Android phones available in today's market have a wide variety of powerful and highly precise sensors. Interesting applications can be built with them, such as a local weather app using weather sensors, an app that analyzes risky driving behavior using motion sensors, and so on. Sensors in external devices such as an Android watch, a body analyzer, and a weight machine, and so on can also be connected and used from your Android app running on your phone.

Moving further, this book will provide the skills required to use sensors with your Android applications. It will walk you through all the fundamentals of sensors and will provide a thorough understanding of the Android sensor framework. You will also get to learn how to write code for the supportive infrastructure such as background services, scheduled and long-running background threads, and databases for saving sensor data. Additionally, you will learn how to connect and use sensors with external devices from your Android app using the Google Fit platform.

By the end of the book, you will be well versed in the use of Android sensors and programming to build interactive applications.

Who this book is written for
This book is targeted at Android developers who want to get a good understanding of sensors and write sensor-based applications, or who want to enhance their existing applications with additional sensor functionality.

What you will learn from this book
- Learn about sensor fundamentals, different types of sensor, and the sensor coordinate system
- Understand the various classes, callbacks, and APIs of the Android sensor framework
- Check all the available sensors on an Android device and know their individual capabilities— for example, their range of values, power consumption, and so on
- Implement sensor fusion using two or more sensors together and learn to compensate for the weakness of one sensor by using the strength of another
- Build a variety of sensor-based, real-world applications such as weather, pedometer, compass, driving events detection, fitness tracker and so on
- Get to know about wake up and non-wake up sensors, wake locks, and how to use sensor batch processing along with the sensor hardware FIFO queue
- Develop efficient battery and processor algorithms using raw sensor data to solve real-world problems

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 4 ‘The Light and Proximity Sensors’
- A synopsis of the book’s content
- More information on Android Sensor Programming By Example
About the Author

Varun Nagpal has been developing mobile apps since 2005 and has developed and contributed to more than 100 professional apps and games on various platforms, such as Android, iOS, Blackberry, and J2ME. Android app development has been his main area of expertise, and he has developed apps for a wide variety of Android devices, such as Android phones, tablets, watches, smart TVs, Android Auto, and Google Glass.

He moved to Chicago in late 2013, and since then, he has become a seasoned mobile architect. He has worked in different roles (mobile architect, technical lead, senior developer, and technical consultant) for a variety of various global clients (Allstate, Verizon, AT&T, Sydbank Denmark, SiS Taiwan, Chams PLC Nigeria, and Nandos South Africa) in order to implement their mobile solutions. He has SCJP (Core Java) and SCWD (JSP and Servlets) certifications from Sun Microsystems and MCP (C#) and MCTS (ASP.NET) certifications from Microsoft. You can find his blogs on mobile technology and white papers written by him on his website at [http://www.varunnagpal.com/](http://www.varunnagpal.com/).

Welcome to *Android Sensor Programming By Example*. This book will provide you the skills required to use sensors in your Android applications. It will walk you through all the fundamentals of sensors and will provide a thorough understanding of the Android Sensor Framework. This book will cover a wide variety of sensors available on the Android Platform. You will learn how to write code for the infrastructure (service, threads, database) required to process high volumes of sensor data. This book will also teach you how to connect and use sensors in external devices (such as Android Wear) from the Android app using the Google Fit platform.

You will learn from many real-world sensor-based applications such, as the Pedometer app to detect daily steps, the Weather app to detect temperature, altitude, absolute and humidity, the Driving app to detect risky driving behavior, and the Fitness tracker app to track heart rate, weight, daily steps, and calories burned.

**What this book covers**

Chapter 1, *Sensor Fundamentals*, provides you a thorough understanding of the fundamentals and framework of Android sensors. It walks you through the different types of sensors and the sensor coordinate system in detail.

Chapter 2, *Playing with Sensors*, guides you through various classes, callbacks, and APIs of the Android Sensor framework. It walks you through a sample application, which provides a list of available sensors and their values and individual capabilities, such as the range of values, power consumption, minimum time interval, and so on.

Chapter 3, *The Environmental Sensors – The Weather Utility App*, explains the usage of various environment sensors. We develop a weather utility app to compute altitude, absolute humidity, and dew point using temperature, pressure, and relative humidity sensors.

Chapter 4, *The Light and Proximity Sensors*, teaches you how to use proximity and light sensors. It explains the difference between wakeup and non-wakeup sensors and explains the concept of the hardware FIFO sensor queue. As a learning exercise, we develop a small application that turns on/off a flashlight using a proximity sensor, and it also adjusts screen brightness using a light sensor.
Chapter 5, *The Motion, Position, and Fingerprint Sensors*, explains the working principle of motion sensors (accelerometer, gyroscope, linear acceleration, gravity, and significant motion), position sensors (magnetometer and orientation), and the fingerprint sensor. We learn the implementation of these sensors with the help of three examples. The first example explains how to use the accelerometer sensor to detect phone shake. The second example teaches how to use the orientation, magnetometer, and accelerometer sensors to build a compass, and in the third example, we learn how to use the fingerprint sensor to authenticate a user.

Chapter 6, *The Step Counter and Detector Sensors – The Pedometer App*, explains how to use the step detector and step counter sensors. Through a real-world pedometer application, we learn how to analyze and process the accelerometer and step detector sensor data to develop an algorithm for detecting the type of step (walking, jogging, sprinting). We also look at how to drive the pedometer data matrix (total steps, distance, duration, average speed, average step frequency, calories burned, and type of step) from the sensor data.

Chapter 7, *The Google Fit Platform and APIs – The Fitness Tracker App*, introduces you to the new Google Fit platform. It walks you through the different APIs provided by the Google Fit platform and explains how to request automated collection and storage of sensor data in a battery-efficient manner without the app being alive in the background all the time. As a learning exercise, we develop a fitness tracker application that collects and processes the fitness sensor data, including the sensor data obtained from remotely connected Android Wear devices.

*Bonus Chapter, Sensor Fusion and Sensor-Based APIs (the Driving Events Detection App)*, guides you through the working principle of sensor-based Android APIs (activity recognition, geo-fence, and fused location) and teaches you various aspects of sensor fusion. Through a real-world application, you will learn how to use multiple sensors along with input from sensor-based APIs to detect risky driving behavior. Through the same application, you will also learn how to develop the infrastructure (service, threads, and database) required to process high volumes of sensor data in the background for a longer duration of time. This chapter is available online at the link https://www.packtpub.com/sites/default/files/downloads/SensorFusionandSensorBasedAPIs_TheDrivingEventDetectionApp_OnlineChapter.pdf
In this chapter, we will learn about proximity and light sensors, and we will develop a small application using them. We will also learn about the concepts of wake locks and wakeup and non-wakeup sensors. We will understand the hardware sensor FIFO queue and what happens to sensors when the application processor goes into suspended mode.

You will learn the following things in this chapter:

- Understanding the light and proximity sensors.
- Understanding requirements for the automatic torchlight and screen brightness app.
- How to use the proximity sensor in the phone and turn phone's flashlight off and on.
- How to use the phone's light sensor and adjust the screen brightness.
- What are wake locks and how should we use them?
- What are wakeup and non-wakeup sensors, and what is the hardware FIFO sensor queue?
Understanding the light and proximity sensors

The light sensor is a part of the environmental sensors, and the proximity sensor is a part of the positional sensors for Android. Both light and proximity sensors can be found in almost every Android device today. The light sensor is responsible for measuring the illuminance in lux. Illuminance is the amount of light striking a surface. It is also known as incident light, where the “incident” is the beam of light actually landing on the surface.

The proximity sensor measures the proximity of any object near the screen. There are two types of proximity sensor found on Android devices.

- The first type of the proximity sensor provides the absolute distance in centimetres between the object and the phone. There are very few phones which support this type of proximity sensor.
- The second type of sensor gives only two values in form of integers, which represents the proximity of the object from the phone. When the object is near, then the value is 0, while when the object is far, then the value is the maximum range of the proximity sensor, which generally lies between 5 to 20 centimeters. For example, the Samsung Galaxy S5 has the far value of 8, while the LG Nexus 5 has the far value of 5.
Most Android devices have the same hardware, which works for both light and proximity sensors, but Android still treats them as logically separate sensors. It depends on whether the individual OEMs (Original Equipment Manufactures) have a single hardware or two separate hardware to support both the logical sensors.

The light and proximity sensor is generally located on the top right-hand section of the phone, a few millimetres to the right of the earpiece. You have to look very carefully to spot the sensor as it's barely visible because of its small size. Generally, it's a pair of small black holes covered under the screen. Some OEMs might choose a different location for the light and proximity sensor, but mostly it will be on the top edge of the phone. For example, Samsung Galaxy S devices have them on the right-hand side of the earpiece, while HTC Android devices have them on the left-hand side of the earpiece.

### The automatic torch light and screen brightness app requirements

As a learning assignment for this chapter, we will be developing a small application that will make use of the light and proximity sensor in the phone to turn on and turn off the flash light and adjust the screen brightness. This app will be running in the foreground Android activity and will start processing the sensor values on `onResume()` and will stop on `onPause()`. We will have the separate activity for each proximity sensor and light sensor, and both will work independently. The following are the high-level requirements for the automatic torch light application:

1. Create an Android activity to process the proximity sensor values.
2. Whenever any object comes close to the phone (the proximity sensor gives the near value), turn on the flashlight, and whenever that object goes away from the phone (the proximity sensor gives the far value), then turn off the flashlight.
3. Create an Android activity to process the light sensor values.
4. Whenever the phone enters any dark area, increase the screen brightness of the phone, and when the phone goes back to a lighted area, decrease the screen brightness.
Time for action – turning the torch light on and off using the proximity sensor

In this section, we will be learning how to use the proximity sensor to turn the camera flash light on and off. As discussed earlier, most proximity sensors return the absolute distance in cm, but some return only the near and far values. The near value is 0 and the far value is the maximum range of the proximity sensor. There are a lot of common use cases for proximity sensors, such as to turn off the screen while the user is on a call, or to detect if the phone is inside the pocket or outside. For our example, we will be turning the camera flashlight on whenever any object comes near the phone screen and turning it off when the object goes far from the phone screen. The proximity sensor has on-change reporting mode, the details of reporting modes are explained in Chapter 1, Sensor Fundamentals. It is fired as soon as the proximity of the object near the phone changes.

The following code shows how to use the proximity sensor to turn the camera flash light on or off.

1. We created a ProximitySensorActivity and followed the standard steps to get values from the sensor, which are implementing the SensorEventListener interface, initiating the SensorManager and Sensor Objects, and checking the availability of the sensors. We also declared the Camera, SurfaceTexture, and Parameters objects required for the camera flashlight to work. We also called the custom initCameraFlashlight() method from onCreate() to initialize the required camera objects:

```java
public class ProximitySensorActivity extends Activity implements SensorEventListener{

    private SensorManager mSensorManager;
    private Sensor mSensor;
    private boolean isSensorPresent;
    private float distanceFromPhone;
    private Camera mCamera;
    private SurfaceTexture mPreviewTexture;
    private Camera.Parameters mParameters;
    private boolean isFlashLightOn = false;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.proximitysensor_layout);
        mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        if(mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
!= null) {
    mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
    isSensorPresent = true;
} else {
    isSensorPresent = false;
}

initCameraFlashLight();
}

2. As a best practice, we registered the listener in the onResume() method and unregistered it in the onPause() method. Inside the custom initCameraFlashlight() method, we initialized the Camera, SurfaceTexture, and Parameters objects required for turning on the flashlight. In the onDestroy() method of the activity, we released the Camera object and set all the initialized object references to null:

@Override
protected void onResume() {
    super.onResume();
    if(isSensorPresent) {
        mSensorManager.registerListener(this, mSensor,
        SensorManager.SENSOR_DELAY_NORMAL);
    }
}

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

public void initCameraFlashLight() {
    mCamera = Camera.open();
    mParameters = mCamera.getParameters();
    mPreviewTexture = new SurfaceTexture(0);
    try {
        mCamera.setPreviewTexture(mPreviewTexture);
    } catch (IOException ex) {
        Log.e(TAG, ex.getLocalizedMessage());
        Toast.makeText(getApplicationContext(), getResources().getText(R.string.error_message), Toast.LENGTH_SHORT).show();
    }
}
After initiating SurfaceTexture, camera, and sensors, we will now write our core logic for the app. In our custom turnTorchLightOn() method, we start the flash light by setting the flash mode to FLASH_MODE_TORCH for the camera parameters and starting the camera preview. Similarly, in the custom turnTorchLightOff() method, we stop the flash light by setting the flash mode to FLASH_MODE_OFF for the camera parameters and stopping the camera preview. Now, we call these methods from the onSensorChanged() method, depending on the distance of any object from the proximity sensor. If the distance of any object from the phone’s proximity sensor is less than the maximum range of the proximity sensor, then we consider the object to be near and call the custom turnTorchLightOn() method; however, if the distance is equal to or greater than the range of the proximity sensor, we consider the object is far and call the turnTorchLightOff() method. We use the isFlashLightOn Boolean variable to maintain the on/off states of the flashlight:

```java
public void turnTorchLightOn() {
    mParameters.setFlashMode(Camera.Parameters.FLASH_MODE_TORCH);
    mCamera.setParameters(mParameters);
    mCamera.startPreview();
    isFlashLightOn = true;
}

public void turnTorchLightOff() {
    mParameters.setFlashMode(mParameters.FLASH_MODE_OFF);
    mCamera.setParameters(mParameters);
    mCamera.stopPreview();
    isFlashLightOn = false;
}

public void onSensorChanged(SensorEvent event) {
    distanceFromPhone = event.values[0];
```
if (distanceFromPhone < mSensor.getMaximumRange()) {
    if (!isFlashLightOn) {
        turnTorchLightOn();
    }
} else {
    if (isFlashLightOn) {
        turnTorchLightOff();
    }
}

What just happened?

We used the standard steps for getting values from the proximity sensor and then used these values to turn on and off the camera flashlight in the phone. If any object comes within the range of the proximity sensor, it will turn on the flashlight; when the object goes away from its range, it will turn off the flashlight. You can determine a sensor's maximum range by using the `getMaximumRange()` method on the proximity sensor object.

Time for action – adjusting the screen brightness using the light sensor

One of the most common use cases for the light sensor is to adjust the screen brightness according to the external lighting conditions. The maximum range of the light sensor might be different on different Android devices, but most of them support from 0 lux to several thousand lux. Lux is the standard unit for measuring the luminance of the light falling on a surface. For our example, we will use a range from 0 to 100 lux, as normal indoor lighting falls within this range. But for sunlight and strong lights the range can go up to 1,000 lux or more. In the sample app, we will increase the screen brightness, when the indoor lighting goes low, and similarly we will decrease the screen brightness when it goes high.

1. We followed the standard steps to get values from the sensor. We select the sensor type to the `TYPE_LIGHT` in the `getDefaultSensor()` method of `SensorManager`. We also called the custom `initScreenBrightness()` method from `onCreate()` to initialize the required content resolver and current window objects:

```java
public class LightSensorActivity extends Activity implements SensorEventListener{
```
private SensorManager mSensorManager;
private Sensor mSensor;
private boolean isSensorPresent;
private ContentResolver mContentResolver;
private Window mWindow;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.lightsensor_layout);

mSensorManager = (SensorManager) this.getSystemService (Context.SENSOR_SERVICE);
if(mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT) != null) {
   mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
   isSensorPresent = true;
} else {
   isSensorPresent = false;
}
initScreenBrightness();
}

2. As a standard practice, we registered the listener in the onResume() method and un-registered it in the onPause() method. Inside the custom initScreenBrightness() method, we initialized the ContentResolver and Window objects. The ContentResolver provides a handle to the system settings and the Window object provides the access to the current visible window. In the onDestroy() method of the activity, we change all the initialized objects references to null:

@Override
protected void onResume() {
super.onResume();
if(isSensorPresent) {
   mSensorManager.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_NORMAL);
}
}

@Override
protected void onPause() {
super.onPause();
if(isSensorPresent) {
   mSensorManager.unregisterListener(this);
}
public void initScreenBrightness()
{
  mContentResolver = getContentResolver();
  mWindow = getWindow();
}

@Override
protected void onDestroy() {
  super.onDestroy();

  mSensorManager = null;
  mSensor = null;
  mContentResolver = null;
  mWindow = null;
}

3. As discussed earlier, we will use a light range from 0 to 100 lux for our example. We will be adjusting the brightness for two objects: one for the current visible window (for which the brightness value lies between 0 and 1), and the second for the system preference (for which the brightness value lies between 0 and 255). In order to use the common brightness value for both the objects, we will stick to a value between 0 and 1, and for system brightness we will scale up by multiplying it by 255. Since we have to increase the screen brightness, as the outside lightening goes low and vice versa, we take the inverse of the light sensor values. Also to keep the range of the brightness value between 0 and 1, we use only light values between 0 and 100. We pass on the inverse of light values obtained from the light sensor in the onSensorChanged() method, as an argument to our custom changeScreenBrightness() method to update the current window and system screen brightness:

    public void changeScreenBrightness(float brightness)
    {
      //system setting brightness values ranges between 0-255
      //We scale up by multiplying by 255
      //This change brightness for over all system settings
      System.putInt(mContentResolver, System.SCREEN_BRIGHTNESS, (int) (brightness*255));
      //screen brightness values ranges between 0 - 1
      //This only changes brightness for the current window
      LayoutParams mLayoutParams = mWindow.getAttributes();
      mLayoutParams.screenBrightness = brightness;
      mWindow.setAttributes(mLayoutParams);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {

float light = event.values[0];
// We only use light sensor value between 0 - 100
// Before sending, we take the inverse of the value
// So that they remain in range of 0 - 1
if(light>0 && light<100) {
    changeScreenBrightness(1/light);
}
}

The app needs to have following three permissions to run the previous two examples:

- `<uses-permission android:name="android.permission.CAMERA" />
- `<uses-permission android:name="android.permission.FLASHLIGHT" />
- `<uses-permission android:name="android.permission.WRITE_SETTINGS" />

The camera permission is required to access the camera object, flashlight permission is required to turn on and turn off the flashlight, and the write settings permission is required to change any system settings.

**What just happened?**

We used light luminance values in lux (coming from the light sensor) to adjust the screen brightness. When it is very dark (almost no light), then the light sensor provides very low sensor values. When we send this low light sensor value (the minimum possible value being 1) to the `changeScreenBrightness()` method, then it makes the screen the brightest by taking the inverse (which is again 1) of the light sensor value and scaling up by multiplying it by 255 (1 * 255 = 255 brightness value). Similarly, under good lighting conditions, when we send a high sensor value (the maximum possible value being 99 in our case), then it makes the screen as dim as possible by taking the inverse (1/99=0.01) of the light sensor value and scaling up by multiplying it by 255 (0.01 * 255 = 2.55 brightness value). One of the easiest ways to test this app is to cover the light sensor with your hand or any opaque object. By doing this, you will observe that when you cover the light sensor, the screen becomes bright, and when you remove the cover, it becomes dim.
Wake locks, wakeup sensors, and the FIFO queue

All Android applications run on a dedicated Application Processor (AP), which is a part of the main CPU of the phone. This application processor is designed in such a way that it goes into the suspended mode when the user is not interacting with the phone. In this suspended mode, it reduces the power consumption by 10 times or more, but this freezes all the applications in the background. To work around this problem, the Android platform provides a solution using wake locks. If an application has to perform some important operation in the background and doesn't want the application processor to go into suspended mode, then it has to request a wake lock from the system’s power service. Once the important operation is completed, it should release the wake lock. Wake lock can be obtained using the PowerManager object, which is provided by the system power service. The newWakeLock() method of PowerManager provides the object of wake lock. This newWakeLock() method accepts the type of wake lock and string tag for identification purposes. Once the wake lock object is instantiated, we need to call the acquire() method of the wake lock object to make it active and the release() method to make it inactive.

The following code snippet shows how to acquire and release a wake lock:

```java
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
    mWakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "myLock");
    mWakeLock.acquire();
    // Do some important work in background.
    mWakeLock.release();
```

Wakeup and non-wakeup sensors

Android categorizes sensors into two types: wakeup and non-wakeup sensors. If the application processor has gone into suspended mode, then the wake up sensor will wake up the application processor from its suspended mode to report the sensor events to the application. The wakeup sensors ensure that their data is delivered independently of the state of the AP and before the maximum reporting latency has elapsed. Non-wakeup sensors do not wake up the AP out of its suspended mode to report data. If the AP is in suspended mode, the sensor events for the non-wakeup sensors are still collected in their hardware FIFO queue, but they are not delivered to the application. From the API level 21 (Lollipop) onward, Android platforms support the isWakeUpSensor() method to check the type of sensor.
The sensor's hardware FIFO queue

Non-wakeup sensors might have their own hardware FIFO queue. This FIFO (First In First Out) queue is used to store the sensor events while the AP is in suspended mode. These sensor events stored in the FIFO queue are delivered to the application when the AP wakes up. If the FIFO queue is too small to store all the events generated, then the older events are dropped to accommodate the newer ones. Some sensors might not have this FIFO queue at all. We can easily check the existence and size of the FIFO queue by using the `maxFifoEventCount()` method on the sensor object. If the value from this method comes to zero, it means that the FIFO count for that sensor doesn't exit.

If your application is using non-wakeup sensors in the background and performing a critical operation, and you don't want the AP to go into suspended mode, then you should use wake lock. But make sure you release the wake lock after the critical operation is done. Wake lock doesn't allow the application processor to go into suspended mode, but it also increases the power consumption. So, we have to make sure we release the wake lock after the critical operation is done; otherwise it will keep on draining the battery.

Summary

In this chapter, we looked at the two new proximity and light sensors and developed a small app using them. We also learned how to turn on and turn off the flashlight using the proximity sensor and adjust the screen brightness using the light sensor. We understood how to wake up the application processor when it's in suspended mode using wake locks. We looked at the wakeup and non-wake up sensors and their FIFO queues.

In the next chapter, we will learn about motion sensors (accelerometer, gyroscope, linear acceleration, gravity, and significant motion) and position sensors (magnetometer and orientation). We will also explore the newly introduced fingerprint sensor.
Where to buy this book

You can buy Android Sensor Programming By Example from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.