Sensors API

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Sensors

- Sensors Overview
- Introduction to Sensors
- Sensor Framework
- Sensor Availability
- Identifying Sensors and Sensor Capabilities
- Monitoring Sensor Events
- Handling Different Sensor Configurations
- Examples

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

Sensors Overview

The Android platform supports three broad categories of sensors:

- 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.

Sensors Overview

- 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.

3.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
  1. Sensor Manager
  2. Sensor
  3. Sensor Event
  4. Sensor EventListener

3.Sensor Framework

3.1 Sensor Manager

- 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.

3.Sensor Framework

3.2 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.

3.Sensor Framework

3.3 Sensor Event

- 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.

3.Sensor Framework

3.4 Sensor Eventlistener

- You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes

3.Sensor Framework

In a typical application you use these sensor-related APIs to perform two basic tasks:

- **Identifying sensors and sensor capabilities** Identifying sensors and sensor capabilities at runtime is useful if your application has features that rely on specific sensor types or capabilities. 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. Likewise, you may want to identify all of the sensors of a given type so you can choose the sensor implementation that has the optimum performance for your application.
- **Monitor sensor events** Monitoring sensor events is how you 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, and the raw sensor data that triggered the event.

Identifying Sensors and Sensor Capabilities

- 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);
```

Identifying Sensors and Sensor Capabilities

- 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 thegetDefaultSensor() 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.  }
```

Identifying Sensors and Sensor Capabilities

```
private SensorManager mSensorManager;
private Sensor mSensor;
..
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

if (mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null){
  List<Sensor> gravSensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
  for(int i=0; i<gravSensors.size(); i++) {
    if ((gravSensors.get(i).getVendor().contains("Google Inc.")) &&
       (gravSensors.get(i).getVersion() == 3)){
      // Use the version 3 gravity sensor.
      mSensor = gravSensors.get(i);
    } }}
else{
  // Use the accelerometer.
  if (mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
    mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  }
  else{
    // Sorry, there are no accelerometers on your device.
    // You can't play this game.
  }}
```

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 theSensor 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 SensorEventobject. 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.

Monitoring Sensor Events

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 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.
  }
```

Monitoring Sensor Events

```
@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);
}}
```

Handling Different Sensor Configurations

- Android does not specify a standard sensor configuration for devices, which means device manufacturers can incorporate any sensor configuration that they want into their Android-powered devices. As a result, devices can include a variety of sensors in a wide range of configurations. For example, the Motorola Xoom has a pressure sensor, but the Samsung Nexus S does not. Likewise, the Xoom and Nexus S have gyroscopes, but the HTC Nexus One does not. If your application relies on a specific type of sensor, you have to ensure that the sensor is present on a device so your app can run successfully. You have two options for ensuring 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.

Handling Different Sensor Configurations

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.  }
```

Handling Different Sensor Configurations

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" />
```

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

Accelerometer

- We will build an application which will change its background color, if it is shuffled. Create a new Android project called *de.sisoft.android.sensor* with an *activity* called *SensorTestActivity*
- Change your layout file to the following code.

```
public class SensorTestActivity extends Activity implements SensorEventListener {
  private SensorManager sensorManager;
  private boolean color = false;
  private View view;
  private long lastUpdate;


/** Called when the activity is first created. */

  @Override
  public void onCreate(Bundle savedInstanceState) {
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    view = findViewById(R.id.textView);
    view.setBackgroundColor(Color.GREEN);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    lastUpdate = System.currentTimeMillis();
  }

  @Override
  public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
      getAccelerometer(event);
    }

  }
```

Accelerometer

```
private void getAccelerometer(SensorEvent event) {
  float[] values = event.values;
  // Movement
  float x = values[0];
  float y = values[1];
  float z = values[2];

  float accelationSquareRoot = (x * x + y * y + z * z)
      / (SensorManager.GRAVITY_EARTH * SensorManager.GRAVITY_EARTH);
  long actualTime = System.currentTimeMillis();
  if (accelationSquareRoot >= 2) //
  {
    if (actualTime - lastUpdate < 200) {
      return;
    }
    lastUpdate = actualTime;
    Toast.makeText(this, "Device was shuffed", Toast.LENGTH_SHORT)
        .show();
    if (color) {
      view.setBackgroundColor(Color.GREEN);

    } else {
      view.setBackgroundColor(Color.RED);
    }
    color = !color;
  }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}
```

Accelerometer

```
  @Override
  protected void onResume() {
    super.onResume();
    // register this class as a listener for the orientation and
    // accelerometer sensors
    sensorManager.registerListener(this,
        sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_NORMAL);
  }

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

Android Proximity Sensor Example

- Every android mobile is shipped with sensors to measure various environmental conditions. In this example we will see how to use the proximity sensors in an android application. Proximity sensors would be useful to reveal the nearness of an object to the phone. We might often experience that our phone screen would turn off when we bring the phone to our ears when we are in a call and the screen will turn on when we take it back. This is because the proximity sensor recognizes the object near the phone.
- In this example we will change the image in the ImageView w.r.t the nearness of any object to the phone. Check the sensor functionality by covering the sensor region (mostly the top left of the phone) with your hand .
- **So lets start**

1. Create a new project **File ->New -> Project ->Android ->Android Application Project**. While creating the new project, name the activity as ProximitySensorActivity(ProximitySensorActivity**.java)** and layout as**sensor_screen.xml**

Android Proximity Sensor Example

2. Now let us design the UI for the SensorActivity i.e **sensor_screen.xml** with an ImageView.
Have two images in drawable folder to differentiate near and far activity.

sensor_screen.xml:

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent“
   tools:context=".SensorActivity" >
      <ImageView android:id="@+id/imageView1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content“
       android:src="@drawable/far" />
</RelativeLayout>
```

Android Proximity Sensor Example

3. Now in the SensorActivity we access the device sensor using **SensorManager,** an instance of this class is got by calling **getSystemService(SENSOR_SERVICE)** . We implement the class with the**SensorEventListener** interface to override its methods to do actions on sensor value change.

ProximitySensor Activity.java:

```
public class ProximitySensor extends Activity implements SensorEventListener {
 private SensorManager mSensorManager;
 private Sensor mSensor;
 ImageView iv;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  // TODO Auto-generated method stub
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
  mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
  iv = (ImageView) findViewById(R.id.imageView1);
```

Android Proximity Sensor Example

```
protected void onResume() {
 super.onResume();
 mSensorManager.registerListener(this, mSensor,
  SensorManager.SENSOR_DELAY_NORMAL);
}
protected void onPause() {
 super.onPause();
 mSensorManager.unregisterListener(this);
}
public void onAccuracyChanged(Sensor sensor, int accuracy) { }
public void onSensorChanged(SensorEvent event) {
 if (event.values[0] == 0) {
  iv.setImageResource(R.drawable.near);//near
 } else {
  iv.setImageResource(R.drawable.far);//far
 } }
}
```

4.Run the project by rightclicking project **Run as → android project.**

Output:

The output of this example would be similar to the one as follows: