ANDROID Sensors

Overview

- 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.
- Likewise, *a weather application* might use a device's *temperature sensor* and *humidity sensor* to calculate and report the dew point.
- Or a *travel application* might use the *geomagnetic field sensor* and *accelerometer* to report a compass bearing.

Types of Sensors

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

Android Sensor Framework

- You can access sensors available on the device and acquire raw sensor data by using the Android sensor framework.
- The sensor framework provides several classes and interfaces that help you perform a wide variety of sensor-related tasks. For example, you can use the sensor framework to do the following:

I. Determine which sensors are available on a device.

II. Determine an individual sensor's capabilities, such as its maximum range, manufacturer, power requirements, and resolution.

III. Acquire raw sensor data and define the minimum rate at which you acquire sensor data.

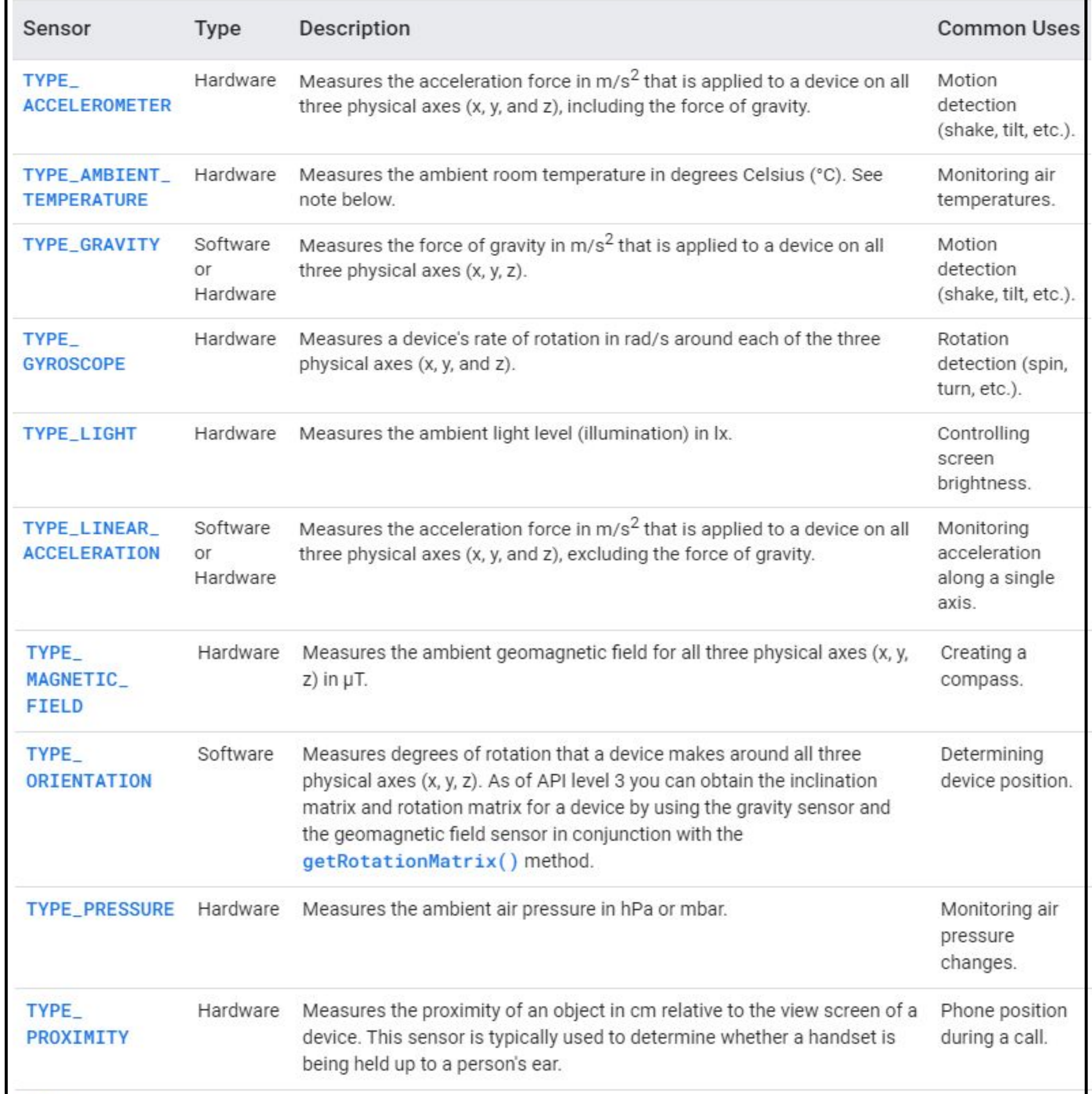
IV. Register and unregister sensor event listeners that monitor sensor changes.

Sensor Hardware/Software

- The Android sensor framework lets you access many types of sensors.
- They are either hardware based or software based.
- **Hardware Based:**
  - They are physical components built into a handset or tablet device.
  - They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change.
- **Software Based:**
  - These sensors are not physical devices, although they mimic hardware-based sensors.
  - Software-based sensors derive their data from one or more of the hardware-based sensors and are sometimes called virtual sensors or synthetic sensors.
  - The linear acceleration sensor and the gravity sensor are examples of software-based sensors.
- Few Android-powered devices have every type of sensor.
- Also, a device can have more than one sensor of a given type. For example, a device can have two gravity sensors, each one having a different range.

Sensor Types supported by Android Platform

Visit: [Full Chart of Sensor types]()



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.
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.

Sensor Framework Classes

- **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:

  I. The raw sensor data,

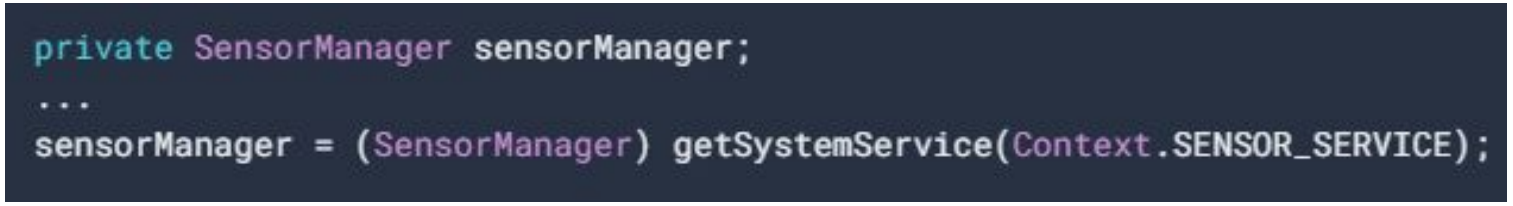
  II. The type of sensor that generated the event,

  III. The accuracy of the data, and the

  IV. 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.

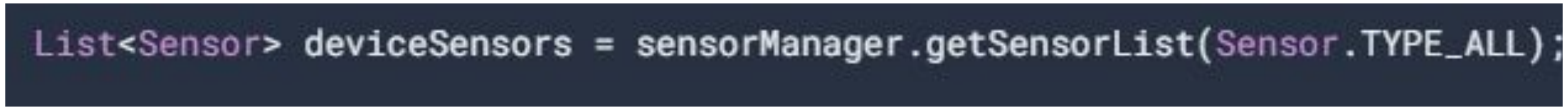
Identifying Sensors: SensorManager

- The Android sensor framework provides several methods that make it easy for you to determine at runtime which sensors are on a device.
- 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.



```
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.

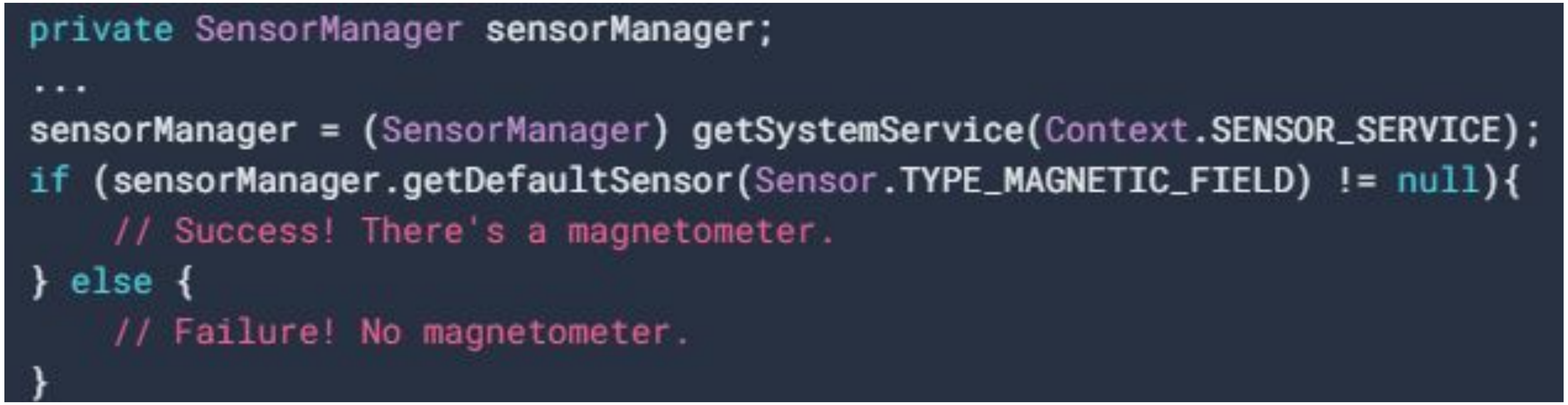


```
List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

- If you want to list all of the sensors of a given type, you could use another constant instead of **TYPE_ALL** such as **TYPE_GYROSCOPE**, **TYPE_LINEAR_ACCELERATION**, or **TYPE_GRAVITY**.

Identifying Sensors: SensorManager

- 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.
}
```

Sensor Capabilities: Sensor Class

- Various methods in Sensor class to get capabilities of Sensor
- **getMinDelay():** Method, which returns the minimum time interval (in microseconds) a sensor can use to sense data.
    - Any sensor that returns a non-zero value for the getMinDelay() method is a streaming sensor. Streaming sensors sense data at regular intervals and were introduced in Android 2.3 (API Level 9). If a sensor returns zero when you call the getMinDelay() method, it means the sensor is not a streaming sensor because it reports data only when there is a change in the parameters it is sensing.
- **getPower():** Method to obtain a sensor's power requirements. (in mA microAmps)
- **getMaximumRange()**: Method to obtain maximum range of measurement.
- **getResolution()**: Method to obtain a sensor's resolution.

Monitoring Sensor Events

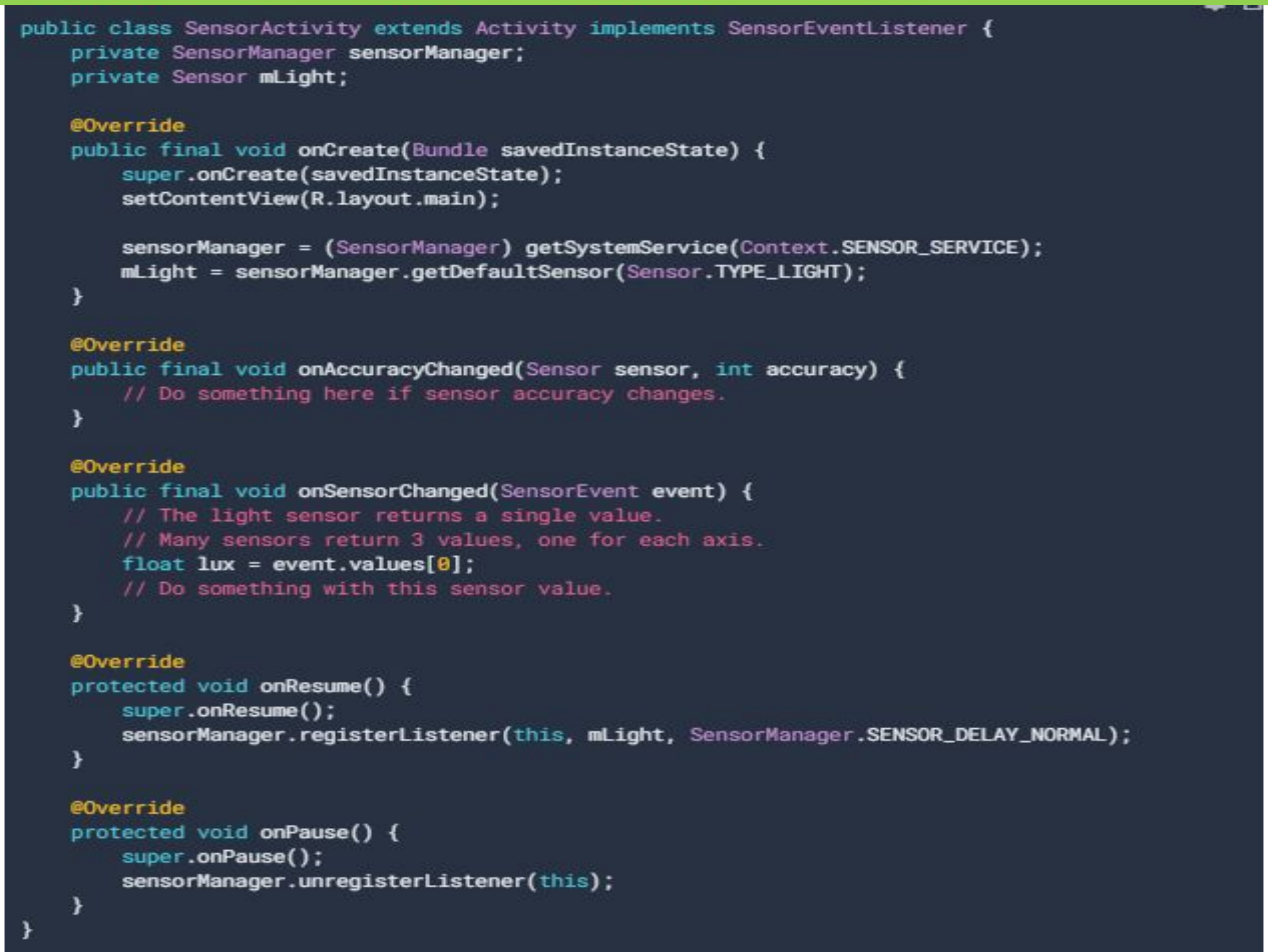
- To monitor raw sensor data you need to implement two callback methods that are exposed through the SensorEventListener interface: **onAccuracyChanged()** and **onSensorChanged()**.

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

- In this case the system invokes the onSensorChanged() method, providing you with a **SensorEvent object**.
- A SensorEvent object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

Sensor Events (Example Code)



```java
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);
    }
}
```

Code Explained

- The parameters in registering a Event Listener:
- registerListener(**SensorEventListener**, **Sensor**, int **data_delay**). This data_delay can be:
  - SENSOR_DELAY_NORMAL
  - SENSOR_DELAY_UI
  - SENSOR_DELAY_GAME
  - SENSOR_DELAY_FASTEST
  - time in microseconds (millionths of a second)
- The data delay (or sampling rate) controls the interval at which sensor events are sent to your application via the onSensorChanged() callback method.
- The default data delay is suitable for monitoring typical screen orientation changes and uses a delay of 200,000 microseconds.
- You can specify other data delays, such as SENSOR_DELAY_GAME (20,000 microsecond delay), SENSOR_DELAY_UI (60,000 microsecond delay), or SENSOR_DELAY_FASTEST (0 microsecond delay). As of Android 3.0 (API Level 11) you can also specify the delay as an absolute value (in microseconds).
- ***Question:*** *Which Delay should you select?*

Sensor Coordinate System

- In general, the sensor framework uses a standard 3-axis coordinate system to express data values.
- For most sensors, the coordinate system is defined relative to the device's screen when the device is held in its default orientation (see figure).
- When a device is held in its default orientation, the X axis is horizontal and points to the right, the Y axis is vertical and points up, and the Z axis points toward the outside of the screen face.
- In this system, coordinates behind the screen have negative Z values. This coordinate system is used by the following sensors:

- Acceleration sensor
- Gravity sensor
- Gyroscope
- Linear acceleration sensor
- Geomagnetic field sensor

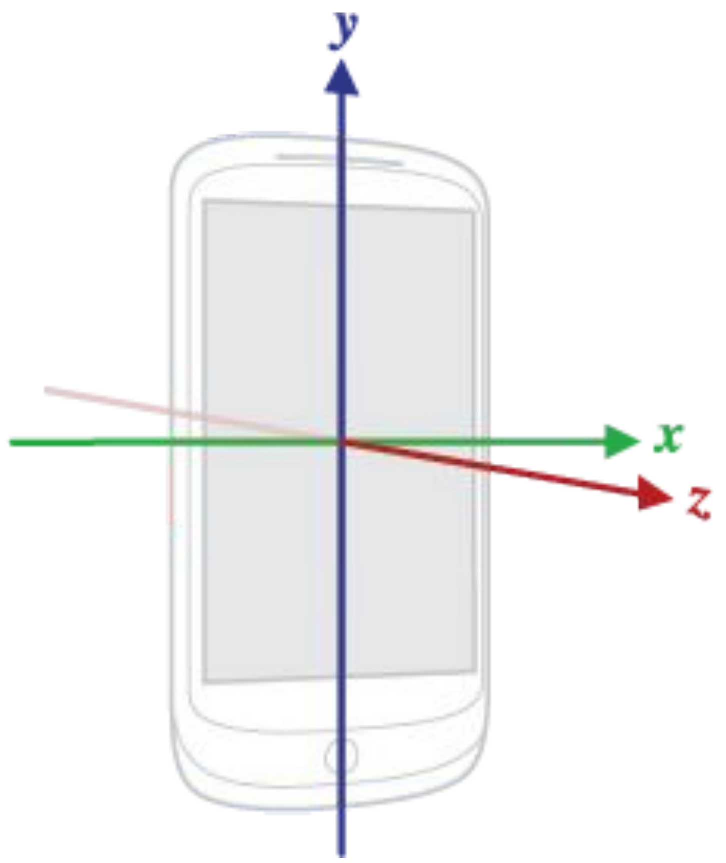


Figure: Coordinate system (relative to a device) that's used by the Sensor API.

Best Practices for Using Sensors

- Only gather sensor data in the foreground
- Unregister sensor listeners
- Test with the Android Emulator
- Don't block the onSensorChanged() method
- Avoid using deprecated methods or sensor types
- Verify sensors before you use them
- Choose sensor delays carefully

THANK YOU