Sensors in Android

IS2205 - Mobile Application Development

Why and What Android Sensors

Add more capabilities to the mobile computer compared to other computing devices such as servers and desktops

The Android platform supports three broad categories of sensors:

- Motion sensors
- Environmental sensors
- Position sensors

Why and What Android Sensors

- Motion sensors (Dynamic Aspects) - These sensors measure acceleration forces and rotational forces along three axes. i.e.: 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. i.e.: barometers, photometers, and thermometers.

- Position sensors (Static Aspects) - These sensors measure the physical position of a device. i.e.: orientation sensors and magnetometers.

Some use cases

- A game might track readings from a device's gravity sensor to identify user gestures and motions, such as tilt, shake, rotation, or swing.
- A weather application might use a device's temperature sensor and humidity sensor to calculate and report weather
- A travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

Activity: Find out **another** sensor that is be available in Android devices and describe a use case for it briefly

How can a developer access a sensor?

Android Sensor Framework in the Android API Framework

Supported Sensor-Related Tasks.

1. Enumerate the sensors available on an Android device.
2. Identify an individual sensor's capabilities i.e: maximum range, manufacturer, power requirements, and resolution.
3. Acquire raw sensor data and define the rate at which you acquire sensor data.
4. **Register** and **unregister** sensor event listeners that monitor sensor changes.

Sensor types: Not all devices are made equal

- Hardware based Sensors - Physical components built into a device. They derive their data by directly measuring specific physical parameters i.e.: acceleration, geomagnetic field strength.

- Software based Sensors - Not physical devices; mimic hardware-based sensors. Software-based sensors derive their data from one or more of the hardware-based sensors. AKA **virtual** sensors or **synthetic** sensors i.e.: linear acceleration sensor, gravity sensor.

Sensor Table

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
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_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_AMBIENT_TEMPERATURE	Hardware	Measures the ambient **room** temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the **device** in degrees Celsius (°C). This sensor implementation varies across devices.Was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Sensor Table...

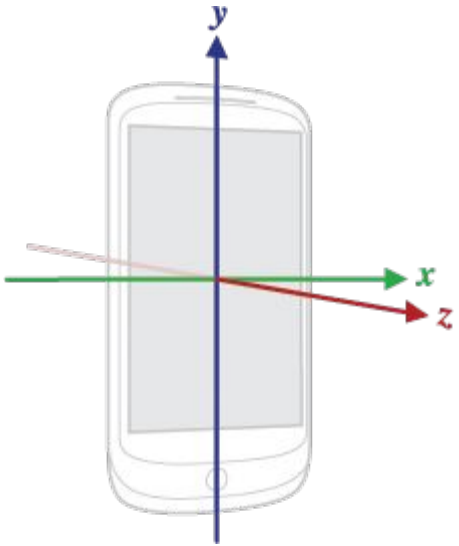
Sensor	Type	Description	Common Uses
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx (lumens per sq.meter).	Controlling screen brightness.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT (microteslas for magnetic flux).	Creating a compass.
TYPE_ORIENTATION	Software	**This constant was deprecated in API level 15.** use **SensorManager.getOrientation()** instead.	Determining device position.

Sensor Table...

Sensor	Type	Description	Common Uses
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa(hectopascal) or mbar(millibar)	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.

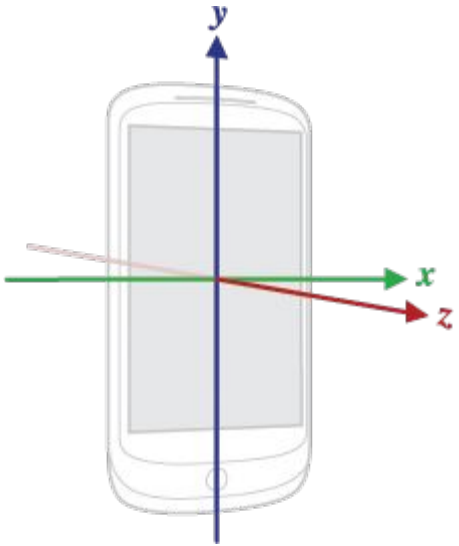
Sensor Coordinate System

- In general, the sensor framework uses a standard 3-axis coordinate system
- This coordinate system is defined relative to the device's screen when the device is held in its default orientation (see the figure).
- When a device is held in its default orientation(as the figure), 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.



Sensor Coordinate System

Questions: When the accelerometer sensor is checked in the given orientation of the device the Y value is approximately +9.8. What will happen to the value if the device starts to free fall from the same position and orientation? Explain the numbers.



For the above orientation if the device is quickly moved to the left side along which axis the value will be changed and what will be the sign of the reading?

Listing and Acquiring sensor object(s)

- Listing all sensors:

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
```

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

- SensorManager is the reference point for acquiring a sensor

```
val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
val sensor: Sensor? = sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY)
```

Acquiring a sensor object...Listener

- Example of a sensor class

```
class SensorActivity : Activity(), SensorEventListener {

    private lateinit var sensorManager: SensorManager
    private var pressure: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        pressure = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE)
    }
```

Acquiring a sensor object...

- Callbacks for Event listening

```
override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
    // Do something here if sensor accuracy changes.
}

override fun onSensorChanged(event: SensorEvent) {
    val millibarsOfPressure = event.values[0]
    // Do something with this sensor data.
}
```

Acquiring a sensor object...

- Acquiring and Releasing

```
override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()
    sensorManager.registerListener(this, pressure, SensorManager.SENSOR_DELAY_NORMAL)
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    sensorManager.unregisterListener(this)
}
```

To Read: Important classes on sensors...

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: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the 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. May need to change based on the application (i.e.: a mobile game, a drawing application)

Filtering by device capabilities

- If you add this element and descriptor to your application's manifest, users will see your application on Google Play only if their device has an accelerometer.
- You may also set required to false if the app can function even without the feature

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

Best practices

Gather sensor data in the foreground

- After Android 9+, apps running in the background have the restrictions of reporting sensor events

Avoid blocking the onSensorChanged(SensorEvent) method

- Sensor data can change at a high rate => System may call the onSensorChanged(.) method often.
- Do as little as possible within the onSensorChanged(.) method.
- If required do processing of sensor data outside of the onSensorChanged(.) method.

Best practices...

Avoid using deprecated methods or sensor types

- Several old methods and constants have been deprecated.
- TYPE_ORIENTATION sensor type has been deprecated get orientation data via the **getOrientation()** method.
- The TYPE_TEMPERATURE sensor type has been deprecated.
- Use the TYPE_AMBIENT_TEMPERATURE sensor type instead on devices that are running Android 4+.

Best practices...

Verify sensors before you using them

- Always verify that a sensor exists on a device before you attempt to acquire data from it.
- Do not assume that a sensor exists simply because it's a frequently-used sensor.
- Device manufacturers are not required to provide any particular sensors in their devices.

Best practices...

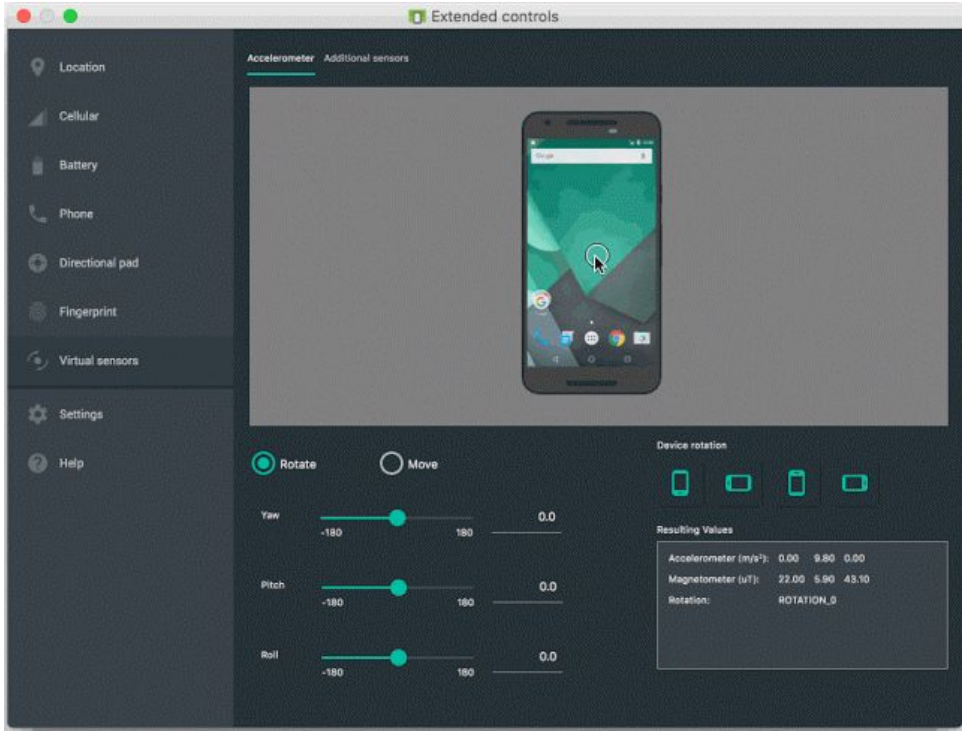
Choose sensor delays carefully

- When registering for sensor events, choose a delivery rate suitable for your use-case.
- Requesting the system to send extra updates that you don't need wastes system resources => More Heat, Quicker battery drains

Best practices...

Test with the Android Emulator and App “SdkControllerSensor” (via USB)

- The emulator uses a connection with an Android device that is running the SdkControllerSensor app.
- The SdkControllerSensor app monitors changes in the sensors on the device and transmits them to the emulator.
- The emulator is then transformed based on the new values that it receives from the sensors on your device.



Summary

Key Takeaways:

Sensors enhance mobile apps with context-aware features.

Use the Sensor Framework to access and manage sensors.

Follow best practices to optimize performance and battery life.

The End