BT04、BT05 SDK

——使用说明书

状态	☐ 草稿 ☐ 评审 √ 发布 ☐ 修订		
版本号	4.5.0		
作者	Forrest wu	**发布时间**	2017.06.13

前　言

尊敬软件工程师，非常感谢您使用天圆蓝牙温湿记录仪产品，此文档为你介绍在 Android、IOS 平台下如何使用我们 SDK 快速开发应用（APP）。

在开发过程中如果遇到问题，请参考示例程序。

目录

一、概述

1. SDK 概述

由于 BT04、BT05 都是基于蓝牙 BLE4.0 开发出设备,所有目标平台必需支持 BLE4.0 硬件。

2. 功能概述

a) 扫描周围 BT04、BT05 设备列表

b) 实时数据、状态显示

c) 修改设备参数

d) 提取设备历史数据

二、SDK 开发准备

1. 平台基本要求

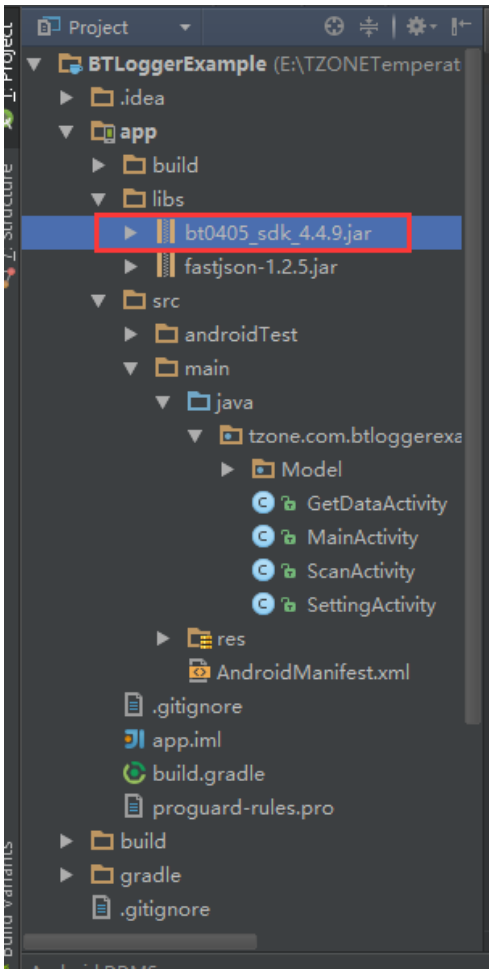
a) Android 4.4 以上

b) IOS 9.3 以上

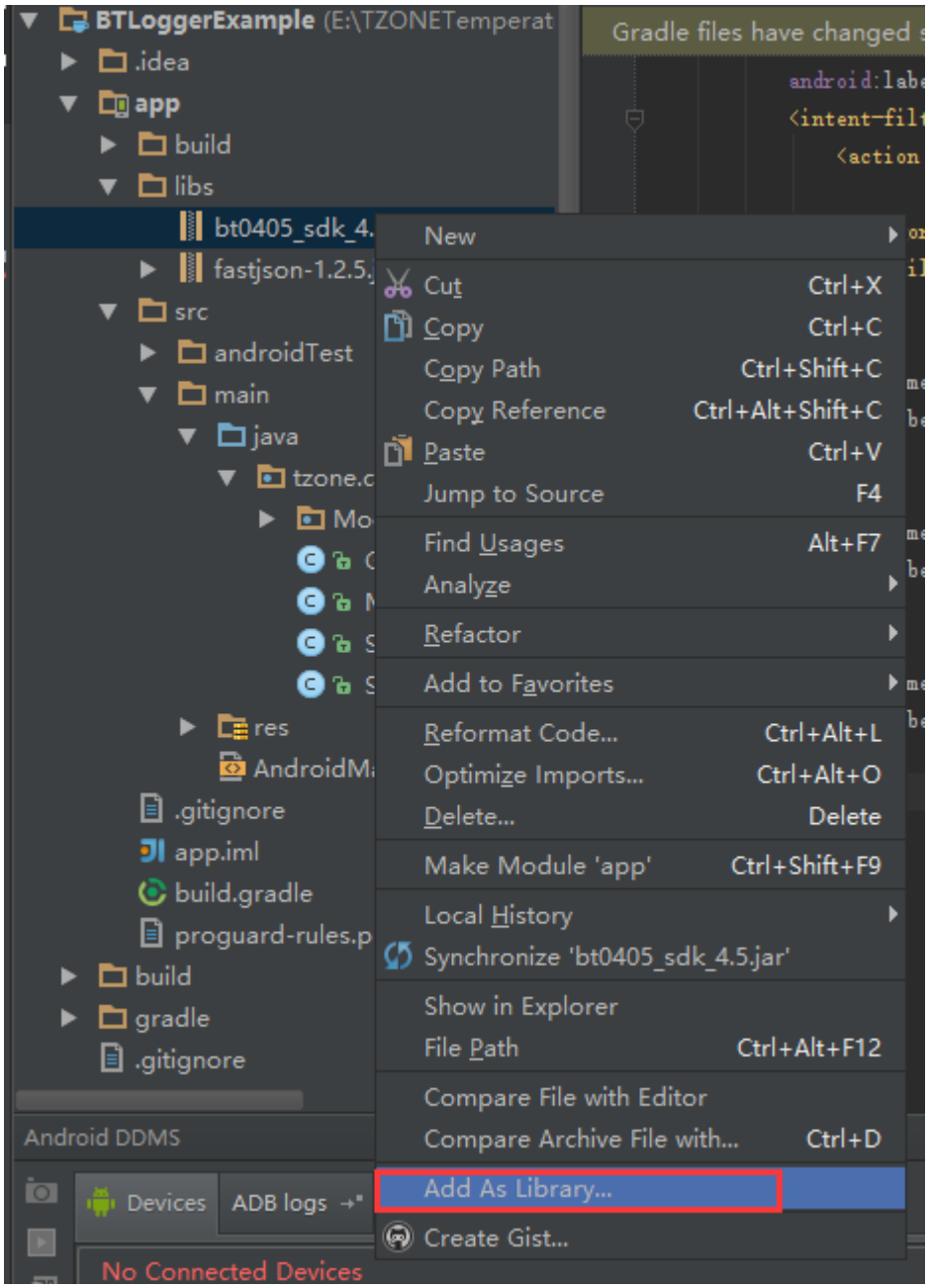
三、Android 如何使用 SDK

1. 如何使用 Android Studio IDE 引用 SDK

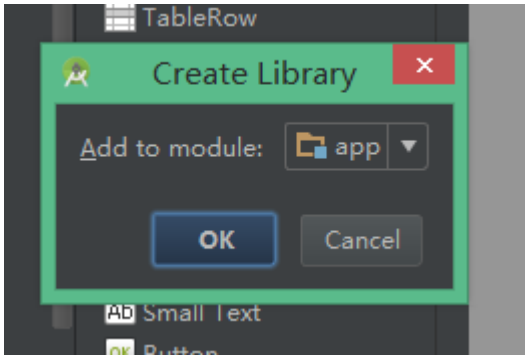
将“bt0405_sdk_4.4.9.jar” 复制到项目 libs 文件夹下，如图：



右键属性”Add as Library”.


BTLoggerExample (E:\TZONETemperat
Gradle files have changed
.idea
app
build
libs
bt0405_sdk_4.
fastjson-1.2.5.
src
androidTest
main
java
tzone.c
res
.gitignore
app.iml
build.gradle
proguard-rules.p
build
gradle
.gitignore
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Reformat Code... Ctrl+Alt+L
Optimize Imports... Ctrl+Alt+O
Delete... Delete
Make Module 'app' Ctrl+Shift+F9
Local History
Synchronize 'bt0405_sdk_4.5.jar'
Show in Explorer
File Path Ctrl+Alt+F12
Compare File with Editor
Compare Archive File with... Ctrl+D
Add As Library...
Create Gist...
Android DDMS
Devices
ADB logs
No Connected Devices

点击“ok”


TableRow
Create Library
Add to module: app
OK
Cancel
Small Text

2. 添加权限。

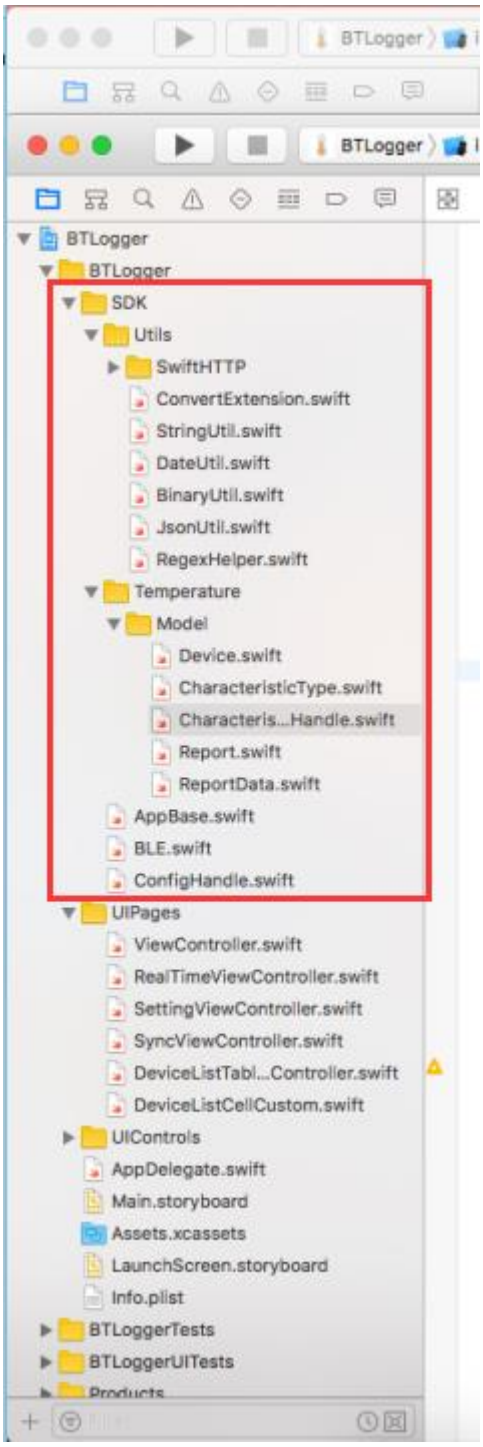
在工程的“ AndroidManifest.xml ”文件中进行添加，请直接拷贝。

```
<!-- 使用蓝牙设备的权限 -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<!-- 管理蓝牙设备的权限 -->
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

四、IOS 如何使用 SDK

1. 如何使用 Xcode IDE 引用 SDK

将红色框内的所有文件复制到自己项目里，如图：



五、使用说明

注：请参考示例程序使用。

1. 扫描周围 BT04、BT05 设备

Android :

a. 实例化 BroadcastService 对象,并初始化。

```
BroadcastService broadcastService = new BroadcastService();
broadcastService.Init(BluetoothAdapter,new LocalBluetoothCallBack{….});
```

b. 开始扫描

```
broadcastService. StartScan();
```

c. 接收扫描到设备

```
new LocalBluetoothCallBack{
        @Override
        public void OnEntered(BLE ble) {
            //第一次进入事件
            Device d = new Device();
            d.fromScanData(ble);
            if(d == null || d.SN == null || d.SN.length() != 8)
                return;
            //保存数据
            ……
        }

        @Override
        public void OnUpdate(BLE ble) {
            //更新事件
        }

        @Override
        public void OnExited(final BLE ble) {
            //离开事件
        }

        @Override
        public void OnScanComplete() {
            //一次扫描结束事件
        }
}
```

d. 结果扫描

```
broadcastService.StopScan();
```

IOS:

a. 实例化 CBCentralManager 对象。

```
var BluetoothAdapter = CBCentralManager(delegate: self, queue: nil);
```

b. 开始扫描

```
self.BluetoothAdapter.scanForPeripherals(withServices: nil, options: nil);
```

c. 接收扫描到设备

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral,
advertisementData: [String : Any], rssi RSSI: NSNumber){
```

```
……
Var M_device = device.fromScanData(peripheral: peripheral, name: peripheral.name, rssi: Int(RSSI), advertisementData: advertisementData)
……
}
```

d. 结束扫描

```
self.BluetoothAdapter.stopScan();
```

2. 实时温湿度

Android :

a. 实例化 BroadcastService 对象,并初始化。

```
BroadcastService broadcastService = new BroadcastService();
broadcastService.Init(BluetoothAdapter,new LocalBluetoothCallBack{….});
```

b. 开始扫描

```
broadcastService. StartScan();
```

c. 接收扫描到设备

```
new LocalBluetoothCallBack{
        @Override
        public void OnEntered(BLE ble) {
            //第一次进入事件

        }

        @Override
        public void OnUpdate(BLE ble) {
            //更新事件
            Device d = new Device();
            d.fromScanData(ble);
            if(d == null || d.SN == null || d.SN.length() != 8)
                return;
            if(d.SN.equals("设备 SN 号")){
                //显示数据
                ……
            }
        }

        @Override
        public void OnExited(final BLE ble) {
            //离开事件
        }

        @Override
        public void OnScanComplete() {
```

```
                    //一次扫描结束事件
                }
        }
```

d. 结果扫描

```
broadcastService.StopScan();
```

IOS:

a. 实例化 CBCentralManager 对象。

```
var BluetoothAdapter = CBCentralManager(delegate: self, queue: nil);
```

b. 开始扫描

```
self.BluetoothAdapter.scanForPeripherals(withServices: nil, options: nil);
```

c. 接收扫描到设备

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber){
    ……
    var d = device.fromScanData(peripheral: peripheral, name: peripheral.name, rssi: Int(RSSI), advertisementData: advertisementData)
    if(d == nil || d.SN == nil || d.characters.count != 8){
        return;
    }
    if(d.SN == “设备 SN 号”){
        //显示数据
        ……
    }
    ……
}
```

d. 结束扫描

```
self.BluetoothAdapter.stopScan();
```

3. 配置设备

Android :

a. 实例化 BroadcastService 对象,并初始化。

```
BroadcastService broadcastService = new BroadcastService();
broadcastService.Init(BluetoothAdapter,new LocalBluetoothCallBack{….});
```

b. 开始扫描

```
broadcastService. StartScan();
```

c. 接收扫描到设备

```
Device _Device = null;
new LocalBluetoothCallBack{
        @Override
        public void OnEntered(BLE ble) {
```

```
            //第一次进入事件
            Device d = new Device();
            d.fromScanData(ble);
            if(d == null || d.SN == null || d.SN.length() != 8)
                return;
            _Device = d;
        }

        @Override
        public void OnUpdate(BLE ble) {
            //更新事件
        }

        @Override
        public void OnExited(final BLE ble) {
            //离开事件
        }

        @Override
        public void OnScanComplete() {
            //一次扫描结束事件
        }
}
```

d．连接设备

broadcastService. StopScan(); //关闭扫描

ConfigService configService = new ConfigService(……);

e. 输入密码

configService.CheckToken(6 位数字密码);

f. 读取设备信息

```
if(HardwareModel.equals("3901"))
    configService.ReadConfig_BT04(Firmware);
else
    configService.ReadConfig_BT05(Firmware);
```

g. 修改设备参数

```
if(HardwareModel.equals("3901"))
    configService.WriteConfig_BT04(newDevice);
else
    configService.WriteConfig_BT05(newDevice);
```

IOS :

a. 实例化 CBCentralManager 对象。

```
var BluetoothAdapter = CBCentralManager(delegate: self, queue: nil);
```

b. 开始扫描

```
self.BluetoothAdapter.scanForPeripherals(withServices: nil, options: nil);
```

c. 接收扫描到设备

```
var M_Device :Device!;
func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber){
    ……
    var d = device.fromScanData(peripheral: peripheral, name: peripheral.name, rssi: Int(RSSI), advertisementData: advertisementData)
    if(d == nil || d.SN == nil || d.characters.count != 8){
        return;
    }
    if(d.SN == “设备 SN 号”){
        M_Device = d;
    }
    ……
}
```

d. 连接设备

```
//停止扫描
BluetoothAdapter.stopScan();
//连接
BluetoothAdapter.connect(m_Device.Peripheral, options: nil);
```

e. 输入密码

```
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic, error: Error?){
    if(m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.uuid) == .Password){
        var token = 6 位数字密码;
        let strToken = StringUtil.PadLeft(token, size: 6);
        var temp = ""
        for i in 0..<NSString(string: strToken).length{
            temp += "0" + StringUtil.SubString(strToken, index: i, len: 1);
        }

        let data = temp.hexToData;
        let characteristic = CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .Password)]
            self.m_Device.Peripheral.writeValue(data, for: characteristic!, type: CBCharacteristicWriteType.withResponse);
    }
}
```

f. 读取设备信息

```
//请求读取设备信息
for characteristic in CharacteristicList {
```

```
                    if(m_CharacteristicHandle.GetCharacteristicType(uuid:
characteristic.key) != .Unknown
                        &&          m_CharacteristicHandle.GetCharacteristicType(uuid:
characteristic.key) != .Password){
                        ReadConfigs.ConfigRequest(uuid: characteristic.key.uuidString);
                        //读取特性值
                        self.m_Device.Peripheral.readValue(for: characteristic.value);
                    }
                }
                ReadConfigs.ConfigRequestComplete();

            //接收设备返回信息
            func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic:
CBCharacteristic, error: Error?){
                ReadConfigs.ConfigRespond(uuid: characteristic.uuid.uuidString);
                print("读取特征 UUID:\(characteristic.uuid) Value:\(characteristic.value)");
                m_Device = m_CharacteristicHandle.SetDevice(device: m_Device, type:
m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.uuid), bytes:
characteristic.value!);

                if(ReadConfigs.IsComplete()){
                    //读取成功
                }
            }
g. 修改设备参数
            //写入修改的参数
            self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
m_Device, type: .TransmitPower), for:
CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .TransmitPower)]!,
type: CBCharacteristicWriteType.withResponse);
            WriteConfigs.ConfigRequest(uuid:
            m_CharacteristicHandle.GetCharacteristicUUID(type: .TransmitPower).uuidString);

            //兼容旧版本 BT04 不带存储
            if (self.m_Device.HardwareModel == "3901" && Int(self.m_Device.Firmware)! <= 5){
                self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
m_Device, type: .Interval), for:
CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .Interval)]!, type:
CBCharacteristicWriteType.withResponse);
                WriteConfigs.ConfigRequest(uuid:
                m_CharacteristicHandle.GetCharacteristicUUID(type: .Interval).uuidString);
            }else{
                self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
m_Device, type: .SaveInterval), for:
```

```
CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .SaveInterval)]!, type:
CBCharacteristicWriteType.withResponse);
        WriteConfigs.ConfigRequest(uuid:
        m_CharacteristicHandle.GetCharacteristicUUID(type: .SaveInterval).uuidString);

        self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
    m_Device, type: .Alarm), for:
    CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .Alarm)]!, type:
    CBCharacteristicWriteType.withResponse);
        WriteConfigs.ConfigRequest(uuid:
        m_CharacteristicHandle.GetCharacteristicUUID(type: .Alarm).uuidString);

        self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
        m_Device, type: .UTC), for:
        CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .UTC)]!, type:
        CBCharacteristicWriteType.withResponse);
        WriteConfigs.ConfigRequest(uuid:
    m_CharacteristicHandle.GetCharacteristicUUID(type: .UTC).uuidString);

        if (self.m_Device.HardwareModel == "3901"){

            self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
            m_Device, type: .RecordStatus), for:
            CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .RecordSta
            tus)]!, type: CBCharacteristicWriteType.withResponse);
            WriteConfigs.ConfigRequest(uuid:
            m_CharacteristicHandle.GetCharacteristicUUID(type: .RecordStatus).uuidString);
        }else{
            if(m_Device.RecordStatus > 0){

                self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
                m_Device, type: .RecordStatus), for:
                CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .Recor
                dStatus)]!, type: CBCharacteristicWriteType.withResponse);
                WriteConfigs.ConfigRequest(uuid:
    m_CharacteristicHandle.GetCharacteristicUUID(type: .RecordStatus).uuidString);

            }

        }

        }

            self.m_Device.Peripheral.writeValue(m_CharacteristicHandle.GetValue(device:
```

```
m_Device, type: .Password), for: CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .Password)]!, type: CBCharacteristicWriteType.withResponse);
        WriteConfigs.ConfigRequest(uuid: m_CharacteristicHandle.GetCharacteristicUUID(type: .Password).uuidString);

        WriteConfigs.ConfigRequestComplete();

//检测写入是否成功
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic, error: Error?){
    WriteConfigs.ConfigRespond(uuid: characteristic.uuid.uuidString);

    if(WriteConfigs.IsComplete()){
        //修改设置参数成功
    }
}
```

4. 提取设备数据

Android :

a. 实例化 BroadcastService 对象,并初始化。

```
BroadcastService broadcastService = new BroadcastService();
broadcastService.Init(BluetoothAdapter,new LocalBluetoothCallBack{….});
```

b. 开始扫描

```
broadcastService. StartScan();
```

c. 接收扫描到设备

```
Device _Device = null;
new LocalBluetoothCallBack{
        @Override
        public void OnEntered(BLE ble) {
            //第一次进入事件
            Device d = new Device();
            d.fromScanData(ble);
            if(d == null || d.SN == null || d.SN.length() != 8)
                return;
            _Device = d;
        }

        @Override
        public void OnUpdate(BLE ble) {
            //更新事件
        }
```

```
        @Override
        public void OnExited(final BLE ble) {
            //离开事件
        }

        @Override
        public void OnScanComplete() {
            //一次扫描结束事件
        }
}
```

d．连接设备

```
broadcastService. StopScan(); //关闭扫描
ConfigService configService = new ConfigService(……);
```

e. 输入密码

```
configService.CheckToken(6 位数字密码);
```

f. 读取设备信息

```
if(HardwareModel.equals("3901"))
    configService.ReadConfig_BT04(Firmware);
else
    configService.ReadConfig_BT05(Firmware);
```

g. 开启接收设备数据通知

```
configService.Sync(true);
```

注意：关闭通知会清掉设备里的数据。configService.Sync(false);

h. 接收数据

```
new IConfigCallBack() {
    ……
    @Override
    public void OnReceiveCallBack(UUID uuid, byte[] data){}
    ……
}
```

i. 解析数据

请参考示例程序

IOS :

a. 实例化 CBCentralManager 对象。

```
var BluetoothAdapter = CBCentralManager(delegate: self, queue: nil);
```

b. 开始扫描

```
self.BluetoothAdapter.scanForPeripherals(withServices: nil, options: nil);
```

c. 接收扫描到设备

```
var M_Device :Device!;
func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral,
```

```
advertisementData: [String : Any], rssi RSSI: NSNumber){
        ……
        var d = device.fromScanData(peripheral: peripheral, name: peripheral.name, rssi: Int(RSSI), advertisementData: advertisementData)
        if(d == nil || d.SN == nil || d.characters.count != 8){
            return;
        }
        if(d.SN == “设备 SN 号”){
            M_Device = d;
        }
        ……
    }
```

d．连接设备

```
    //停止扫描
    BluetoothAdapter.stopScan();
    //连接
    BluetoothAdapter.connect(m_Device.Peripheral, options: nil);
```

e. 输入密码

```
    func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic, error: Error?){
        if(m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.uuid) == .Password){
            var token = 6 位数字密码;
            let strToken = StringUtil.PadLeft(token, size: 6);
            var temp = ""
            for i in 0..<NSString(string: strToken).length{
                temp += "0" + StringUtil.SubString(strToken, index: i, len: 1);
            }

            let data = temp.hexToData;
            let characteristic = CharacteristicList[m_CharacteristicHandle.GetCharacteristicUUID(type: .Password)]
                self.m_Device.Peripheral.writeValue(data, for: characteristic!, type: CBCharacteristicWriteType.withResponse);
        }
    }
```

f. 读取设备信息

```
    //请求读取设备信息
    for characteristic in CharacteristicList {
            if(m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.key) != .Unknown
                && m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.key) != .Password){
                ReadConfigs.ConfigRequest(uuid: characteristic.key.uuidString);
```

```
                //读取特性值
                self.m_Device.Peripheral.readValue(for: characteristic.value);
            }
        }
        ReadConfigs.ConfigRequestComplete();

    //接收设备返回信息
    func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?){
        ReadConfigs.ConfigRespond(uuid: characteristic.uuid.uuidString);
        print("读取特征  UUID:\(characteristic.uuid) Value:\(characteristic.value)");
        m_Device = m_CharacteristicHandle.SetDevice(device: m_Device, type: m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.uuid), bytes: characteristic.value!);

        if(ReadConfigs.IsComplete()){
            //读取成功
        }
    }
```

g. 开启接收设备数据通知

```
    if (m_Device.HardwareModel == "3901"){
        if(Int(m_Device.Firmware)! < 17){
                // The current device firmware is not supported
                return;
            }
    }else if(m_Device.HardwareModel == "3A01"){
            if(Int(m_Device.Firmware)! < 5){
                // The current device firmware is not supported
                return;
            }
    }else{
            // The current device firmware is not supported
            return;
    }

    if(m_Device.SavaCount > 0){
            _SyncCount = m_Device.SavaCount;
    }
    IsSync = enable;
    IsSynced = false;
    SyncProgress = 0;
    self.m_Device.Peripheral.setNotifyValue(true, for: CharacteristicList[CBUUID(string:"27763B21-999C-4D6A-9FC4-C7272BE10900")]!);
```

注意：关闭通知会清掉设备里的数据。

h. 接收数据

```
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic:
CBCharacteristic, error: Error?){
        ......
        let bytes:Data = characteristic.value!;
        ......
}
```

i. 解析数据

请参考示例程序