Remote Sensor

Programming Guide

Version 1.0

Revision History

Version	Date	Description
0.1	2014-01-20	Initial draft
0.2	2014-01-21	Modified class name (prefix)
0.3	2014-01-22	Modified sample code
0.4	2014-01-27	Modified picture
0.5	2014-01-28	Added comments
0,6	2014-02-06	Modified sample source and permission
0.7	2014-02-11	Added new sensor types (TYPE_HEART_RATE, TYPE_WEARING_STATE)
0.8	2014-02-17	Added feature types (TYPE_GEAR_MANAGER, TYPE_GEAR_FIT_MANAGER, TYPE_REMOTE_SENSOR_SERVICE) for the isFeatureEnabled() method
0.9	2014-02-20	Added notes for registerListener() and req-ustTriggerSensor() method
1.0	2014-02-24	Removed TYPE_HEART_RATE sensor

Table of Contents

1. Overview

Remote Sensor allows you to retrieve user activity, pedometer (step counter) and wearing state data from a wearable device to use in your application. The host device on which your application is running requests the data, and the wearable device transmits it.

You can use Remote Sensor to:

- Get user activity data from the wearable device.

  When the user of a wearable device starts running or walking, the application on the host device can be notified.

- Get pedometer data from the wearable device.

  The application on the host device can get the user's step count.

- Get the state whether the user wears wearable device or not

  The application on the host device can get the user's wearing state of wearable device.

1.1. Architecture

The following figure shows the Remote Sensor architecture.

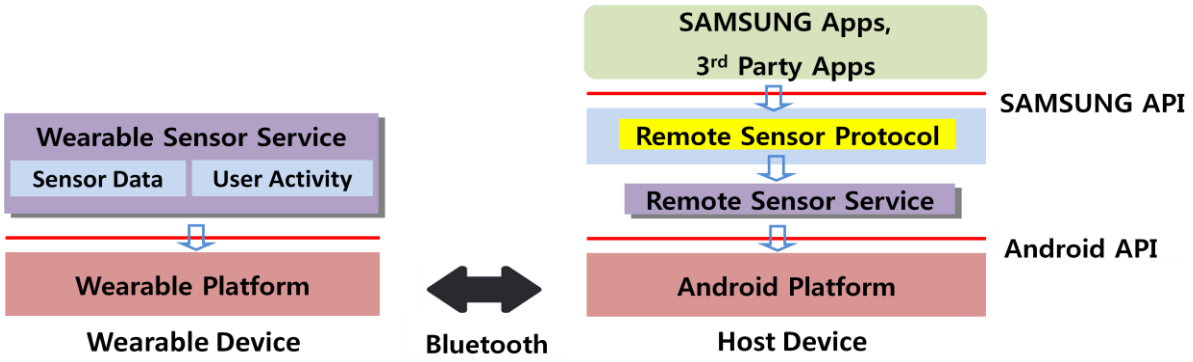

Figure 1: Remote Sensor architecture

The architecture consists of:

- **Applications:** One or more applications that use Remote Sensor.
- **Remote Sensor Protocol**: Components for accessing the wearable device sensor data.
- **Remote Sensor Service**: Components for handling channel mixing, and the data cache for applications on the host device. Depending on the type of the wearable device, either the enhanced Samsung Accessory Protocol (eSAP) or the Wearable Communication Protocol provides the Bluetooth communication.

- **Wearable Sensor Service**: Components for handling Bluetooth connectivity and channel management.

1.2. Class Diagram

The following figure shows the Remote Sensor classes and interfaces that you can use in your application.

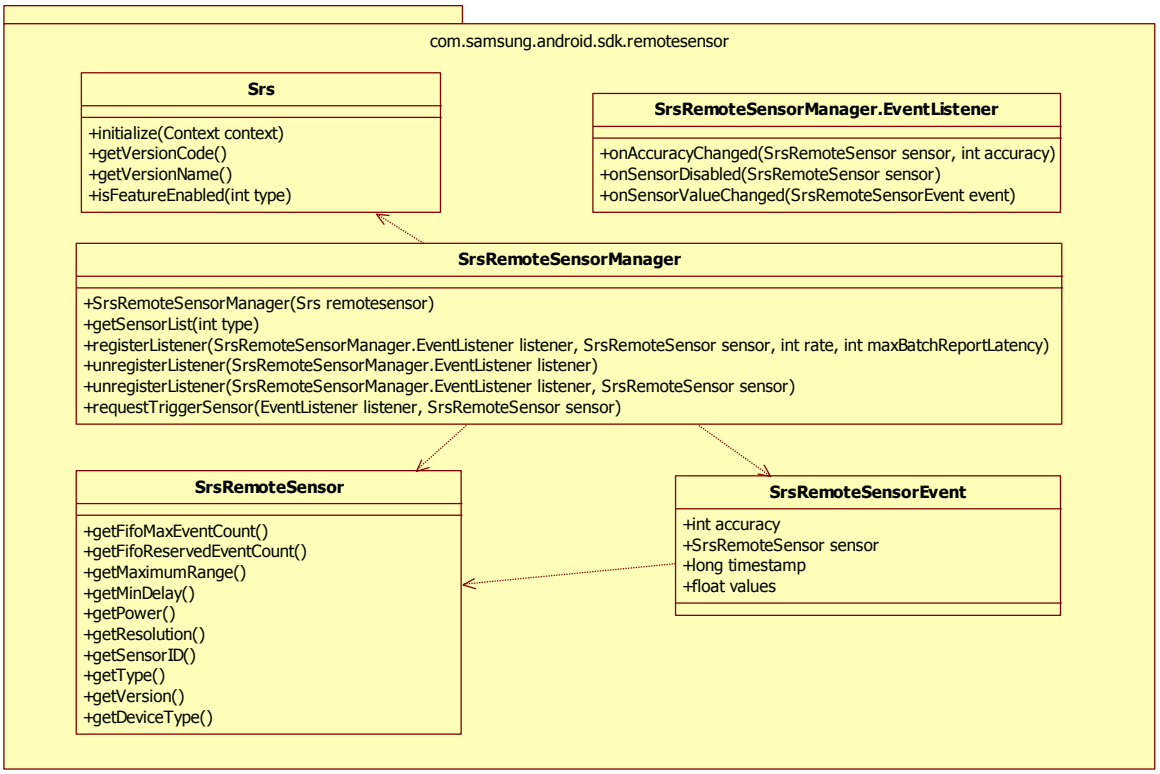

Figure 2: Remote Sensor classes and interfaces

The Remote Sensor classes and interfaces include:

- **Srs:** Initializes the Remote Sensor package.
- **SrsRemoteSensorManager:** Gets the sensor list, registers or unregisters event listeners, and receives events.
- **SrsRemoteSensor:** Describes the attributes for each sensor type.
- **SreRemoteSensorEvent:** Contains the sensor data, including the timestamp.
- **SrsRemoteSensorManager.EventListener:** Listens for accuracy changes, value changes, or stopped events.

1.3. Supported Platforms

- Android 4.3 (Jelly Bean API 18) or above supports Remote Sensor.

1.4. Supported Features

Remote Sensor supports the following features:

- Getting user activity data from the wearable device

  When the user of a wearable device starts running or walking, the application on the host device can be notified.

- Getting pedometer data from the wearable device

  The application on the host device can get the user's step count.

- Get the state whether the user wears wearable device or not

  The application on the host device can get the user's wearing state of wearable device.

1.5. Components

- Components
  - remotesensor-v1.0.0.jar
- Imported packages:
  - com.samsung.android.sdk.remotesensor

1.6. Installing the Package for Eclipse

To install Remote Sensor for Eclipse:

1. Add the following files file to the libs folder in Eclipse:

   - remotesensor-v1.0.0.jar
     This is the remote sensor package.
   - accessory-v1.0.0.jar
     This is the Bluetooth communication package used by the Remote Sensor Service.
   - sdk-v1.0.0.jar
     This is the Samsung Mobile SDK package.

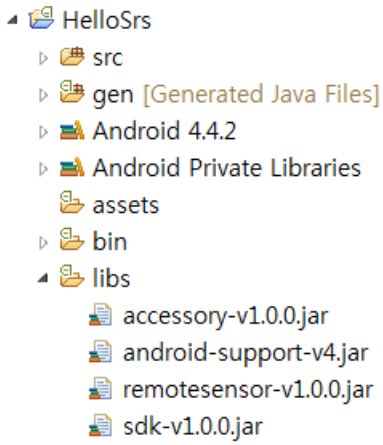

Figure 3: libs folder in Eclipse

2. Add the following permissions to your Android manifest file:

```
<uses-permission
    android:name="com.samsung.android.sdk.permission.REMOTE_SENSOR_SERVICE">
</uses-permission>
```

2. Hello Srs

Hello Remote Sensor is a simple program that:

1. Creates Srs and SrsRemoteSensorManager instances.
2. Implements, registers, and starts an SrsRemoteSensorManager.EventListener instance.
3. Handles Srs events in the `onSensorValueChanged()` method.

```
public class RemoteSensorActivity extends Activity implements EventListener {

	SrsRemoteSensorManager mServiceManager = null;
	TextView activitySensorText;
	TextView activityValueText;

	TextView pedoSensorText;
	TextView pedoValueText;

	List<SrsRemoteSensor> activitySensorList;
	List<SrsRemoteSensor> pedoSensorList;
	Srs remoteSensor = null;

	SrsRemoteSensor userActivitySensor = null;
	SrsRemoteSensor pedometerSensor = null;

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

		remotesensor = new Srs();
		try {
			remotesensor.initialize(this.getApplicationContext());
		} catch (SsdkUnsupportedException e) {
			switch (e.getType ()) {
			    case SsdkUnsupportedException.LIBRARY_NOT_INSTALLED:
				// Handle the exception
				break;
			    case SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED:
				// Handle the exception
				break;
			    default:
				// Handle the exception
				break;
			}
		}

		mSensorManager = new SrsRemoteSensorManager(remotesensor);
	}

	public void getPedometerSensorInfo (View v){

		pedoSensorList =
		    mServiceManager.getSensorList(SrsRemoteSensor.TYPE_PEDOMETER);
		SrsRemoteSensor sensor;
		sensor = pedoSensorList.get(0);
```

```
            pedoSensorText.setText(sensor.toString());
      }

      public void getPedometerEvent (View view){

            pedometerSensor = pedoSensorList.get(0);
            mServiceManager.registerListener(this, pedometerSensor,
                              SrsRemoteSensorManager.SENSOR_DELAY_NORMAL, 0);
      }

      public void stopPedometerEvent(View view){
            SrsRemoteSensor     sensor;
            sensor = pedoSensorList.get(0);
            mServiceManager.unregisterListener(this, sensor);
      }

      @Override
      protected void onResume() {
            super.onResume();
            mSensorManager.registerListener(this, pedometerSensor,
                              SrsRemoteSensorManager.SENSOR_DELAY_NORMAL, 0);
       }

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

      // Called when the accuracy of a sensor has changed.
      @Override
      public void onAccuracyChanged(SrsRemoteSensor sensor, int accuracy){

      }

      // Called when sensor values have changed.
      @Override
      public void onSensorValueChanged(final SrsRemoteSensorEvent event){

            runOnUiThread(new Runnable() {
              @Override
              public void run() {
                if (event.sensor.getType() == SrsRemoteSensor.TYPE_PEDOMETER) {
                   pedoValueText.setText("Step Count : (" +
                                    Float.toString(event.values[0]) + ")");
                }
              }
            });
      }

      // Called when sensor is disabled in case of device disconnected.
      @Override
      public void onSensorDisabled(SrsRemoteSensor sensor){

      }
}
```

3. Using the Srs Class

The Srs class provides the following methods:

- `initialize()` initializes Remote Sensor. You need to initialize the Remote Sensor package before you can use it.
- `getVersionCode()` gets the Remote Sensor version number as an integer.
- `getVersionName()` gets the Remote Sensor version name as a string.
- `isFeatureEnabled(int type)` checks if the Remote Sensor feature is available on the host device.

```
remotesensor = new Srs();
try {
        remotesensor.initialize(this.getApplicationContext());
} catch (SsdkUnsupportedException e) {
        switch (e.getType ()) {
             case SsdkUnsupportedException.LIBRARY_NOT_INSTALLED:
                  // Handle the exception
                  break;
             case SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED:
                 // Handle the exception
                 break;
             default:
                 // Handle the exception
                break;
        }
}
```

3.1. Using the initialize() Method

The Srs.`initialize()` method:

- Initializes the Remote Sensor package
- Checks if the device supports the Remote Sensor package
- Checks if the Remote Sensor package libraries are installed on the device.

```
void initialize(Context context) throws SsdkUnsupportedException
```

If the Remote Sensor package fails to initialize, the `initialize()` method throws an SsdkUnsupportedException exception. To find out the reason for the exception, check the exception message.

```
try {
       remoteSensor.initialize (this.getApplicationContext ());
} catch (SsdkUnsupportedException e) {
    switch (e.getType ()) {
       case SsdkUnsupportedException.LIBRARY_NOT_INSTALLED:
          if ((remoteSensor.isFeatureEnabled (Srs.TYPE_GEAR_MANAGER) ==
                                                                    false) &&
               (remoteSensor.isFeatureEnabled (Srs.TYPE_GEAR_FIT_MANAGER) ==
                                                                    false)) {
             invokeInstallOption (R.string.manager_msg_str);
          }
          if (remoteSensor.isFeatureEnabled (Srs.TYPE_REMOTE_SENSOR_SERVICE) ==
                                                                  false) {
             invokeInstallOption (R.string.rss_msg_str);
          }
          break;
       case SsdkUnsupportedException.LIBRARY_UPDATE_IS_REQUIRED:
          break;
       default:
          break;
    }
}
```

3.2. Handling SsdkUnsupportedException

If an SsdkUnsupportedException exception is thrown, check the exception message type using `SsdkUnsupportedException.getType()`.The SsdkUnsupportedException can be thrown when the context argument is wrong.

Note: The Gear Manager(or Gear Fit Manager) application and Remote Sensor Service should be installed in the host. If those applications are not installed SsdkUnsupportedException is thrown. An remote sensor application can check this situation in the `isFeatureEnabled()`.

3.3. Using the isFeatureEnabled(type) Method

There are three types, TYPE_GEAR_MANAGER, TYPE_GEAR_FIT_MANAGER and TYPE_REMOTE_SENSOR_SERVICE. If this method returns false, a user has to download and install the appropriate application in the host device. That means, Gear Manager (or Gear Fit Manager) application and Remote sensor service. This method is thrown if it is called before Srs. Initialize() is called.

Note: Gear Manager (or Gear Fit Manager) should be installed before Remote sensor application. If the order is reversed, the remote sensor application may not work correctly.

4. Using the Remote Sensor Package

The Remote Sensor package supports the following sensor types:

- **TYPE_PEDOMETER**: If the host device requests the pedometer sensor data, the wearable device sends pedometer data periodically.
- **TYPE_USER_ACTIVITY**: If the host device requests the user activity sensor data, for example, notifications when the user starts to walk or run, the wearable device sends the data when the event occurs.
- **TYPE_WEARING_STATE**: If the host device requests the wearing state data, the wearable device sends the state data once.

4.1. Binding to a Remote Sensor Service

Binding occurs automatically within `Srs.initialize()`, which checks if the Remote Sensor Service is running, binds to it, and creates a socket to communicate with the wearable device. Depending on the type of wearable device, `Srs.initialize()` uses the eSAP or the Wearable Communication Protocol. You do not need to consider the exact protocol when coding your application.

```
remotesensor.initialize(this.getApplicationContext());
```

4.2. Registering a Listener for Sensor Events

To register a listener for a remote sensor:

1. Create an SrsRemoteSensorManager instance.
2. Discover the physical and virtual sensors in connected wearable devices. You can discover a specific sensor type, for example, TYPE_PEDOMETER, or all sensors (TYPE_ALL).
3. Register a listener. The rate argument determines the interval between successive events. The following values are available:
    - SENSOR_DELAY_SLOW
    - SENSOR_DELAY_NORMAL
    - SENSOR_DELAY_FAST
    - SENSOR_DELAY_FASTEST

The maxBatchReportLatency argument specifies, in microseconds, the maximum batching time. If the maxBatchReportLatency is set, the events are gathered up and delivered later in a batch.

Note: For the TYPE_PEDOMETER sensor, the rate argument has no effect, since the interval is about 5 minutes (to optimize the sensor power consumption). However, if another application has registered the TYPE_PEDOMETER sensor earlier, and pedometer data is consequently already available, you can receive your batch of data in less than 5 minutes. For the TYPE_USER_ACTIVITY sensor, the event delay is approximately 4 ~ 5 seconds. This value corresponds to 4 ~ 8 steps and

prevents noise. If this method is called for trigger sensor like TYPE _WEARING_SENSOR, it will respond only once. After event notification, the listener will be unregistered automatically.

```
public void getPedometerSensorInfo(View v){

	pedoSensorList =
		mServiceManager.getSensorList(SrsRemoteSensor.TYPE_PEDOMETER);
	SrsRemoteSensor sensor;
	sensor = pedoSensorList.get(0);
	pedoSensorText.setText(sensor.toString());
}
```

```
public void getPedometerEvent(View view){

	pedometerSensor = pedoSensorList.get(0);
	mServiceManager.registerListener(this, pedometerSensor,
				SrsRemoteSensorManager.SENSOR_DELAY_NORMAL, 0);
}
```

The requestTriggerSensor() method handles only TYPE_WEARING _STATE sensor. This is trigger sensor and responds only once. This method should be called every time the data is needed. Because this method is for one time data request, unregistering step is not necessary

Note: For the TYPE_WEARING_STATE sensor, the event comes once and the event delay is approximately 1 second. The requestTriggerSensor() returns false if it receives TYPE_PEDOMETER sensor or TYPE_USER_ACTIVITY sensor.

```
public void getWearingStateEvent (int position) {
	wearingSensor = wearingSensorList.get (0);
	mServiceManager.requestTriggerSensor (this, wearingSensor);
}
```

4.3. Receiving Sensor Events

After an application has registered a listener for a specific sensor, the onSensorValueChanged() method is called periodically or whenever an event occurs (for example, the user starts to walk or run). The SrsRemoteSensorEvent instance has a timestamp and sensor data.

This event happens once for the TYPE_WEARING_STATE sensor.

With the TYPE_PEDOMETER sensor, values[0] has a step counter started at the start of the day(00h:00m:00s).

With the TYPE_USER_ACTIVITY sensor, values[0] can have two values.

- 0.0f (UNKNOWN_STATE)
- 1.0f (WALK)
- 2.0f (RUN)

With the TYPE_WEARING_STATE sensor, values[0] can have two values.

- 0 .0f (not wearing state)
- 1.0f (wearing state)

Note: Remote Sensor ver. 1.0 does not support the onAccuracyChanged().

```
// Called when the accuracy of a sensor has changed.
@Override
public void onAccuracyChanged(SrsRemoteSensor sensor, int accuracy){

}

// Called when sensor values have changed.
@Override
public void onSensorValueChanged(final SrsRemoteSensorEvent event){

       runOnUiThread(new Runnable() {
         @Override
         public void run() {
           if (event.sensor.getType() == SrsRemoteSensor.TYPE_PEDOMETER) {
             pedoValueText.setText("Step Count : (" +
                                   Float.toString(event.values[0]) + ")");
           }
          }
       });
}

// Called when sensor is disabled in case of device disconnecting.
@Override
public void onSensorDisabled(SrsRemoteSensor sensor){

}// Called when sensor is disabled in case of device disconnected.
@Override
public void onSensorDisabled(SrsRemoteSensor sensor){
}
}
```

To save power, call `unregisterListener()` in the `onPause()` method and call `registerListener()` again in the `onResume()` method. The wearable device resets the pedometer data daily. When `onResume()` is called again, at most 1 day of pedometer data is available.

Note: Make sure to always disable sensors you do not need, especially when your activity is paused. Failure to do so can drain the battery in just a few hours. Note that the system does not disable sensors automatically when the screen turns off.

```
@Override
protected void onResume(){
      super.onResume();
      mSensorManager.registerListener (this, sensor, SrsRemoteSensorManager.
                                                      SENSOR_DELAY_NORMAL, 0);
}

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

Copyright

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co., Ltd. cannot accept responsibility for any errors or omissions or for any loss occurred to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co. Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of Samsung Electronics Co. Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

For more information, please visit http://developer.samsung.com/