Unity3DBrainLinkProSDK V1.0.0

Date: 0902 2021

Author: ZiDian

Zhu

SDK Version: 1.0.0

MCU: 3.1

Table of Contents

Unity3D BrainLinkProSDK Development Manual

Introduction

This guide will guide you how to use the Unity3D BrainLinkProSDK to get brainwave data from Macrotellect's hardware. This will allow your mobile application to receive and use brainwave data such as BLEMIND and BLEGRAVITY, and you can get them via Bluetooth, Macrotellect's hardware, and the file resource Unity3D BrainLinkProSDK.

Function:
Receive brain wave data.

The file contains:

- API reference (this document)
- Asset/iOS
- Asset/Android

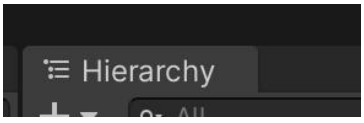
Supported hardware devices:

- Data format with battery life
- BrainLink_Pro

Supported iOS/Android version:
iOS 9.0 + / Android SDK 26+

- Supported Unity 3D version:
  Unity2019 +

Unity3D using method

- Create new gameObject“ThinkGearManager”,
- 

- Add ThinkGearMnanger.cs script
- Detailed method ---Demo scene and Demo.cs script

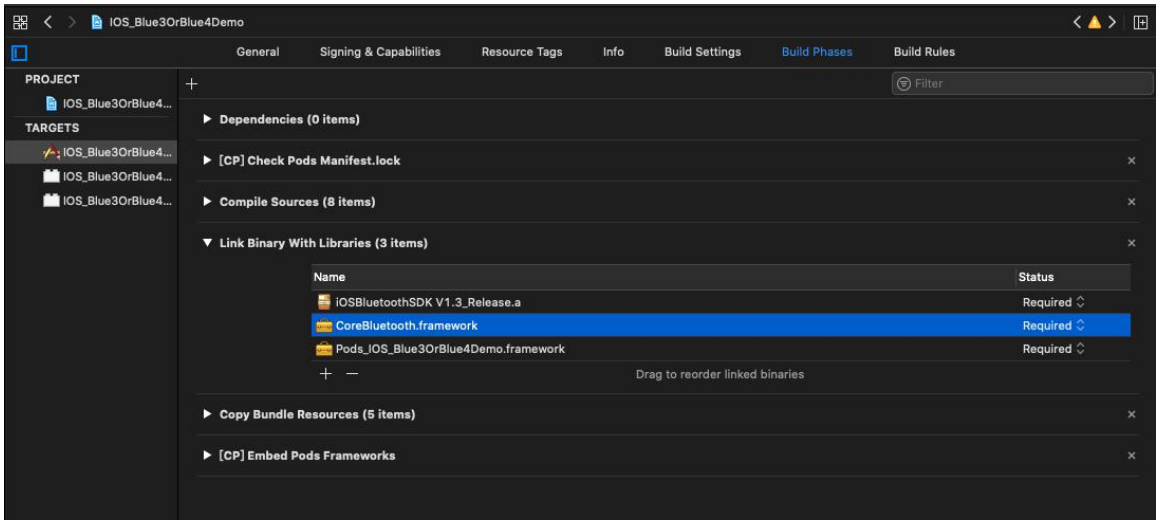
iOS configuration：

Step 1:

1.1 in Xcode project TARGETS — Build Phases import IOS system framework as following

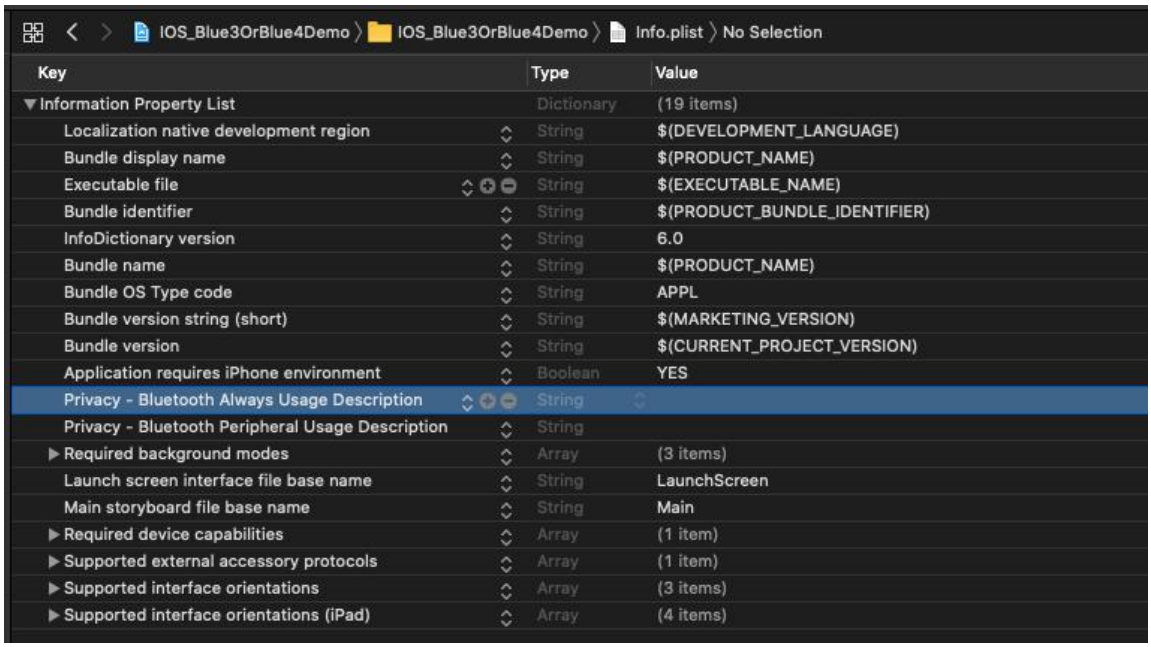
- CoreBluetooth.framework

As picture ：

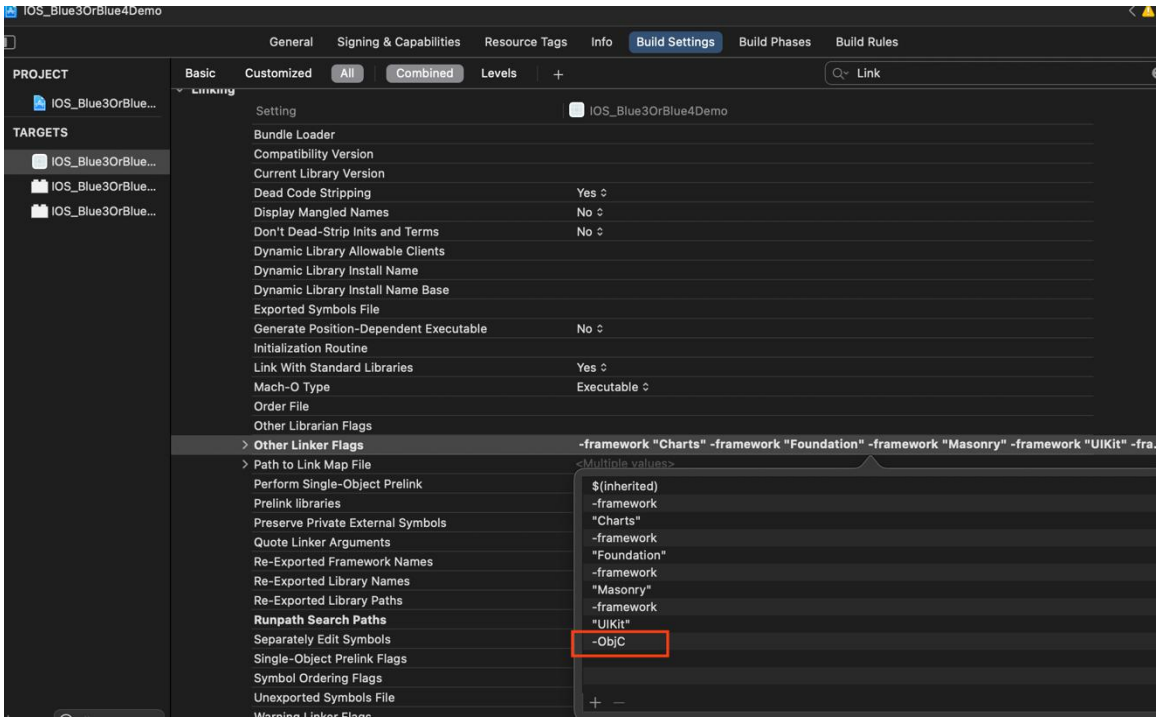


add Bluetooth permission to Info.plist （IOS13 Need to add Bluetooth permission Privacy — Bluetooth Always Usage Description，Privacy — Bluetooth Peripheral Usage Descriptio）

As picture ：

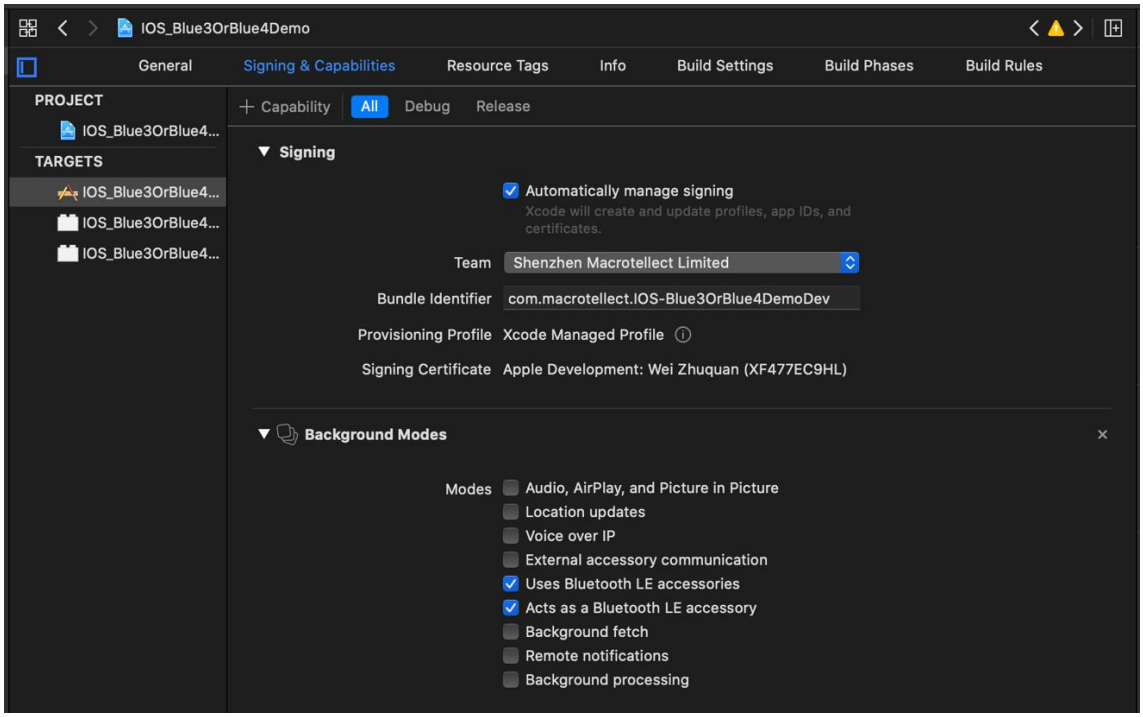


TARGETS Build Settingslink———>add from “Other Linker Flags ： —ObjC



1.2 If you want Bluetooth to run in the backend, please set as follows, you don’t need to set it if you don’t need it

As picture：



Step 2:

IN UnityAppController.mm file

Import header file

```
#import "Blue4Manager.h"
```

Add method at the end of the file

```
extern "C"{
    void Connect()
    {
        // Bluetooth connection settings
        [Blue4Manager logEnable:YES];

                                                    [[Blue4Manager shareInstance]
configureBlueNames:@[@"BrainLink_Pro",@"BrainLink_Lite",@"BrainLink",@"BrainLink_Lite_P",@"BrainL
ink_Lite",@"ROYWOS",@"BrainLink_Pink"] ableDeviceSum:1];
        // Bluetooth connection successful
        [Blue4Manager shareInstance].blueConBlock = ^(NSString *markKey) {
            if ([markKey isEqualToString:@"1"]) {
                // Determine the connected device
                NSLog(@"A Device Bluetooth connection is successful");
                UnitySendMessage("ThinkGearManager", "ReceiveContentState", "yes");
            }
        };
```

```
        // Bluetooth disconnection callback
        [Blue4Manager shareInstance].blueDisBlock =    ^(NSString *markKey){
            if ([markKey isEqualToString:@"1"]) {
                // Determine the connected device
                NSLog(@"A Device Bluetooth disconnect");
                UnitySendMessage("ThinkGearManager", "ReceiveContentState", "no");
//                    UnitySendMessage("ThinkGearManager", "ReceiveBlueToothType", "");
            }
        };

        //Data callback of the first device (A) Data callback of other devices as hzlblueDataBlock_B
与 hzlblueDataBlock_A's same coding method
        [Blue4Manager shareInstance].hzlblueDataBlock_A = ^(HZLBlueData *blueData, BlueType conBT,
BOOL isFalseCon) {
            if (conBT == BlueType_Pro) {
                if (blueData.bleDataType == BLEMIND) {
                    // If the signal value is 0, the Bluetooth device is worn
                    // Note: If the Bluetooth device is connected but not worn, the signal value
is greater than 0 and less than or equal to 200
//                    UnitySendMessage("ThinkGearManager", "ReceiveBlueToothType",
"4_0");

                    UnitySendMessage("ThinkGearManager", "ReceivePoorSignal", [[NSString
stringWithFormat:@"%d",blueData.signal] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveBatteryCapacity",
[[NSString                    stringWithFormat:@"%d",blueData.batteryCapacity]
cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveAttention", [[NSString
stringWithFormat:@"%d",blueData.attention] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveMeditation", [[NSString
stringWithFormat:@"%d",blueData.meditation] cStringUsingEncoding:NSUTF8StringEncoding]);

                    UnitySendMessage("ThinkGearManager", "ReceiveDelta", [[NSString
stringWithFormat:@"%d",blueData.delta] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveTheta", [[NSString
stringWithFormat:@"%d",blueData.theta] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveLowAlpha", [[NSString
stringWithFormat:@"%d",blueData.lowAlpha] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveHighAlpha", [[NSString
stringWithFormat:@"%d",blueData.highAlpha] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveLowBeta", [[NSString
stringWithFormat:@"%d",blueData.lowBeta] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveHighBeta", [[NSString
stringWithFormat:@"%d",blueData.highBeta] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveLowGamma", [[NSString
stringWithFormat:@"%d",blueData.lowGamma] cStringUsingEncoding:NSUTF8StringEncoding]);
```

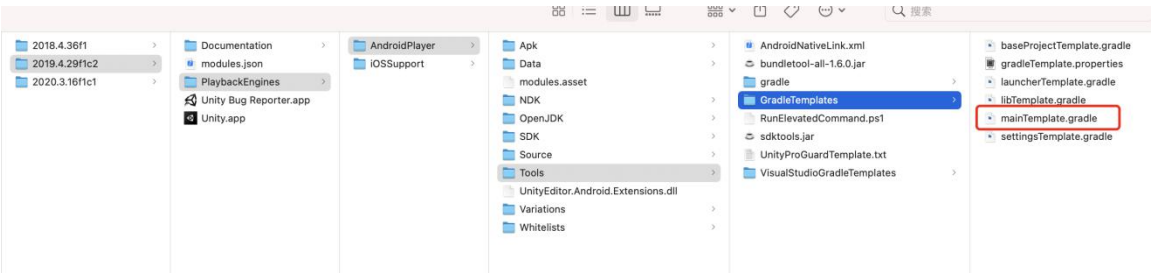
```
                    UnitySendMessage("ThinkGearManager", "ReceiveHighGamma", [[NSString
stringWithFormat:@"%d",blueData.highGamma] cStringUsingEncoding:NSUTF8StringEncoding]);

                    UnitySendMessage("ThinkGearManager", "ReceiveHeaetRate", [[NSString
stringWithFormat:@"%d",[blueData.heartRate intValue]] cStringUsingEncoding:NSUTF8StringEncoding]);

                    UnitySendMessage("ThinkGearManager", "ReceiveTemperature", [[NSString
stringWithFormat:@"%f",[blueData.temperature floatValue]] cStringUsingEncoding:NSUTF8StringEncoding]);

                    UnitySendMessage("ThinkGearManager", "ReceiveGrind4_0", [[NSString
stringWithFormat:@"%d",[blueData.grind intValue]] cStringUsingEncoding:NSUTF8StringEncoding]);

                    UnitySendMessage("ThinkGearManager", "ReceiveAp4_0", [[NSString
stringWithFormat:@"%d",blueData.ap] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveHardwareversion4_0",
[blueData.hardwareVersion cStringUsingEncoding:NSUTF8StringEncoding]);
                }
                else if (blueData.bleDataType == BLEGRAVITY) {
                    UnitySendMessage("ThinkGearManager", "ReceiveXValue", [[NSString
stringWithFormat:@"%d",blueData.xvlaue] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveYValue", [[NSString
stringWithFormat:@"%d",blueData.yvlaue] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveZValue", [[NSString
stringWithFormat:@"%d",blueData.zvlaue] cStringUsingEncoding:NSUTF8StringEncoding]);
                }
                else if (blueData.bleDataType == BLERaw) {
                    UnitySendMessage("ThinkGearManager", "ReceiveRawdata", [[NSString
stringWithFormat:@"%d",blueData.raw] cStringUsingEncoding:NSUTF8StringEncoding]);
                }
            }

            else if (conBT == BlueType_Jii){
                if (blueData.bleDataType == BLEMIND) {
                    UnitySendMessage("ThinkGearManager", "ReceiveAttention", [[NSString
stringWithFormat:@"%d",blueData.attention] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveMeditation", [[NSString
stringWithFormat:@"%d",blueData.meditation] cStringUsingEncoding:NSUTF8StringEncoding]);
                    UnitySendMessage("ThinkGearManager", "ReceiveBatteryCapacity",
[[NSString stringWithFormat:@"%d",blueData.batteryCapacity]
cStringUsingEncoding:NSUTF8StringEncoding]);

                    UnitySendMessage("ThinkGearManager", "ReceiveBlutToothType", "4_0");
                }
            }
            else if (conBT == BlueType_Lite) {
//                  UnitySendMessage("ThinkGearManager", "ReceiveBlueToothType",
"");
```

```
                    if (blueData.bleDataType == BLEMIND) {
                        UnitySendMessage("ThinkGearManager", "ReceivePoorSignal", [[NSString
stringWithFormat:@"%d",blueData.signal] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveAttention", [[NSString
stringWithFormat:@"%d",blueData.attention] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveMeditation", [[NSString
stringWithFormat:@"%d",blueData.meditation] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveBatteryCapacity",
[[NSString stringWithFormat:@"%d",0] cStringUsingEncoding:NSUTF8StringEncoding]);

                        UnitySendMessage("ThinkGearManager", "ReceiveDelta", [[NSString
stringWithFormat:@"%d",blueData.delta] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveTheta", [[NSString
stringWithFormat:@"%d",blueData.theta] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveLowAlpha", [[NSString
stringWithFormat:@"%d",blueData.lowAlpha] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveHighAlpha", [[NSString
stringWithFormat:@"%d",blueData.highAlpha] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveLowBeta", [[NSString
stringWithFormat:@"%d",blueData.lowBeta] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveHighBeta", [[NSString
stringWithFormat:@"%d",blueData.highBeta] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveLowGamma", [[NSString
stringWithFormat:@"%d",blueData.lowGamma] cStringUsingEncoding:NSUTF8StringEncoding]);
                        UnitySendMessage("ThinkGearManager", "ReceiveHighGamma", [[NSString
stringWithFormat:@"%d",blueData.highGamma] cStringUsingEncoding:NSUTF8StringEncoding]);
                    }
                    else if (blueData.bleDataType == BLERaw) {
                        UnitySendMessage("ThinkGearManager", "ReceiveRawdata", [[NSString
stringWithFormat:@"%d",blueData.raw] cStringUsingEncoding:NSUTF8StringEncoding]);
                    }
                }

                if (isFalseCon) {
                    NSLog(@"A Fake device connection");
                }
            };
        [[Blue4Manager shareInstance] connectBlue4];



}

void disConnect(){
    [[Blue4Manager shareInstance] disConnectBlue4];
}
```

```
}
```

Android Configuration：

Find in the unity installation directory mainTemplate.gradle



Add：implementation 'androidx.appcompat:appcompat:1.1.0'

Unity3D BrainLinkProSDK V1.0.0 iOS API Reference

HZLBlueData Reference

Overview

This class is the data model

Enum

```
typedef enum : NSUInteger {
    BlueType_NO = 0,
    BlueType_Lite,
/* Connected to BrainLink_Lite data format device, there are BLEMIND, BLERaw type data */
    BlueType_Pro,
/* Connected to BrainLink_Pro data format device, with BlemIND, BLEGRAVITY, BLERaw type data */
    BlueType_Jii,
/* Connected is Jii */
}BlueType;

typedef NS_ENUM(NSUInteger,BLEDATATAYPE){
        BLEMIND = 0,            // Brainwave data
        BLEGRAVITY,             // Gravity data
        BLERaw,                  // Raw blink data
};
```

Brainwave data：

- signal, Device wearing quality

- attention，Attention Level
- meditation，Relaxation Level
- delta，
- theta，
- lowAlpha，
- highAlpha，
- lowBeta，
- highBeta，
- lowGamma，
- highGamma，
- ap，Appreciation Level
- batteryCapacity，Battery capacity percentage
- hardwareVersion，Device firmware version
- grind
- grind　　Blink
- temperature　Temperature
- heartrate　　Heart Rate

Gravity data：

- xvlaue，
- yvlaue，
- zvlaue

Raw Blink Data

- raw，
- blinkeye

Annotation：

Connect Jii，Only signal，attention，meditation，batteryCapacity，ap

Connect BrainLink_Lite，Only signal，attention，meditation，delta，theta，lowAlpha，highAlpha，lowBeta，highBeta，lowGamma，highGamma，raw，blinkeye

Instructions of some Instance Property

- signal：Signal Value。When the signal is 0，it means that it has been worn，when the signal value is greater than 0 and less than or equal to 200，it means that the hardware and the mobile phone have been connected via Bluetooth
- batteryCapacity：Battery capacity percentage
- ap：Appreciation Level
- hardwareVersion：hardware version. The first version value is 255，when you update the hardware successfully，the version value of the hardware will become smaller
- xvlaue：　X axis value of gravity sensor swing forward and backward pitch angle
- yvlaue：　Y axis value of gravity sensor Swing left and right Yaw angle
- zvlaue：　Z axis value of gravity sensor Wing swing roll angle

Blue4Manager Reference

Overview

This class deals with the interaction between Macrotellect hardware and Bluetooth devices

Instance Property

Callback for successful Bluetooth connection

@property(nonatomic,copy)Blue4Connect blueConBlock;

Bluetooth disconnection callback

@property (nonatomic,copy) BlueConnectdismiss blueDisBlock;

Note: Bluetooth devices are A B C D E F in the order of connection 。

Use the above method, for example, there are 6 data callbacks (hzlblueDataBlock_A,hzlblueDataBlock_B.....), In order to ensure the independence of data, data between various devices can be accepted at the same time without affecting each other.

Up to 6 Bluetooth 4.0 devices can be connected, and 6 can be connected but it is difficult to connect successfully。

If you want to use a single connection, the input parameter of ableDeviceSum is 1, just call hzlblueDataBlock_A。

Data callback of each device

@property(nonatomic,copy)Blue4DataBlock hzlblueDataBlock_A;
@property(nonatomic,copy)Blue4DataBlock hzlblueDataBlock_B;
@property(nonatomic,copy)Blue4DataBlock hzlblueDataBlock_C;
@property(nonatomic,copy)Blue4DataBlock hzlblueDataBlock_D;
@property(nonatomic,copy)Blue4DataBlock hzlblueDataBlock_E;
@property(nonatomic,copy)Blue4DataBlock hzlblueDataBlock_F;

Connection status of each device

@property (nonatomic,assign)BOOL connected_A;
@property (nonatomic,assign)BOOL connected_B;
@property (nonatomic,assign)BOOL connected_C;
@property (nonatomic,assign)BOOL connected_D;
@property (nonatomic,assign)BOOL connected_E;
@property (nonatomic,assign)BOOL connected_F;

Method

Whether to print log or not by default

+ (void)logEnable:(BOOL)enable;

Initialization (singleton)

+ (instancetype)shareInstance;

Connection configuration

Parameter Description:

blueNames: The name of the device that can be connected (Bluetooth 4.0 device)

NSArray *blueNames = @[@"BrainLink",@"BrainLink_Pro",@"jii@jii-***"];

1.jii@jii- Indicates that the device name with jii- prefix can be connected. There is jii@ which means it is a jii device @ The following is the device name *** means that the prefix is the same

```
/*! @ brief connection configuration (only for Macrotellect internal testing)
    appSoleCode: app Unique code
    defaultBlueNames: An array of default connectable Bluetooth names
    ableDeviceSum: Number of Bluetooth devices that can be connected
    result: the name of the device that can be connected when"back"
-(void)configureBlueNamesWithAppSoleCode:(NSString *)appSoleCode defaultBlueNames:(NSArray *)defaultBlueNames ableDeviceSum:(int)ableDeviceSum result:(void(^)(NSArray*))result;
*/
```

ableDeviceSum: Number of Bluetooth devices that can be connected

```
-(void)configureBlueNames:(NSArray *)blueNames ableDeviceSum:(int)deviceSum
```

Connect a Bluetooth device

```
-(void)connectBlue4;
```

Disconnect the Bluetooth device

```
-(void)disConnectBlue4;
```

Manually test fake connections (Fakeonnection definition: When signal is equal to 0 and the 10 consecutive values of attention and medition remain unchanged, it is considered a fakeonnection, The SDK will disconnect the Bluetooth connection of the current device and automatically connect again)

```
-(void)testAFalseCon:(BOOL)isTest; // Manually test the fake connection of A device
-(void)setTestToZero;// Cancel all manual test fake connections
```

Unity3D BrainLinkProSDK V1.0.0 Android API Reference

UnityThinkGear.cs script

SetBLLinstenner(string objectName) this method initiate detection,parameter is mount receive call back method's script game object ,in this demo, it is ThinkGearManager,use ConnectBluetooth()connection method after initiate detection

The callback method is in the ThinkGearManager.cs script ReceiveXX