# Unity3DBrainLinkProSDK V1.0.5

Date: 0402 2023

Author: Liang Fang

SDK Version: 1.0.5

MCU: 4.201

**Update record:**

1. V1.0.5 Manual scan

2. V1.0.4 Added the switch of Ap and angle value

3. V1.3.3 Added new Bluetooth data fields HRV

4. V1.0.1 Android SDK optimization, adding detailed development steps

5. V1.0.0 Unity3D Mobile SDK

**Table of Contents**

[Unity3DBrainLinkProSDK V1.0.5 1](#_Toc131342008)

[Unity3D BrainLinkProSDK Development Manual 4](#_Toc131342009)

[Introduction 4](#_Toc131342010)

[iOS configuration: 5](#_Toc131342011)

[Android Configuration: 13](#_Toc131342012)

[Unity3D BrainLinkProSDK V1.0.5 iOS API Reference 16](#_Toc131342013)

[HZLBlueData Reference 16](#_Toc131342014)

[Blue4Manager Reference 18](#_Toc131342015)

[Unity3D BrainLinkProSDK V1.0.5 Android API Reference 20](#_Toc131342016)

## 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

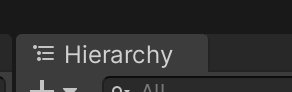
Supported iOS/Android version:

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

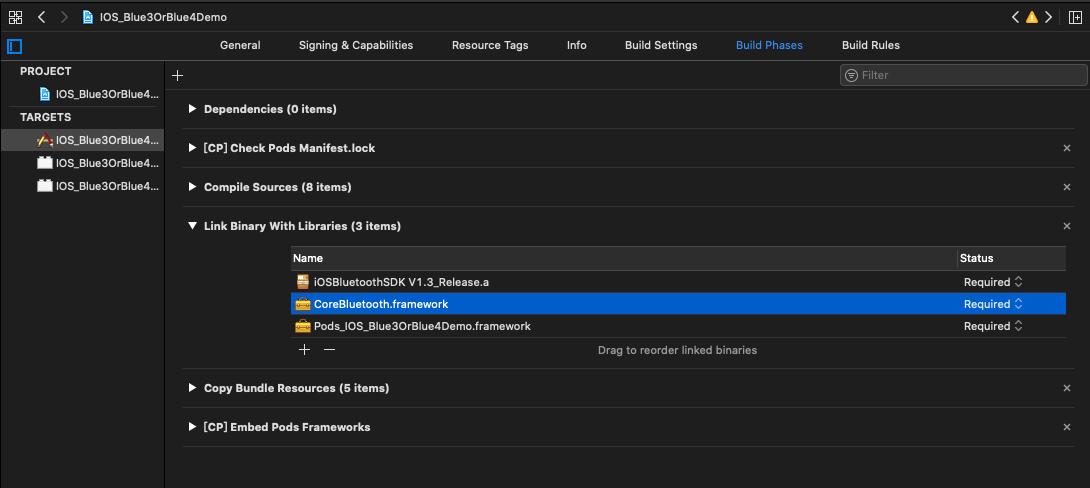
### iOS configuration:

**Step 1:**

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

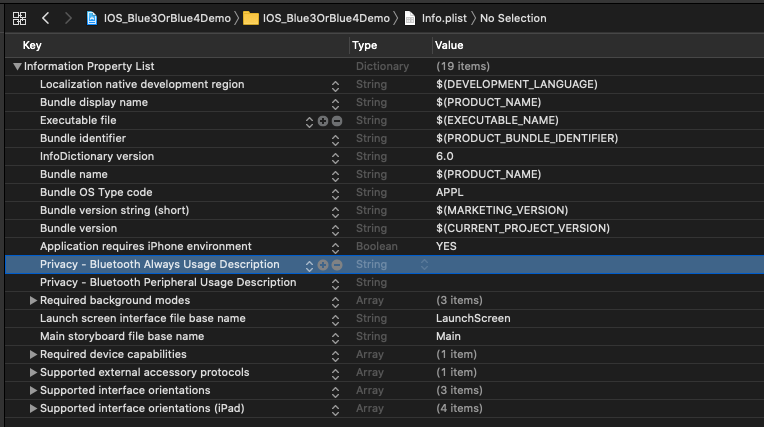
* CoreBluetooth.framework

As picture ：

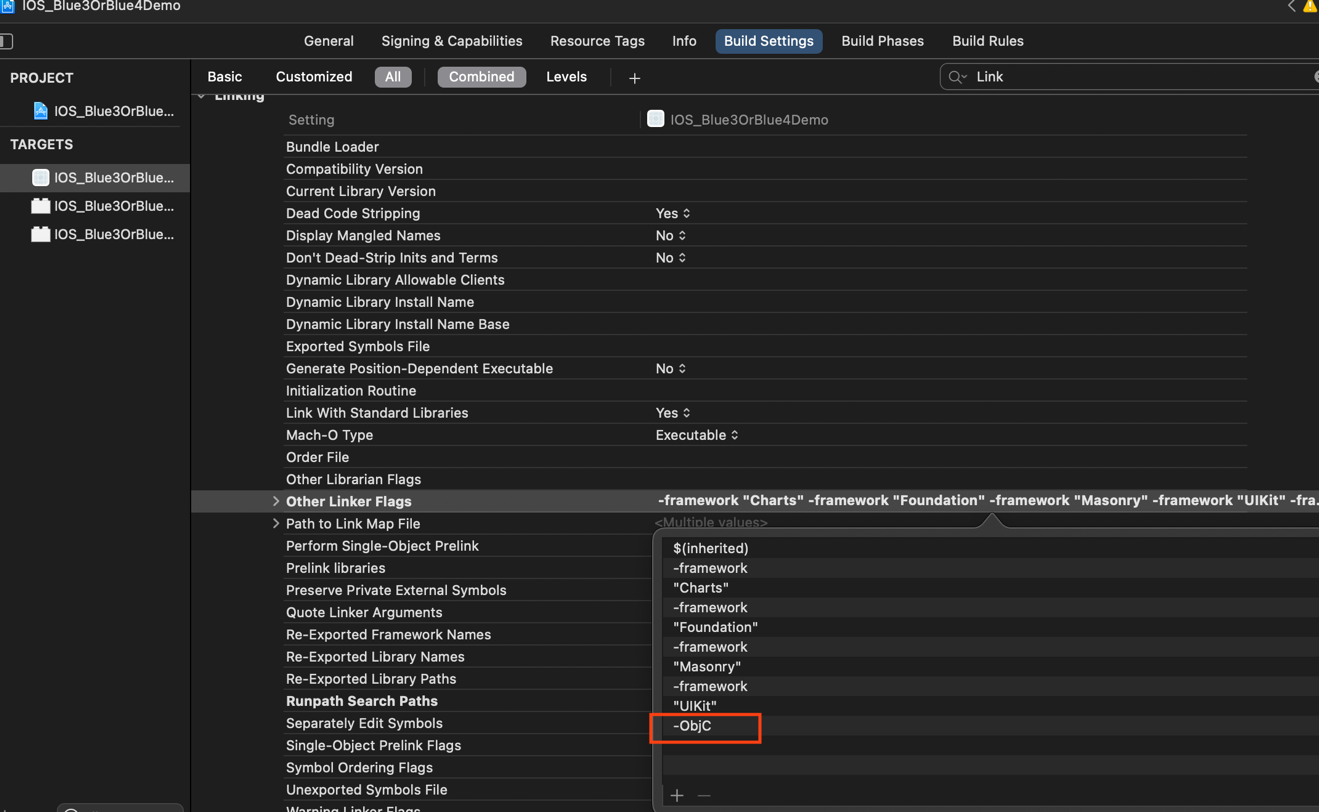
****

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

As picture ：

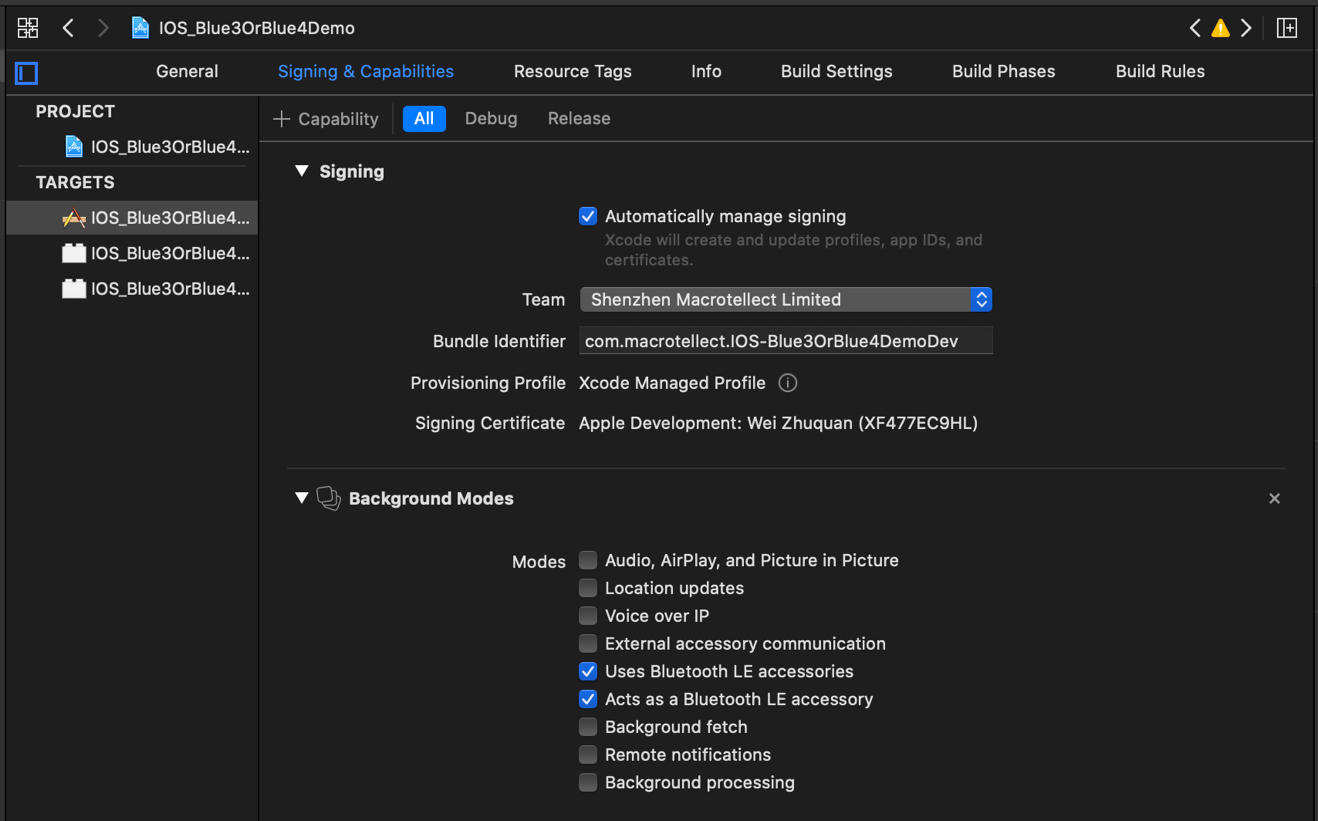
****

All TARGETS Build Settingslink--->add from “Other Linker Flags ：-ObjC，Bitcode is set to No



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 SendSettings(char \*settings){

NSData \*data = [[NSString stringWithCString:settings encoding:NSUTF8StringEncoding] dataUsingEncoding:NSUTF8StringEncoding];

[[Blue4Manager shareInstance] writeToProAWithCode:data];

}

void SetWhiteList(char \*whiteList){

\_refreshDevices = [NSMutableArray array];

NSString \*list = [NSString stringWithCString:whiteList encoding:NSUTF8StringEncoding];

NSArray \*blueNames = [list componentsSeparatedByString:@","];

// Bluetooth connection settings

[Blue4Manager logEnable:YES];

[[Blue4Manager shareInstance] configureBlueNames:blueNames ableDeviceSum:1];

[[Blue4Manager shareInstance] bluePermission:^(BluePermission\_state state, NSString \*log) {

NSLog(@"BluePermission\_state====%@",log);

if (state == Blue4\_Unauthorized) {

}

else if (state == Blue4\_PoweredOn) {

[\_refreshDevices removeAllObjects];

[[Blue4Manager shareInstance] scanBlue4WithScannedWithScanTime:12 blue4DeviceBlock:^(ScannedDevice \*blue4Device) {

BOOL isFind = false;

for (ScannedDevice \*ss in \_refreshDevices) {

if ([ss.peripheral.identifier.UUIDString isEqualToString: blue4Device.peripheral.identifier.UUIDString]) {

isFind = true;

}else{

}

}

if(!isFind){

[\_refreshDevices addObject:blue4Device];

NSLog(@"==name===%@", blue4Device.name);

NSLog(@"==identifier===%@", blue4Device.peripheral.identifier);

}

NSString \*nameIdentifierRiss = [NSString stringWithFormat:@"%@,%@,%d",blue4Device.name,blue4Device.peripheral.identifier,blue4Device.RSSI.intValue];

UnitySendMessage("ThinkGearManager", "DeviceFound", [nameIdentifierRiss cStringUsingEncoding:NSUTF8StringEncoding]);

}];

}

else if (state == Blue4\_PoweredOff) {

}

}];

[Blue4Manager shareInstance].endScanBlue4Block = ^{

NSLog(@"scan is over");

};

// Determine the connected device

NSLog(@"A Device Bluetooth connection is successful");

  // 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]);

NSString \*hrvStr = @"";

if (blueData.HRV != nil) {

for (int i = 0; i < blueData.HRV.count; i++) {

if(i >= 1){

hrvStr = [hrvStr stringByAppendingString:[NSString stringWithFormat:@",%dms",[blueData.HRV[i] intValue]]];

}else{

hrvStr = [hrvStr stringByAppendingString:[NSString stringWithFormat:@"%dms",[blueData.HRV[i] intValue]]];

}

}

UnitySendMessage("ThinkGearManager", "ReceiveHRV", [hrvStr 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] connectBlue4WithIsAuto:NO];

}

void Scan(){

[\_refreshDevices removeAllObjects];

[[Blue4Manager shareInstance] scanBlue4WithScannedWithScanTime:12 blue4DeviceBlock:^(ScannedDevice \*blue4Device) {

BOOL isFind = false;

for (ScannedDevice \*ss in \_refreshDevices) {

if ([ss.peripheral.identifier.UUIDString isEqualToString: blue4Device.peripheral.identifier.UUIDString]) {

isFind = true;

}else{

}

}

if(!isFind){

[\_refreshDevices addObject:blue4Device];

NSLog(@"==name===%@", blue4Device.name);

NSLog(@"==identifier===%@", blue4Device.peripheral.identifier);

}

NSString \*nameIdentifierRiss = [NSString stringWithFormat:@"%@,%@,%d",blue4Device.name,blue4Device.peripheral.identifier,blue4Device.RSSI.intValue];

UnitySendMessage("ThinkGearManager", "DeviceFound", [nameIdentifierRiss cStringUsingEncoding:NSUTF8StringEncoding]);

}];

}

void ConnectDevice(char \*identifier){

for (ScannedDevice \*device in \_refreshDevices) {

if ([device.peripheral.identifier.UUIDString isEqualToString:[NSString stringWithCString:identifier encoding:NSUTF8StringEncoding]]) {

[[Blue4Manager shareInstance] manuallyConnetBlue4ForScannedDevice:device];

}

}

}

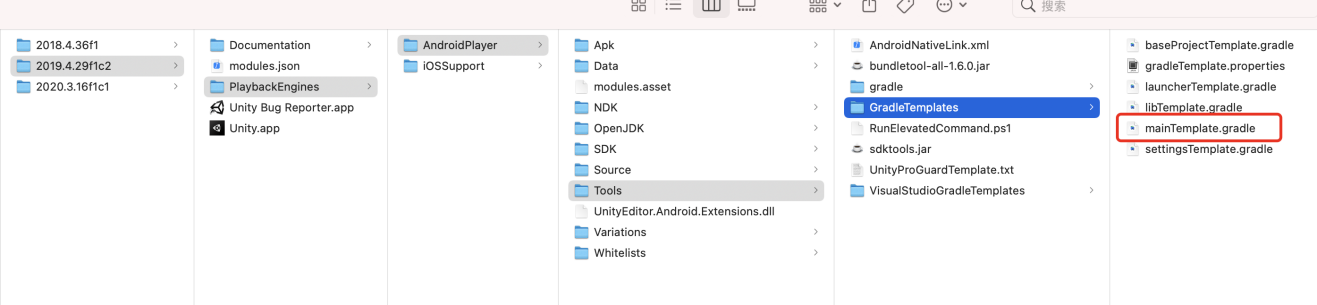
// void DisConnect(){

// [[Blue4Manager shareInstance] disConnectBlue4];

// }

### Android Configuration:

1.Find in the unity installation directory mainTemplate.gradle



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

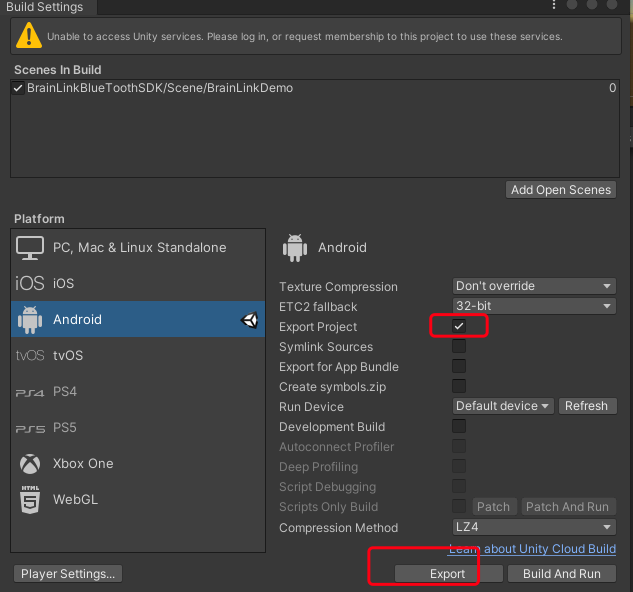
dependencies {

implementation fileTree(dir: 'libs', include: ['\*.jar'])

implementation 'androidx.appcompat:appcompat:1.1.0'

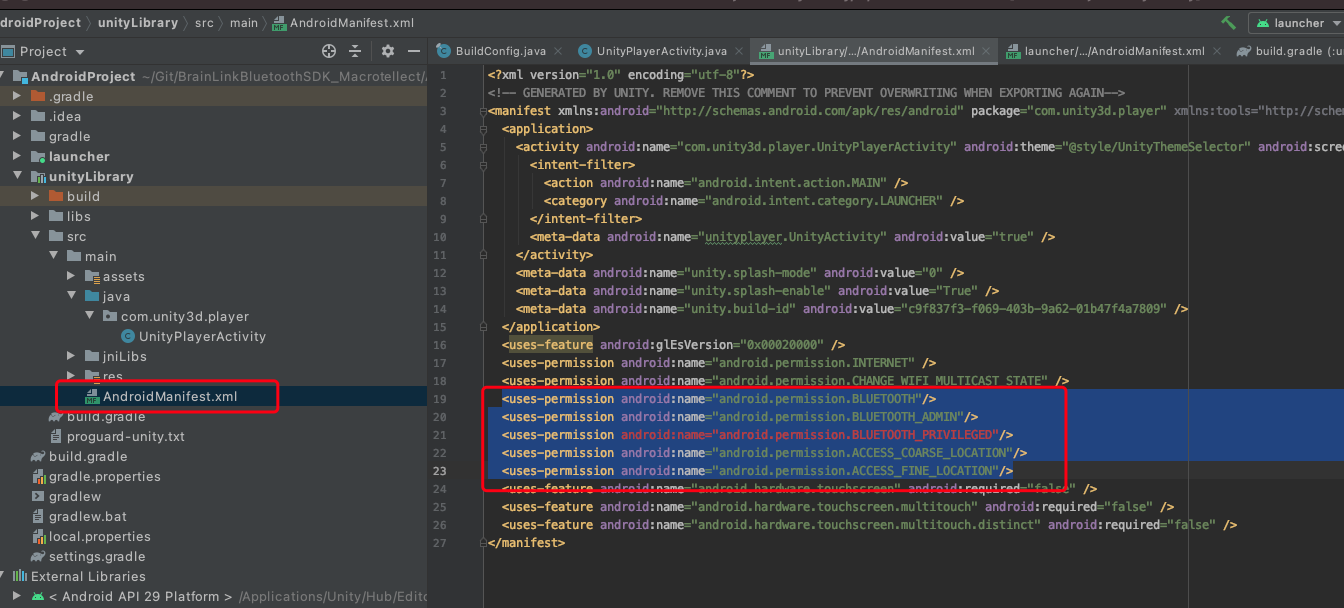
\*\*DEPS\*\*}

2. Export the android project of unity as shown below



1. In the exported android project, add Bluetooth and positioning permissions to AndroidManifest.xml in the unityLibrary directory, as follows

<uses-permission android:name="android.permission.BLUETOOTH"/>  
<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/>  
<uses-permission android:name="android.permission.BLUETOOTH\_PRIVILEGED"/>  
<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>  
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>



Add authorization code to UnityPlayerActivity, as follows

requestPermissions();

String[] permissions = new String[]{

        android.Manifest.permission.*BLUETOOTH*,

        Manifest.permission.*ACCESS\_COARSE\_LOCATION*,

        Manifest.permission.*ACCESS\_FINE\_LOCATION*

};

private void requestPermissions() {

    ArrayList<String> permissionList = new ArrayList<>();

    for (int i = 0; i < permissions.length; i++) {

        if (ActivityCompat.*checkSelfPermission*(UnityPlayerActivity.this, permissions[i]) != PackageManager.*PERMISSION\_GRANTED*) {

            permissionList.add(permissions[i]);

        }

    }

    if (!permissionList.isEmpty()) {

        String[] mPermissions = permissionList.toArray(new String[permissionList.size()]);

        ActivityCompat.*requestPermissions*(UnityPlayerActivity.this, mPermissions, 100);

    }

}

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

    if (requestCode == 100) {

        for (int i = 0; i < grantResults.length; i++) {

            if (grantResults[i] == PackageManager.*PERMISSION\_DENIED*) {

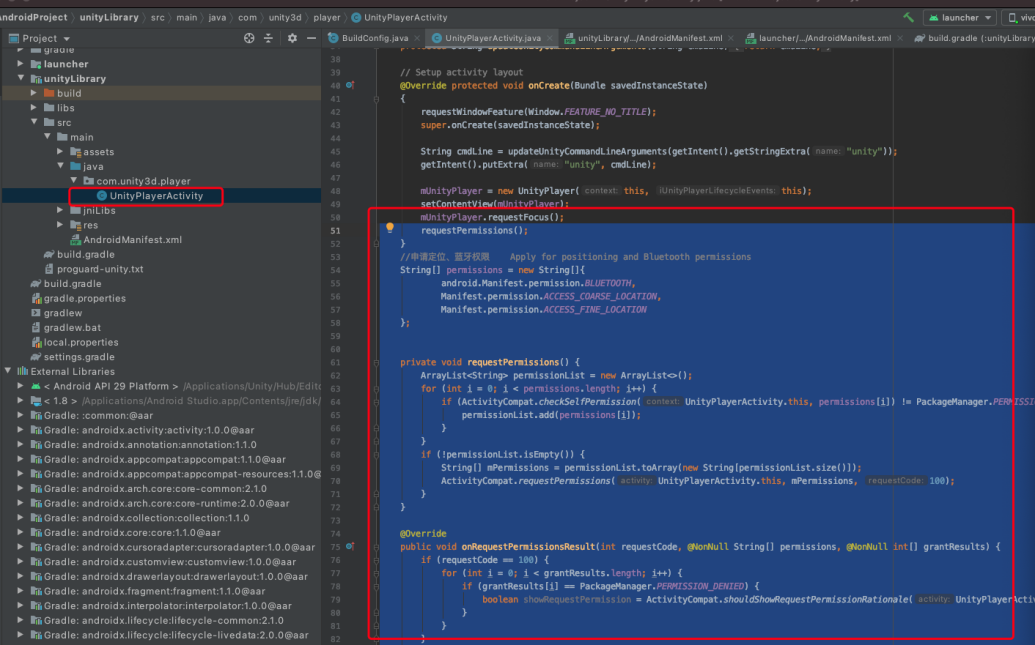
                boolean showRequestPermission = ActivityCompat.*shouldShowRequestPermissionRationale*(UnityPlayerActivity.this, permissions[i]);

            }

        }

    }

}



## Unity3D BrainLinkProSDK V1.0.5 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
* HRV Heart rate variability

**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 V1.0.5 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