Introducing the Sensor Manager

The Sensor Manager is used to manage the sensor hardware available on an Android device. Use getSystemService to get a reference to the Sensor Service as shown in the code snippet below:

```
String service_name = Context.SENSOR_SERVICE;
SensorManager sensorManager = (SensorManager)getSystemService(service_name);
```

The following sections look closely at how to use the Sensor Manager to monitor orientation and acceleration, but the pattern shown here can be used to monitor sensor results from any available hardware sensor:

```
SensorListener mySensorListener = new SensorListener() {
public void onSensorChanged(int sensor, float[] values) {
// TODO Deal with sensor value changes
}
public void onAccuracyChanged(int sensor, int accuracy) {
// TODO Auto-generated method stub
}
};
```

The SensorListener interface is used to listen for Sensor value and accuracy changes. Implement the onSensorChanged method to react to value changes. The sensor parameter identifi es the sensor that triggered the event, while the values float array contains the new values detected by that sensor.

Implement onAccuracyChanged to react to changes in a sensor's accuracy. The sensor parameter again identifi es the sensor that triggered the event, while the accuracy parameter indicates the new accuracy of that sensor using one of the constants:

- SensorManager.SENSOR_STATUS_ACCURACY_HIGH Indicates that the sensor is reporting with the highest possible accuracy.
- SensorManager.SENSOR_STATUS_ACCURACY_LOW Indicates that the sensor is reporting with low accuracy and needs to be calibrated.
- SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM Indicates that the sensor data is of average accuracy, and that calibration might improve the readings.
- SensorManager.SENSOR_STATUS_UNRELIABLE Indicates that the sensor data is unreliable, meaning that either calibration is required or readings are not currently possible.

The Sensor Manager includes constants to help identify the sensor triggering the change event. The following list includes the sensors for which constants are currently defi ned. Some or all of these sensors will be available to your applications depending on the hardware available on the host device:

- SensorManager.SENSOR_ACCELEROMETER Is an accelerometer sensor that returns the current acceleration along three axes in meters per second squared (m/s^2). The accelerometer is explored in greater detail later in this chapter.
- SensorManager.SENSOR_ORIENTATION Is an orientation sensor that returns the current orientation on three axes in degrees. The orientation sensor is explored in greater detail later in this chapter.
- SensorManager.SENSOR_LIGHT Is an ambient-light sensor that returns a single value describing the ambient illumination in lux.
- SensorManager.SENSOR_MAGNETIC_FIELD Is a sensor used to determine the current magnetic field measured in microteslas (⌠T) along three axes.
- SensorManager.SENSOR_PROXIMITY Is a proximity sensor that returns a single value describing the distance between the device and the target object in meters (m).
- SensorManager.SENSOR_TEMPERATURE Is a thermometer sensor that returns the ambient temperature in degrees Celsius (°C).

To receive notifi cations of changes from a particular sensor, create a Sensor Listener as described previously, and register it with the Sensor Manager specifying the sensor that should trigger the Listener and the rate at which the sensor should update, as shown in the following code snippet:

```
sensorManager.registerListener(mySensorListener,
SensorManager.SENSOR_TRICORDER,
SensorManager.SENSOR_DELAY_FASTEST);
```

The Sensor Manager includes the following constants (shown in descending order of responsiveness) to let you select a suitable update rate:

- SensorManager.SENSOR_DELAY_FASTEST Specifi es the fastest possible sensor update rate.
- SensorManager.SENSOR_DELAY_GAME Selects an update rate suitable for use in controlling games.
- SensorManager.SENSOR_DELAY_NORMAL Specifi es the default update rate.
- SensorManager.SENSOR_DELAY_UI Specifi es a rate suitable for updating UI features.

The rate you select is not binding; the Sensor Manager may return results faster or slower than you specify, though it will tend to be faster. To minimize the associated resource cost of using the sensor in your application you should try to select the slowest suitable rate.

An overloaded registerListener method is also available that applies the default (SENSOR_DELAY_ NORMAL) update rate, as shown below:

```
sensorManager.registerListener(mySensorListener,
SensorManager.SENSOR_TRICORDER);
```