EEG algorithm SDK for Android: Development Guide

September 21, 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

About the Android SDK

This document will guide you through the process of generating algorithm outputs from different NeuroSky Proprietary Mind Algorithms *using* **NeuroSky EEG Algorithm SDK for Android** *with* EEG data collected by NeuroSky Biosensor System (e.g. TGAM module or MindWave Mobile Headset).

This development guide is intended for *Android application developers* who are already familiar with standard Android development using **Android Studio/Eclipse**. If you are not already familiar with developing for Android, please first visit Android's developer web site for instruction and tools to develop Android apps.

Important: .

- Requires minimum Android API version 16 or later

EEG Algorithm SDK for Android Contents

- EEG Algorithm SDK for Android: Development Guide (this document)
- EEG algorithms descriptions
- EEG Algorithm SDK library: **libs/**
  - *<CPU ARCH>*/libNskAlgo.so (compatible CPU architectures: *arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, x86_64*)
- EEG Algorithm SDK Java interface: **jar/**
  - NskAlgoSdk.jar
- Algo SDK Sample project

Note: .

- The sample project was created with **Android Studio**. However, developers can still use any familiar IDE to start his own Android project
- The minimum requirements for the sample project is **API 19**

Application development

Introduction

We recommend developers to use our COMM SDK for Android in their application. Comm SDK reduces the complexity of managing EEG Algorithm SDK connections and handles data stream parsing. With the help of our SDKs, the application could be as simple as passing the data received and parsed by the Comm SDK to specific function call(s) at Algo SDK. Specific EEG algorithm index would then be returned accordingly.

Important: .

- NeuroSky Comm SDK can only communicate with one paired device at a time.

EEG Algorithm Sample Project

For collecting EEG data from NeuroSky Biosensor System (e.g. MindWave Mobile headset) with Android device, please refer to our COMM SDK for Android document.

Algo SDK Sample is an sample Android application using Communication (Comm) SDK to connect to NeuroSky Biosensor System (e.g. MindWave Mobile headset) and Algorithm (Algo) SDK for algorithmic computation for specific NeuroSky Mind algorithm, including Attention, Meditation, Appreciation, Mental Effort (with secondary algorithm) and Familiarity (with secondary algorithm).

1. Pairing NeuroSky MindWave Mobile Headset with Android device
2. Import the Algo SDK Sample project (gradle build) with Android Studio
3. Select Build –> Rebuild Project to build the "app" and install the "Algo SDK Sample" app by Run Run "app"

Important: .

- The sample project compiles with **Android Studio**. However, the sample code inside still compiles with any other Android application development IDEs (e.g. **Eclipse**)
- The sample project only demonstrates how to iterate with the EEG Algo SDK.
- **Comm SDK** enclosed in the sample project is **version 1.0.4**. Please make sure you are using the latest stable Comm SDK version and make proper changes on sample project if needed.

API Documentation

The **EEG Algorithm SDK API Reference** in this section contains descriptions of the classes and protocols available in the EEG Algorithm Android API.

Data Types

See the **NskAlgoSdk.jar** in the SDK package

```
/* EEG data signal quality definitions */
public enum NskAlgoSignalQuality {
    NSK_ALGO_SQ_GOOD            (0), /* Good signal quality */
    NSK_ALGO_SQ_MEDIUM          (1), /* Medium signal quality */
    NSK_ALGO_SQ_POOR            (2), /* Poor signal quality */
    NSK_ALGO_SQ_NOT_DETECTED    (3); /* No signal detected. It probably is caused by bad sensor
contact */
}

/* SDK state definitions */
public enum NskAlgoState {
    /* SDK state */
    /*
       Algo SDK is initialized (Reason code is omitted),
       host application should never receive this state
    */
    NSK_ALGO_STATE_INITED                   (0x0100),
    /* Algo SDK is performing analysis. */
    NSK_ALGO_STATE_RUNNING                  (0x0200),
    /*
       Algo SDK is collecting baseline data (Reason code is omitted).

       When baseline data collection is done, SDK state should change
       to NSK_ALGO_STATE_RUNNING and start data analysis
     */
    NSK_ALGO_STATE_COLLECTING_BASELINE_DATA  (0x0300),
    /*
       Algo SDK stops data analysis/baseline collection.

       State will only change to stop if previous state is NSK_ALGO_STATE_RUNNING or
       NSK_ALGO_STATE_COLLECTING_BASELINE_DATA
     */
    NSK_ALGO_STATE_STOP                     (0x0400),
    /*
       Algo SDK pauses data analysis due to poor signal quality or paused by user.

       State will only change to pause if previous state is NSK_ALGO_STATE_RUNNING
     */
    NSK_ALGO_STATE_PAUSE                    (0x0500),
    /* Algo SDK is uninitialized (Reason code is omitted) */
    NSK_ALGO_STATE_UNINTIED                 (0x0600),
```

```
    /*
       Algo SDK is analysing provided bulk data (i.e. NSK_ALGO_DataStream()
       is invoked with NSK_ALGO_DATA_TYPE_BULK_EEG.

       Note: SDK state will change to NSK_ALGO_STATE_STOP after analysing data
     */
    NSK_ALGO_STATE_ANALYSING_BULK_DATA       (0x0800),

    NSK_ALGO_STATE_MASK                      (0xFF00),

    /* Reason for state change */
    /* RESERVED */
    NSK_ALGO_REASON_CONFIG_CHANGED           (0x0001),
    /* RESERVED */
    NSK_ALGO_REASON_USER_PROFILE_CHANGED     (0x0002),
    /* RESERVED */
    NSK_ALGO_REASON_CB_CHANGED               (0x0003),
    /* Stopped/Paused by user (i.e. NskAlgoStop()/NskAlgoPause() is invoked) */
    NSK_ALGO_REASON_BY_USER                  (0x0004),
    /* RESERVED */
    NSK_ALGO_REASON_BASELINE_EXPIRED         (0x0005),
    /* RESERVED */
    NSK_ALGO_REASON_NO_BASELINE                    (0x0006),
    /*
       SDK state changes due to signal quality changes.

       e.g. NSK_ALGO_STATE_PAUSE + NSK_ALGO_REASON_SIGNAL_QUALITY means SDK pauses data analysis
due to poor signal quality
       e.g. NSK_ALGO_STATE_RUNNING + NSK_ALGO_REASON_SIGNAL_QUALITY means SDK resumes data analysis
due to signal resuming from poor signal quality
     */
    NSK_ALGO_REASON_SIGNAL_QUALITY           (0x0007),

    /* FOR EVALUATION BUILD ONLY */
    /* SDK has been expired */
    NSK_ALGO_REASON_EXPIRED                  (0x0008),
    /* Evaluation license key error */
    NSK_ALGO_REASON_KEY_ERROR                (0x0009),
    /* Internet connection error */
    NSK_ALGO_REASON_INTERNET_ERROR           (0x000A),
    NSK_ALGO_REASON_MASK                     (0x00FF);
}

/* EEG algorithm type definitions */
public enum NskAlgoType {
    NSK_ALGO_TYPE_INVALID   (0x0000),
    NSK_ALGO_TYPE_AP            (0x0001),        /* Appreciation */
    NSK_ALGO_TYPE_ME        (0x0002),        /* Mental effort */
    NSK_ALGO_TYPE_ME2       (0x0004),   /* Mental effort secondary algorithm */
    NSK_ALGO_TYPE_F         (0x0008),   /* Familiarity */
    NSK_ALGO_TYPE_F2        (0x0010),   /* Familiarity secondary algorithm */
    NSK_ALGO_TYPE_YY        (0x0020),   /* YinYang */
    NSK_ALGO_TYPE_ET        (0x0040),   /* eTensity */
    NSK_ALGO_TYPE_ATT       (0x0100),   /* Attention */
    NSK_ALGO_TYPE_MED       (0x0200),   /* Meditation */
    NSK_ALGO_TYPE_BLINK     (0x0400),   /* Eye blink */
    NSK_ALGO_TYPE_CR        (0x0800),   /* Creativity */
    NSK_ALGO_TYPE_AL        (0x1000),   /* Alertness */
```

```
    NSK_ALGO_TYPE_CP        (0x2000),   /* Cognitive Preparedness */
    NSK_ALGO_TYPE_BP        (0x4000);   /* EEG Bandpower */
}

/* Incoming EEG data type definitions (data from COMM SDK or recorded EEG data) */
public enum NskAlgoDataType {
    NSK_ALGO_DATA_TYPE_EEG      (0x01),     /* Raw EEG data */
    NSK_ALGO_DATA_TYPE_ATT      (0x02),     /* Attention data */
    NSK_ALGO_DATA_TYPE_MED      (0x03),     /* Meditation data */
    NSK_ALGO_DATA_TYPE_PQ       (0x04),     /* Poor signal quality data */
    NSK_ALGO_DATA_TYPE_BULK_EEG (0x05),     /* Bulk of EEG data */
    NSK_ALGO_DATA_TYPE_MAX      (0x06);
}

/* Brain Conditioning Quantification data type definitions */
public enum NskAlgoBCQType {
    NSK_ALGO_BCQ_TYPE_VALUE     (0x01),     /* BCQ algorithm value is available */
    NSK_ALGO_BCQ_TYPE_VALID     (0x02),     /* BCQ validation result is available */
    NSK_ALGO_BCQ_TYPE_BOTH      (0x03);     /* Both BCQ algorithm value and validation result are
available */
}
```

SDK Listener Methods

See the **NskAlgoSdk.jar** in the SDK package.

setOnStateChangeListener

EEG Algo SDK state change notification listener method

```
// Required
public void setOnStateChangeListener(NskAlgoSdk.OnStateChangeListener listener);
```

Note: .

- Developer will always need to check with the SDK state and perform proper GUI handling

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnStateChangeListener(new NskAlgoSdk.OnStateChangeListener() {
    @Override
    public void onStateChange(int state, int reason) {
        Log.i(TAG, "On State Change: " + state + " [reason:" + reason + "]");
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnAPAlgoIndexListener

Appreciation Algorithm index notification listener method

```
// Required
public setOnAPAlgoIndexListener(OnAPAlgoIndexListener listener);
```

Note: .

- Appreciation algorithm has an optional output interval of 1,2,3,4 or 5 seconds
- Algorithm output: 1-Not at all, 2-Low, 3-Medium, and 4-High
- Please refer to EEG algorithm description for the definition and further details of the algorithm.
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnAPAlgoIndexListener(new NskAlgoSdk.OnAPAlgoIndexListener() {
    @Override
    public void onAPAlgoIndex(float value) {
        Log.i(TAG, "NskAlgoAPAlgoIndexListener: AP: " + value);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnMEAlgoIndexListener

Mental Effort Algorithm index notification listener method

```
// Optional
public void setOnMEAlgoIndexListener(OnMEAlgoIndexListener listener);
```

Note: .

- Mental effort algorithm has an optional output interval of 1,2,3,4 or 5 seconds
- Algorithm output: -100 to 100
- Please refer to EEG algorithm description for the definition and further details of the algorithm.
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnMEAlgoIndexListener(new NskAlgoSdk.OnMEAlgoIndexListener() {
    @Override
    public void onMEAlgoIndex(float abs_me, float diff_me, float max_me, float min_me) {
        Log.i(TAG, "NskAlgoMEAlgoIndexListener: ME: abs:" + abs_me + ", diff:" + diff_me + " [" +
min_me + ":" + max_me + "]");
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnME2AlgoIndexListener

Mental Effort Secondary Algorithm index notification listener method

```
// Optional
public void setOnME2AlgoIndexListener(OnME2AlgoIndexListener listener);
```

Note: .

- Please refer to EEG algorithm description for the definition and further details of the algorithm.
- Algorithm indices will be output based on configured output interval (i.e. output rate.) between 30 seconds and 36000 seconds (10 hours) with a step of 1 second (i.e. 30s, 31s, 32s, …, 36000s)
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnME2AlgoIndexListener(new NskAlgoSdk.OnME2AlgoIndexListener() {
    @Override
    public void onME2AlgoIndex(float total_me, float me_rate, float changing_rate) {
        Log.i(TAG, "NskAlgoME2AlgoIndexListener: ME: total ME:" + total_me + ", ME rate:" + me_rate
+ ", Changing rate:" + changing_rate);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnFAlgoIndexListener

Familiarity Algorithm index notification listener method

```
// Optional
public void setOnFAlgoIndexListener(OnFAlgoIndexListener listener);
```

Note: .

- Familiarity algorithm has an optional output interval of 1,2,3,4 or 5 seconds
- Algorithm output: -100 to 100
- Please refer to EEG algorithm description for the definition and further details of the algorithm.
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnFAlgoIndexListener(new NskAlgoSdk.OnFAlgoIndexListener() {
    @Override
    public void onFAlgoIndex(float abs_f, float diff_f, float max_f, float min_f) {
        Log.i(TAG, "NskAlgoMEAlgoIndexListener: F: abs:" + abs_f + ", diff:" + diff_f + " [" +
min_f + ":" + max_f + "]");
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnF2AlgoIndexListener

Familiarity Secondary Algorithm index notification listener method

```
// Optional
public void setOnF2AlgoIndexListener(OnF2AlgoIndexListener listener);
```

Note: .

- For the definition of algorithm indices, please refer to EEG algorithm description.
- Algorithm indices will be output based on configured output interval (i.e. output rate.) between 30 seconds and 36000 seconds (10 hours) with a step of 1 second (i.e. 30s, 31s, 32s, …, 36000s)
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnF2AlgoIndexListener(new NskAlgoSdk.OnF2AlgoIndexListener() {
    @Override
```

```
    public void onF2AlgoIndex(int progress_level, float f_degree) {
        Log.i(TAG, "NskAlgoF2AlgoIndexListener: F: Progress level:" + progress_level + ", F
degree:" + f_degree);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnAttAlgoIndexListener

Attention Algorithm index notification listener method

```
// Optional
public void setOnAttAlgoIndexListener(OnAttAlgoIndexListener listener);
```

Note: .

- Attention algorithm has a fixed output interval of 1 second
- Attention algorithm index ranges from 0 to 100 where higher the attention index, higher the attention level
- In situation when there are more than one algorithm running at the same time (e.g. Attention and Appreciation), algorithm indices from Attention will still be returned when SDK state is **COLLECTING BASELINE**.

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnAttAlgoIndexListener(new NskAlgoSdk.OnAttAlgoIndexListener() {
    @Override
    public void onAttAlgoIndex(int value) {
        Log.i(TAG, "NskAlgoAttAlgoIndexListener: Attention:" + value);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnMedAlgoIndexListener

Meditation Algorithm index notification listener method

```
// Optional
public void setOnMedAlgoIndexListener(OnMedAlgoIndexListener listener);
```

Note: .

- Meditation algorithm has a fixed output interval of 1 second
- Meditation algorithm index ranges from 0 to 100 where higher the meditation index, higher the meditation level
- In situation when there are more than one algorithm running at the same time (e.g. Meditation and Appreciation), algorithm indices from Meditation will still be returned when SDK state is **COLLECTING BASELINE**.

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnMedAlgoIndexListener(new NskAlgoSdk.OnMedAlgoIndexListener() {
    @Override
    public void onMedAlgoIndex(int value) {
        Log.i(TAG, "NskAlgoMedAlgoIndexListener: Meditation:" + value);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnEyeBlinkDetectionListener

Eye blink detection notification listener method

```
// Optional
public void setOnEyeBlinkDetectionListener(OnEyeBlinkDetectionListener listener);
```

Note: .

- No baseline data collection will be needed
- Eye blink strength will be returned once eye blink is detected when Algo SDK state is **RUNNING**

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnEyeBlinkDetectionListener(new NskAlgoSdk.OnEyeBlinkDetectionListener() {
    @Override
    public void onEyeBlinkDetection(int value) {
        Log.i(TAG, "NskAlgoEyeBlinkDetectionListener: Strength:" + value);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
```

```
        });
    }
});
```

setOnBPAlgoIndexListener

EEG Band Power notification listener method

```
// Optional
public void setOnBPAlgoIndexListener(OnBPAlgoIndexListener listener);
```

Note: .

- EEG bandpower algorithm has a fix output rate as 1 second
- Algorithm output unit: dB
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnBPAlgoIndexListener(new NskAlgoSdk.OnBPAlgoIndexListener() {
    @Override
    public void onBPAlgoIndex(float delta, float theta, float alpha, float beta, float gamma) {
        Log.i(TAG, "NskAlgoBPAlgoListener: Delta:" + delta + " Theta:" + theta + " Alpha:" + alpha
+ " Beta:" + beta + " Gamma:" + gamma);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnCRAlgoIndexListener

Creativity notification listener method

```
// Optional
public void setOnCRAlgoIndexListener(OnCRAlgoIndexListener listener);
```

Note: .

- Creativity algorithm value (i.e. cr_value) has an optional output interval of 1,2,3,4 or 5 seconds
- BCQ validation (i.e. type == NSK_ALGO_BCQ_TYPE_VALID or NSK_ALGO_BCQ_TYPE_BOTH) will only be outputed once during the trial
- Algorithm output: -1.0 to 1.0
- Please refer to EEG algorithm description for the definition and further details of the algorithm
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnCRAlgoIndexListener(new NskAlgoSdk.OnCRAlgoIndexListener() {
    @Override
    public void onCRAlgoIndex(int type, float cr_value, boolean bBCQ_Valid) {
        Log.i(TAG, "NskAlgoCRAlgoListener: type:" + type + " cr_value:" + cr_value + " bBCQ_Valid:"
+ bBCQ_Valid);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnALAlgoIndexListener

Alertness notification listener method

```
// Optional
public void setOnALAlgoIndexListener(OnALAlgoIndexListener listener);
```

Note: .

- Alertness algorithm value (i.e. al_value) has an optional output interval of 1,2,3,4 or 5 seconds
- BCQ validation (i.e. type == NSK_ALGO_BCQ_TYPE_VALID or NSK_ALGO_BCQ_TYPE_BOTH) will only be outputed once during the trial
- Algorithm output: -1.0 to 1.0
- Please refer to EEG algorithm description for the definition and further details of the algorithm
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnALAlgoIndexListener(new NskAlgoSdk.OnALAlgoIndexListener() {
    @Override
    public void onALAlgoIndex(int type, float al_value, boolean bBCQ_Valid) {
        Log.i(TAG, "NskAlgoCRAlgoListener: type:" + type + " al_value:" + al_value + " bBCQ_Valid:"
+ bBCQ_Valid);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnCPAlgoIndexListener

Cognitive Preparedness notification listener method

```
// Optional
public void setOnCPAlgoIndexListener(OnCPAlgoIndexListener listener);
```

Note: .

- Cognitive Preparedness algorithm value (i.e. cp_value) has an optional output interval of 1,2,3,4 or 5 seconds
- BCQ validation (i.e. type == NSK_ALGO_BCQ_TYPE_VALID or NSK_ALGO_BCQ_TYPE_BOTH) will only be outputed once during the trial
- Algorithm output: -1.0 to 1.0
- Please refer to EEG algorithm description for the definition and further details of the algorithm
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnCPAlgoIndexListener(new NskAlgoSdk.OnCPAlgoIndexListener() {
    @Override
    public void onCPAlgoIndex(int type, float cp_value, boolean bBCQ_Valid) {
        Log.i(TAG, "NskAlgoCPAlgoListener: type:" + type + " cp_value:" + cp_value + " bBCQ_Valid:"
+ bBCQ_Valid);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnETAlgoIndexListener

eTensity notification listener method

```
// Optional
public void setOnETAlgoIndexListener(OnETAlgoIndexListener listener);
```

Note: .

- eTensity algorithm index (i.e. et_index) has an optional output interval of 1,2,3,4 or 5 seconds
- Algorithm output: 1 to 4
- Please refer to EEG algorithm description for the definition and further details of the algorithm
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnETAlgoIndexListener(new NskAlgoSdk.OnETAlgoIndexListener() {
    @Override
    public void onETAlgoIndex(float et_index) {
        Log.i(TAG, "NskAlgoETAlgoListener: index:" + et_index);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

setOnYYAlgoIndexListener

Yin-Yang notification listener method

```
// Optional
public void setOnYYAlgoIndexListener(OnYYAlgoIndexListener listener);
```

Note: .

- Yin-Yang algorithm index (i.e. yy_index) has an optional output interval of 5 or 10 seconds
- Algorithm output: -1 (Unpleasant emotion), 0 (Neutral emotion) or 1 (Pleasant emotion)
- Please refer to EEG algorithm description for the definition and further details of the algorithm
- **No** algorithm index will be returned until the SDK state equals to **RUNNING/ANALYSING BULK DATA** with EEG data fed

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnYYAlgoIndexListener(new NskAlgoSdk.OnYYAlgoIndexListener() {
    @Override
    public void onYYAlgoIndex(float yy_index) {
        Log.i(TAG, "NskAlgoYYAlgoListener: index:" + yy_index);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

NskAlgoSignalQualityListener

EEG data signal quality notification listener method

```
// Optional
public void setOnSignalQualityListener(OnSignalQualityListener listener);
```

Note: .

- Signal Quality was measured and reported at a fixed output interval of 1 second
- SDK state will be changed from **RUNNING/COLLECTING BASELINE** to **PAUSE** when signal quality is poor or sensor off-head is detected. It would return to its previous state (e.g. **RUNNING**) when the signal quality returns to normal
- There will be no signal quality notification when SDK state is **ANALYSING BULK DATA**

Example

```
NskAlgoSdk nskAlgoSdk = new NskAlgoSdk();

nskAlgoSdk.setOnSignalQualityListener(new NskAlgoSdk.OnSignalQualityListener() {
    @Override
    public void onSignalQuality(int level) {
        Log.i(TAG, "On Signal Quality: " + level);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // change UI elements here
            }
        });
    }
});
```

SDK Utility Methods

NskAlgoInit

Initialize the Algo SDK with supported algorithm type(s).

```
/**
  * @brief    Required: Initialize the Algo SDK with supported algorithm types.
  *
  * @param    type       : Algorithm type(s) (see NskAlgoType)
  * @param    dataPath   : An user data path to store user data
  * @param    licenseKey : License key chain
  * @retval   Return 0 on operation success or else fail
  */
int NskAlgoInit(int algoTypes, String dataPath, String licenseKey);
```

Example 1 - Single algorithm

```
int ret = NskAlgoInit (NSK_ALGO_TYPE_AP, "/User/Documents", "LICENSE_KEY");
ASSERT(ret==0);
```

Example 2 - Multiple algorithms

```
int ret = NskAlgoInit (NSK_ALGO_TYPE_AP+ NSK_ALGO_TYPE_MED, "/User/Documents", "LICENSE_KEY");
ASSERT(ret==0);
```

NskAlgoUninit

Uninitialize the Algo SDK

```
/**
  * @brief    Required: Uninitialize the Algo SDK
  *                 Note: if SDK state is NSK_ALGO_STATE_RUNNING or
NSK_ALGO_STATE_COLLECTING_BASELINE_DATA, then SDK state will change to NSK_ALGO_STATE_STOP with
reason NSK_ALGO_REASON_BY_USER before SDK is uninitialized
  * @retval    Return 0 on operation success or else fail
  */
int NskAlgoUninit ();
```

NskAlgoAlgoVersion

Get the Algo SDK version.

```
/**
  * @brief    Optional: Get the Algo SDK version
  *                     Format: M.m.p, where M is major version, m is minor version and p is patch
version
  *
  * @param    type    : Specify the supported Algo type version to be queried
  * @retval   Null terminated string
  */
String NskAlgoAlgoVersion (int type);
```

Example

```
int ret = NskAlgoInit(NSK_ALGO_TYPE_AP, "/User/Documents");
String apVersionStr = NS_NULL;
ASSERT(ret==0);

apVersionStr = NskAlgoAlgoVersion(NSK_ALGO_TYPE_AP);
if (apVersionStr != NS_NULL) {
        Log.d(TAG, "Appreciation Algo ver.: %s\n", apVersionStr);
}
```

NskAlgoSdkVersion

Get the Algo SDK version.

```
/**
  * @brief    Optional: Get the Algo SDK version
  *                Format: M.m.p, where M is major version, m is minor version and p is patch
version
  *
  * @retval    Null terminated string
  */
String NskAlgoSdkVersion ();
```

Example

```
int ret = NskAlgoInit(NSK_ALGO_TYPE_AP, "/User/Documents");
String sdkVersionStr = null;
ASSERT(ret==0);

sdkVersionStr = NskAlgoSdkVersion();
if (sdkVersionStr != NS_NULL) {
        Log.d(TAG, "EEG Algo ver.: %s\n", sdkVersionStr);
}
```

NskAlgoSetConfig

```
/**
  * @brief    Optional: To configure the supported algorithm type.
  *                Note: output interval for NSK_ALGO_TYPE_ATT and NSK_ALGO_TYPE_MED cannot be
changed for now (always equals to 1 seconds for SDK V3.0)
  *
  * @param    type    : Algo type to be configure
  * @param    config  : Configuration
  * @retval   Return 0 on operation success or else fail
  */
int NskAlgoSetConfig (int type, NskAlgoConfig config);
```

Example

```
// Setting Appreciation index output interval to 5 seconds
int ret;
ret = NskAlgoInit(NSK_ALGO_TYPE_AP, "/User/Documents");
ASSERT(ret==0);

ret = NskAlgoSetConfig(NSK_ALGO_TYPE_AP, new NskAlgoConfig(5));
ASSERT(ret==0);
```

Example: Configure BCQ algorithms (Creativity, Alertness or Cognitive Preparedness)

```
// Setting Creativity index output interval to 5 seconds with light threshold and validate every 30
seconds
int ret;
ret = NskAlgoInit(NSK_ALGO_TYPE_CR, "/User/Documents");
ASSERT(ret==0);

ret = NskAlgoSetConfig(NSK_ALGO_TYPE_CR, new NskAlgoConfig(5,
NskAlgoConfig.NSK_ALGO_BCQ_THRESHOLD_LIGHT, 30));
ASSERT(ret==0);
```

Note: .

- Please refer to specific algorithm description for the specific range of output interval supported, e.g. Attention and Meditation support only output rate = 1s, while Appreciation supports an output rate from 1 to 5s.

NskAlgoDataStream

EEG data stream input from NeuroSky Biosensor System (e.g. TGAM or MindWave Mobile headset).

```
/**
  * @brief    Required: EEG data stream input from NeuroSky Biosensor System (e.g. TGAM or MindWave
Mobile headset)
  *             When type = NSK_ALGO_DATA_TYPE_PQ, dataLength = 1
  *             When type = NSK_ALGO_DATA_TYPE_EEG, dataLength = 512 (i.e. 1 second EEG raw data)
  *             When type = NSK_ALGO_DATA_TYPE_ATT, dataLength = 1
  *             When type = NSK_ALGO_DATA_TYPE_MED, dataLength = 1
  *             When type = NSK_ALGO_DATA_TYPE_BULK_EEG, dataLength = N*512 (i.e. N continous
seconds of EEG raw data)
  *             Note 1: In case of type = NSK_ALGO_DATA_TYPE_BULK_EEG, caller should NOT release
the data buffer until SDK state changes back to NSK_ALGO_STATE_STOP
  *             Note 2: In case of type = NSK_ALGO_DATA_TYPE_BULK_EEG, the first 5 seconds of data
will be used as baseline data
  *
  * @param    type       : Data type
  * @param    data       : Data stream array
  * @param    dataLenght : Size of the data stream
  * @retval   Return 0 on operation success or else fail
  */
int NskAlgoDataStream (int type, short data[], int dataLenght);
```

Note: .

- For the data format from NeuroSky Biosensor System, please refer to TGAM Communication Protocol

Important: .

- There are different data output giving out from NeuroSky Biosensor System.
- EEG Algo SDK handles only the following **4** data output for now. They are:
  - Poor Signal Quality
  - EEG Raw Data
  - Attention
  - Meditation

Example 1 - Handling realtime EEG data

```
public void onDataReceived(int datatype, int data, Object obj) {
    // You can handle the received data here
    // You can feed the raw data to algo sdk here if necessary.
    //Log.i(TAG,"onDataReceived");
    switch (datatype) {
        case MindDataType.CODE_ATTENTION:
            short attValue[] = {(short)data};
            NskAlgoDataStream(NskAlgoDataType.NSK_ALGO_DATA_TYPE_ATT.value, attValue, 1);
            break;
        case MindDataType.CODE_MEDITATION:
            short medValue[] = {(short)data};
            NskAlgoDataStream(NskAlgoDataType.NSK_ALGO_DATA_TYPE_MED.value, medValue, 1);
            break;
        case MindDataType.CODE_POOR_SIGNAL:
            short pqValue[] = {(short)data};
            NskAlgoDataStream(NskAlgoDataType.NSK_ALGO_DATA_TYPE_PQ.value, pqValue, 1);
            break;
        case MindDataType.CODE_RAW:
            raw_data[raw_data_index++] = (short)data;
            if (raw_data_index == 512) {
                NskAlgoDataStream(NskAlgoDataType.NSK_ALGO_DATA_TYPE_EEG.value, raw_data,
raw_data_index);
                    raw_data_index = 0;
            }
            break;
        default:
            break;
    }
}
```

Example 2 - Analysing recorded EEG data (bulk data analysis)

```
Log.d(TAG, "Reading raw data");
try {
    inputStream = assetManager.open("raw_data_em.bin");
    raw_data = readData(inputStream, 512*600);
    inputStream.close();
    NskAlgoDataStream(NskAlgoDataType.NSK_ALGO_DATA_TYPE_BULK_EEG.value, raw_data, 512*600);
} catch (IOException e) {
}
Log.d(TAG, "Finished reading data");
```

NskAlgoStart

Start processing data from NskAlgoDataStream() call.

```
/**
  * @brief    Required: Start processing data from NskAlgoDataStream() call
  *                Note: SDK state will changed to NSK_ALGO_STATE_RUNNING /
NSK_ALGO_STATE_COLLECTING_BASELINE_DATA when previous state is NSK_ALGO_STATE_STOP /
NSK_ALGO_STATE_PAUSE / NSK_ALGO_STATE_INITED
  *
  * @param    bBaseline: Always be false [RESERVED]
  * @retval   Return 0 on operation success or else fail
  */
int NskAlgoStart (boolean bBaseline);
```

Note: .

- SDK state will only change to **RUNNING/COLLECTING BASELINE** by invoking **NSK_ALGO_Start()**

NskAlgoPause

Pause processing/collecting data.

```
/**
  * @brief    Required: Pause processing/collecting data
  *                Note: SDK state will changed to NSK_ALGO_STATE_PAUSE with reason
NSK_ALGO_REASON_BY_USER
  *
  * @retval   Return 0 on operation success or else fail
  */
int NskAlgoPause ();
```

Note: .

- SDK state will change to **PAUSE**
- No algorithm index callback will not be invoked unless **NskAlgoStart()** function is invoked again
- When SDK state is ANALYSING BULK DATA, then **NskAlgoPause()** will always return *non-zero* (i.e. no effect)

NskAlgoStop

Stop processing/collecting data.

```
/**
  * @brief    Required: Stop processing/collecting data.
  *                Note: SDK state will changed to NSK_ALGO_STATE_STOP with reason
NSK_ALGO_REASON_BY_USER
  *
  * @retval   Return 0 on operation success or else fail
  */
int NskAlgoStop ();
```


Note: .

- SDK state will change to **STOP**
- No algorithm index callback will be invoked unless **NskAlgoStart()** method is invoked again
- **NskAlgoStop()** requires recollection of baseline data once restart (Exception for Attention and Meditation) while **NskAlgoPause()** doesn't.

Applications

SDK State Diagram

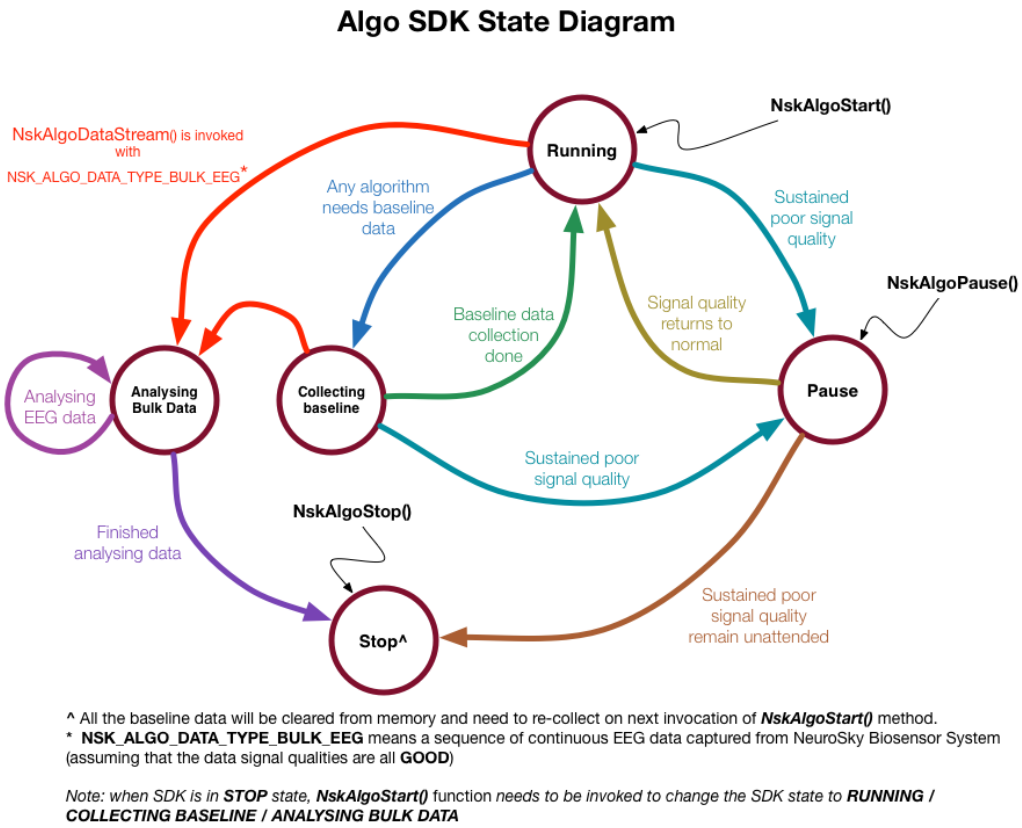

Figure 3.1: Algo SDK state diagram

Application of Appreciation Algorithm

- Selecting Appreciation Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
    1. **Baseline Data Collection**

  - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
  - Suggested baseline data collection protocol

2. **Appreciation Index Computation**
  - Appreciation indexes are computed and returned at a configurable **algorithm output interval**

Note: .

- The default output interval of Appreciation is 1 second, i.e. one new Appreciation index every second (it also represents the minimal output interval)

- Definition of Appreciation index

Appreciation Index	Appreciation or Enjoyment level
1	Not at all
2	Low
3	Medium
4	High

- Please find below the block diagram showing the operation procedures and interaction between EEG data blocks, function calls, algorithm and application outputs in different output rate conditions.

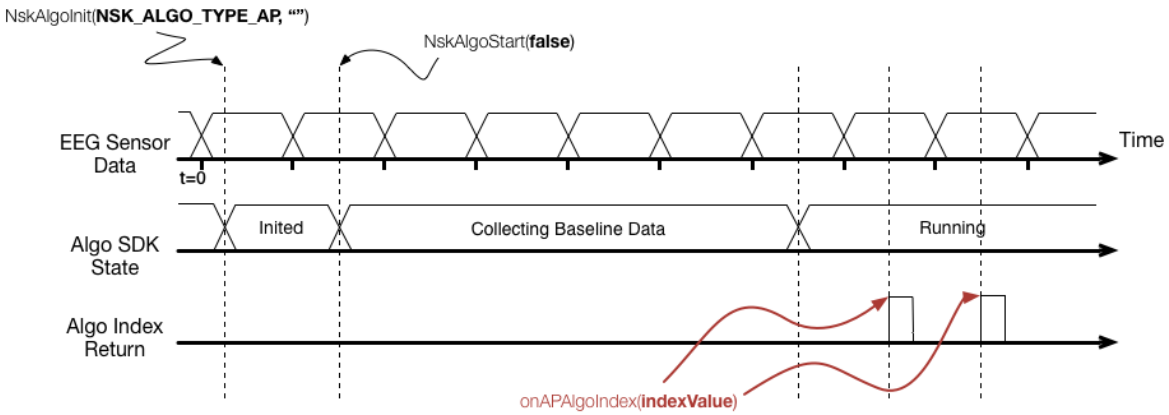

Figure 3.2: Time diagram on getting Appreciation Indices

Note: .

- Please note that user might not have fully engaged in the application (e.g. video watching) in the first few seconds. In such case, it's possible that the obtained algorithm results for that period might not truly reflect user's *expected* mental state induced by the application.

Application of Attention Algorithm

- Selecting Attention Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- Attention index will be returned every 1 second when Algo SDK state is **RUNNING**
- Attention index ranges from **0 to 100**. The higher the index, the higher the attention level

Note: .

- Attention has a fixed output interval of 1 second, i.e. one new Attention index every second
- In the case of selecting multiple algorithms (e.g. Attention and Appreciation), there would still be Attention index returning while Appreciation or other algorithms is collecting baseline data (i.e. Algo SDK state is **COLLECTING BASELINE**)

Application of Meditation Algorithm

- Selecting Meditation Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- Meditation index will be returned every 1 second when Algo SDK state is **RUNNING**
- Meditation index ranges from **0 to 100**. The higher the index, the higher the meditation level

Note: .

- Meditation has a fixed output interval of 1 second, i.e. one new Meditation index every second
- In the case of selecting multiple algorithms (e.g. Meditation and Appreciation), there would still be Meditation index returning while Appreciation or other algorithms is collecting baseline data (i.e. Algo SDK state is **COLLECTING BASELINE**)

Application of Eye Blink Detection

- Selecting Eye Blink Detection by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- Eye blink strength will be returned once eye blink is detected when Algo SDK state is **RUNNING**

Note: .

- No baseline data collection will be needed

Application of EEG Bandpower Algorithm

- Selecting EEG bandpower algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- EEG bandpowers (delta, theta, alpha, beta and gamma in dB) will be returned in 5 seconds after Algo SDK state is **RUNNING** and the EEG bandpower values will be returned in every second

Note: .

- No baseline data collection will be needed

Application of Creativity Algorithm

- Selecting Creativity Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
     - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
     - Suggested baseline data collection protocol
  2. **Creativity Value and Quantification Computation**
     - Creativity values are computed and returned at a configurable **algorithm output interval**
     - Creativity quantification validation are returned at a configurable **algorithm window interval**

Note: .

- The default output interval of Creativity value and quantification is 1 seconds and 30 seconds respectively.

Application of Alertness Algorithm

- Selecting Alertness Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
     - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)

  - Suggested baseline data collection protocol

2. **Alertness Value and Quantification Computation**
   - Alertness values are computed and returned at a configurable **algorithm output interval**
   - Alertness quantification validation are returned at a configurable **algorithm window interval**

Note: .

- The default output interval of Alertness value and quantification is 1 seconds and 30 seconds respectively.

Application of Cognitive Preparedness Algorithm

- Selecting Cognitive Preparedness Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
     - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
     - Suggested baseline data collection protocol
  2. **Cognitive Preparedness Value and Quantification Computation**
     - Cognitive Preparedness values are computed and returned at a configurable **algorithm output interval**
     - Cognitive Preparedness quantification validation are returned at a configurable **algorithm window interval**

Note: .

- The default output interval of Cognitive Preparedness value and quantification is 1 seconds and 30 seconds respectively.

Application of eTensity Algorithm

- Selecting eTensity Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**

  - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
  - Suggested baseline data collection protocol

2. **eTensity Index**
  - eTensity indexes are computed and returned at a configurable **algorithm output interval**

Note: .

- The default output interval of eTensity index is 1 second.

Application of YinYang Algorithm

- Selecting YinYang Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
    - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
    - Suggested baseline data collection protocol
  2. **YinYang Index**
    - YinYang indexes are computed and returned at a configurable **algorithm output interval**

Note: .

- The default output interval of YinYang index is 5 second.

Application of Mental Effort Algorithm

- Selecting Mental Effort Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
    - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
    - Suggested baseline data collection protocol
  2. **Mental Effort Index Computation**

  - Mental Effort index will be output as a group of four values: **Maximum value of Absolute Mental Effort**, **Minimum value of Absolute Mental Effort**, **Absolute Mental Effort** & **Differential Mental Effort**
  - Mental Effort indexes are computed and returned at a configurable **algorithm output interval**

Note: .

- The default output interval of Mental Effort is 1 second, i.e. one new Mental Effort index every second (it also represents the minimal output interval)

- Definition of Mental Effort index: Mental Effort Application Development Guide
- Please find below the block diagram showing the operation procedures and interaction between EEG data blocks, function calls, algorithm and application outputs in different output rate conditions.

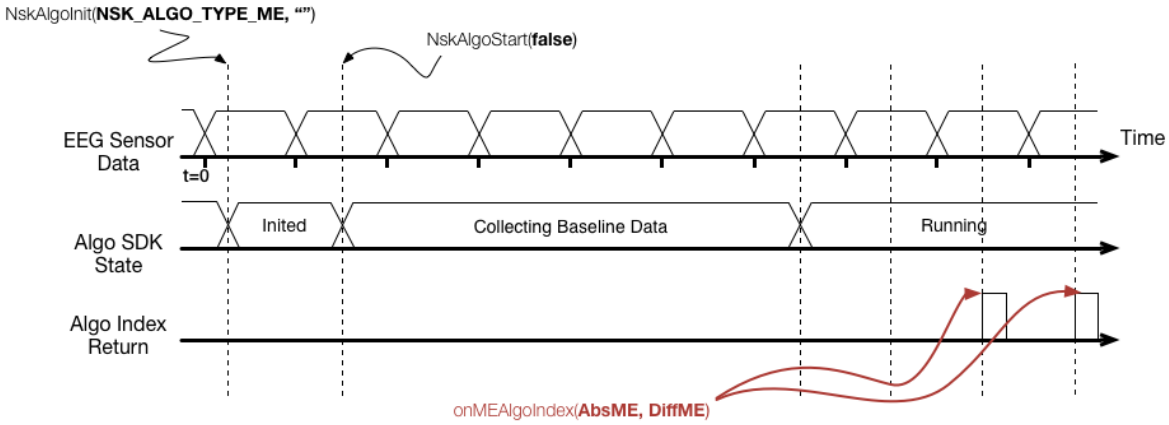

Figure 3.3: Time diagram on getting Mental Effort Indices

Application of Mental Effort Secondary Algorithm

- Selecting Mental Effort Secondary Algorithm (*NSK_ALGO_TYPE_ME2*) by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
     - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
     - Suggested baseline data collection protocol
  2. **Mental Effort Secondary Algorithm Index Computation**

- Mental Effort Secondary Algorithm Indexes consist of three output values: **Total Mental Effort**, **Mental Effort Rate** & **Mental Effort Changing Rate**
- They are computed and returned at a configurable **algorithm output interval**

Note: .

- Mental Effort Secondary Algorithm has a configurable output interval from 30 seconds to 36000 seconds (10 hours) with a step of 1 second (i.e. 30s, 31s, 32s, …, 36000s).

- For the definition of algorithm indices, please refer to EEG algorithm description.
- Please find below the block diagram showing the operation procedures and interaction between EEG data blocks, function calls, algorithm and application outputs in different output rate conditions.

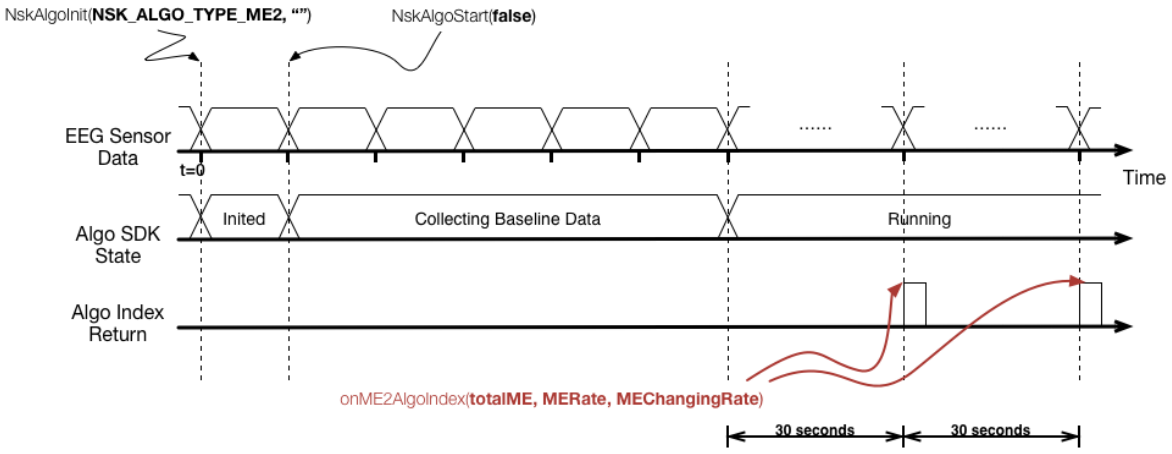

Figure 3.4: Time diagram on getting Mental Effort Indices

Note: .

- Different with Mental Effort and Appreciation that the first valid Mental Effort Secondary Index will **only** be ready after *output interval* seconds after finishing *baseline collection*

Important: .

- There are some special handles on Mental Effort Secondary Algorithm, as it's an extended version of Mental Effort:
  - **Realtime Data Analysis**
    1. selecting ***both*** **Mental Effort** and **Mental Effort Secondary** algorithms, **Mental Effort** algorithm will always with an output rate of **5 seconds** and **Mental Effort Secondary** algorithm will be output based on the configured output interval
    2. selecting ***only*** **Mental Effort Secondary** algorithm, ***only*** **Mental Effort Secondary** Index will be output based on the corresponding output interval
  - **Bulk Data Analysis (offline mode)**
    1. selecting ***both*** **Mental Effort** and **Mental Effort Secondary** algorithms, **Mental Effort** algorithm will always with an output rate of **5 seconds** and **Mental Effort Secondary** algorithm will be output based on the configured output interval
    2. selecting ***only*** **Mental Effort Secondary** algorithm, ***only*** **Mental Effort Secondary** Index will be output based on the corresponding output interval

Application of Familiarity Algorithm

- Selecting Familiarity Algorithm by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
     - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
     - Suggested baseline data collection protocol
  2. **Familiarity Index Computation**
     - Familiarity index will be output as a group of four values: **Maximum value of Absolute Familiarity**, **Minimum value of Absolute Familiarity**, **Absolute Familiarity** & **Differential Familiarity**
     - Familiarity indexes are computed and returned at a configurable **algorithm output interval**

Note: .

- The default output interval of Familiarity is 1 second, i.e. one new Familiarity index every second (it also represents the minimal output interval)

- Definition of Familiarity index: Familiarity Application Development Guide

- Please find below the block diagram showing the operation procedures and interaction between EEG data blocks, function calls, algorithm and application outputs in different output rate conditions.

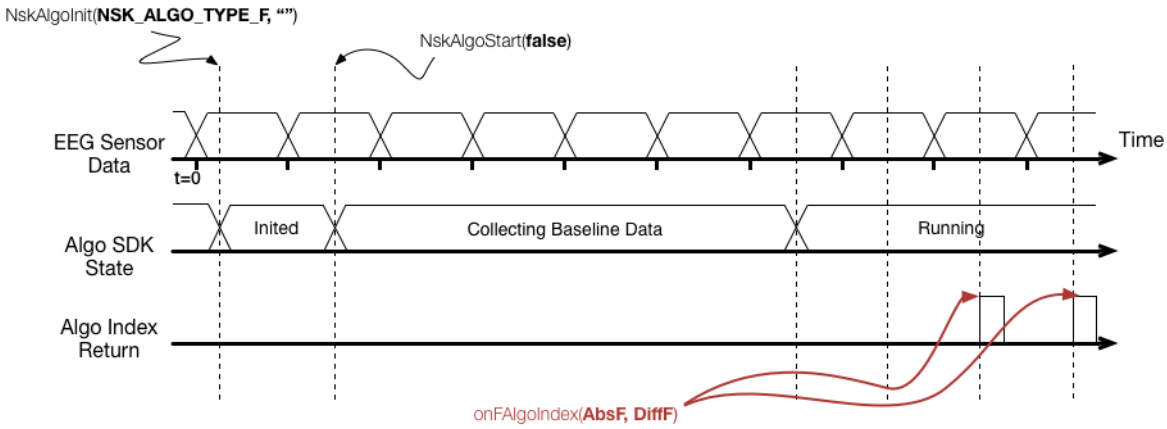

Figure 3.5: Time diagram on getting Familiarity Indices

Application of Familiarity Secondary Algorithm

- Selecting Familiarity Secondary Algorithm (*NSK_ALGO_TYPE_F2*) by invoking **NskAlgoInit()** method
- Starting EEG data analysis by invoking **NskAlgoStart()** method
- **Procedure**
  1. **Baseline Data Collection**
     - When previous Algo SDK state is **INITED/STOPPED**, then the first 5s EEG raw data will be used as baseline data (i.e. SDK state will be **COLLECTING BASELINE**)
     - Suggested baseline data collection protocol
  2. **Familiarity Secondary Algorithm Index Computation**
     - Familiarity Secondary Algorithm Indexes consist of three output values: **Progress Level** & **Familiarity Degree**
     - They are computed and returned at a configurable **algorithm output interval**

Note: .

- Familiarity Secondary Algorithm has a configurable output interval from 30 seconds to 36000 seconds (10 hours) with a step of 1 second (i.e. 30s, 31s, 32s, …, 36000s).

- For the definition of algorithm indices, please refer to EEG algorithm description.

- Please find below the block diagram showing the operation procedures and interaction between EEG data blocks, function calls, algorithm and application outputs in different output rate conditions.

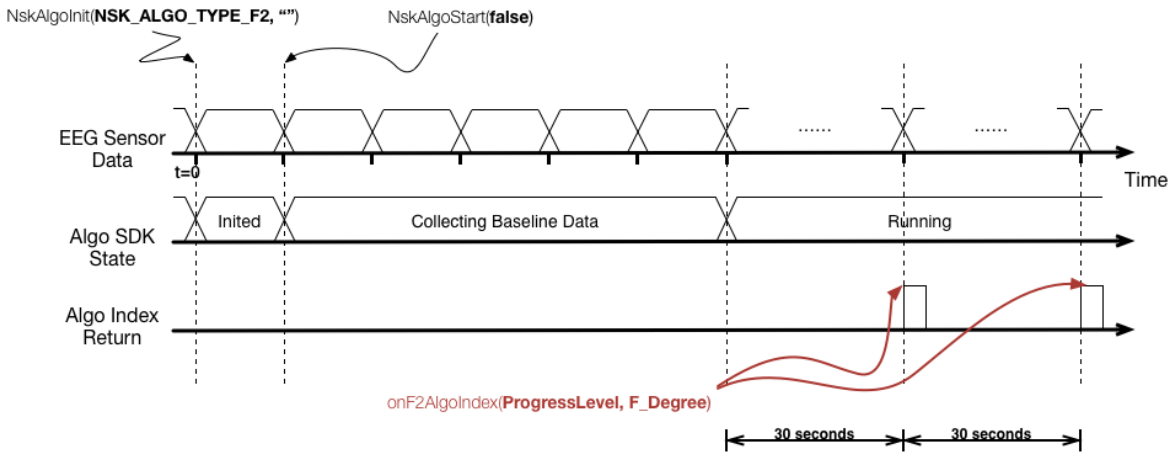

Figure 3.6: Time diagram on getting Familiarity Indices

Note: .

- Similar as Mental Effort Secondary algorithm that the first valid Familiarity Secondary Index will **only** be ready after *output interval* seconds after finishing *baseline collection*

Important: .

- There are some special handles on Familiarity Secondary Algorithm, as it's an extended version of Familiarity:
  - **Realtime Data Analysis**
    1. selecting ***both*** **Familiarity** and **Familiarity Secondary** algorithms, **Familiarity** algorithm will always with an output rate of **5 seconds** and **Familiarity Secondary** algorithm will be output based on the configured output interval
    2. selecting ***only*** **Familiarity Secondary** algorithm, ***only*** **Familiarity Secondary** Index will be output based on the corresponding output interval
  - **Bulk Data Analysis (offline mode)**
    1. selecting ***both*** **Familiarity** and **Familiarity Secondary** algorithms, **Familiarity** algorithm will always with an output rate of **5 seconds** and **Familiarity Secondary** algorithm will be output based on the configured output interval
    2. selecting ***only*** **Familiarity Secondary** algorithm, ***only*** **Familiarity Secondary** Index will be output based on the corresponding output interval

SDK Operations

Pause and Resume

- Assuming SDK is in **RUNNING/COLLECTING BASELINE** state
- Pausing EEG algorithm data analysis by invoking **NskAlgoPause()** method
- Resuming EEG algorithm data analysis by invoking **NskAlgoStart()** method

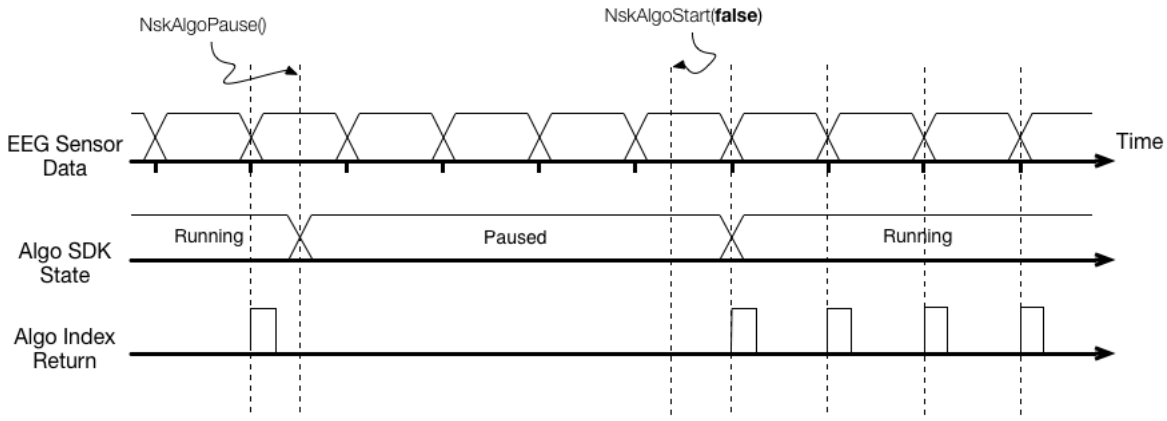

Figure 3.7: Time diagram on Pause/Resume SDK

Note: .

- There will be no effect on **NskAlgoPause()** when previous SDK state is not **RUNNING/COLLECTING BASELINE**
- When SDK state is **ANALYSING BULK DATA**, then **NskAlgoPause()** will always return non-zero (i.e. no effect)

Stop and Start

- Assuming SDK is in **RUNNING/COLLECTING BASELINE/ANALYSING BULK DATA** state
- Stopping EEG algorithm data analysis by invoking **NskAlgoStop()** method
- Restart EEG algorithm data analysis by invoking **NskAlgoStart()** method

Note: .

- SDK will need to re-collect baseline data after **NskAlgoStop()** (i.e. SDK state will change to **COLLECTING BASELINE** right after invoking **NskAlgoStart()** method)

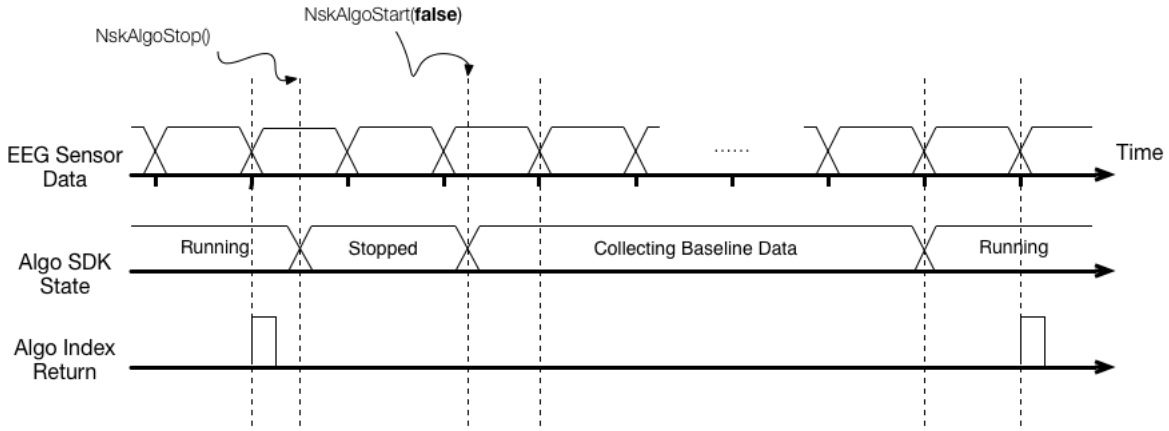

Figure 3.8: Time diagram on Stop/Start SDK

Note: .

- There will be no effect on **NskAlgoStop()** when previous SDK state is not **RUNNING/COLLECTING BASELINE/ANALYSING BULK DATA**

Customize Algorithm Output Interval

- Algorithm output interval can be configured at any time once SDK has been initialized
- The new configured output interval will become effective based on last index returned

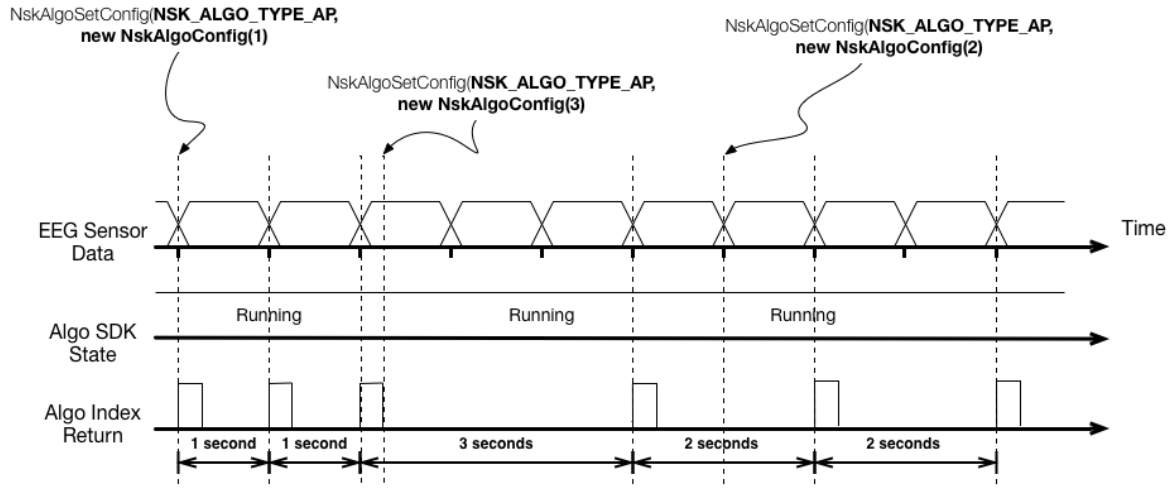

Figure 3.9: Time diagram on configuring algorithm output interval

Note: .

- Different algorithm may have different **minimum/default** output interval

Bulk EEG Data Analysis

Bulk EEG data analysis allows the application to perform **offline** analysis of recorded EEG data.

Below shows difference between handling realtime (directly from NeuroSky Biosensor System) and offline (recorded EEG data from NeuroSky Biosensor System) EEG raw data:

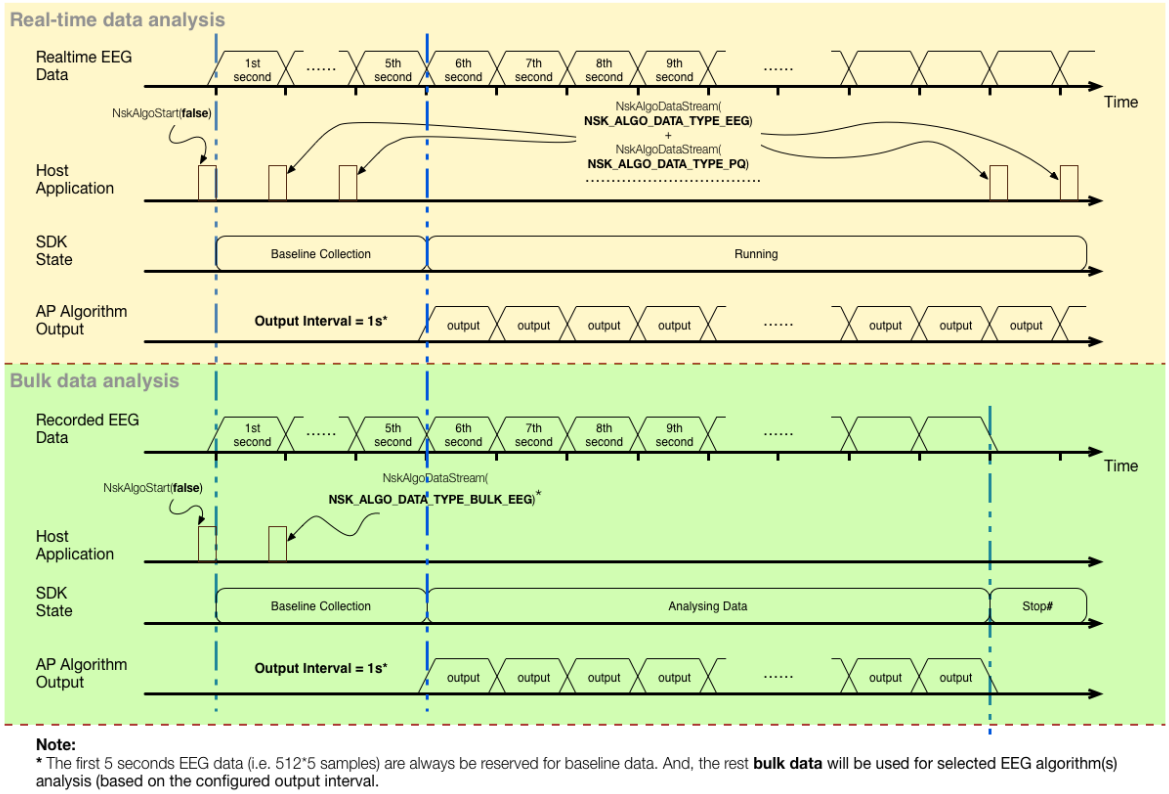

Figure 3.10: Time diagram on analysing EEG in real-time / offline

Note: .

- Only Appreciation, Mental Effort, Mental Effort Secondary, Familiarity and Familiarity Secondary algorithm support *bulk EEG data analysis*

Frequently Asked Questions