ThinkGear Development Guide for Android

April 5, 2013

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 NeuroSky's **ThinkGear SDK for Android** to write Android apps that can utilize bio-signal data from NeuroSky's ThinkGear family of bio-sensors (which includes the CardioChip family of products). This will enable your Android apps to receive and use bio-signal data such as EEG and ECG/EKG acquired from NeuroSky's sensor hardware.

This guide (and the entire **ThinkGear SDK for Android** for that matter) is intended for programmers who are already familiar with standard Android development using Eclipse and Google's AndroidSDK. 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+AndroidSDK development environment and create typical Android apps.

If you are already familiar with creating typical Android apps, then the next step is to make sure you have downloaded NeuroSky's **ThinkGear SDK for Android**. Chances are, if you're reading this document, then you already have it. If not, the SDK can be downloaded from http://store.neurosky.com/products/developer-tools-3-android.

ThinkGear SDK for Android Contents

- ThinkGear SDK for Android: Development Guide (this document)
- ThinkGear SDK for Android: API Reference
- ThinkGearBase.jar library
- ThinkGearPackX.jar library
- example ThinkGear project for Android

You'll find the "API Reference" in the `reference/` folder of the TG-SDK, the "ThinkGearBase.jar" and "ThinkGearPackX.jar" in the `libs/` folder, and the "HelloEEG, HelloEKG or other example projects" in the `Sample Projects/` folder.

Important: If you have a beta test or other special version of the library jar files, the files may instead be named TgLABsBase.jar and TgLABsPackX.jar. Also for the jar files with "Pack" in the name, the following character indicates an option Pack. For example TgLABsPackA.jar or ThinkGearPackB.jar.

Supported ThinkGear Hardware

The ThinkGear SDK for Android must be used with a ThinkGear-compatible hardware sensor device. The following ThinkGear-compatible hardware devices are currently supported:

- MindWave (RF)
- MindWave Mobile

- MindBand
- MindTune
- MindSet
- ThinkCap (*some older models are no longer supported)
- BrainAthlete
- CardioChip Starter Kit Unit

Important: Before using any Android app that uses the TG-SDK for Android, make sure you have paired the ThinkGear sensor hardware to your Android device by carefully following the instructions in the User Manual that came with each ThinkGear hardware device!

Your First Project:

Your SDK may include a sample project that demonstrates how to setup, connect, and handle data to a ThinkGear device. Add the project to your Eclipse IDE by following these steps. (Tested with Eclipse IDE for Java Developers Version: Juno Service Release 1 on Mac OS 10.8.2)

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 **Hello** project folder in the **Sample Projects** 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 or emulator just like any typical Android app.

Note: If there are problems, try the following:

- Right-click on the project and select **Properties**. Click on the **Android** section and make sure a build target of at least **Android 2.3.3** is selected.
- Before trying to connect, make sure you have paired your ThinkGear hardware device to your Android device via your Android device's Bluetooth settings!
- In Eclipse's Package Explorer, expand the `libs/` folder of your project, right-click (or Ctrl-click for Mac users) on **ThinkGearBase.jar**, and select **Build Path** > **Add to Build Path** ALSO add the **ThinkGearPackX.jar** that is in the folder. (or TgLABsBase.jar and TgLABsPackX.jar)
- 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.

You may use this same process to build other sample projects.

Developing Your Own ThinkGear-enabled Apps for Android

Preparing Your Android Project

- First, make sure the "Android build target" for your project is at least Android 2.3.3.
    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 2.3.3** is selected.
- Then, you must add the `ThinkGearBase.jar` and `ThinkGearPackX.jar`library files to your project:
    1. With your Android project open in Eclipse, use the Package Explorer in Eclipse to create a `lib` folder at the root folder of your project
    2. Use Windows Explorer (or Finder on Mac) to copy the `ThinkGear` jar files from the `lib` folder of your **TG-SDK for Android** to your Eclipse project's `lib` folder.
    3. In the Eclipse Package Explorer, right-click (Ctrl-click for Mac users) on `ThinkGear-Base.jar` select **Build Path** » **Add to build path**. ALSO add the `ThinkGearPackX.jar`
- Assuming your app will be connecting to the ThinkGear hardware device via bluetooth, you will need to enable the `BLUETOOTH` permission in your app's manifest file:

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

Creating the TGDevice Object

A `TGDevice` object is used to manage a single connection to a single ThinkGear hardware device. This guide will only cover the most common connecting case, which is where your Android app will be connecting to the ThinkGear hardware device via standard Android Bluetooth using the `TGDevice(BluetoothAdapter, Handler)` constructor. (The alternate `TGDevice(InputStream, OutputStream, Handler)` constructor should only be used for special, uncommon use cases).

- Import the following packages into your Activity:

```
import com.neurosky.thinkgear.*;

import com.android.bluetooth.*;
import com.android.util.Log;
```

- Declare a `TGDevice` object and a `BluetoothAdapter` object in your Activity:

```
public class HelloEEGActivity extends Activity {
    //...
    TGDevice         tgDevice  = null;
    BluetoothAdapter btAdapter = null;
    //...
```

- Initialize the `btAdapter` and `tgDevice` in the onCreate() method:

```
    public void onCreate(Bundle savedInstanceState) {
        //...
        btAdapter = BluetoothAdapter.getDefaultAdapter();
        if( btAdapter != null ) {
            tgDevice = new TGDevice( btAdapter, handler );
        }
        //...
    }
```

Note: We'll discuss the `handler` object in the next section.

Receiving and Handling Data Messages

The TGDevice will communicate to your app via messages sent to a Handler object. To create a Handler object in your app to process incoming data:

```
    private final Handler handler = new Handler() {

        @Override
        public void handleMessage( Message msg ) {

            switch( msg.what ) {
                case TGDevice.MSG_MODEL_IDENTIFIED:
                    // connection has been established and
                    // the device identified.
                    // now configure any special options

                    if (setBlinkDetectionEnabled(true)) {
                         // return true, means success

                         Log.v( "HelloEEG", "BlinkDetection is Enabled");
                    }
                    else {
                         // return false, meaning not supported because:
                         //   + connected hardware doesn't support
                         //   + conflict with another option already set
                         //   + not support by this version of the SDK

                         Log.v( "HelloEEG", "BlinkDetection can not be Enabled");
                    }
                    break;

                case TGDevice.MSG_STATE_CHANGE:
                    switch( msg.arg1 ) {
                        case TGDevice.STATE_IDLE:
                            break;
                        case TGDevice.STATE_ERR_BT_OFF:
                            break;
                        case TGDevice.STATE_CONNECTING:
                            break;
                        case TGDevice.STATE_ERR_NO_DEVICE:
                            break;
                        case TGDevice.STATE_NOT_FOUND:
                            break;
                        case TGDevice.STATE_CONNECTED:
                            tgDevice.start();
                            break;
                        case TGDevice.STATE_DISCONNECTED:
                            break;
                        default:
                            break;
                    } /* end switch on state change type */
                    break;

                case TGDevice.MSG_POOR_SIGNAL:
                    Log.v( "HelloEEG", "PoorSignal:  " + msg.arg1 );
                    break;

                case TGDevice.MSG_ATTENTION:
                    Log.v( "HelloEEG", "Attention:  " + msg.arg1 );
                    break;

                case TGDevice.MSG_RAW_DATA:
                    int rawValue = msg.arg1;
                    break;

                case TGDevice.MSG_EEG_POWER:
                    TGEegPower ep = (TGEegPower)msg.obj;
                    Log.v( "HelloEEG", "Delta:  " + ep.delta );
                    break;

                case TGDevice.MSG_ERR_CFG_OVERRIDE:
                    Log.v( "HelloEEG", "Feature Not Allowed:  " + msg.arg1);
                    break;

                case TGDevice.MSG_ERR_NOT_PROVISIONED:
                    Log.v( "HelloEEG", "Feature Not Included:  " + msg.arg1);
                    break;

                default:
                    break;

            } /* end switch on message type */

        } /* end handleMessage() */

    }; /* end Handler */
```

The type of each Message is determined by examining the value of `msg.what`, while the actual data value for the Message is available from either `msg.arg1` (for simple values) or `msg.obj` (for more complex values).

TGDevice Message Types (msg.what)

msg.what	Description	Data
MSG_MODEL_IDENTIFIED	The TGDevice is connected	The connection has been made, packets are being received and the type of hardware has been identified. Features can now be enabled.
MSG_ERR_CFG_OVERRIDE	A user Configuration has been overridden	A user set configuration has been overridden as incompatible with the connected hardware or software, indication in the `arg1` field of the Message (see TGDevice Configuration Problems table below)
MSG_ERR_NOT_PROVISIONED	An attempt was made to configure a feature not included.	An attempt was made to configure a feature that is not included in this release. An indication is stored in the `arg1` field of the Message (see TGDevice Configuration Problems table below)
MSG_STATE_CHANGE	The TGDevice has changed state	The state change info is stored in the `arg1` field of the Message (see TGDevice States table below)
MSG_POOR_SIGNAL	Bio-signal quality/status	The status or quality of the bio-signal is stored in the `arg1` field of the Message
MSG_RAW_DATA	Raw sample value	The single-channel raw sample value (most ThinkGear devices) is stored as an int in the `arg1` field of the Message
MSG_RAW_MULTI	Multi-channel raw values	The multi-channel raw sample data (only available from certain ThinkGear devices) are stored as a **TGRawMulti** object in the `obj` field of the Message
MSG_RAW_MULTI_NEW	Multi-channel raw values	The multi-channel raw sample data (only available from certain ThinkGear devices) are stored as a **TGRawMulti** object in the `obj` field of the Message

msg.what	Description	Data
MSG_ATTENTION	Attention level	The attention level of the user is stored in the `arg1` field of the Message
MSG_MEDITATION	Meditation level	The meditation level of the user is stored in the `arg1` field of the Message
MSG_ZONE	Zone level	The Zone level of the user is stored in the `arg1` field of the Message
MSG_BLINK	Strength of detected blink	The blink strength of the user's blink is stored in the `arg1` field of the Message
MSG_EEG_POWER	EEG powers	The EEG powers of the user are stored as a **TGEegPower** object in the `obj` field of the Message
MSG_THINKCAP_RAW	Multi-channel raw values	The multi-channel raw sample data (only available from ThinkCap devices) are stored as a **TGRawMulti** object in the `obj` field of the Message
MSG_POSITIVITY	-100.0 to 100.0	Values indicate the subject's emotion or mood. The more positive values mean the subject is in a positive emotion or approach motivation, and the more negative values mean more negative emotion or withdrawal motivation.
MSG_FAMILIARITY	floating point	Can be used to compare a test subjects familiarity with a newly learned motor skill. One minute of collected data could constitute a trial, and will produce a familiarity index value. The familiarity index of separate trials of the motor skill can be compared.
MSG_DIFFICULTY	floating point	Can be used to compare the difficult a test subjects finds a newly learned motor skill. One minute of collected data could constitute a trial, and will produce a difficulty index value. The difficulty index of separate trials of the motor skill can be compared.

msg.what	**Description**	**Data**
MSG_HEART_RATE	Heart rate	The heart rate data is stored as an int in the `arg1` field of the Message
MSG_EKG_RRINT	R-to-R interval	The R-to-R interval in milliseconds is stored as an int in the `arg1` field of the Message
MSG_RELAXATION	Relaxation level	The relaxation level of the user is stored as an int in the `arg1` field of the Message
MSG_RESPIRATION	Respiration Rate	The respiration rate of the user is stored as a **Float** object in the `obj` field of the Message
MSG_HEART_AGE	Heart Age	The heart age of the user is stored as an int in the `arg1` field of the Message
MSG_HEART_AGE_5MIN	Heart Age	The heart age of the user is stored as an int in the `arg1` field of the Message
MSG_EKG_TRAIN_STEP	Train Step	The train step value is stored as an int in the `arg1` field of the Message
MSG_EKG_TRAINED	Trained	Does not return anything
MSG_EKG_IDENTIFIED	Identified	The identified result is stored as a string in the `arg1` field of the Message

Note: Depending on the type of ThinkGear hardware device that your TGDevice is connected to, some of the data types listed above will never be sent to your app's Handler, since those data types may not be applicable for that ThinkGear hardware device. See the developer specs for each ThinkGear hardware device that your app may support for details.

Important: Pay particular attention to `MSG_STATE_CHANGE` messages (which indicate what the `TGDevice` object is currently doing), and to `MSG_POOR_SIGNAL` messages (which indicates the current state of the physical ThinkGear hardware device). For example, if the `MSG_POOR_SIGNAL` is indicating that the physical ThinkGear hardware device isn't even being worn by a user at the moment (a value of 200 on ThinkGear EEG devices, or a value of 0 on ECG devices), then the app needs to treat any incoming raw data values or EEG power values accordingly, possibly by ignoring them as appropriate for the app.

For more detailed information about each data type such as Attention, Meditation, or Relaxation, please use the ThinkGear Data Types section below as a reference guide.

For information that may be device-specific, additionally refer to the specs and development notes that are available for each specific ThinkGear hardware device.

TGDevice States

State	Description
STATE_IDLE	Initial state of the TGDevice. Not connected to any ThinkGear hardware device after stop()
STATE_ERR_BT_OFF	The Android device's Bluetooth adapter is not turned on or enabled during TGDevice() constructor or connect()
STATE_CONNECTING	Attempting to connect to a ThinkGear hardware device during connect()
STATE_ERR_NO_DEVICE	There are no Bluetooth devices paired to this Android device
STATE_NOT_FOUND	Could not find any of the ThinkGear hardware devices that are paired to this Android device
STATE_CONNECTED	The data stream has been opened
STATE_DISCONNECTED	The connection to the module is lost after close()

TGDevice Configuration Problems

State	Description
ERR_MSG_BLINK_DETECT	not supported by either hardware or software, not allowed on EKG devices
ERR_MSG_TASKFAMILIARITY	not supported by either hardware or software, not allowed on EKG devices or when Positivity is enabled
ERR_MSG_TASKDIFFICULTY	not supported by either hardware or software, not allowed on EKG devices or when Positivity is enabled
ERR_MSG_POSITIVITY	not supported by either hardware or software, not allowed on EKG devices or when Familiarity or Difficulty is enabled
ERR_MSG_RESPIRATIONRATE	not supported by either hardware or software, not allowed on EEG devices

Using the TGDevice Object

With the Handler object ready to receive data and the TGDevice object created, your app can now instruct the TGDevice object to try to connect to your physical ThinkGear hardware device and start processing the incoming data.

- Call the `connect()` method of your `tgDevice` to start the connection process. The `tgDevice` will search through your Android device's list of all paired Bluetooth devices, and try to connect to the first one that it recognizes as a ThinkGear-compatible device (see Supported Devices):

```
tgDevice.connect( true );
```

Note: Setting the `connect()` method's `rawEnabled` parameter to `true` allows raw sample data to be sent to your app. Setting to `false` will prevent raw data from being parsed and sent to your app, possibly saving some CPU from parsing and saving your app's Handler from having to receive many raw sample messages that your app may not need to use.

- After successfully finding and connecting to a ThinkGear hardware device through Bluetooth, the tgDevice will send a **`STATE_CONNECTED`** Message to your app's Handler. To start receiving data upon receiving this message, call the tgDevice's `start()` method:

```
tgDevice.start();
```

- When your app no longer needs data from the TGDevice, close the connection by calling the `close()` method:

```
tgDevice.close();
```

At this point, your app should now be able to connect to a ThinkGear hardware device and receive values! To learn more details about the **TGDevice** class, you may refer to the **ThinkGear SDK for Android: API docs** at any time, found in the `reference/` folder of your **ThinkGear SDK for Android**.

Or, read on to the next section below to find out what your app can (or should) do with some of those ThinkGear values it's now receiving.

ThinkGear Data Types

The ThinkGear data types are generally divided into three groups: data types that are only applicable for EEG sensor devices, types that are only applicable for ECG/EKG (CardioChip) sensor devices, and data types that are typically applicable to all ThinkGear-based devices, including EEG and ECG/EKG.

General

These data types are generally available from most or all types of ThinkGear hardware devices.

POOR_SIGNAL/SENSOR_STATUS

This integer value provides an indication of how good or how poor the bio-signal is at the sensor. This value is typically output by all ThinkGear hardware devices once per second.

This is an extremely important value for any app using ThinkGear sensor hardware to always read, understand, and handle. Depending on the use cases for your app and users, your app may need to alter the way it uses other data values depending on the current value of `POOR_SIGNAL/SIGNAL_STATUS`. For example, if this value is indicating that the bio-sensor is not currently contacting the subject, then any received RAW_DATA or EEG_POWER values during that time should be treated as floating noise not from a human subject, and possibly discarded based on the needs of the app. The value should also be used as a basis to prompt the user to possibly adjust their sensors, or to put them on in the first place.

In order to interpret this value you must first decide which type of NeuroSky sensor you are using. As the interpretation is different for the different types of sensors.

For EEG sensor hardware: A value of 0 indicates that the bio-sensor is not able to detect any obvious problems with the signal at the sensor. Higher values from 1 to 199 indicate increasingly more detected problems with the signal. A value of 200 means the sensor contacts detect that they are not even all properly in contact with a conductive subject (for example, the EEG headset may currently not even be worn on any person's head).

For ECG/EKG (CardioChip) sensor hardware: A value of 200 indicates the bio-sensor contacts are all currently in contact with a conductive subject (such as a user's skin), while a value of 0 indicates the opposite: that not all the contacts are in proper contact with a conductive subject.

Poor signal may be caused by a number of different things. In order of severity, they are:

- Sensor, ground, or reference electrodes not being on a person's head/body (i.e. when nobody is wearing the ThinkGear equipment).
- Poor contact of the sensor, ground, or reference electrodes to a person's skin (i.e. hair in the way, or headset which does not properly fit a person's head, or headset not properly placed on the head).

- Excessive motion of the wearer (i.e. moving head or body excessively, jostling the headset/sensor).
- Excessive environmental electrostatic noise (some environments have strong electric signals or static electricity buildup in the person wearing the sensor).
- Excessive biometric noise (i.e. unwanted EMG, EKG/ECG, EOG, EEG, etc. signals)

For EEG modules, a certain amount of noise is unavoidable in normal usage of ThinkGear sensor hardware, and both NeuroSky's filtering technology and algorithms have been designed to detect, correct, compensate for, account for, and tolerate many types of signal noise. Most typical users who are only interested in using the eSense values, such as Attention and Meditation, do not need to worry as much about the `POOR_SIGNAL` Quality value, except to note that the Attention and Meditation values will not be updated while `POOR_SIGNAL` is greater than zero, and that the headset is not being worn while `POOR_SIGNAL` is 200. The `POOR_SIGNAL` Quality value is more useful to some applications which need to be more sensitive to noise (such as some medical or research applications), or applications which need to know right away when there is even minor noise detected.

RAW_DATA

This data type supplies the raw sample values acquired at the bio-sensor. The sampling rate (and therefore output rate), possible range of values, and interpretations of those values (conversion from raw units to volt) for this data type are dependent on the hardware characteristics of the ThinkGear hardware device performing the sampling. You must refer to the documented development specs of each type of ThinkGear hardware that your app will support for details.

As an example, the majority of ThinkGear devices sample at 512Hz, with a possible value range of -32768 to 32767.

As another example, to convert TGAT-based EEG sensor values (such as TGAT, TGAM, MindWave Mobile, MindWave, MindSet) to voltage values, use the following conversion:

```
(rawValue * (1.8/4096) / 2000
```

Note that ECG/EKG raw values from CardioChip/BMD10X-based devices must use a different conversion.

RAW_MULTI

This data type is not currently used by any current commercially-available ThinkGear products. It is kept here for backwards compatibility with some end-of-life products, and as a placeholder for possible future products.

EEG

These data types are only available from EEG sensor hardware devices, such as the MindWave Mobile, MindSet, MindBand, and TGAM chips and modules.

ATTENTION

This int value reports the current eSense Attention meter of the user, which indicates the intensity of a user's level of mental "focus" or "attention", such as that which occurs during intense concentration and directed (but stable) mental activity. Its value ranges from 0 to 100. Distractions, wandering thoughts, lack of focus, or anxiety may lower the Attention meter levels. See eSense\texttrademark Meters below for details about interpreting eSense levels in general.

By default, output of this Data Value is enabled. It is typically output once a second.

MEDITATION

This unsigned one-byte value reports the current eSense Meditation meter of the user, which indicates the level of a user's mental "calmness" or "relaxation". Its value ranges from 0 to 100. Note that Meditation is a measure of a person's **mental** levels, not **physical** levels, so simply relaxing all the muscles of the body may not immediately result in a heightened Meditation level. However, for most people in most normal circumstances, relaxing the body often helps the mind to relax as well. Meditation is related to reduced activity by the active mental processes in the brain, and it has long been an observed effect that closing one's eyes turns off the mental activities which process images from the eyes, so closing the eyes is often an effective method for increasing the Meditation meter level. Distractions, wandering thoughts, anxiety, agitation, and sensory stimuli may lower the Meditation meter levels. See "eSense Meters" below for details about interpreting eSense levels in general.

By default, output of this Data Value is enabled. It is typically output once a second.

eSense™ Meters

For all the different types of eSenses (i.e. Attention, Meditation), the meter value is reported on a relative eSense scale of 1 to 100. On this scale, a value between 40 to 60 at any given moment in time is considered "neutral", and is similar in notion to "baselines" that are established in conventional EEG measurement techniques (though the method for determining a ThinkGear baseline is proprietary and may differ from conventional EEG). A value from 60 to 80 is considered "slightly elevated", and may be interpreted as levels being possibly higher than normal (levels of Attention or Meditation that may be higher than normal for a given person). Values from 80 to 100 are considered "elevated", meaning they are strongly indicative of heightened levels of that eSense.

Similarly, on the other end of the scale, a value between 20 to 40 indicates "reduced" levels of the eSense, while a value between 1 to 20 indicates "strongly lowered" levels of the eSense. These levels may indicate states of distraction, agitation, or abnormality, according to the opposite of each eSense.

ZONE

This value reports the current performance Zone of the subject. It's value ranges from 0 to 3. And this value is sent only when the subject transitions from one Zone to another.

This algorithm uses the Attention and Mediation values to guide a subject to their best performance.

To be in the Elite Zone (3), the subject must hold their Attention level a value of at least 82 and simultaneously holding their Meditation level steady or increasing.

To be in the Intermediate Zone (2), the subject must hold their Attention level a value of at least 67 and simultaneously holding their Meditation level steady or increasing.

To be in the Beginner Zone (1), the subject must hold their Attention level a value of at least 53 and simultaneously holding their Meditation level steady or increasing.

The Not Ready Zone (0) is all Attention levels below 53 and subjects from the Beginner Zone whose Meditation levels are decreasing.

All Zone calculations are suspended and values reset if the sensor doesn't appear to be in good contact with a human.

Note: This is a different implementation of performance Zone compared to other NeuroSky products. Reference: Golf Putting Training Algorithm v 2.0, September 2012, Dr. KooHyoung Lee.

BLINK

This int value reports the intensity of the user's most recent eye blink. Its value ranges from 1 to 255 and it is reported whenever an eye blink is detected. The value indicates the relative intensity of the blink, and has no units.

The Detection of Blinks must be enabled.

```
if (setBlinkDetectionEnabled(true)) {
     // return true, means success

     Log.v( "HelloEEG", "BlinkDetection is Enabled");
}
else {
     // return false, meaning not supported because:
     //  + connected hardware doesn't support
     //  + conflict with another option already set
     //  + not support by this version of the SDK

     Log.v( "HelloEEG", "BlinkDetection can not be Enabled");
}
```

The current configuration can be retrieved.

```
if (getBlinkDetectionEnabled()) {
     // return true, means it is enabled

     Log.v( "HelloEEG", "BlinkDetection is configured");
}
else {
     // return false, meaning not currently configured

     Log.v( "HelloEEG", "BlinkDetection is NOT configured");
}
```

Note: If these methods are called before the MSG_MODEL_IDENTIFIED has been received, it is considered a request to be processed when the connected equipment is identified. It is possible to Enable this feature and later find that it is no longer enabled. Once the connected equipment has been identified, if the request is incompatible with the hardware or software it will be overridden and the MSG_ERR_CFG_OVERRIDE message sent to provide notification.

EEG_POWER

This Data Value represents the current magnitude of 8 commonly-recognized types of EEG frequency bands.

The eight EEG powers are: delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low-alpha (7.5 - 9.25Hz), high-alpha (10 - 11.75Hz), low-beta (13 - 16.75Hz), high-beta (18 - 29.75Hz), low-gamma (31 -

39.75Hz), and mid-gamma (41 - 49.75Hz). These values have no units and are only meaningful for comparison to the values for the other frequency bands within a sample.

By default, output of this Data Value is enabled, and it is output approximately once a second.

THINKCAP_RAW

This data type is not currently used by any current commercially-available ThinkGear products. It is kept here for backwards compatibility with some end-of-life products, and as a placeholder for possible future products.

POSITIVITY

Values -100 to +100, indicates that the subject is attentive, the more negative values mean the subject is less attentive and the more positive values mean more attentive

Note: This feature is not currently available.

FAMILIARITY

This algorithm seeks to represent the subject's Familiarity with a motor skill, it may be used together with the Difficulty algorithm to examine different aspects of learned motor skills. But it may also be used independently.

It can be used to compare a test subjects familiarity with a newly learned motor skill. One minute of collected data constitutes a trial, and will produce a familiarity index value for this individual. The familiarity index of separate trials of the motor skill can be compared for the same individual.

The familiarity index is a reported as a floating point number. They have no units and are only meaningful in comparison to other values collected from the same individual. A useful presentation is to calculate the percentage change from a baseline (or the last) trial and the current trial. Note that the difference may be positive or negative. Examine the HelloEEG sample application for one example of how to use this information.

The Calculation of Task Familiarity must be enabled.

```
if (setTaskFamiliarityEnable(true)) {
     // return true, means success

     Log.v( "HelloEEG", "TaskFamiliarity is Enabled");
}
else {
     // return false, meaning not supported because:
     //  + connected hardware doesn't support
     //  + conflict with another option already set
     //  + not support by this version of the SDK

     Log.v( "HelloEEG", "TaskFamiliarity can not be Enabled");
}
```

The current configuration can be retrieved.

```
if (getTaskFamiliarityEnable()) {
     // return true, means it is enabled

     Log.v( "HelloEEG", "TaskFamiliarity is configured");
}
else {
     // return false, meaning not currently configured

     Log.v( "HelloEEG", "TaskFamiliarity is NOT configured");
}
```

It is possible to configure Task Familiarity to run continuously. Configuration of this feature does not change the basic Task Familiarity configuration.

```
if (setTaskFamiliarityRunContinuous(true)) {
     // return true, means success

     Log.v( "HelloEEG", "TaskFamiliarity Continuous operation");
}
else {
     // return false, meaning not supported because:
     //  + connected hardware doesn't support
     //  + conflict with another option already set
     //  + not support by this version of the SDK

     Log.v( "HelloEEG", "TaskFamiliarity normal operation ");
}
```

The current configuration can be retrieved.

```
if (getTaskFamiliarityRunContinuous()) {
     // return true, means it is enabled

     Log.v( "HelloEEG", "TaskFamiliarity Continuous operation");
}
else {
     // return false, meaning not currently configured

     Log.v( "HelloEEG", "TaskFamiliarity normal operation");
}
```

Note: If these methods are called before the MSG_MODEL_IDENTIFIED has been received, it is considered a request to be processed when the connected equipment is identified. It is possible to Enable this feature and later find that it is no longer enabled. Once the connected equipment has been identified, if the request is incompatible with the hardware or software it will be overridden and the MSG_ERR_CFG_OVERRIDE message sent to provide notification.

Note: This algorithm is resource and computation intensive. If you need to run with the Android Debugger, be aware that this calculation may take many minutes (perhaps 5) to complete when the debugger is engaged. It will complete and present it's results. Without the debugger engaged, this calculation will complete in a few seconds.

DIFFICULTY

This algorithm seeks to represent the subject's Mental Effort, it may be used together with the Familiarity algorithm to examine different aspects of learned motor skills. But it may also be used independently.

It can be used to compare how difficult a test subjects finds a newly learned motor skill. One minute of collected data constitutes a trial, and will produce a difficulty index value for this individual. The difficulty index of separate trials of the motor skill can be compared for the same individual.

The difficulty index is a reported as a floating point number. They have no units and are only meaningful in comparison to other values collected from the same individual. An useful presentation is to calculate the percentage change from an initial (or baseline) trial and the current trial. Note that the difference may be positive or negative. Examine the HelloEEG sample application for one example of how to use this information.

The Calculation of Task Difficulty must be enabled.

```
if (setTaskDifficultyEnable(true)) {
     // return true, means success

     Log.v( "HelloEEG", "TaskDifficulty is Enabled");
}
else {
     // return false, meaning not supported because:
     //  + connected hardware doesn't support
     //  + conflict with another option already set
     //  + not support by this version of the SDK

     Log.v( "HelloEEG", "TaskDifficulty can not be Enabled");
}
```

The current configuration can be retrieved.

```
if (getTaskDifficultyEnable()) {
     // return true, means it is enabled

     Log.v( "HelloEEG", "TaskDifficulty is configured");
}
else {
     // return false, meaning not currently configured

     Log.v( "HelloEEG", "TaskDifficulty is NOT configured");
}
```

It is possible to configure Task Difficulty to run continuously. Configuration of this feature does not change the basic Task Difficulty configuration.

```
if (setTaskDifficultyRunContinuous(true)) {
     // return true, means success

     Log.v( "HelloEEG", "TaskDifficulty Continuous operation");
}
else {
     // return false, meaning not supported because:
     //  + connected hardware doesn't support
     //  + conflict with another option already set
     //  + not support by this version of the SDK

     Log.v( "HelloEEG", "TaskDifficulty normal operation ");
}
```

The current configuration can be retrieved.

```
if (getTaskDifficultyRunContinuous()) {
     // return true, means it is enabled

     Log.v( "HelloEEG", "TaskDifficulty Continuous operation");
}
else {
     // return false, meaning not currently configured

     Log.v( "HelloEEG", "TaskDifficulty normal operation");
}
```

Note: If these methods are called before the MSG_MODEL_IDENTIFIED has been received, it is considered a request to be processed when the connected equipment is identified. It is possible to Enable this feature and later find that it is no longer enabled. Once the connected equipment has been identified, if the request is incompatible with the hardware or software it will be overridden and the MSG_ERR_CFG_OVERRIDE message sent to provide notification.

Note: This algorithm is resource and computation intensive. If you need to run with the Android Debugger, be aware that this calculation may take many minutes (perhaps 5) to complete when the debugger is engaged. It will complete and present it's results. Without the debugger engaged, this calculation will complete in a few seconds.

ECG/EKG

These data types are only available from ECG/EKG sensor (CardioChip) hardware devices, such as the CardioChip Starter Kit Unit and BMD10X chips and modules.

HEART_RATE

This int value reports the current heart rate of the user, in units of beats per minute (BPM). Unlike many other commonly seen reports of heart rate from other devices, this value is calculated precisely in real time based on the actual time between each and every one of the user's actual R-peaks. This results in a very precise and continuous reporting of Heart Rate that changes with the actual beat-to-beat fluctuations of every single one of the user's actual heart beats.

To easily get an "smoothed, averaged" heart rate value which is more commonly seen as reported by other ECG/EKG devices, use these values as inputs to the Smoothed Heart Rate described below.

Smoothed Heart Rate

Typically, when viewing a "Heart Rate" value on many ECG/EKG devices, a "smoothed" value is displayed so that there aren't rhythmic fluctuations in the viewed heart rate based on the subject's natural HRV rhythms. The same sort of "smoothed" effect can be achieved against the precise HEART_RATE values, by using the `getAcceleration()` method of the `HeartRateAcceleration` class provided in this SDK.

See the section on Heart Rate Acceleration for a description of how to calculate the Smoothed Heart Rate, and then refer to the API Reference for full details on the `HeartRateAcceleration` class.

Heart Rate Acceleration

A potentially useful metric of Heart Rate is the acceleration rate. A positive acceleration value indicates the user's heart rate is speeding up by a certain number of BPM over a given period of time (such as over 10 seconds), while a negative acceleration value indicates the user's heart rate is slowing down by a certain number of BPM over a given period. When starting exercise, or during rest after exercise, this acceleration metric could be used as an indicator of how quickly a person's heart is speeding up to match the activity, or how quickly it is slowing down back to normal, respectively.

To calculate Heart Rate Acceleration (and/or Smoothed Heart Rate), first initialize a HeartRateAcceleration() object in your app:

```
HeartRateAcceleration heartRateAcceleration = new HeartRateAcceleration();
```

This will initialize the calculation to use a period of 10 second. (You can instead choose to use the overloaded constructors to initialize the calculation using a longer or shorter period of time, as appropriate for your app).

Then, whenever a new Heart Rate value becomes available to your app, get the Smoothed Heart Rate and acceleration values like this:

```
int[] result = heartRateAcceleration.getAcceleration( heartRate, poorSignal );
if( result[0] != -1 ) {
    int smoothedHeartRate = result[0];
    int heartRateAcceleration = result[1];
}
```

Refer to the API Reference documentation for full details of the `HeartRateAcceleration` class.

Target Heart Rate for Physical Training

Given information about a user's age and gender, it is possible to determine a target range of heart rates for them to achieve particular physical training "zones". Combined with the HEART_RATE information reported by the sensor, an app could advise a user whether their current heart rate is within their target training zone (such as right after a workout).

To determine the target range of heart rates for a person, first create a `TargetHeartRate` object:

```
TargetHeartRate targetHeartRate = new TargetHeartRate();
```

Then, at any time, to determine the target range of heart rates (min to max values) for a a user to achieve a particular physical training zone, use the `getTargetHeartRate()` method:

```
int age = 25;
String gender = "Male";
String zone = "Aerobic";
int[] range = targetHeartRate.getTargetHeartRate( age, gender, zone );

int lowerBound = range[0];
int upperBound = range[1];
```

The `lowerBound` and `upperBound` could then be compared to the user's HEART_RATE or Smoothed Heart Rate to determine if the user is within the their target range for the target physical training zone.

The `gender` must be either `"Male"` or `"Female"`. The `zone` must be one of:

- `"Light Exercise"`
- `"Weight Loss"`
- `"Aerobic"`
- `"Conditioning"`
- `"Athletic"`

If any of the arguments are incorrect, then the method will return an int[] of `-1, -1` .

Important: The heart rate should not be measured while the user is engaged in physical activity; the user should temporarily stop the activity and then measure their heart rate.

(References)

1. http://www.heart.org/HEARTORG/GettingHealthy/PhysicalActivity/Target-Heart-Rates_UCM_434341_
2. http://www.cdc.gov/physicalactivity/everyone/measuring/heartrate.html
3. http://www.heart.com/heart-rate-chart.html
4. http://www.thewalkingsite.com/thr.html

Heart Fitness Level

Given a person's age, gender, and current resting heart rate, it is possible to get a general idea of the person's current heart health and fitness, labeling them as one of "Poor", "Below Average", "Average", "Above Average", "Good", "Excellent", or "Athlete".

To determine the heart fitness level for a person, first create a `HeartFitnessLevel` object:

```
HeartFitnessLevel heartFitnessLevel = new HeartFitnessLevel();
```

Then, once you have the age, gender, and current *resting* heart rate of the person, use the `getHeart-FitnessLevel()` method:

```
int age = 25;
String gender = "Male";  // "Male" or "Female"
int restingHR = 60;
String heartFitnessLevel = heartFitnessLevel.getHeartFitnessLevel( age, gender, restingHR );
```

The gender must be one of "Male" or "Female", otherwise the method will simply return the empty string ("").

The `heartFitnessLevel` will be returned as one of "Poor", "Below Average", "Average", "Above Average", "Good", "Excellent", or "Athlete".

(References)

1. http://www.topendsports.com/testing/heart-rate-resting-chart.htm

RELAXATION

The Relaxation data value gives an indication of whether a user's heart is showing indications of relaxation, or is instead showing indications of excitation, stress, or fatigue, based on the user's Heart Rate Variability (HRV) characteristics. It is reported on a scale from 1 to 100. High Relaxation values tend to indicate a state of relaxation, while low values tend to indicate excitation, stress, or fatigue.

To receive these values via MSG_RELAXATION messages to your app's Handler, simply have the TGDevice connected to a ThinkGear ECG/EKG sensor (CardioChip), and a user contacting the ECG/EKG sensor hardware properly for at least one minute continuously with a good, clean signal (SENSOR_STATUS == 200 for 1 minute). If the signal is interrupted, and SENSOR_STATUS becomes anything other than 200, then this calculation is reset and starts over, requiring another minute of clean data to report a MSG_RELAXATION.

For best results, the user should be sitting calmly during data collection.

(References)

1. Neurosci Biobehav Rev. 2009 Feb; 33(2): 71-80. Epub 2008 Jul 30. Heart rate variability explored in the frequency domain: a tool to investigate the link between heart and behavior. Montano N, Porta A, Cogliati C, Costantino G, Tobaldini E, Casali KR, Iellamo F.
2. Int J Cardiol. 2002 Jul; 84(1): 1-14. Functional assessment of heart rate variability: physiological basis and practical applications. Pumprla J, Howorka K, Groves D, Chester M, Nolan J.
3. International Conference on Computer and Automation Engineering. A Review of Measurement and Analysis of Heart Rate Variability. Dipali Bansal, Munna Khan, A. K. Salhan.
4. Neurosci Biobehav Rev. 2009 Feb; 33(2): 81-8. Epub 2008 Aug 13. Claude Bernard and the heart-brain connection: further elaboration of a model of neurovisceral integration. Thayer JF, Lane RD.

RESPIRATION

The Respiration data value reports a user's approximate respiration rate in breaths per minute. It is calculated from the user's ECG/EKG and Heart Rate Variability (HRV) characteristics.

To receive these values via MSG_RESPIRATION Messages to your app's Handler, simply have the TGDevice connected to a ThinkGear ECG/EKG sensor (CardioChip), and a user contacting the ECG/EKG sensor hardware properly for at least 64 seconds continuously with a good, clean signal (SENSOR_STATUS >= 200 for 64 seconds). Once the first MSG_RESPIRATION Message is received, updated values will be sent via subsequent MSG_RESPIRATION Messages every 10 seconds. If the signal is ever interrupted, and SENSOR_STATUS becomes anything less than 200, then this calculation is reset and starts over, requiring another 64 seconds of clean data to report a MSG_RESPIRATION.

For best results, the user should be sitting calmly during data collection.

(References)

1. Rosenthal, Talma, Ariela Alter, Edna Peleg, and Benjamin Gavish. "Device-guided breathing exercises reduce blood pressure: ambulatory and home measurements." American Journal of Hypertension. 14. (2001): 74–76.

The Calculation of Respiration Rate must be enabled. And once enabled it will run continuously until disabled.

```
if (setRespirationRateEnable(true)) {
     // return true, means success

     Log.v( "HelloEKG", "RespirationRate is Enabled");
}
else {
     // return false, meaning not supported because:
     //  + connected hardware doesn't support
     //  + conflict with another option already set
     //  + not support by this version of the SDK

     Log.v( "HelloEKG", "RespirationRate can not be Enabled");
}
```

The current configuration can be retrieved.

```
if (getRespirationRateEnable()) {
     // return true, means it is enabled

     Log.v( "HelloEKG", "RespirationRate is configured");
}
else {
     // return false, meaning not currently configured

     Log.v( "HelloEKG", "RespirationRate is NOT configured");
}
```

Note: If these methods are called before the MSG_MODEL_IDENTIFIED has been received, it is considered a request to be processed when the connected equipment is identified. It is possible to Enable this feature and later find that it is no longer enabled. Once the connected equipment has been identified, if the request is incompatible with the hardware or software it will be overridden and the MSG_ERR_CFG_OVERRIDE message sent to provide notification.

Note: This algorithm is resource and computation intensive. If you need to run with the Android Debugger, be aware that this calculation may take many minutes (perhaps 5) to complete. It will complete and present it's results. Without the debugger engaged, this calculation will complete in a few seconds.

Heart Risk Awareness

The Heart Risk Awareness data value aims to raise awareness if the HRV is very low, as low HRV has been shown to be associated with increased risk of mortality.

To determine the Heart Risk Awareness for a person, first create a NeuroSkyHeartMeters object:

```
    NeuroSkyHeartMeters neuroSkyHeartMeters = new NeuroSkyHeartMeters();
```

Then, use one of the following two methods to calculate:

Using R-R Intervals Collection

Whenever your app's Handler receives a MSG_EKG_RRINT Message, save the R-R Interval value into a buffer. Once you have at least 60 R-R Intervals in the buffer, use the `calculateHeartRiskAware( Integer[] rrIntervalInMS )` method of the `NeuroSkyHeartMeters` class:

```
    private final Handler handler = new Handler() {

        ArrayList<Integer> temp_rrintBuffer = new ArrayList<Integer>();
        Integer[] rrinterBuffer = new Integer[60];

        @Override
        public void handleMessage( Message msg ) {

            switch( msg.what ) {

              //...

              case MSG_EKG_RRINT:
                  temp_rrintBuffer.add( msg.arg1 );
                  if( temp_rrintBuffer.size()==60 ) {
                      for( int i = 0; i<60; i++ ) {
                          rrintBuffer[i] = temp_rrintBuffer.get(i);
                      }
                      temp_rrintBuffer.clear();
                      int heartRiskAwareness = neuroSkyHeartMeters.calculateHeartRiskAware(
rrintBuffer );
                  }
                  break;

                //...

            } /* end switch on message type */

        } /* end handleMessage() */

    }; /* end Handler */
```

Using Storage Data

Simply use the `calculateHeartRiskAware( String fileName )` method in the NeuroSkyHeart-Meters class:

```
    int heartRiskAwarness = neuroSkyHeartMeters.calculateHeartRiskAware( "john" );
```

Note: The parameter "fileName" in the method is the name of the file that stored calculated heart age.

Results

The return value will be a "Heart Risk Awareness" index that will be one of "0", "1","2" or "3". The following information could be provided by the app to the user based on their "Heart Risk Awareness":

HeartRiskAwareness = 0

Your HRV does not appear to be low at this time. Low HRV has been shown to be related to increased risk of heart attack and mortality. This means your HRV suggests you currently have limited or no risk.

HeartRiskAwareness = 1

Your HRV is a relatively low. Low HRV has been shown to be related to increased risk of heart attack and mortality. It is recommended that you stay active and be careful about what you eat. You could eat foods that can prevent heart attack, such as nuts, fish, coarse grains, vegetables, and you may also drink green tea.

HeartRiskAwareness = 2

Your HRV is low. Low HRV has been shown to be related to increased risk of heart attack and mortality. Bad habits that influence the health of your heart include consumption of food with high fat and sugar content, smoking, drinking alcohol, lack of exercise, high mental pressure, and long periods of sleep deprivation. It is recommended that you change your bad habits by drinking only a moderate amount of alcohol, eating healthy, getting appropriate exercise, controlling your body weight, developing good sleeping habits, and keeping a peaceful state of mind.

HeartRiskAwareness = 3

Your HRV is very low. Low HRV has been shown to be related to increased risk of heart attack and mortality. It is recommended that you change some of your habits. You may consider to quit smoking, stop drinking alcohol. You should also make sure to get appropriate amount of exercise, control your body weight, develop good sleeping habits, eat more fiber and less salt, and keep a peaceful state of mind. Symptoms of heart attack include chest pain, shoulder pain, trouble breathing, poor digestion, and severe fatigue. Please see a doctor if these symptoms occur to you.

(for References, see Heart Age)

HEART_AGE

The Heart Age data value provides an indication of the relative age of a subject heart, based on their Heart Rate Variability (HRV) characteristics as compared to the general population. A low HRV is associated with an increased risk of mortality, and is represented by a Heart Age that is possibly higher than the user's biological age (such as a 35 year old with HRV characteristics that suggest a heart age of 45). The calculation will take into account the user's reported biological age. Use of this data value is only recommended for subjects that are at least 10 years old (biological age).

To receive these values via MSG_HEART_AGE Messages to your app's Handler, first set the user's biological age via the TGDevice object: `tgDevice.inputAge = 25` (of course replacing 25 with the user's actual age). Then, simply have the TGDevice connected to a ThinkGear ECG/EKG sensor (CardioChip), and a user contacting the ECG/EKG sensor hardware properly for at least 60 heart beats continuously with a good, clean signal (SENSOR_STATUS >= 200 for 60 heart beats). If the signal is ever interrupted, and SENSOR_STATUS becomes anything less than 200, then this calculation is reset and starts over, requiring another 60 heart beats of clean data before it can report a MSG_HEART_AGE.

For best results, the user should be sitting calmly during data collection.

An example of how your app and users could potentially use this information would be if your app displayed messages like the following to the user based on their Heart Age value:

Adolescent heart: < 25 years old

Your heart age is xx years old, which is greater/less than your actual age by xx years. Your young heart age allows you to be energetic and to think actively, which helps you deal with demanding work and exercise. A young heart also needs to be taken care of. It is recommended that you avoid staying up late at night, get appropriate amounts of exercise, and maintain a peaceful and positive attitude. You should also eat more fresh fruits and vegetables and cut down on fatty foods to keep your heart at its young state.

Young heart: 26 – 39

Your heart age is xx years old, which is greater/less than your actual age by xx years. You have a mature heart. While in a tense working environment, please don't forget to get good amounts of sleep and exercise, eat well, and take good care of yourself.

Middle-aged heart: 40 – 55

Your heart age is xx years old, which is greater/less than your actual age by xx years. Please pay close attention to your heart health and plan your work and life accordingly to lessen the burden on your heart. It is recommenced that you eat foods that are good for your heart, such as fish, whole grains, beans, nuts, vegetables, red wine, and green tea. You should also get a reasonable amount of exercise to strengthen your heart.

Young elderly heart: 56 – 70

Your heart age is xx years old, which is greater/less than your actual age by xx years. Your heart's function is taking a step towards old age. It is recommended that you live with discipline and avoid straining your body or becoming overly excited or nervous. You should also have regular physical examinations and eat more foods that are good for your heart, such as fish, coarse grains, beans, nuts, vegetables, red wine, and green tea. It is also important for you to get a reasonable amount of exercise so that your heart continues to work effectively.

Elderly heart: >70 years old

Your heart age is xx years old, which is greater/less than your actual age by xx years. It is recommended that you regularly visit your doctor to get physical examinations and carefully follow your doctor's instructions in order to prevent and treat heart disease. You should also get appropriate amounts of exercise and keep a peaceful state of mind. Living with discipline and eating healthy can improve the function of your cardiovascular system and prevent heart disease.

(References)

1. Res Sports Med. 2010 Oct; 18(4):263-9. Age and heart rate variability after soccer games. Yu S, Katoh T, Makino H, Mimuno S, Sato S.

2. J Am Coll Cardiol. 1998 Mar 1; 31(3): 593-601. Twenty four hour time domain heart rate variability and heart rate: relations to age and gender over nine decades. Umetani K, Singer DH, McCraty R, Atkinson M.

3. Am J Cardiol. 2010 Apr 15; 105(8): 1181-5. Epub 2010. Relation of high heart rate variability to healthy longevity. Zulfiqar U, Jurivich DA, Gao W, Singer DH.

4. Cardiovasc Electrophysiol. 2003 Aug; 14(8): 791-9. Circadian profile of cardiac autonomic nervous modulation in healthy subjects: differing effects of aging and gender on heart rate variability. Bonnemeier H, Richardt G, Potratz J, Wiegand UK, Brandes A, Kluge N, Katus HA.

5. Pacing Clin Electrophysiol. 1996 Nov; 19(11 Pt 2): 1863-6. Changes in heart rate variability with age. Reardon M, Malik M.

Personalization

This algorithm allows the TGDevice to try to recognize a connected user based on their ECG/EKG data. To use it, one or more users should "train" their ECG/EKG data into the TGDevice. Then, whenever the TGDevice is reading ECG data from a user, it can attempt to identify which of the trained users (if any) is the one it is reading from.

SEE APPENDIX C and APPENDIX D for the steps to enable this feature.

There are two steps to use the Personalization algorithm:

Training

The first part records ECG/EKG data of user, using the `EKGstartLongTraining( String userName )` method in the TGDevice class. If the user then keeps good, clean contact with the ECG/EKG sensor hardware properly, a `MSG_EKG_TRAIN_STEP` Message will be sent to your app's Handler to show which step you are currently on. After two steps are finished, it will send a `MSG_EKG_TRAINED` Message to your app's Handler to indicate that the recording is finished, then your Handler should use the `EKGstopTraining()` method to stop.

Detection

This part is used to recognize user based on saved data from the first part. To recognize user, invoke the `EKGstartDetection()` method. Then, if the user keeps a good, clean contact with the ECG/EKG sensor hardware, the TGDevice will send a `MSG_EKG_IDENTIFIED` Message to your app's Handler. The return value will be one of the registered user names, or "Unknown".

```
private final Handler handler = new Handler() {

    @Override
    public void handleMessage( Message msg ) {

        switch( msg.what ) {

            //...

            case MSG_EKG_TRAIN_STEP:
                int trainStep = msg.arg1;
                break;

            case MSG_EKG_TRAINED:
                tgDevice.EKGstopTraining();
                break;

            case MSG_EKG_IDENTIFIED:
                String result = msg.obj;
                // result is the text of the userName for the matched profile
                // or "Unknown" if there isn't a good match
                tgDevice.EKGstopDetection();
                break;

            //...

        } /* end switch on message type */

    } /* end handleMessage() */

}; /* end Handler */

//...

tgDevice.EKGstartLongTraining( "John" );

//...

tgDevice.EKGstartDetection();
```

EKG_RRINT

Whenever an R-peak is detected along a user's PQRST ECG/EKG, then a MSG_EKG_RRINT Message is sent to your app's Handler indicating the R-R interval, in milliseconds, since the last R-peak.

Proper App Design

Important: Before releasing an app for real-world use, make sure your app considers or handles the following:

- If your app's Handler receives a `MSG_STATE_CHANGE` Message with any value other than `STATE_CONNECTING` or `STATE_CONNECTED`, it should carefully handle each possible error situation with an appropriate message to the user via the app's UI. Not handling these error cases well in the UI almost always results in an extremely poor user experience of the app. Here are some examples:
  - If a `STATE_ERR_BT_OFF` Message is received, the user should be prompted to turn on their Bluetooth adapter, and then they can try again.
  - If a `STATE_ERR_NO_DEVICE` Message is received, the user should be reminded to first pair their ThinkGear hardware device to their Android device's Bluetooth, according to the instructions they received with their ThinkGear hardware device.
  - If a `STATE_NOT_FOUND` Message is received, the user should be reminded to check that their ThinkGear hardware device is properly paired to their Android device (same as the `STATE_ERR_NO_DEVICE` case), and if so, that their ThinkGear hardware device is turned on, in range, and has enough battery or charge.
  - See TGDevice States for more info.
- Always make sure your app is handling the POOR_SIGNAL/SENSOR_STATUS Data Type. It is output by almost all ThinkGear devices, and provides important information about whether the sensor is properly in contact with the user. If it is indicating some sort of problem (problem == not 0 for EEG devices, or not 200 for ECG/EKG devices), then your app should notify the user to properly wear the ThinkGear hardware device, and/or disregard any other reported data values while the POOR_SIGNAL/SENSOR_STATUS continues to indicate a problem, as appropriate for your app.
- To make the user experience consistent, familiar, and easy-to-learn and use for end customers across platforms and devices, your app should be designed to follow the guidelines and conventions described in NeuroSky's App Standards.

Troubleshooting

Note: There are currently no known issues. If you encounter any bugs or issues, please visit http://support.neurosky.com, or contact support@neurosky.com.

If you need further help, you may visit http://developer.neurosky.com to see if there is any new information.

To contact NeuroSky for support, please visit http://support.neurosky.com, or send email to support@neurosky.com.

For developer community support, please visit our community forum on http://www.linkedin.com/groups/NeuroSky-Brain-Computer-Interface-Technology-3572341

Important Notices

The algorithms included in this SDK are solely for promoting the awareness of personal wellness and health and are not a substitute for medical care. The algorithms are not to be used to diagnose, treat, cure or prevent any disease, to prescribe any medication, or to be a substitute for a medical device or treatment. In some circumstances, the algorithm may report false or inaccurate results. The descriptions of the algorithms or data displayed in the SDK documentation, are only examples of the particular uses of the algorithms, and NeuroSky disclaims responsibility for the final use and display of the algorithms internally and as made publically available.

The algorithms may not function well or may display accurate data if the user has a pacemaker.

All ECG data should be collected while the user is seated quietly, breathing regularly, with minimal movement, for best results.

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.

IN NO EVENT SHALL NEUROSKY BE LIABLE FOR ANY SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING BUT NOT LIMITED TO LOSS OF PROFITS OR INCOME, WHETHER OR NOT NEUROSKY HAD KNOWLEDGE, THAT SUCH DAMAGES MIGHT BE INCURRED.

APPENDIX A: Additional References

- http://developer.android.com/guide/topics/wireless/bluetooth.html

APPENDIX B: UART (Non-Bluetooth) Connections

For Android devices, such as phones or tablets, that are designed to be physically connected to ThinkGear hardware devices via UART (especially CardioChip), the TG-SDK for Android offers an alternate constructor and connect method:

```
public TGDevice( InputStream s, OutputStream o, Handler h );
public synchronized void connectStream( boolean rawEnabled );
```

Instead of taking a `BluetoothAdapter` in the constructor and using Bluetooth for communication, it instead takes an `InputStream` and `OutputStream` for communication. The `connectStream()` method is then analogous for this constructor as the `connect()` method for the Bluetooth-based constructor.

To use this API of the SDK, you must first prepare the Android/Linux system as follows:

1. Physically connect the UART pins of the ThinkGear hardware device to the UART pins of the Android device
2. Make sure the Linux OS layer beneath the Android OS layer makes the UART port available as a serial port, such as "/mnt/uart1"

Then, because Android does not provide APIs to directly access serial ports, we must go around the Android layer to the Linux serial port layer using JNI:

1. Compile the `uart.cpp` file (provided separately by NeuroSky, along with a makefile) into a native library file called `libJRDBCM.lib` that JNI will be able to call. This library will make it possible for your app to open the serial port.
2. Use JNI in your app to call the `SerialJNI_open()` method from within your app. This will open a `FileDescriptor`, from which you can obtain a `FileInputStream` and `FileOutputStream` for use with the TGDevice constructor:

```
public class Native {
    static {
        System.loadLibrary( "JRDBCM" );
    }
}

String dev = "/mnt/uart1";

FileDescriptor serialPort = Native.SerialJNI_open( dev );
InputStream iStream = new FileInputStream( serialPort );
OutputStream oStream = new FileOutputStream( serialPort );

TGDevice tgDevice = new TGDevice( iStream, oStream, handler );
tgDevice.connectStream( true );
```

From here on out, the TG API is exactly the same as for Bluetooth described in the rest of this document. Just remember to close your `FileDescriptor` using `Native.SerialNJI.close()` when you are done with the TGDevice, *after* the TGDevice is closed.

APPENDIX C: Enabling the use of Personalization

Your code to initialize the `btAdapter` and `tgDevice` must first enable the Personalization feature. It must also enable writing to the Android file system. SEE APPENDIX D.

```
public void onCreate(Bundle savedInstanceState) {
    //...
    btAdapter = BluetoothAdapter.getDefaultAdapter();
    if( btAdapter != null ) {
        TGDevice.ekgPersonalizationEnabled = true;

        tgDevice = new TGDevice( btAdapter, handler );
    }
    //...
}
```

APPENDIX D: Enabling writing to the Android File System

For an Android application to be allowed to create files in the local file system a line must be added to your projects AndroidManifest.xml.

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