

Android Studio

CSC3054 / CSC7054

Introduction to Android Sensors

What is a sensor?

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 dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

Types of sensors

| Sensor | Description |
|------------------------------|---|
| Motion Sensors | These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors. |
| Position Sensors | These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers. |
| 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. |

Why use sensors?

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:

- 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.

Introduction to Sensors

The Android sensor framework lets you access many types of sensors. Unfortunately, you cannot test the accelerometer on the Android emulator. Some of these sensors are hardware-based and some are software-based. Hardware-based sensors 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 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. Table 1 summarizes the sensors that are supported by the Android platform.

Few Android-powered devices have every type of sensor. For example, most handset devices and tablets have an accelerometer and a magnetometer, but fewer devices have barometers or thermometers. 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 | 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 ($^{\circ}\text{C}$). 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 <code>getRotationMatrix()</code> 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 <code>TYPE_AMBIENT_TEMPERATURE</code> sensor in API Level 14 | Monitoring temperatures. |

Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the `android.hardware` package and includes the following classes and interfaces:

| Class | Description |
|----------------------------------|---|
| <code>SensorManager</code> | 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. |
| <code>Sensor</code> | 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. |
| <code>SensorEvent</code> | 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. |
| <code>SensorEventListener</code> | 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 and Sensor Capabilities

The Android sensor framework provides several methods that make it easy for you to determine at runtime which sensors are on a device. The API also provides methods that let you determine the capabilities of each sensor, such as its maximum range, its resolution, and its power requirements.

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:

```
private SensorManager mSensorManager;  
...  
mSensorManager = (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:

```
List<Sensor> deviceSensors = mSensorManager.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`.

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

Public methods of the Sensor Class

This can be used to determine the capabilities and attributes of individual sensors. This is useful if you want your application to behave differently based on which sensors or sensor capabilities are available on a device.

| Method | Description |
|---|---|
| <code>getResolution()</code> and <code>getMaximumRange()</code> | Obtains a sensor's resolution and maximum range of measurement. |
| <code>getPower()</code> | Obtains a sensor's power requirements. |
| <code>getMinDelay()</code> | This method returns the minimum time interval (in microseconds) a sensor can use to sense data. It is useful because it lets you determine the maximum rate at which a sensor can acquire data. |

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()`. The Android system calls these methods whenever the following occurs:

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`
- `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
- The new data that the sensor recorded.

An Example

The following code shows how to use the `onSensorChanged()` method to monitor data from the light sensor. This example displays the raw sensor data in a `TextView` that is defined in the `activity_main.xml` file as `sensor_data`.

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

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

        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mLight = mSensorManager.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();
        mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
    }
}
```

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

Detecting sensors at runtime

If your application uses a specific type of sensor, but doesn't rely on it, you can use the sensor framework to detect the sensor at runtime and then disable or enable application features as appropriate. For example, a navigation application might use the temperature sensor, pressure sensor, GPS sensor, and geomagnetic field sensor to display the temperature, barometric pressure, location, and compass bearing. If a device doesn't have a pressure sensor, you can use the sensor framework to detect the absence of the pressure sensor at runtime and then disable the portion of your application's UI that displays pressure. 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) {
    // Success! There's a pressure sensor.
}
else {
    // Failure! No pressure sensor.
}
```

Using Google Play filters to target specific sensor configurations

If you are publishing your application on Google Play you can use the `<uses-feature>` element in your manifest file to filter your application from devices that do not have the appropriate sensor configuration for your application. The `<uses-feature>` element has several hardware descriptors that let you filter applications based on the presence of specific sensors. The sensors you can list include: accelerometer, barometer, compass (geomagnetic field), gyroscope, light, and proximity. The following is an example manifest entry that filters apps that do not have an accelerometer:

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

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 should set the descriptor to `android:required="true"` only if your application relies entirely on a specific sensor.

If your application uses a sensor for some functionality, but still runs without the sensor, you should list the sensor in the `<uses-feature>` element, but set the descriptor to `android:required="false"`. This helps ensure that devices can install your app even if they do not have that particular sensor. This is also a project management best practice that helps you keep track of the features your application uses. Keep in mind, if your application uses a particular sensor, but still runs without the sensor, then you should detect the sensor at runtime and disable or enable application features as appropriate.

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 1). 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:

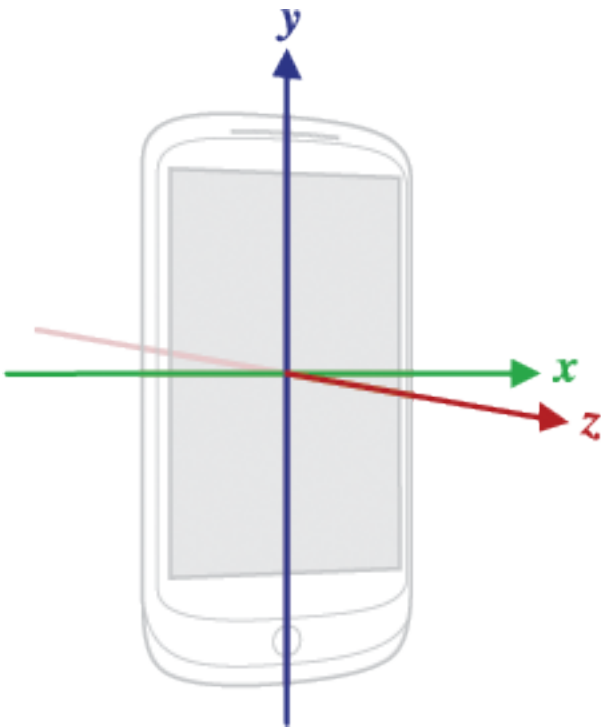


Figure 1. Coordinate system (relative to a device) that's used by the Sensor API.