** are **available** on a device.
- Determine **individual sensor's capabilities**, such as its maximum range, manufacturer, power requirements, and resolution.
- **Acquire raw sensor data** and define the minimum rate at which you acquire sensor data.
- **Register and unregister sensor event listeners** that monitor sensor changes.

Example of Sensors



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.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.



Sensor	Type	Description	Common Uses
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.

Steps for using sensors

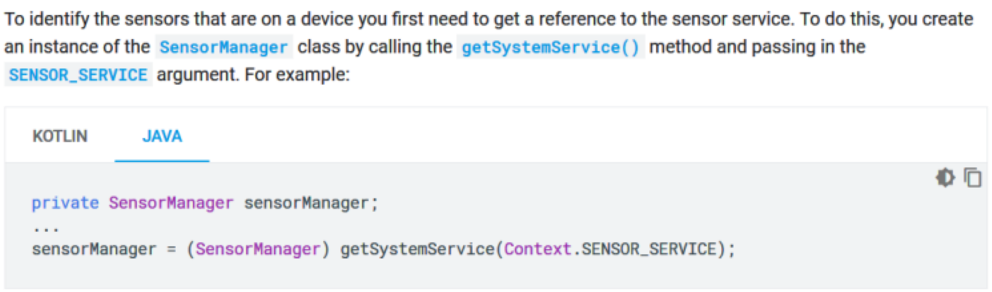
A typical application based on sensor-related APIs requires to perform two basic tasks:

- **Identifying sensors and sensor capabilities:** (at runtime), useful if the application has features that rely on specific sensor types or capabilities.

  For example, you may want to identify all the sensors that are present on a device and disable any application features that rely on sensors that are not present.

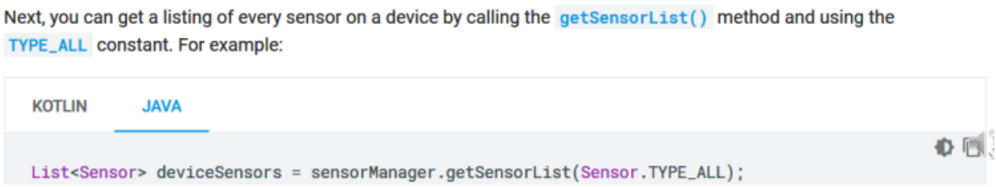
- **Monitor sensor events**: to acquire raw sensor data.
  - A **sensor event** occurs every time a sensor detects a change in the parameters it is measuring.
  - A **sensor event** provides you with four pieces of information:
    - The **name of the sensor** that triggered the event.
    - The **timestamp** for the event.
    - The **accuracy** of the event.
    - The **raw sensor data** that triggered the event.

Identifying the Available Sensors



To identify the sensors that are on a device you first need to get a reference to the sensor service. To do this, you create an instance of the `SensorManager` class by calling the `getSystemService()` method and passing in the `SENSOR_SERVICE` argument. For example:

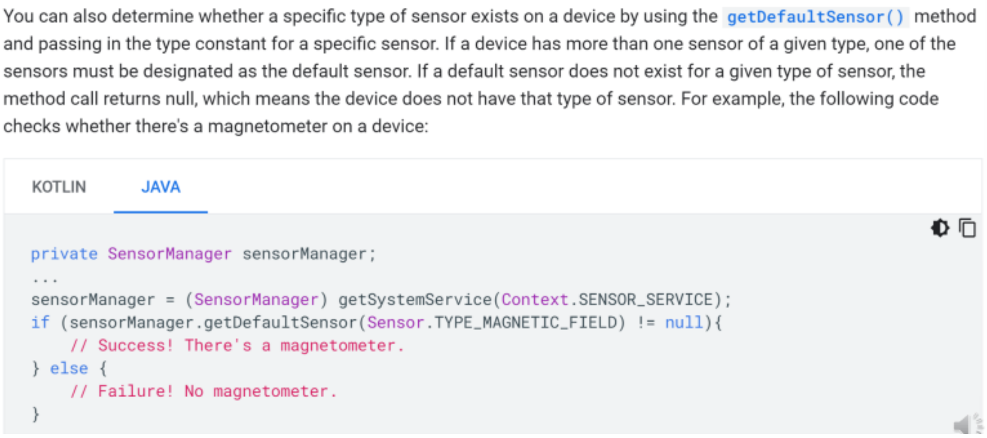
```java
private SensorManager sensorManager;
...
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```



Next, you can get a listing of every sensor on a device by calling the `getSensorList()` method and using the `TYPE_ALL` constant. For example:

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

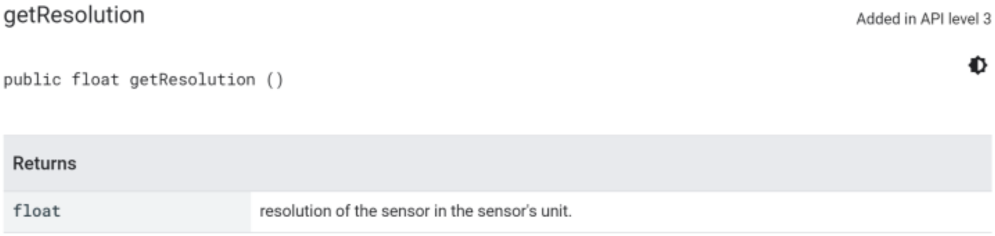
Finding a specific sensor



You can also determine whether a specific type of sensor exists on a device by using the `getDefaultSensor()` method and passing in the type constant for a specific sensor. If a device has more than one sensor of a given type, one of the sensors must be designated as the default sensor. If a default sensor does not exist for a given type of sensor, the method call returns null, which means the device does not have that type of sensor. For example, the following code checks whether there's a magnetometer on a device:

```
private SensorManager sensorManager;
...
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
    // Success! There's a magnetometer.
} else {
    // Failure! No magnetometer.
}
```

Capabilities and attributes of sensors

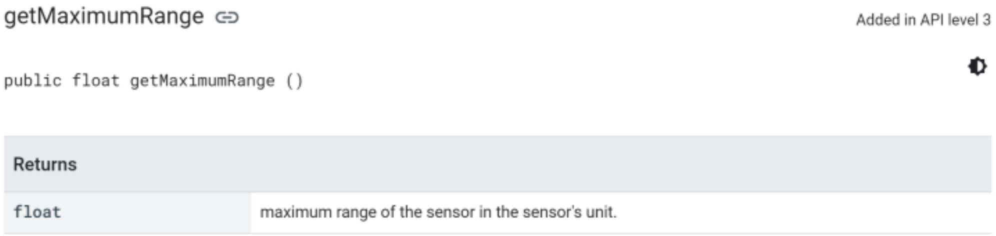


getResolution

Added in API level 3

```
public float getResolution ()
```

Returns	
`float`	resolution of the sensor in the sensor's unit.



getMaximumRange

Added in API level 3

```
public float getMaximumRange ()
```

Returns	
`float`	maximum range of the sensor in the sensor's unit.

Monitoring Sensor Events

To monitor raw sensor data you need to **implement two callback methods** that are exposed through the SensorEventListener interface: ***onSensorChanged()*** and ***onAccuracyChanged()***.

The Android system calls these methods whenever the following events occur:

- **A sensor reports a new value.**

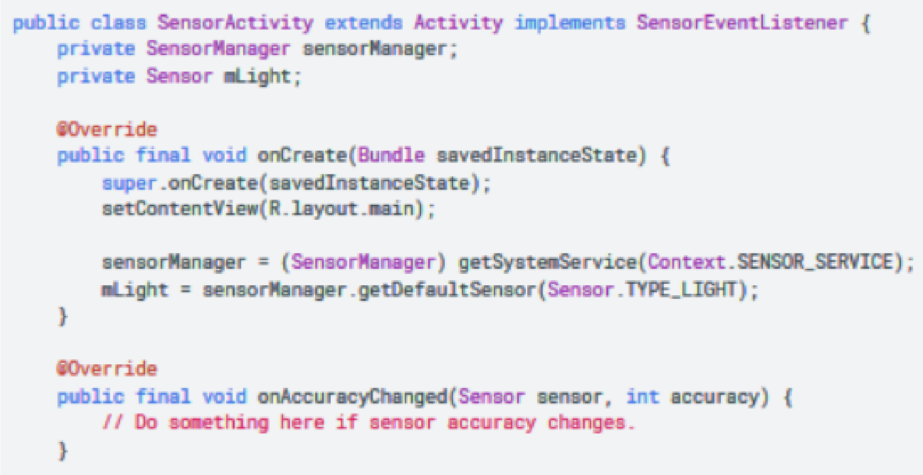
  In this case the system invokes the ***onSensorChanged()*** method, providing you with a SensorEvent object.

- **A sensor's accuracy changes.**

  In this case the system invokes the ***onAccuracyChanged()*** method, providing you with a reference to the Sensor object that changed and the new accuracy of the sensor.

onSensorChanged() Example

The following code shows how to use the onSensorChanged() method to monitor data from the light sensor:

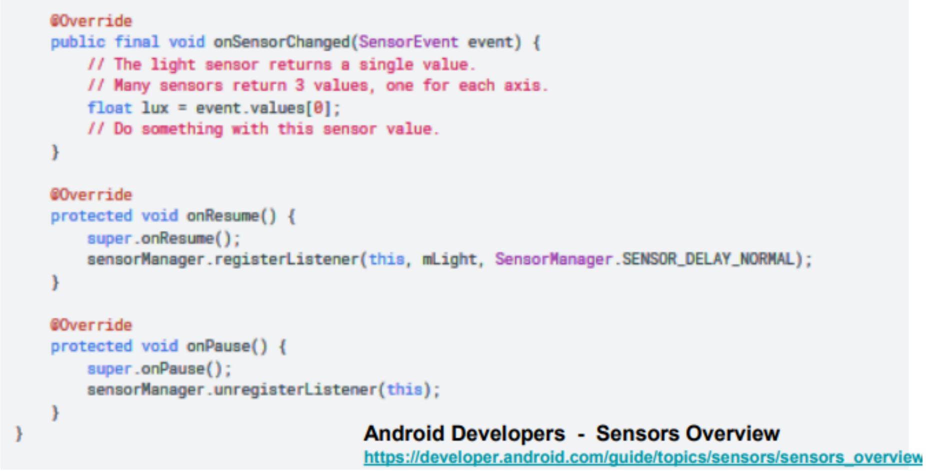


```
public class SensorActivity extends Activity implements SensorEventListener {
    private SensorManager sensorManager;
    private Sensor mLight;

    @Override
    public final void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
    }

    @Override
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Do something here if sensor accuracy changes.
    }
```



```
    @Override
    public final void onSensorChanged(SensorEvent event) {
        // The light sensor returns a single value.
        // Many sensors return 3 values, one for each axis.
        float lux = event.values[0];
        // Do something with this sensor value.
    }

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

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(this);
    }
}
```

Android Developers - Sensors Overview
https://developer.android.com/guide/topics/sensors/sensors_overview