iOS BLE5.0 Quick Start

Routine download

Link to download: https://github.com/WITMOTION/WitBluetooth_BWT901BLE5_0

Introduction to routines

Routine introduction

1. The routine demonstrates how to use the iOS Bluetooth 5.0 sensor SDK developed by wit-motion

2. This routine will demonstrate how to search and connect Bluetooth 5.0 sensors, and control the sensors

3. Please be familiar with the use of wit-motion Bluetooth 5.0 sensor before using the routine, and understand the protocol of the sensor

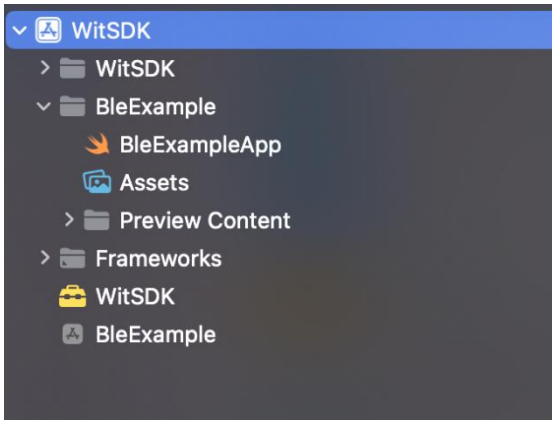
4. This routine is developed in swift language and can be applied to your iPhone/iPad/MaciOS

Routine Directory

The routine project directory is as follows

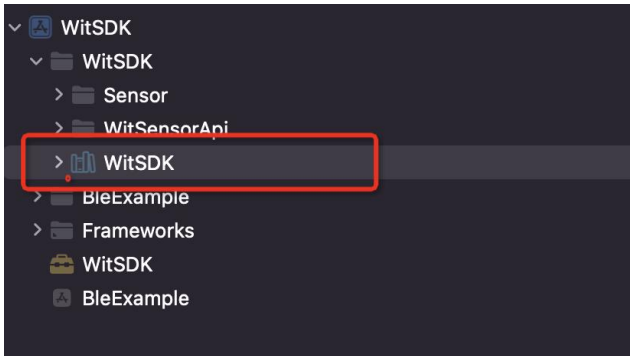
Frameworks： The dependency file of the project, please import “WitSDK” into your project before running the project.

AppMainView： There is only one code file in the routine, and all logic codes are in this file, and there are no other files



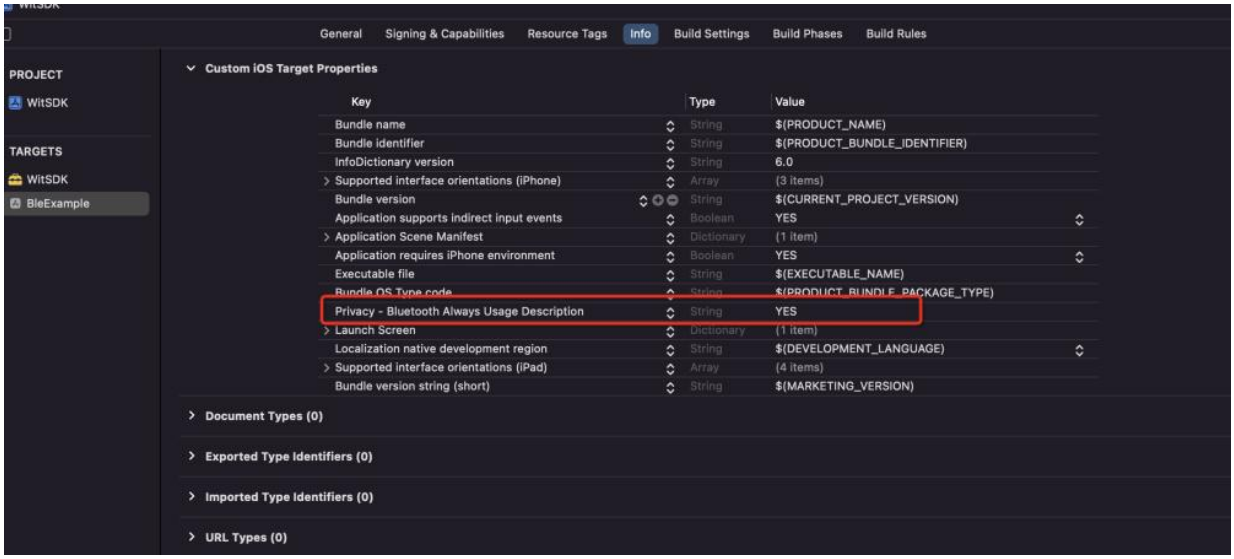
Routine dependency

This routine depends on the WitSDK project. If you want to port it to your own app, please port the routine "WitSDK" under the "Frameworks" folder to your app.

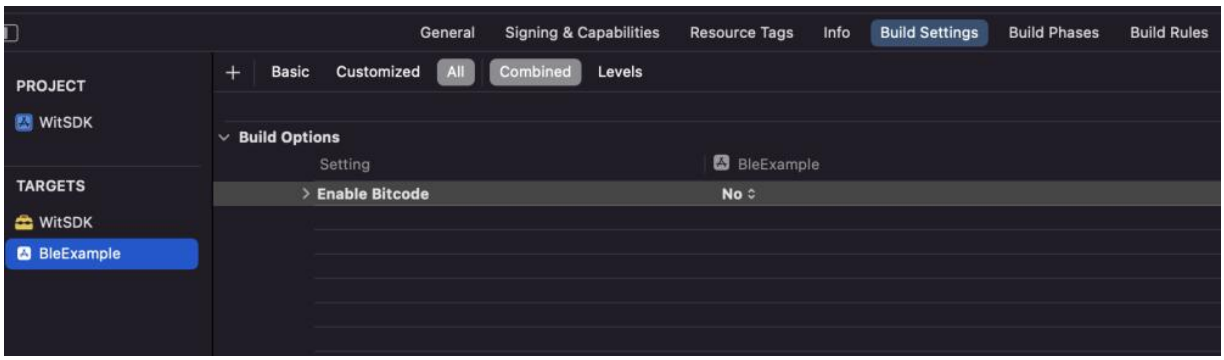


Project Permissions

Before this project can run, you need to add permission to connect to Bluetooth. Add the Key value under Info to Privacy - Bluetooth Always Usage Description, and the Value to YES



Add the Key value to Bitcode under Info, and the Value to NO



Search Device

Search Bluetooth

You can use the "WitBluetoothManager.startScan" method to start scanning devices. When a Bluetooth 5.0 device is found, the information of the Bluetooth device will be displayed first; at the same time, after choosing to connect to Bluetooth, it will prompt whether the connection is successful.

```
    // MARK: 开始扫描设备
    // MARK: Start scanning for devices
    func scanDevices() {
        print("开始扫描周围蓝牙设备 Start scanning for surrounding bluetooth devices")
```

```
        // 移除所有的设备，在这里会关闭所有设备并且从列表中移除
        // Remove all devices, here all devices are turned off and removed from the list
        removeAllDevice()
        // 注册蓝牙事件观察者
        // Registering a Bluetooth event observer
        self.bluetoothManager.registerEventObserver(observer: self)
        // 开启蓝牙扫描
        // Turn on bluetooth scanning
        self.bluetoothManager.startScan()
    }

    // MARK: 如果找到低功耗蓝牙传感器会调用这个方法
    // MARK: This method is called if a Bluetooth Low Energy sensor is found
    func onFoundBle(bluetoothBLE: BluetoothBLE?) {
        if isNotFound(bluetoothBLE) {
            print("\(String(describing: bluetoothBLE?.peripheral.name)) 找到一个蓝牙设备
found a bluetooth device")
            self.deviceList.append(Bwt901ble(bluetoothBLE: bluetoothBLE))
        }
    }

    // 判断设备还未找到
    // Judging that the device has not been found
    func isNotFound(_ bluetoothBLE: BluetoothBLE?) -> Bool{
        for device in deviceList {
            if device.mac == bluetoothBLE?.mac {
                return false
            }
        }
        return true
    }

    // MARK: 当连接成功时会在这里通知您
    // MARK: You will be notified here when the connection is successful
    func onConnected(bluetoothBLE: BluetoothBLE?) {
        print("\(String(describing: bluetoothBLE?.peripheral.name)) 连接成功")
```

```
    }

    // MARK: 当连接失败时会在这里通知您
    // MARK: Notifies you here when the connection fails
    func onConnectionFailed(bluetoothBLE: BluetoothBLE?) {
        print("\(String(describing: bluetoothBLE?.peripheral.name)) 连接失败")
    }

    // MARK: 当连接断开时会在这里通知您
    // MARK: You will be notified here when the connection is lost
    func onDisconnected(bluetoothBLE: BluetoothBLE?) {
        print("\(String(describing: bluetoothBLE?.peripheral.name)) 连接断开")
    }
```

Stop Searching

Existing searches can be stopped by the "WitBluetoothManager.stopScan" method

```
    // MARK: 停止扫描设备
    // MARK: Stop scanning for devices
    func stopScan(){
        // 删除蓝牙事件观察器
        self.bluetoothManager.removeEventObserver(observer: self)
        // 移除监听新找到的传感器
        self.bluetoothManager.stopScan()
    }
```

Receive Sensor Data

Get Data

The sensor data can be obtained through the "getDeviceData" method. "getDeviceData" needs to receive a key value, which is stored in the "WitSensorKey" class

```
    // MARK: 获得设备的数据，并且拼接为字符串
    // MARK: Get the data of the device and concatenate it into a string
    func getDeviceDataToString(_ device:Bwt901ble) -> String {
        var s = ""
        s  = "\(s)name:\(device.name ?? "")\r\n"
        s  = "\(s)mac:\(device.mac ?? "")\r\n"
        s  = "\(s)version:\(device.getDeviceData(WitSensorKey.VersionNumber) ?? "")\r\n"
        s  = "\(s)AX:\(device.getDeviceData(WitSensorKey.AccX) ?? "") g\r\n"
        s  = "\(s)AY:\(device.getDeviceData(WitSensorKey.AccY) ?? "") g\r\n"
        s  = "\(s)AZ:\(device.getDeviceData(WitSensorKey.AccZ) ?? "") g\r\n"
        s  = "\(s)GX:\(device.getDeviceData(WitSensorKey.GyroX) ?? "") °/s\r\n"
        s  = "\(s)GY:\(device.getDeviceData(WitSensorKey.GyroY) ?? "") °/s\r\n"
        s  = "\(s)GZ:\(device.getDeviceData(WitSensorKey.GyroZ) ?? "") °/s\r\n"
        s  = "\(s)AngX:\(device.getDeviceData(WitSensorKey.AngleX) ?? "") °\r\n"
        s  = "\(s)AngY:\(device.getDeviceData(WitSensorKey.AngleY) ?? "") °\r\n"
        s  = "\(s)AngZ:\(device.getDeviceData(WitSensorKey.AngleZ) ?? "") °\r\n"
        s  = "\(s)HX:\(device.getDeviceData(WitSensorKey.MagX) ?? "") μt\r\n"
        s  = "\(s)HY:\(device.getDeviceData(WitSensorKey.MagY) ?? "") μt\r\n"
        s  = "\(s)HZ:\(device.getDeviceData(WitSensorKey.MagZ) ?? "") μt\r\n"
        s  =
"\(s)Electric:\(device.getDeviceData(WitSensorKey.ElectricQuantityPercentage) ??
"") %\r\n"
        s  = "\(s)Temp:\(device.getDeviceData(WitSensorKey.Temperature) ?? "") °C\r\n"
        return s
    }
```

Record Data

When the device is turned on, you can call the "registerListenKeyUpdateObserver" method of bwt901ble, and when the sensor data is updated, bwt901ble will call the "onRecord" method to notify you to record data

```
// MARK: 打开设备
// MARK: Turn on the device
func openDevice(bwt901ble: Bwt901ble?){
    print("打开设备 MARK: Turn on the device")

    do {
        try bwt901ble?.openDevice()
        // 监听数据
        // Monitor data
        bwt901ble?.registerListenKeyUpdateObserver(obj: self)
    }
    catch{
        print("打开设备失败 Failed to open device")
    }
}


// MARK: 当需要记录传感器的数据时会在这里通知您
// MARK: You will be notified here when data from the sensor needs to be recorded
func onRecord(_ bwt901ble: Bwt901ble) {
    // 您可以在这里获得传感器的数据
    // You can get sensor data here
    let deviceData =  getDeviceDataToString(bwt901ble)
    // 打印到控制台,您也可以在这里把数据记录到您的文件中
    // Prints to the console, where you can also log the data to your file
    print(deviceData)
}
```

Set Sensor

Accelerometer Calibration

Accelerometer calibration can be done by calling "appliedCalibration" of Bwt901ble. Remember to unlock the registers before accelerometer calibration

```
// MARK: 加计校准
// MARK: Addition calibration
func appliedCalibration(){
    for device in deviceList {

        do {
            // 解锁寄存器
            // Unlock register
            try device.unlockReg()
            // 加计校准
            // Addition calibration
            try device.appliedCalibration()
            // 保存
            // save
            try device.saveReg()

        }catch{
            print("设置失败 Set failed")
        }
    }
}
```

Magnetic Field Calibration

You can control the start of magnetic field calibration and the end of magnetic field calibration by calling the "startFieldCalibration" and "endFieldCalibration" methods of Bwt901ble. After starting the magnetic field calibration, please rotate 2-3 circles around the x y z axes of the sensor. If you don't know the magnetic field calibration, you can Consult our technical support.

Start Magnetic Field Calibration

```
// MARK: 开始磁场校准
// MARK: Start magnetic field calibration
func startFieldCalibration(){
    for device in deviceList {
        do {
            // 解锁寄存器
            // Unlock register
            try device.unlockReg()
            // 开始磁场校准
            // Start magnetic field calibration
            try device.startFieldCalibration()
            // 保存
            // save
            try device.saveReg()
        }catch{
            print("设置失败 Set failed")
        }
    }
}
```

End Magnetic Field Calibration

```
// MARK: 结束磁场校准
    // MARK: End magnetic field calibration
    func endFieldCalibration(){
        for device in deviceList {
            do {
                // 解锁寄存器
                // Unlock register
                try device.unlockReg()
                // 结束磁场校准
                // End magnetic field calibration
                try device.endFieldCalibration()
                // 保存
                // save
                try device.saveReg()
            }catch{
                print("设置失败 Set failed")
            }
        }
    }
```

Read Sensor Register

This demonstration demonstrates how to read the 03 register of the sensor. Use the readRge method to read the sensor data. After reading, you can use getDeviceData to obtain the data of the register.

```
    // MARK: 读取 03 寄存器
    // MARK: Read the 03 register
    func readReg03(){
        for device in deviceList {
            do {
                // 读取 03 寄存器，等待 200ms，如果没读到可以把读取时间延长或多读几次
```

```
                // Read the 03 register and wait for 200ms. If it is not read out, you can
extend the reading time or read it several times
                try device.readRge([0xff ,0xaa, 0x27, 0x03, 0x00], 200, {
                    let reg03value = device.getDeviceData("03")
                    // 输出结果到控制台
                    // Output the result to the console
                    print("\(String(describing: device.mac)) reg03value:
\(String(describing: reg03value))")
                })
            }catch{
                print("设置失败 Set failed")
            }
        }
    }
```

WitBluetoothManager-API Description

Class function introduction: Bluetooth manager, manage the Bluetooth of your device, you can use it to search for Bluetooth connection objects

Method Definition	Function Description	Parameter Description
startScan()	Start scanning for bluetooth devices	none
stopScan()	Stop scanning for bluetooth devices	none
sregisterEventObserver(observer:IBluetoothEventObserver)	Register Bluetooth	observer: bluetooth

	device event listener	event observer
removeEventObserver(observer:IBluetoothEventObserver)	Cancel Bluetooth device event monitoring	observer: bluetooth event observer

Bwt901ble-Class API description

Class function introduction: It represents a Bluetooth 5.0 sensor. You can use an instance of this class to quickly obtain sensor data and send quick commands such as accelerometer calibration and magnetic field calibration.

Method Definition	Function Description	Parameter Description
openDevice()	turn on the device	none
closeDevice()	turn off the device	none
getDeviceData(_key:String)	Get device data	key: data key value
appliedCalibration()	Accelerometer Calibration	none
startFieldCalibration()	Start Magnetic Field Calibration	none
endFieldCalibration()	End magnetic field calibration	none
sendProtocolData(_data:[UInt8], _waitTime:Int64)	send protocol data	data: the data to be sent

sendData (_data:[UInt8], waitTime:Int64])	send data	waitTime: waiting time
readRge(_data:[UInt8], waitTime:Int64,_callback: @escaping()->Void)	read register	data: read command waitTime: waiting time callback: callback method
writeRge(_data:[UInt8] ,_waitTime:Int64)	write register	data: set command waitTime: waiting time
unlockReg()	unlock register	none
saveReg()	save register	none

English Description

IOS sample program description (English).pdf