BT04、BT05 SDK

——Instruction Manual

Statues	□ Draft □ Review √ Release □ Revision		
Version number	4.5.0		
The author	Forrest wu	**Release date**	2017.06.13

PREFACE

Dear software engineer,

Thank you very much for the use of Tzone bluetooth temperature and humidity recorder products,

This document shows you how to use our SDK to develop applications (APP) on Android and IOS platforms.

In the development process if you encounter problems, please refer to the sample program.

Contents

I. OUTLINE

1. SDK outline

As BT04, BT05 are developed under the Bluetooth 4.0, all platforms must support Bluetooth 4.0 hardware.

2. Function outline

a) Scan BT04, BT05

b) Real time data, Indicator status

c) Modify the device parameters

d) Extract history data from device

II. SDK Development Preparation

1. Basic requirements of the platform

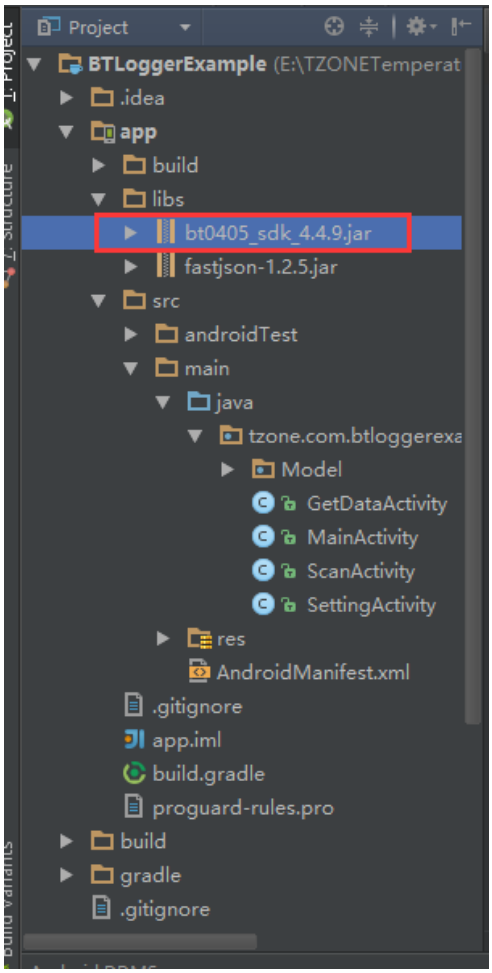
a) Android 4.4 above

b) IOS 9.3 above

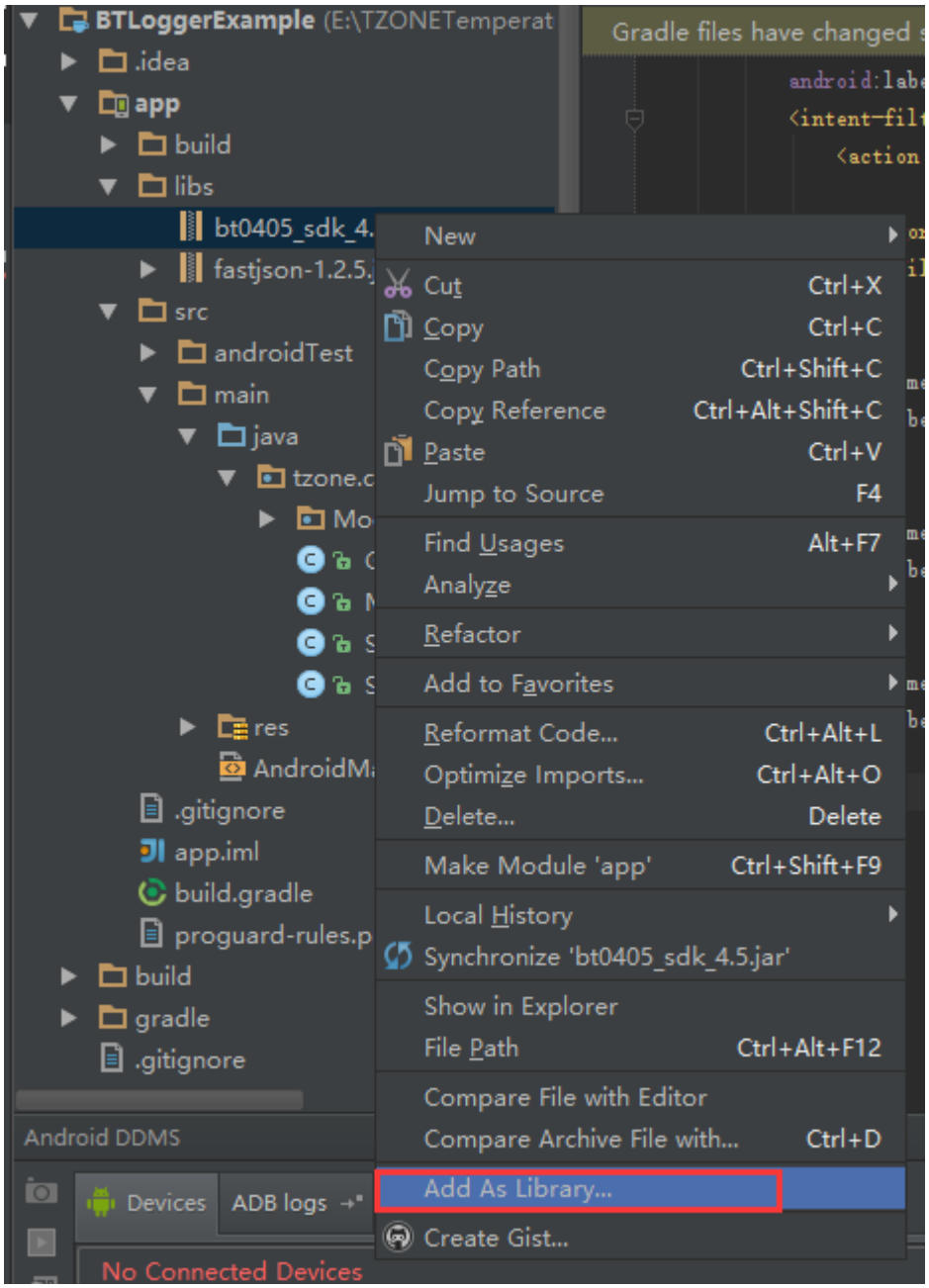
III. How to use the SDK on Android

1. How to have the Android studio IDE applies to SDK

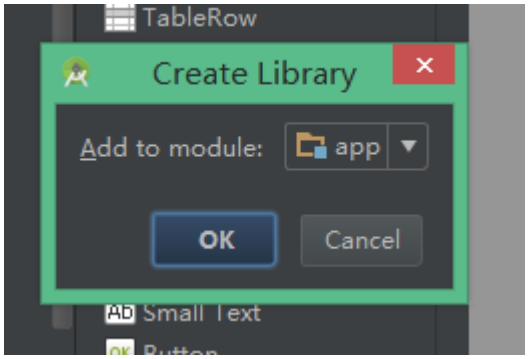
Copy“bt0405_sdk_4.4.9.jar” to “libs”folder，as shown below：



Mouse right click ”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

click“ok”


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

2. Add permission.

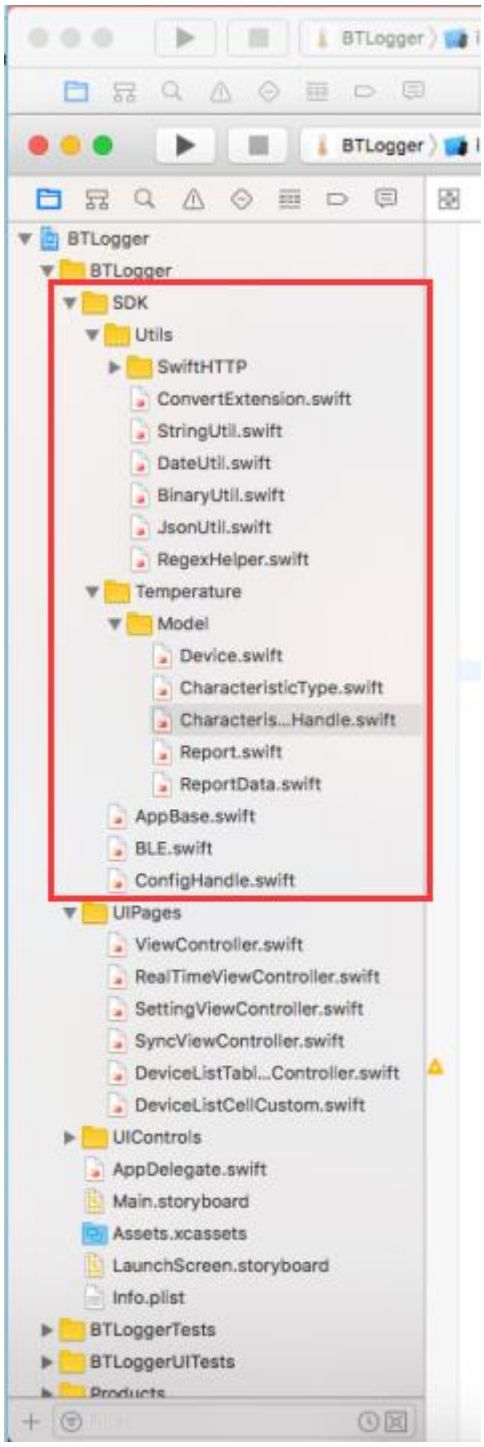
Find this folder“ AndroidManifest.xml ”copy below codes into this folder.

```
<!—Permission with bluetooth device -->
<uses-permission android:name="android.permission.BLUETOOTH" />
<!—Management bluetooth device permission -->
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

IV. How to use SDK on IOS

1. How to have Xcode apply to SDK

Copy all the files marked of red frame into your Project, as shown:



V. Instruction

Note：Please follow the sample program.

1. Scan BT04, BT05 device

Android :

a. Instantiate the BroadcastService, and initialize it.

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

b. Start scan

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

c. Scan enters into device

```
new LocalBluetoothCallBack{
        @Override
        public void OnEntered(BLE ble) {
            // First time entrance
            Device d = new Device();
            d.fromScanData(ble);
            if(d == null || d.SN == null || d.SN.length() != 8)
                return;
            //Save data
            ……
        }

        @Override
        public void OnUpdate(BLE ble) {
            //updating
        }

        @Override
        public void OnExited(final BLE ble) {
            //leave
        }

        @Override
        public void OnScanComplete() {
            // Once scan ends
        }
}
```

d. Stop scan

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

IOS:

a. Instantiate the CBCentralManager

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

b. Start scan

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

c. Scan enters into device

```
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. Stop scan

self.BluetoothAdapter.stopScan();

2. Real time temperature

Android :

a.

Instantiate BroadcastService and initialize it.

BroadcastService broadcastService = new BroadcastService();

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

b. Start scan

broadcastService. StartScan();

c. Scan enters into device

```
new LocalBluetoothCallBack{
        @Override
        public void OnEntered(BLE ble) {
            //First time entrance

        }

        @Override
        public void OnUpdate(BLE ble) {
            //updating
            Device d = new Device();
            d.fromScanData(ble);
            if(d == null || d.SN == null || d.SN.length() != 8)
                return;
            if(d.SN.equals("Device's SN number")){
                //data presentation
                ……
            }
        }

        @Override
        public void OnExited(final BLE ble) {
            //leave
        }

        @Override
```

```
            public void OnScanComplete() {
                //Once scan ends
            }
        }
```

d. Stop scan

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

IOS:

a. Instantiate CBCentralManager

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

b. start scan

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

c. Scan enters into 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 == “device’s SN number”){
        //data presentation
        ……
    }
    ……
}
```

d. stop scan

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

3. Configure device

Android :

a. Instantiate BroadcastService , initialized it.

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

b. Star scan

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

c. scan enters into device

```
Device _Device = null;
new LocalBluetoothCallBack{
        @Override
```

```
        public void OnEntered(BLE ble) {
            //First time entrance
            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) {
            //updating
        }

        @Override
        public void OnExited(final BLE ble) {
            //leave
        }

        @Override
        public void OnScanComplete() {
            //Once scan ends
        }
}
```

d. Connect device

```
broadcastService. StopScan(); //close scan
ConfigService configService = new ConfigService(……);
```

e. Enter password

```
configService.CheckToken(6 digit password);
```

f. Read device information

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

g. Modify device parameters

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

IOS :

a. Instantiate CBCentralManager

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

b. Star scan

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

c. Scan enters into device

```
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 == “device’s SN number”){
        M_Device = d;
    }
    ……
}
```

d. Connect device

```
//Stop scan
BluetoothAdapter.stopScan();
//connect
BluetoothAdapter.connect(m_Device.Peripheral, options: nil);
```

e. entre password

```
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic, error: Error?){
    if(m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.uuid) == .Password){
        var token =6 digit password;
        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. Read device information

```
// Request to read device information
```

```
    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);
                // Read the property value
                self.m_Device.Peripheral.readValue(for: characteristic.value);
            }
        }
        ReadConfigs.ConfigRequestComplete();

    // Receive the returning information from the device
    func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic:
CBCharacteristic, error: Error?){
        ReadConfigs.ConfigRespond(uuid: characteristic.uuid.uuidString);
        print("read characters 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()){
            //read suceessfully
        }
    }
```

g. Modify device's parameters

```
    // Write the modified parameters
    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);

    // Compatible with older versions BT04 without storage
    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: .RecordStatus)]!, 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: .RecordStatus)]!, 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();

// Check whether the write is successful
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic,
error: Error?){
    WriteConfigs.ConfigRespond(uuid: characteristic.uuid.uuidString);

    if(WriteConfigs.IsComplete()){
        // Modify setting parameters successfully
    }
}
```

4. Read device information

Android :

a. Instantiate BroadcastService and initialized it

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

b. Start scan

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

c. Scan enters into device

```
    Device _Device = null;
    new LocalBluetoothCallBack{
            @Override
            public void OnEntered(BLE ble) {
                //First time enterance
                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) {
                //updating
            }
```

```
        @Override
        public void OnExited(final BLE ble) {
            //leave
        }

        @Override
        public void OnScanComplete() {
            //once scan ends
        }
}
```

d. Connect device

broadcastService. StopScan(); //close scan

ConfigService configService = new ConfigService(……);

e. enter password

configService.CheckToken(6 digit password);

f. read device information

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

g. Turn on the receiving device data notification

configService.Sync(true);

Note: Closing the notification, will clears the data in the device. configService.Sync(false);

h. Receive data

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

i. Analytical data

Please refer to the sample program

IOS :

a. Instantiate CBCentralManager.

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

b. Start scan

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

c. Scan enters into device

```
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 == “Device’s SN number”){
        M_Device = d;
    }
    ……
}
```

d. Connect device

```
//stop scan
BluetoothAdapter.stopScan();
//connect
BluetoothAdapter.connect(m_Device.Peripheral, options: nil);
```

e. enter password

```
func peripheral(_ peripheral: CBPeripheral, didWriteValueFor characteristic: CBCharacteristic, error: Error?){
    if(m_CharacteristicHandle.GetCharacteristicType(uuid: characteristic.uuid) == .Password){
        var token = 6 digit password;
        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. Read device information

```
// Request to read device information
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);
                    // Read the property value
                    self.m_Device.Peripheral.readValue(for: characteristic.value);
                }
            }
            ReadConfigs.ConfigRequestComplete();

        // Receive the returning informations from device
        func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?){
            ReadConfigs.ConfigRespond(uuid: characteristic.uuid.uuidString);
            print("Read characters  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()){
                //read sucessfully
            }
        }
```

g. Turn on the receiving device data notification

```
        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")]!);

Note: Closing notification will clear device data

h. Receive data

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

i. Analytical data

Please refer to the sample program