eSense Android Library Documentation

Installation and Usage

Author	Pervasive Systems
Organization	Nokia Bell Labs, Cambridge
Last Update	May 2, 2019

Contents

1 Overview 3

2 Getting Started 4

2.1 What We Tested: Device, Android, API, IDE 4

2.2 Create the Android project 4

2.3 Import the library 4

2.3.1 Source code 5

2.3.2 AAR library 5

2.4 Run the application 5

3 Use eSense Library 6

3.1 Import the library in your Android project 6

3.2 Create the `ESenseManager` instance 6

3.3 Connect the eSense device 7

3.4 Obtain the sensor data 7

3.5 Change the configuration of the eSense device & Read device events 8

1 Overview

This document describes how to use the eSense library on the Android platform. The eSense library provides a set of intuitive APIs to interact with the eSense device. It can be seen as a wrapper to access and operate the eSense device via Bluetooth Low Energy (BLE), which is described in the **eSense BLE Specifications** document.

Please note that playing and recording audio are performed via the Bluetooth Classic interface and are not supported by the eSense library described in this document.

The eSense library is implemented using asynchronous programming and non-blocking execution with simple event-based APIs. `ESenseManager` is your starting point for all eSense actions. Once you have the manager instance, you can manage the connection with eSense, change the eSense configuration, and read data from eSense via the listener interface. The library provides three types of the listener interfaces, `ESenseConnectionListener`, `ESenseEventListener`, and `ESenseSensorListener`.

- `ESenseConnectionListener` is used to receive notifications when the connection status has changed.
- `ESenseEventListener` is used to receive notifications when there is a new eSense event.
- `ESenseSensorListener` is used to receive notifications when there is new sensor data.

The detailed information about APIs can be found in the JavaDoc file provided.

2 Getting Started

2.1 What We Tested: Device, Android, API, IDE

The minimum SDK version of the library is set to Android 6.0 (API 23) and we tested the eSense library in the following environments using Android Studio.

Device model	Android version	API level
Motorola Nexus 6	Android 7.1.1	API 25
Pixel 2	Android 8.0.0	API 26
Pixel 3	Android 9.0.0	API 28

We expect that the library would operate on other environments, especially on the Android reference phones or maybe even on earlier than API 23, but do not guarantee it.

2.2 Create the Android project

1. Create a new Android project
2. Make sure that your project's *minSdkVersion* is at API 23 or higher.
3. Add the following permission and feature declarations to your AndroidManifest.xml (inside the `manifest` tag)

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
```

2.3 Import the library

The eSense library is provided with two forms, the source code and Android Archive (AAR). Choose one of the two depending on your preference.

2.3.1 Source code

1. Create the package folder under your Android project and name it to `io.esense.esenselib`.
2. Put the java files in the zip file under the package.

2.3.2 AAR library

1. Put the `esense-lib.aar` file into the 'libs' folder on your Android project.
2. Add the following implementation declaration under `dependencies` in `build.gradle` for the application module.

```
implementation 'io.esense.esenselib:esense-lib:1.0@aar'
```

3. Add the repository declaration in the same `build.gradle`.

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

2.4 Run the application

1. Rebuild the project.
2. The first time the app starts make sure to allow it to access the phone location. This is necessary to use the BLE on Android.
    a. Alternatively, enable the Location permission in the App info panel accessible from the Apps & notifications section in Settings.

3 Use eSense Library

3.1 Import the library in your Android project

```
import io.esense.esenselib.*;
```

3.2 Create the `ESenseManager` instance

```
manager = new ESenseManager(name, MainActivity.this.getApplicationContext(),
eSenseConnectionListener);
```

Note that 'name' should be set to the name of the eSense device you are using.

To create the `ESenseManager`, you must implement the `ESenseConnectionListener` Interface. The `ESenseconnectionListener` interface exposes methods to receive notifications from `ESenseManager` when the connection status has changed.

The `ESenseConnectionListener` interface provides the following abstract methods.

Modifier and Type	Method and Description
void	**`onConnected`**`(`**`ESenseManager`** `manager)` Called when the connection has been successfully made
void	**`onDeviceFound`**`(`**`ESenseManager`** `manager)` Called when the device with the specified name has been found during a scan
void	**`onDeviceNotFound`**`(`**`ESenseManager`** `manager)` Called when the device with the specified name has not been found during a scan
void	**`onDisconnected`**`(`**`ESenseManager`** `manager)` Called when the device has been disconnected

3.3 Connect the eSense device

Connect the eSense device using `connect()`. You can also specify the connection timeout by using `connect(int)`. Once the onConnected() event has been received it is possible to interact with the eSense device, for example reading sensor data or changing its configuration.

Always make sure to disconnect the device when you don't need it anymore. Failing to do so can drain the battery significantly.

3.4 Obtain the sensor data

Register a sensor listener and specify the sampling rate at which to receive the sensor data.

```
manager.registerSensorListener(eSenseSensorListener, 100);
```

Note that the sampling rate is only a hint to the system. Sensor events may be received faster or slower than the specified rate, depending on the Bluetooth communication status and parameter values (refer to the configuration of the BLE connection interval in the eSense BLE Specification for more details).

To obtain the sensor data, you must implement the `ESenseSensorListener` interface. The methods in `ESenseSensorListener` are called when there is new sensor data. The `ESenseSensorListener` interface provides the following abstract method.

Modifier and Type	Method and Description
void	**`onSensorChanged`**`(`**`ESenseEvent`**` evt)` Called when there is new sensor data available

The `ESenseEvent` class represents an eSense event and holds information such as packet index, timestamp and the sensor's values. Since eSense does not keep real-time information, the timestamp in the event object is added by the receiving phone.

You can read the accelerometer and gyroscope data using `getAccel()` and `getGyro()` methods in the `ESenseEvent` class, respectively. These methods return the ADC values. You can also obtain the converted values by using `convertAccToG()` for acceleration in g and `convertGyroToDegPerSecond()` for rotational speed in degrees/second. For the conversion,

you need to read the configuration from the eSense device regarding accelerometer/gyroscope range and pass it to the `convertAccToG()` and `convertGyroToDegPerSecond()` methods.

Always make sure to unregister the sensor listener when you don't need it anymore. Failing to do so can drain the battery significantly.

3.5 Change the configuration of the eSense device & Read device events

1. Configuration of the eSense device

The `ESenseManager` exposes methods to change the configuration of the eSense device. With the library, you can change the device name using `setDeviceName()`, change the advertising and connection interval using `setAdvertisementAndConnectiontInterval()`, and change the IMU sensor configuration using `setSensorConfig()`. You can also get the corresponding information using `getDeviceName()`, `getAdvertisementAndConnectionInterval()`, and `getSensorConfig()`, respectively.

For more information about advertising and connection interval. Please refer to the **eSense-BLE-Specification** document.

As for the sensor config, the library allows to change the full scale range and low pass filter configuration of accelerometer and gyroscope. The possible options are specified in the following enum classes: `ESenseConfig.AccRange`, `ESenseConfig.AccLPF`, `ESenseConfig.GyroRange`, and `ESEnseConfig.GYROLPF`.

To use `convertAccToG()` and `convertGyroToDegPerSecond()` for the conversion of ADC values, obtain the `ESenseConfig` instance by using `getSensorConfig()` and pass it to the conversion methods.

2. Device events (battery voltage, factory accelerometer offset, and button events)

The eSense library allows applications to read various device events including battery voltage, factory accelerometer offset, and button events. Use `getBatteryVoltage()` and `getAccelerometerOffset()` to read the battery voltage and factory accelerometer offset,

respectively. The information of the button events does not require any method to be called and is delivered via the `ESenseEvenetListener` automatically when the button event has changed.

3. Register and Implement ESenseEventListener

Register an event listener to the `ESenseManager` instance.

```
manager.registerEventListener(eSenseEventListener);
```

To obtain the event result such as the advertising/connection interval and the sensor config, you must implement the `ESenseEventListener` interface. The `ESenseEventListener` interface provides the following abstract methods.

Modifier and Type	Method and Description
void	**`onAdvertisementAndConnectionIntervalRead`**`(int minAdvertisementInterval, int maxAdvertisementInterval, int minConnectionInterval, int maxConnectionInterval)` Called when the information on advertisement and connection interval has been received
void	**`onBatteryRead`**`(double voltage)` Called when the information on battery voltage has been received
void	**`onButtonEventChanged`**`(boolean pressed)` Called when the button event has changed
void	**`onSensorConfigRead`**`(ESenseConfig config)` Called when the information on sensor configuration has been received
void	**`onAccelerometerOffsetRead(int`** `offsetX, int offsetY, int offsetZ)` Called when the information on accelerometer offset has been received