Stream SDK for Android: Development Guide

July 14, 2016

NeuroSky®
Brain-Computer Interface Technologies

The NeuroSky® product families consist of hardware and software components for simple integration of this biosensor technology into consumer and industrial end-applications. All products are designed and manufactured to meet consumer thresholds for quality, pricing, and feature sets. NeuroSky sets itself apart by providing building block component solutions that offer friendly synergies with related and complementary technological solutions.

Contents

Introduction

This guide will teach you how to use **Stream SDK for Android** to write Android applications that can connect your android device and get bio-signal data from NeuroSky's bio-sensors.

This guide (and the entire **Stream SDK for Android** for that matter) is intended for programmers who are already familiar with standard Android development using Eclipse and Google's Android SDK. If you are not already familiar with developing for Android, please first visit http://developer.android.com to learn how to set up your Eclipse + Android SDK development environment and create typical Android apps.

Stream SDK contents

- Stream SDK for Android Development Guide (this document)
- libStreamSDK.jar
- libNSUART.so
- TGStreamDemo(Sample Project)
- demo(.apk file)
- ChangLog.txt

Supported NeuroSky Hardware

- Starter Kit
- MindWave Mobile

Run the Demo

There is a sample project that demonstrates how to setup, connect, and handle data to a Starter Kit device. Add the project to your Eclipse IDE by following steps:

1. In Eclipse, select **File—> New —> Project…**
2. In the New Project wizard, expand the Android section and select **Android Project from Existing Code** and click **Next**
3. Click on **Browse…** and locate the **TGStreamDemo** folder and click **Open**
4. Click the checkbox for **Copy projects to workspace** and click **Finish**
5. Click **Finish** to exit the wizard
6. Have your Android device connected to your computer
7. At this point, you should be able to browse the code, make modifications, compile, and deploy the app to your device just like any typical Android app
8. Right click on the project in the Package or Project Explorer, choose "Run As"and pick "Android Application"

Note: If there are problems, try the following:

- The sample project is generated and tested by Eclipse Standard Android SDK 4.4.2, if there is any issue related to Eclipse version or Android SDK driver, please update yours.
- Right-click on the project and select Properties. Click on the Android section and make sure a build target of at least **Android 2.2.3 (API 10)** is selected.
- In Eclipse's Package Explorer, expand the libs/ folder of your project, right-click (or Ctrl-click for Mac users) on libStreamSDK.jar, and select **Build Path** > **Add to Build Path**.
- Check the Eclipse—> Run—> Run Configurations.. to make sure the application is running on the correct device. On the Target tab you can pick "Always prompt to pick device".
- Take a look in the Eclipse Console, sometimes it indicates that it needs to be restarted, so exit Eclipse and restart it.
- If you see the message "The selection cannot be launched, and there are no recent launches, then right click on the project in the Package or Project Explorer, choose "Run As" and pick "Android Application", then try to run again.

Develop with Stream SDK

- First, make sure the "Android build target" for your project is at least Android2.2.3.

1. Right-click on your project and select **Properties**.
2. Click on the **Android** section and make sure a build target of at least **Android 2.2.3** is selected.

- Then, you must add the libStreamSDK.jar library file to your project:

1. With your Android project open in Eclipse, use the Package Explorer in Eclipse to create a libs folder at the root folder of your project
2. Use Windows Explorer (or Finder on Mac) to copy the libStreamSDK.jar file from the libs folder of your **Stream SDK for Android** to your Eclipse project's libs folder.
3. Use Windows Explorer (or Finder on Mac) to copy the libNSUART.so file from the libs/armeabi folder of your **Stream SDK for Android** to your Eclipse project's libs/armeabi folder. If armeabi foler does not exist in libs directory, use your file explorer to create this folder then move the so library file into the armeabi folder.
4. In the Eclipse Package Explorer, right-click (Ctrl-click for Mac users) on libStreamSDK.jar select **Build Path » Add to build path**.

- You have to add permissions list below to your projects AndroidManifest.xml.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

If you are using Bluetooth, you have to add Bluetooth permissions:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

TgStreamReader

TgStreamReader is used to manage the connection details, and return the parsed data. Import the following packages into your Activity:

```
import com.neurosky.connection.*;
```

TgStreamReader has four constructors:
1. public TgStreamReader(InputStream is,TgStreamHandler tgStreamHandler): used to read back raw data from file and parse them.

2. public TgStreamReader(BluetoothAdapter ba, TgStreamHandler tgStreamHandler): connect NeuroSky's hardware with Android device via Bluetooth, get the stream and parse the data.

3. public TgStreamReader(BluetoothDevice mBluetoothDevice,TgStreamHandler tgStreamHandler): connect to the given BluetoothDevice, get the stream and parse the data.

4. public TgStreamReader(String devName, TgStreamHandler tgStreamHandler): connect NeuroSky's hardware with Android device via UART, get the stream and parse the data. libNSUART.so should be add to project when using this constructor.

If we connect the NeuroSky's hardware with Android device via Bluetooth, you can create TGStreamReader like this:

```
tgStreamReader = new TgStreamReader(mBluetoothAdapter,tgStreamHandler );
```

Please make sure that there is only 1 or 0 TgStreamReader object exists at any time. If you want to create a new TgStreamReader object, please destroy the before one first. You can do it like this:

```
if(tgStreamReader!= null){
                tgStreamReader.stop();
                tgStreamReader.close();
                tgStreamReader= null;
}
```

TgStreamHandler

TgStreamHandler handles the callback messages,

```
private TgStreamHandler tgStreamHandler = new TgStreamHandler(){
@Override
                public void onDataReceived(int datatype, int data, Object obj) {
                Message msg = LinkDetectedHandler.obtainMessage();
                        msg.what = datatype;
                        msg.arg1 = data;
                        msg.obj = obj;
                        LinkDetectedHandler.sendMessage(msg);
                         }


                @Override
                public void onStatesChanged(int connectionStates) {
                        switch(connectionStates){
                        case ConnectionStates.STATE_CONNECTED:
                                tgStreamReader.start();
                                break;
                        }

                }

                @Override
                public void onRecordFail(int flag) {
                        // TODO Auto-generated method stub
                }

                @Override
                public void onChecksumFail(byte[] payload, int length, int checksum) {
```

```
                        // TODO Auto-generated method stub
                }
        };
```

When receive data, onDataReceived() is called.
When the state changes, onStatesChanged() is called.

BodyDataType

If you are using Starter kit, BodyDataType will be used to distinguish the returned data type.

Type	Description	Data
CODE_POOR_SIGNAL	init status	int
CODE_RAW	parsed raw data	int
CODE_HEATRATE	heart rate	int

An example of BodyDataType usage:

```
@Override
public void onDataReceived(int datatype, int data, Object obj) {
        // TODO Auto-generated method stub
        Message msg = LinkDetectedHandler.obtainMessage();
        msg.what = datatype;
        msg.arg1 = data;
        msg.obj = obj;
        LinkDetectedHandler.sendMessage(msg);
        //Log.i(TAG,"onDataReceived");
}

private Handler LinkDetectedHandler = new Handler() {

                @Override
                public void handleMessage(Message msg) {

                        switch (msg.what) {

                        case BodyDataType.CODE_RAW:
                                updateWaveView(msg.arg1);//0
                                break;

                        case BodyDataType.CODE_HEATRATE:
                                Log.d(TAG, "CODE_HEATRATE" + msg.arg1);
                                tv_hr.setText("" +msg.arg1 );
                                break;

                        case BodyDataType.CODE_POOR_SIGNAL:
                                int poorSignal = msg.arg1;
                                Log.d(TAG, "poorSignal:" + poorSignal);
                                tv_ps.setText(""+msg.arg1);

                                break;

                        default:
```

```
                        break;
                    }
                    super.handleMessage(msg);
                }
        };
```

MindDataType

If you are using MindWave Mobile, MindDataType will be used to distinguish the returned data type.

Type	Description	Data
CODE_POOR_SIGNAL	init status	int
CODE_RAW	parsed raw data	int
CODE_ATTENTION	attention	int
CODE_MEDITATION	meditation	int
CODE_EEGPOWER	EEGPower	EEGPower object
CODE_FILTER_TYPE	MindWave Mobile 1.5 filter type	int

Enum FilterType

FILTER_50HZ(4), FILTER_60HZ(5)

An example of MindDataType usage:

```
@Override
public void onDataReceived(int datatype, int data, Object obj) {
        // TODO Auto-generated method stub
        Message msg = LinkDetectedHandler.obtainMessage();
        msg.what = datatype;
        msg.arg1 = data;
        msg.obj = obj;
        LinkDetectedHandler.sendMessage(msg);
        //Log.i(TAG,"onDataReceived");
}

private Handler LinkDetectedHandler = new Handler() {

                @Override
                public void handleMessage(Message msg) {

                        switch (msg.what) {
                        case MindDataType.CODE_FILTER_TYPE:
                            Log.d(TAG,"CODE_FILTER_TYPE: " + msg.arg1 );

                            break;
                        case MindDataType.CODE_RAW:
                                updateWaveView(msg.arg1);
                                //You can put the raw data into Algo SDKs here
                                break;
                        case MindDataType.CODE_MEDITATION:
                                Log.d(TAG, "HeadDataType.CODE_MEDITATION " + msg.arg1);
                                tv_meditation.setText("" +msg.arg1 );
```

```
                        break;
                    case MindDataType.CODE_ATTENTION:
                        Log.d(TAG, "CODE_ATTENTION " + msg.arg1);
                        tv_attention.setText("" +msg.arg1 );
                        break;
                    case MindDataType.CODE_EEGPOWER:
                        EEGPower power = (EEGPower)msg.obj;
                        if(power.isValidate()){
                                tv_delta.setText("" +power.delta);
                                tv_theta.setText("" +power.theta);
                                tv_lowalpha.setText("" +power.lowAlpha);
                                tv_highalpha.setText("" +power.highAlpha);
                                tv_lowbeta.setText("" +power.lowBeta);
                                tv_highbeta.setText("" +power.highBeta);
                                tv_lowgamma.setText("" +power.lowGamma);
                                tv_middlegamma.setText("" +power.middleGamma);
                        }
                        break;
                    case MindDataType.CODE_POOR_SIGNAL://
                        int poorSignal = msg.arg1;
                        Log.d(TAG, "poorSignal:" + poorSignal);
                        tv_ps.setText(""+msg.arg1);

                        break;

                    default:
                        break;
                    }
                    super.handleMessage(msg);
                }
            };
```

SDK States

States	Description	int value
STATE_INIT	signal value	0
STATE_CONNECTING	connect() or connectAndStart() called	1
STATE_CONNECTED	connect() called successful	2
STATE_WORKING	start() called sucessful	3
STATE_STOPPED	stop() called	4
STATE_DISCONNECTED	close() called	5
STATE_COMPLETE	use file as data source, and parse finished	6
STATE_FAILED	connect() called failed	100
STATE_RECORDING_START	startRecordRawData() called successful	7
STATE_RECORDING_END	stopRecordRawData() called successful	8
STATE_ERROR	unhandled exception happened	101
STATE_GET_DATA_TIME_OUT	Read thread can't get data last 5 seconds(timeout time)	9

States Diagram

states diagram shows as follow:

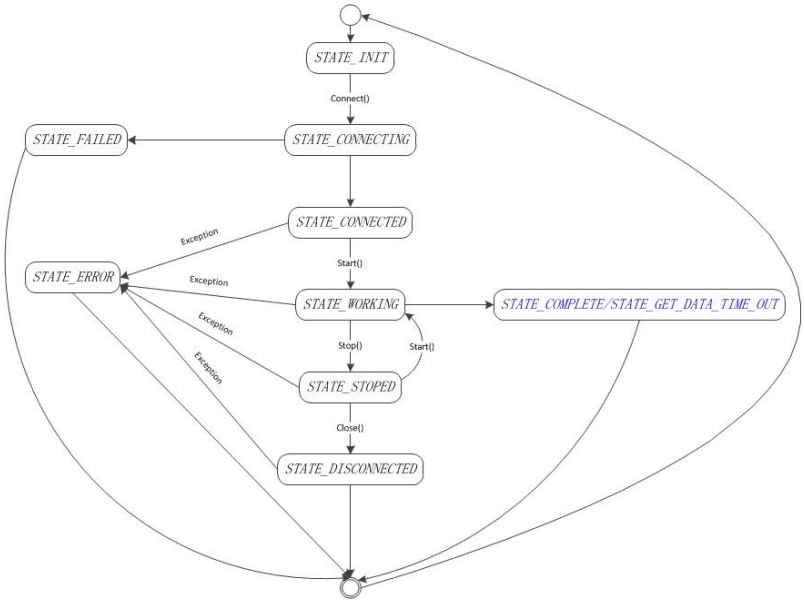

Figure 3.1:

note: STATE_COMPLETE only used in read file condition

API Reference

Note: If you want to know the usage of these API, please refer to the sample project.

getVersion

Return the SDK version

```
public static String getVersion()
```

Return

SDK version string.

Constructors

```
public TgStreamReader(InputStream is, TgStreamHandler tgStreamHandler)
```

This constructor is used to read back raw data from file and parse them. You have to get the Input-Stream first.

```
public TgStreamReader(BluetoothAdapter ba, TgStreamHandler tgStreamHandler)
```

This constructor is used to connect NeuroSky's hardware with Android phone via Bluetooth, get the stream and parse the data.

```
public TgStreamReader(BluetoothDevice mBluetoothDevice,TgStreamHandler tgStreamHandler)
```

This constructor is used to connect NeuroSky's hardware with given BluetoothDevice, get the stream and parse the data.

```
public TgStreamReader(String devName, TgStreamHandler tgStreamHandler)
```

This constructor is used to connect NeuroSky's hardware with Android phone via UART, get the stream and parse the data.

setParser

Set parser for stream reader. ParserType include PARSER_TYPE_DEFAULT, PARSER_TYPE_CHAMPION, and sampleRate include 1024 and 2048. If you are not planning to use champion parser, you don't have to call this function.

```
public void setParser(ParserType parserType, int sampleRate)
```

Parameter

parserType currently only support PARSER_TYPE_CHAMPION here.
sampleRate 1024 or 2048.

changeBluetoothDevice

Change a BluetoothDevice object for current TgStreamReader(Only when using constructor TgStreamReader(BluetoothDevice, TgStreamHandler)). You can implement change device feature by this method without closing the app.

```
public void changeBluetoothDevice(BluetoothDevice bluetoothDevice)
```

Parameter

bluetoothDevice BluetoothDevice object

startLog

Enable the debug log. Some logs in this SDK are controlled by a log flag. If you call this method, the flag will be enabled, and you will get more log by logcat.

```
public void startLog()
```

stopLog

Disable the debug log. You will get less SDK log by calling this method.

```
public void stopLog()
```

setRecordStreamFilePath

Set the file path for recording the stream data, the default path is /sdcard /neurosky/year-month-day-hour-minute-second-RawPackageRecording.txt

```
public void setRecordStreamFilePath(String filePath)
```

Parameter

filePath File path for stream data

startRecordRawData

Set the record flag for reading thread. If the setRecordStreamFilePath() is not called yet, the default file path is used. If there are some exceptions occured, onRecordFail() in TgStreamHandler will be called. The int parameter flag indicates the fail reason: 1 means getting the file path failed, even the default path; 2 means read thread is not working; 3 means recording is started, you can not call it again; 4 means opening the file failed.

```
public void startRecordRawData()
```

stopRecordRawData

Stop recording the stream data and flush the stored file.

```
public void stopRecordRawData()
```

connect

Connect to the target. Normally, you should call connectAndStart() .

```
public void connect()
```

start

Start the reading thread. Normally, you should call connectAndStart().

```
public void start()
```

connectAndStart

The process of connect will take several seconds or more , if you don't want to handle the waiting time, use this method instead. This method will call start() automatically when connect successful. If you want to manage more detail between connect and start, call connect() and start() separately.

```
public void connectAndStart()
```

stop

Stop the read thread. stopRecordRawData() will be called if startRecordRawData() is called successfully before.

```
public void stop()
```

close

Close the connect thread, if it has not finished. Release the Bluetooth socket.

```
public void close()
```

setReadFileBlockSize

Set the block size for reading file thread, the default size is 8.

```
public void setReadFileBlockSize(int size)
```

Parameter

size The size of the reading block.

setReadFileDelay

If the working thread reads file too fast, give a tiny delay after reading once, the default delay time is 2ms.

```
public void setReadFileDelay( int delayTime)
```

Parameter

delayTime The delay time for reading once.

setGetDataTimeOutTime

set the timeout time for read thread, the default time is 5s.

```
public void setGetDataTimeOutTime(int seconds)
```

Parameter

seconds Time in second for geting data time out.

redirectConsoleLogToDocumentFolder

Redirect the console log to file under `/sdcard/neruosky/Console_log/yyyyMMdd_HH_mm_ss.txt`. This is a static function.

```
public static void redirectConsoleLogToDocumentFolder()
```

stopConsoleLog

Stop the console log process. This is a static function. If you want to log all the logs in whole running time, you don't have to call this method.

```
public static void stopConsoleLog()
```

MWM15_getFilterType

Get the working filter type information. Only support MindWave Mobile1.5. This is an asynchronous call, it will return immediately, and the filter type will return by TgStreamHandler.onDataReceived with data type MindDataType.CODE_FILTER_TYPE. Please confirm the state is connected before calling this funciton.

```
public void MWM15_getFilterType()
```

MWM15_setFilterType

Set the filter type with FilterType.FILTER_50HZ or FilterType.FILTER_60HZ. The value depends on the alternating current frequency(or AC Frequency) of your country or territory, for examle, USA 60HZ, China 50HZ. If you are not sure about it, please google “alternating current frequency YOUR LOCATION”. This function only support MindWave Mobile1.5. This is an asynchronous call, it will return immediately. Please confirm the state is connected before calling this funciton. If you want to check the result, please call MWM15_getFilterType.

```
public void MWM15_setFilterType(FilterType type)
```

Parameter

FilterType type Enum for filter type, include FILTER_50HZ and FILTER_60HZ

Warnings and Disclaimer of Liability

THE ALGORITHMS MUST NOT BE USED FOR ANY ILLEGAL USE, OR AS COMPONENTS IN LIFE SUPPORT OR SAFETY DEVICES OR SYSTEMS, OR MILITARY OR NUCLEAR APPLICATIONS, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE ALGORITHMS COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. YOUR USE OF THE SOFTWARE DEVELOPMENT KIT, THE ALGORITHMS AND ANY OTHER NEUROSKY PRODUCTS OR SERVICES IS "AS-IS," AND NEUROSKY DOES NOT MAKE, AND HEREBY DISCLAIMS, ANY AND ALL OTHER EXPRESS AND IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.