MWM Comm SDK for iOS

December 27, 2017

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

NeuroSky MWM Development Guide

Introduction

This guide will teach you how to use **NeuroSky MWM(MindWave Mobile) Comm SDK for iOS** to write iOS applications that can acquire bio-signal data from NeuroSky's Hardware. This will enable your iOS apps to receive and use bio-signal data such as EEG acquired via Classic BlueTooth and Bluetooth low energy as a Stream.

This guide (and the entire **NeuroSky MWM Comm SDK for iOS** for that matter) is intended for programmers who are already familiar with standard iOS development using Xcode and Apple's iOS SDK. If you are not already familiar with developing for iOS, please first visit Apple's web site for instruction and tools to develop iOS apps.

If you are already familiar with creating typical iOS apps, then the next step is to make sure you have downloaded NeuroSky's **MWM Comm SDK for iOS**. Chances are, if you're reading this document, then you already have it.

MWM Comm SDK for iOS Contents

- Development Guide (this document)
- API Reference (this document)
- SDK static library and headers
  - libMWMSDK.a
  - MWMDelegate.h
  - MWMDevice.h
  - MWMEnum.h
- MWMSample example project for iOS

You'll find the "libMWMSDK.a" in the `libn/` folder, and the "MWMSample example project" in the `SampleProject/` folder.

Supported NeuroSky Hardware

The following NeuroSky hardware are currently supported:

- MindWave Mobile - TGAM for Mind (EEG)
- MindWave Mobile plus
- BrainLink

Important: Before using any iOS application that uses the MWM Comm SDK for iOS, make sure you have paired the NeuroSky Hardware to your iOS device by carefully following the instructions in the User Manual that came with each NeuroSky Hardware!

Supported iOS Version

- Support iOS Version 7.0 later

SDK Features

- Support Automatic Reference Counting in this release.

Your First Project: HelloMWMiOS

Important: Apple has announced that simulator not support External Accessory frameworks. Testing External Accessory applications will require access to a real iOS device going forward. For how to set up a test environment using real iOS device, please visit iOS Developer Library: App Distribution Guide.

HelloMWMiOS is a sample project we've included in the **MWM Comm SDK for iOS** that demonstrates how to setup, connect, and handle data to a NeuroSky Hardware. Add the project to your Xcode environment by following these steps:

1. In Xcode, select **File** —> **Open** —>
2. Browse in the MWM SDK to select the SampleProject directory
3. Click the Open button
4. Update the code signing options in the project target settings
5. Select **Product** —> **Run** to compile, link and start HelloMWMiOS in the Xcode.

Note: This is an example application. It may not be completely compliant with Apple's guidelines for building deploy-able applications.

At this point, you should be able to browse the code, make modifications, compile, and deploy the app to your device just like any typical iOS application.

Develop Your Own NeuroSky Hardware Enabled Apps for iOS

For most applications, using the MWM Comm SDK for iOS API is recommended. It reduces the complexity of managing NeuroSky Hardware's accessory connections and handles parsing of the data stream from NeuroSky Hardware's accessory. To make a brainwave-sensing application, all you need to do is to import a library, add the requisite setup and tear-down functions, and assign a delegate object to which accessory event notifications will be dispatched.

Some limitations of the MWM Comm SDK for iOS API include:

- Can only communicate with one attached NeuroSky Hardware Enabled accessory

The "MWM API Reference" contains descriptions of the classes and protocols available in the MWM iOS API.

The MWM Comm SDK for iOS also includes the `MWMSample` sample project, which is a simple demo iOS application that displays the connection with NeuroSky Hardware.

Configure Your Environment

In order for you app to communicate with any NeuroSky hardware module, you must include the `Supported external accessory protocols` key in your app's `Info.plist` file.

This key contains an array of strings that identify the communications protocols that your app supports.

Add `com.neurosky.thinkgear` and `com.neurosky.cardio` to the list of supported external accessory protocols.

Your project window should now look similar to this:

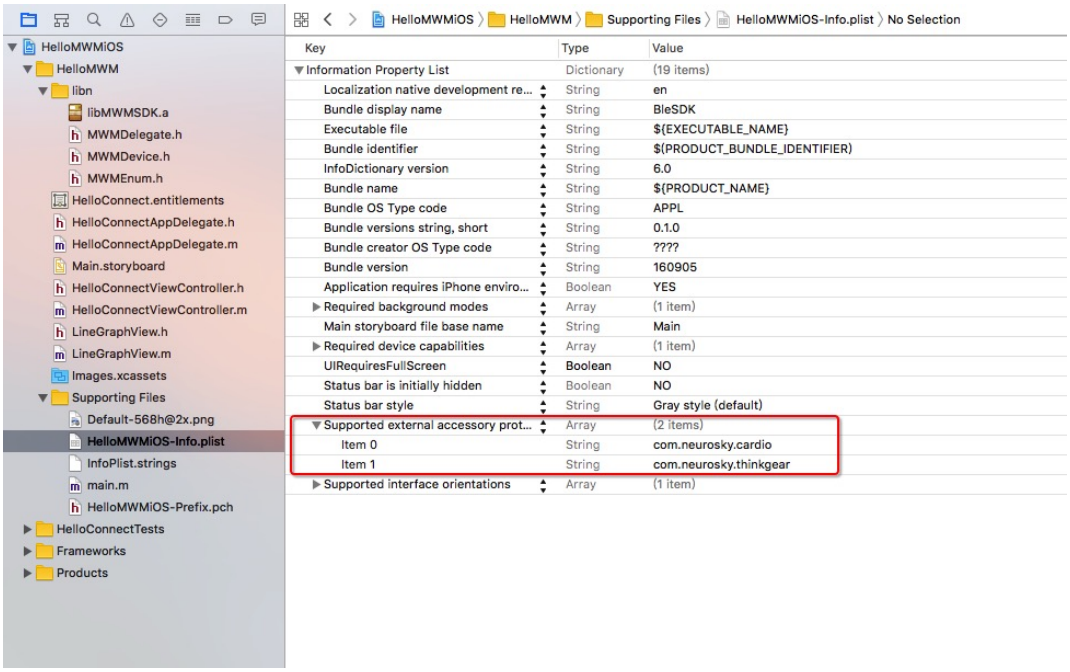

Figure 1.1: Add Supported External Accessory

Copy the following directories from the `lib` directory in the SDK for iOS into the `MWMSDK` group in your project:

- `libMWMSDK.a`
- `MWMDelegate.h`
- `MWMDevice.h`
- `MWMEnum.h`

Your project window should now look similar to this:

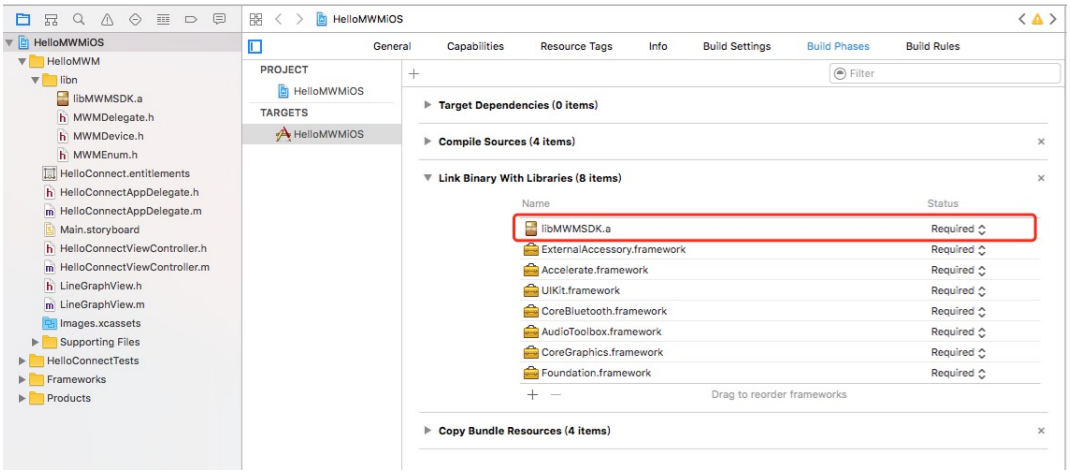

Figure 1.2: Copy MWM SDK iOS library to project

Next, add the ExternalAccessory frameworks to the project.

1. Navigate to your project settings
2. Select your target
3. Select Build Phases
4. Expand **Link Binary With Libraries**
5. Click on + and select `libMWMSDK.a` and click **Add**

Your project window should now look similar to this:

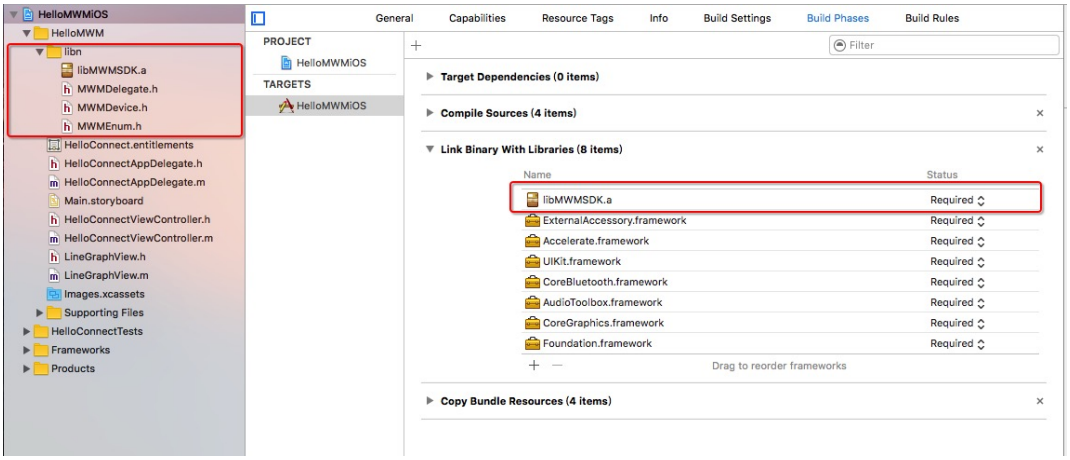

Figure 1.3: Add frameworks to project

Then import the appropriate header files into the requisite classes.

Set Up the MWMDevice

Set up the `MWMDevice` should be performed as early as necessary. Typically, this would be in the `viewDidLoad` method in the `ViewController` class. Simply add the following two lines to that method:

```
mwDevice = [ MWMDevice sharedInstance];
[ mwDevice setDelegate: self];
```

This sets up the shared `MWMDevice` instance. The delegate can be set to any class that implements the `MWMDelegate` protocol — in this case, it's an instance of `ViewController`.

Handle Data Received

Since the delegate object was set to be a `ViewController` instance, we have to edit its class definition to indicate support of the `MWMDelegate` protocol. In the sample project file, the class definition in `ViewController.h` looks similar to the following:

```
@interface ViewController : UIViewController
```

Simply modify the definition in the following way:

```
@interface ViewController : UIViewController<MWMDelegate>
```

And in the implementation file, implement the method. A few `NSLog` calls are provided as a trivial example. Check the "Stream API Reference" for a full list of the supported data.

```
-(void)eegSample:(int) sample;

-(void)eSense:(int)poorSignal Attention:(int)attention Meditation:(int)meditation;

-(void)eegPowerDelta:(int)delta Theta:(int)theta LowAlpha:(int)lowAlpha HighAlpha:(int)highAlpha;

-(void)eegPowerLowBeta:(int)lowBeta HighBeta:(int)highBeta LowGamma:(int)lowGamma
MidGamma:(int)midGamma;
```

Handle Accesssory Connection and Disconnection

The `MWMDelegate` protocol also specifies `didConnect:` and `didDisconnect` for the delegate object to handle accessory connection and disconnection . Add the following method definitions to the header file:

```
-(void)didConnect;
```

In the implementation file, implement these methods:

```
-(void)didDisconnect;
```

Connect and Disconnect the Device

When your application is ready to receive the EEG data, call the `connectDevice:` method. In the sample project, this is done by click button `Select`.

```
-(void)connectDevice:(NSString *)deviceID;
```

You will also need a matching call to `disconnectDevice` click button `Disconnect`.

```
//disconnect
-(void)disconnectDevice;

//Device found
-(void)deviceFound:(NSString *)devName MfgID:(NSString *)mfgID DeviceID:(NSString *)deviceID;
```

Further Considerations

- Provide a consistant user experience by adhering to the guidelines set by the NeuroSky Developer Application Standards document.

Configure Your Environment

You'll first need to add the External Accessory framework to your Xcode project. See the previous section about configuring your environment on how to add the ExternalAccessory.framework.

NeuroSky MWM API Reference

Overview

The `MWMDevice` class handles connections between a NeuroSky Hardware accessory and an iOS device.

Configuration

Get the Shared Stream Manager

- `+ (MWMDevice *)sharedInstance`

Get SDK Version

- `-(NSString *) getVersion`

Scan Device

- `-(void)scanDevice`

Stop Scan Device

- `-(void)stopScanDevice`

Connect or Disconnect HardWare

- `-(void)connectDevice:(NSString *)deviceID`
- `-(void)disconnectDevice`

Configure the Notch Filter

- `-(void)writeConfig:(TGMWMConfigCMD)cmd`
- `-(void)readConfig`

Note: These two methods are disabled in MindWave Mobile and BrainLink. But they are enabled in MindWave Mobile plus. Please confirm your device before using them.

Record log

- `-(void)enableConsoleLog:(BOOL)enabled`
- `-(NSString *)enableLoggingWithOptions:(unsigned)option`
- `-(void)stopLogging`

Property

delegate

The object that acts as the delegate of the MWMDevice.

```
@property (nonatomic, weak) id<MWMDelegate> delegate;
```

Note

The delegate receives notifications about changes to the status of the NeuroSky Hardware Enabled accessory, as well as data received notifications. The delegate must adopt the `MWMDelegate` protocol.

Class Methods

sharedInstance

Return the shared `MWMDevice` object for the iOS-based device.

```
+ (MWMDevice *)sharedInstance
```

Return Value

The shared `MWMDevice` object.

Note

You should always use this method to obtain the `MWMDevice` object, rather than creating an instance directly.

Note: MWMDevice instance is Singleton. So please use this method the get instance to make sure that there's only one instance in current program.

Instance Methods

getVersion

Return the version string of MWMDevice.

```
-(NSString *) getVersion;
```

Calling this to get the version of current SDK to make sure the update of it.

scanDevice

Call this method to scan the connected BT devices and broadcasting BLE devices.

```
-(void)scanDevice;
```

This method can be used for scanning both BT and BLE devices. All of them will be returned by the delegate **-(void)deviceFound:(NSString*)devName MfgID:(NSString*)mfgID DeviceID:(NSString*)deviceID;** method. You can connect to headset by deviceID in this method.

Note: While scanning MindWave Mobile, please make sure that the device has been paired and connected. For the MindWave Mobile plus, please power on first and then the device will be founded by **deviceFound:** delegate method. In one word, connect to the device with its way.

stopScanDevice

Call this method to stop scanning BLE devices.

```
-(void)stopScanDevice;
```

This method can be used for stopping scanning BLE devices.

Note: This method only works on stop scanning BLE Devices. Not for BT ones.

connectDevice

Call this method to connect MindWave device by device ID.

```
-(void)connectDevice:(NSString *)deviceID;
```

Call this method to connect to headset Device by deviceID which can be got by the **-(void)deviceFound:** delegate method.

Note: Take it easy to connect to MindWave Mobile or MindWave Mobile plus with this method. The SDK can recognize the type of Headset Bluetooth module.

disconnectDevice

Call this method to disconnect MindWave device from iOS Device.

```
-(void)disconnectDevice;
```

writeConfig

Call this method to set notch filter for MindWave device.

```
-(void)writeConfig:(TGMWMConfigCMD)cmd;
```

readConfig

Call this method to get current baud rate and notch filter for MindWave device.

```
-(void)readConfig;
```

enableConsoleLog

Calling the enable or disable the log output while debugging project with Xcode.

```
-(void)enableConsoleLog:(BOOL)enabled;
```

enableLoggingWithOptions

Calling this method to log data from Headset into files with options.

```
-(void)enableLoggingWithOptions:(unsigned)option;

typedef NS_ENUM(NSUInteger, LoggingOptions){
    LoggingOptions_Raw  = 1,
    LoggingOptions_Processed = 1 << 1,
};

 [[MWMDevice sharedInstance] enableLoggingWithOptions:LoggingOptions_Processed |
LoggingOptions_Raw];
```

stopLogging

Call this method to disable enableLoggingWithOptions method and stop logging data from Headset into files. .

```
-(void)stopLogging
```

Note: Disable logging, will prevent new files from being created. But files that are currently open may continue to be written until they are closed by the SDK.

Enum

Declared in `MWMEnum.h`

TGMWMConfigCMD

```
typedef NS_ENUM(NSUInteger, TGMWMConfigCMD){
    TGMWMConfigCMD_ChangeNotchTo_50,
    TGMWMConfigCMD_ChangeNotchTo_60
};
```

TGBleExceptionEvent

```
typedef NS_ENUM(NSUInteger, TGBleExceptionEvent){
    TGBleUnexpectedEvent = 0,
    TGBleConfigurationModeCanNotBeChanged = 1,
    TGBleFailedOtherOperationInProgress = 2,
    TGBleConnectFailedSuspectKeyMismatch = 3,
    TGBlePossibleResetDetect = 4,
    TGBleNewConnectionEstablished = 5,
    TGBleStoredConnectionInvalid = 6,
    TGBleConnectHeadSetDirectoryFailed = 7,
    TGBleBluetoothModuleError = 8,
    TGBleNoMfgDatainAdvertisement = 9,
};
```

LoggingOptions

```
typedef NS_ENUM(NSUInteger, LoggingOptions){
    LoggingOptions_Raw  = 1,
    LoggingOptions_Processed = 1 << 1,
};
```

MWM Delegate Protocol Reference

Overview

The `MWMDeviceDelegate` protocol defines methods for handling accessory event notifications dispatched from a `MWMDevice` object.

Protocol Definition

Device found call back

- `- deviceFound:`

Connection trigger call back

- `- didConnect:`
- `- didDisconnect:`

Sample raw data call back

- `- eegSample:`

Emotion Sense call back

- `- eSense:`

EEGPower data call back

- `- eegPowerDelta:`
- `- eegPowerLowBeta:`

Hardware configuration call back

- `- mwmBaudRate:`

BLE Exception Event call back

- `- exceptionMessage:`

Instance Methods

deviceFound:

Tells the delegate that device was found.

```
-(void)deviceFound:(NSString *)devName MfgID:(NSString *)mfgID DeviceID:(NSString *)deviceID;
```

Parameters

- `devName` — name of device
- `mfgID` — mfgID of Device
- `deviceID` — device id used for connecting device

Note: Both MindWave Mobile and MindWave Mobile plus can be returned by this method if founded.

didConnect:

Tells the delegate that device was connected.

```
-(void)didConnect;
```

didDisconnect:

Tells the delegate that device was disconnected.

```
-(void)didDisconnect;
```

eegSample:

Tells the delegate that raw data was received from Hardware.

```
-(void)eegSample:(int) sample;
```

Parameters

- `sample` — raw data from device

eSense:

Tells the delegate that emotion sense data was received from Hardware.

```
-(void)eSense:(int)poorSignal Attention:(int)attention Meditation:(int)meditation;
```

Parameters

- `poorSignal` — poor signal value of device. 0: good signal, 200: bad signal
- `attention` — attention value from device
- `meditation` — meditation value from device

eegPower

Tells the delegate that EEG Power data was received from the accessory

```
-(void)eegPowerDelta:(int)delta Theta:(int)theta LowAlpha:(int)lowAlpha HighAlpha:(int)highAlpha;

-(void)eegPowerLowBeta:(int)lowBeta HighBeta:(int)highBeta LowGamma:(int)lowGamma
MidGamma:(int)midGamma;
```

Parameters

- `delta` — the delta EEG power band
- `theta` — the theta EEG power band
- `lowAlpha` — the lowAlpha EEG power band
- `highAlpha` — the highAlpha EEG power band
- `lowBeta` — the lowBeta EEG power band
- `highBeta` — the highBeta EEG power band
- `lowGamma` — the lowGamma EEG power band
- `middleGamma` — the middleGamma EEG power band

mwmBaudRate:

Tells the delegate that configuration was received from the accessory

```
-(void)mwmBaudRate:(int)baudRate NotchFilter:(int)notchFilter;
```

Parameters

- `baudRate` — value of baud rate. 0:57600, 1: 115200
- `notchFilter` — value of notch filter. 0: 50Hz, 1: 60Hz

exceptionMessage:

Tells the delegate that what BLE Exception Message happens.

```
-(void)exceptionMessage:(TGBleExceptionEvent)eventType;
```

Parameters

- `eventType` — type of BLE Exception Message