Mobile Sensors Support

In Android Platform

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.

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

* **Motion sensors**: These sensors measure acceleration forces and rotational forces along three axis . 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.

Sensors can be accessed on the device and raw sensor data can be acquired 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:

* Determine which sensors are available on a device.
* Determine an 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.

Some of the sensors are hardware based and some are software based. Hardware-based sensors derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change.

Software-based 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.

Sensor types supported by the android platform:

|  |  |  |  |
| --- | --- | --- | --- |
| Sensor | Type | Description | Common Use |
| 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) | motion detection (shake, tilt, etc..). |
| TYPE\_AMBIENT\_TEMPERATURE | Hardware | Measures the ambient room temperature in degrees Celsius (°C) | monitoring air temperatures |
| TYPE-GRAVITY | Hardware or Software | 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 | Measure the ambient light level | Controlling screen brightness |
| TYPE\_MAGNETIC\_FIELD | Hardware | Measures the ambient geomagnetic field for all three physical axes (x, y, z) | Creating a compass |
| 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\_ORIENTATION | Software | Measures degrees of rotation that a device makes around all three physical axes (x, y, z) | Determining device position |

**Sensor Framework**

Sensors can be accessed and acquire new data from it by using the Android sensor framework, includes these main classes and interfaces:

**Sensor Manager** --> 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 acquire orientation information.

**Sensor** --> Use this class to create an instance of a specific sensor. This class also provides methods that lets you determine a sensor’s capabilities.

**Sensor Event** --> The system uses this class to create a sensor event object, which provides information about a sensor event. This object includes: raw sensor data, the type of sensor that generated the event, the accuracy of the data, the timestamp of the event.

**Sensor Event Listener** --> This interface can be used to create two callback methods that receive notifications (sensor events) when the sensor values change or sensor accuracy changes.

In a typical application sensor-related APIs are used to perform 2 main tasks:

* **Identify sensors and sensor capabilities**:

It is useful if the application has features that rely on specific sensor types, for example, you may want to identify all of the sensors that are present on a device and disable any application features that rely on sensors that are not present.

* **Monitor sensor events**:

Monitoring sensor events is how 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 four pieces of information:

1. The Name of the sensor that triggered the event.
2. The timestamp for the event.
3. The accuracy of the event.
4. The raw sensor data that triggered the event.

**Sensor Availability**

While sensor availability varies from device to device, it can also vary between android versions.

**Identifying sensors and sensor capabilities :-**

Using android SensorManager , you can get a list of all available sensors on a device by calling getSensorList function on SensorManager object passing Sensor.TYPE\_ALL argument to it.

The method returns list of Sensor objects to be used to get sensor type, name, power, delay between sensor events and maximum range,..etc.

//-------------------------------------------------------------------------------------//

SensorManager sensorManager = (SensorManager)getSystemService(Context.SENSOR\_SERVICE);

**List**<Sensor> sensorList = sensorManager.getSensorList(Sensor.TYPE\_ALL);

String sensorInfo = "";

**for** (Sensor s : sensorList)

{ sensorInfo= sensorInfo + s.getName()+ "\n"; }

//-------------------------------------------------------------------------------------//

**Monitoring sensor events:**

To monitor raw sensor data two callback methods needs to be implemented through the SensorEventListener interface: onAccuracyChanged() & onSensorChanged() , the Android system calls these methods whenever the following occurs:

1. **A sensor’s accuracy changes.**

When the system invokes the onAccuracyChanged() method it is because a sensor reports with different accuracy, providing 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\_ACCURACY\_UNRELIABLE.

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

In this case the system invokes the onSensorChanged() method , providing 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 & the new data that the sensor recorded.

**Handling Different Sensor Configuration:**

Android does not specify a standard sensor configuration for devices, which means device manufacturers can incorporate any sensor configuration that they want in their Android-powered devices. As a result, devices can include a variety of sensors in a wide range of configurations. If an application relies on a specific type of sensor, the owner of the application has to ensure that the sensor is present on a device of it to run successful.

**There are 2 options to ensure that a given sensor is present on a device:**

* Detect sensors at runtime and enable or disable application features as appropriate.
* Use Google Play filters to target devices with specific sensor configurations.

**Detecting sensors at runtime**

If an application uses specific type of sensor, but doesn’t rely on it, the sensor framework can be used to detect the sensor at runtime and then disable or enable application features as appropriate. For example, the following code checks whether there’s a pressure sensor on a device :-

//-------------------------------------------------------------------------------------//

Private SensorManager mSensorManager;

MSensorManager = (SensorManager) getSystemService(Context.SENSOR\_SERVICE);

If (mSensorManager.getDefaultSensor(Sensor.TYPE\_PRESSURE) != null){

// There is a pressure sensor

} else {

//No pressure sensor

}

//-------------------------------------------------------------------------------------//

**Using Google Play filters to target specific sensor configurations**

If an application gets published on Google Play and it need specific sensors to be able to operate, developers can use the <uses-feature> element in the manifest file to filter the application from the device that do not have the appropriate sensor configuration for the application. The <uses-feature> element has several hardware descriptors to filter applications based on the presence of specific sensors. Example :-

//-------------------------------------------------------------------------------------//

<uses-feature android:name=”android.hardware.sensor.accelerometer”

Android:required=”true” />

//-------------------------------------------------------------------------------------//

If this element and descriptor is added to the application’s manifest, users will see the application on Google Play only if their device has an accelerometer.

The descriptor should be set to Android:required=”true” only if the application relies entirely on a specific sensor. If the application uses a sensor for some functionality but still operates without the sensor, then the sensor should be listed in the <uses-feature> element, but the descriptor should be set to android:required=”false”. This helps ensure that the device can install the application even if they don’t have that particular sensor.

**Best Practices for Accessing & using sensors**

1. **Only gather sensor data in the foreground.**

On devices running android 9 (codename is “pie”) or higher, apps running in the background have the following restrictions:

* Sensors that use continuous reporting mode, such as accelerometer and gyroscopes, don’t receive events, so it will be useless to try to gather sensor data when it’s in the background.

1. **Unregister sensor listeners**

Be sure to unregister a sensor’s listener when you are done using the sensor or when the sensor activity pauses. If a sensor listener is registered and it’s activity is paused, the sensor will continue to acquire data and use battery resources unless you unregister the sensor. The following code shows how to use the onPause() method to unregister a listener:-

//-------------------------------------------------------------------------------------//

Private SensorManager mSensorManager;

@Override

Protected void onPause() {

Super.onPause();

MSensorManager.unregisterListener(this);

}

//-------------------------------------------------------------------------------------//

1. **Test with the Android Emulator**

The Android Emulator includes a set of virtual sensor controls that allow you to test sensors such as accelerometer, ambient temperature, proximity, light and others.

The emulator uses a connection with an Android device, it 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.

**4. Verify sensors before you use them**

Always verify that a sensor exists on a device before you attempt to acquire data from it. Don't 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.

**5. Choose sensor delays carefully**

When you register a sensor with the [registerListener()](https://developer.android.com/reference/android/hardware/SensorManager.html#registerListener(android.hardware.SensorEventListener,%20android.hardware.Sensor,%20int)) method, be sure you choose a delivery rate that is suitable for your application or use-case. Sensors can provide data at very high rates. Allowing the system to send extra data that you don't need wastes system resources and uses battery power.