IOS Example Program Description

Download the routine

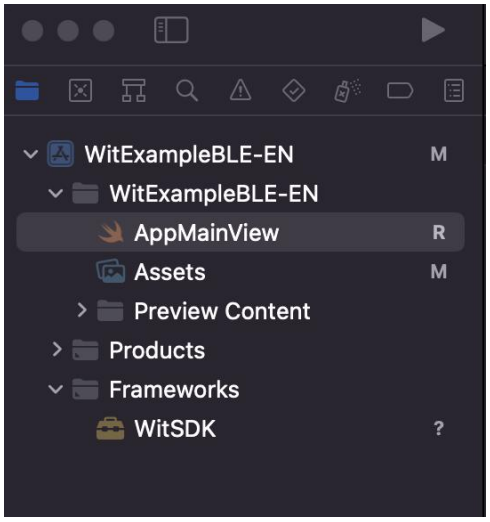
Go to the link below to download the sample

Routines Introduction

Routines Introduce
1.This example demonstrates how to use the iOS Bluetooth 5.0 sensor SDK developed by WitMotion
2.This example will demonstrate how to search for and connect to Bluetooth 5.0 sensors, and control the sensors
3.Please be familiar with the use of Bluetooth 5.0 sensor and understand the sensor protocol before using the routine
4.This routine is developed in swift language and can be applied to your iPad

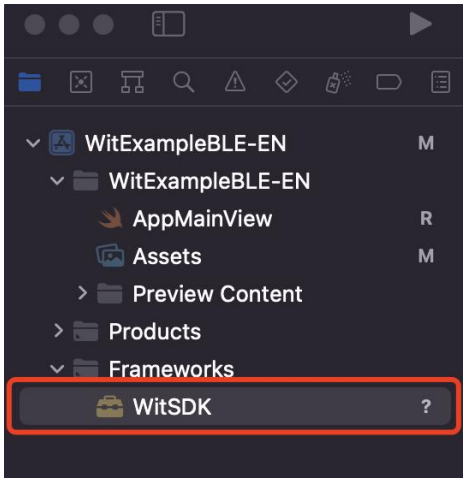
Routine directory

The directory of the routine project 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, all logic code is in this file, and there are no other files



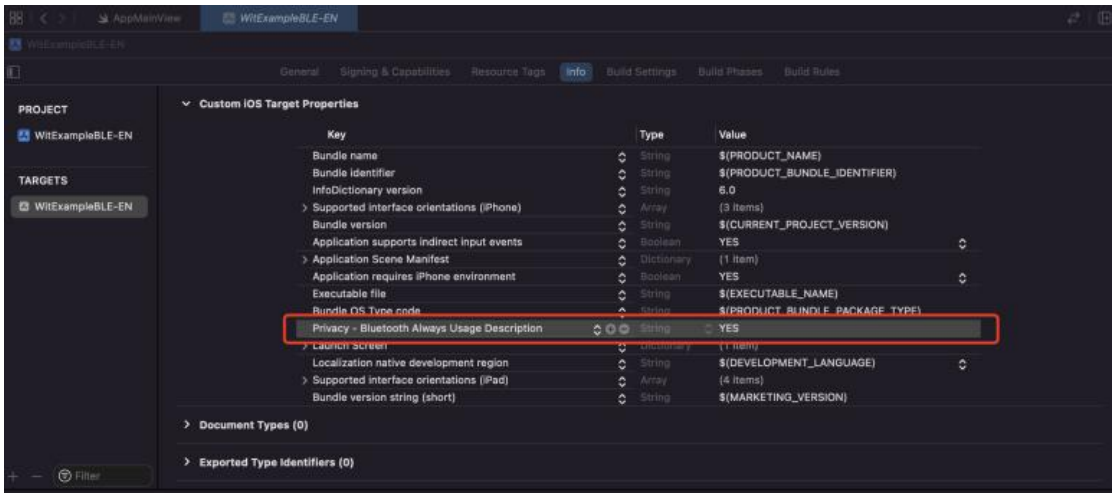
Routine dependencies

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



Project permissions

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



Search device

Search for bluetooth

You can start scanning devices through the scanDevices method. When a Bluetooth 5.0 device is found, the information of the Bluetooth device will be displayed first; at the same time, when you choose to connect to Bluetooth, it will also prompt whether the connection is successful or not

```
// MARK: Start scanning for devices
```

```
    func scanDevices() {
        print("Start scanning for surrounding bluetooth devices")
        // Remove all devices, here all devices are turned off and removed from the list
        removeAllDevice()
        // Registering a Bluetooth event observer
        self.bluetoothManager.registerEventObserver(observer: self)
        // Turn on bluetooth scanning
        self.bluetoothManager.startScan()
    }

    // 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)) Find a Bluetooth device")
            self.deviceList.append(Bwt901ble(bluetoothBLE: bluetoothBLE))
        }
    }

    // Determine if 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: You will be notified here when the connection is successful
    func onConnected(bluetoothBLE: BluetoothBLE?) {
        print("\(String(describing: bluetoothBLE?.peripheral.name)) Successfully connected")
    }

    // MARK: Notifies you here when the connection fails
    func onConnectionFailed(bluetoothBLE: BluetoothBLE?) {
        print("\(String(describing: bluetoothBLE?.peripheral.name)) Failed to connect")
    }

    // MARK: Notifies you here when the connection is lost
    func onDisconnected(bluetoothBLE: BluetoothBLE?) {
        print("\(String(describing: bluetoothBLE?.peripheral.name)) Connection disconnected")
    }
```

Stop the search

Existing searches can be stopped by the stopScan method

```
    // MARK:Stop scanning for devices
     func stopScan(){
          // Remove the bluetooth event watcher
          self.bluetoothManager.removeEventObserver(observer: self)
          // Remove listening for newly found sensors
          self.bluetoothManager.stopScan()
     }
```

Receive sensor data

Get the 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:Get the data of the device and concatenate it into a string
     func getDeviceDataToString(_ device:Bwt901ble) -> String {
          var s = ""
          s   =  "\(s)Device name:\(device.name ?? "")\r\n"
          s   =  "\(s)Device Bluetooth address:\(device.mac ?? "")\r\n"
          s    =  "\(s)Version number:\(device.getDeviceData(WitSensorKey.VersionNumber) ??
"")\r\n"
          s   =  "\(s)Acceleration X:\(device.getDeviceData(WitSensorKey.AccX) ?? "") g\r\n"
          s   = "\(s)Acceleration Y:\(device.getDeviceData(WitSensorKey.AccY) ?? "") g\r\n"
          s   = "\(s)Acceleration Z:\(device.getDeviceData(WitSensorKey.AccZ) ?? "") g\r\n"
          s    =  "\(s)Angular velocity X:\(device.getDeviceData(WitSensorKey.GyroX) ?? "")  °
/s\r\n"
          s    = "\(s)Angular velocity Y:\(device.getDeviceData(WitSensorKey.GyroY) ?? "")  °
/s\r\n"
          s    = "\(s)Angular velocity Z:\(device.getDeviceData(WitSensorKey.GyroZ) ?? "")  °
/s\r\n"
          s   =  "\(s)Angle X:\(device.getDeviceData(WitSensorKey.AngleX) ?? "")  °\r\n"
          s   = "\(s)Angle Y:\(device.getDeviceData(WitSensorKey.AngleY) ?? "")  °\r\n"
          s   = "\(s)Angle Z:\(device.getDeviceData(WitSensorKey.AngleZ) ?? "")  °\r\n"
          s   =  "\(s)Magnetic field X:\(device.getDeviceData(WitSensorKey.MagX) ?? "")  μt\r\n"
          s   = "\(s)Magnetic field Y:\(device.getDeviceData(WitSensorKey.MagY) ?? "")  μt\r\n"
          s   = "\(s)Magnetic field Z:\(device.getDeviceData(WitSensorKey.MagZ) ?? "")  μt\r\n"
          //                                   s                                                    =
"\(s)Electricity:\(device.getDeviceData(WitSensorKey.ElectricQuantityPercentage) ?? "") %\r\n"
          return s
```

```
}
```

Record data

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

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

    do {
        try bwt901ble?.openDevice()
        // Monitor data
        bwt901ble?.registerListenKeyUpdateObserver(obj: self)
    }
    catch{
        print("Failed to open device")
    }
}
// MARK: You will be notified here when data from the sensor needs to be recorded
func onRecord(_ bwt901ble: Bwt901ble) {
    // You can get sensor data here
    let deviceData =    getDeviceDataToString(bwt901ble)
    // prints to the console, where you can also log the data to your file
    print(deviceData)
}
```

Set up the sensor

Addition calibration

Addition calibration can be done by calling appliedCalibration of Bwt901ble. Remember to unlock the register before adding meter calibration

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

        do {
            // Unlock the register
            try device.unlockReg()
            // Addition calibration
            try device.appliedCalibration()
```

```
                // Save
                try device.saveReg()

            }catch{
                print("Setting failed")
            }
        }
    }
```

Magnetic Field Calibration

Magnetic field calibration can be controlled by calling the startFieldCalibration and endFieldCalibration methods of Bwt901ble to control the start and end of magnetic field calibration. After starting magnetic field calibration, please make 2-3 turns around each axis of the sensor's x, y, and z axes. If you do not know about magnetic field calibration, you can consult us technical support.

Start magnetic field calibration

```
    // MARK: Start magnetic field calibration
    func startFieldCalibration(){
        for device in deviceList {
            do {
                // Unlock the register
                try device.unlockReg()
                // Addition calibration
                try device.startFieldCalibration()
                // Save
                try device.saveReg()
            }catch{
                print("Setting failed")
            }
        }
    }
```

End the magnetic field calibration

```
    // MARK: End the magnetic field calibration
    func endFieldCalibration(){
        for device in deviceList {
            do {
                // Unlock the register
                try device.unlockReg()
                // End the magnetic field calibration
                try device.endFieldCalibration()
                // Save
                try device.saveReg()
```

```
            }catch{
                print("Setting failed")
            }
        }
    }
```

Read sensor registers

This demonstration shows how to read the 03 register of the sensor. Use the readRge method to read the sensor data. After reading, you can get the register data through getDeviceData.

```
// MARK: Read the 03 register
    func readReg03(){
        for device in deviceList {
            do {
                // 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")
                    // Output the result to the console
                    print("\(String(describing: device.mac)) reg03value: \(String(describing:
                    reg03value))")
                })
            }catch{
                print("Setting failed")
            }
        }
    }
```

WitBluetoothManager-class API introduce

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)	Registering Bluetooth device event listeners	observer:bluetooth event watcher
removeEventObserver(observer:IBluetoothEventObserver)	Cancel Bluetooth device event listening	observer:bluetooth event watcher

Bwt901ble-class API introduce

Class function introduction: Represents a Bluetooth 5.0 sensor. You can use an instance of this class to quickly obtain sensor data and send shortcut commands such as meter calibration, magnetic field calibration, etc

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
appliedCalibration()	Addition calibration	None
startFieldCalibration()	Start the magnetic field calibration	None
endFieldCalibration()	End the magnetic field calibration	None
sendProtocolData(_data:[UInt8], _waitTime:Int64)	Send protocol data	data:Data to send waitTime:Wait time
sendData（_data:[UInt8, waitTime:Int64]）	Send data	data:Data to send waitTime:Wait time
readRge(_data:[UInt8], waitTime:Int64,_callback: @escaping()->Void)	Read registers	data:Read command waitTime:Wait time callback:Callback method
writeRge(_data:[UInt8] ,_waitTime:Int64)	Write to register	data:Set command waitTime:Wait time
unlockReg()	Unlock the register	None
saveReg()	Save the register	None