ISDT 深圳艾斯特创新科技有限公司

CM1620 蓝牙与 IOS 系统实现通讯

CM1620 Bluetooth and iOS Communication System

1、概述 Overview

1.1 使用的编程语言以及通信方式

Programming language used and communication method

编程语言：swift5.0

Programming language: swift5.0

使用的 Framework: Core Bluetooth

Framework used: Core Bluetooth

Core Bluetooth 官方文档： Core Bluetooth official documents:

https://developer.apple.com/documentation/corebluetooth

这里采用手机作为蓝牙中心设备，CM1620 作为外围设备。

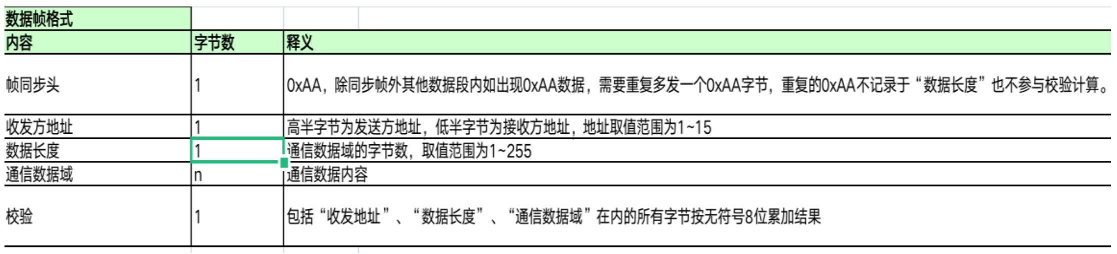
The mobile phone is used as the Bluetooth central device, and the CM1620 is used as the peripheral device.

蓝牙调试工具：nRF Connent

Bluetooth debugging tool: nRF Connect

1.2 通讯协议格式

1.2 Communication protocol format



数据帧格式

内容	字节数	释义
帧同步头	1	0xAA，除同步帧外其他数据段内如出现0xAA数据，需要重复多发一个0xAA字节，重复的0xAA不记录于"数据长度"也不参与校验计算。
收发方地址	1	高半字节为发送方地址，低半字节为接收方地址，地址取值范围为1~15
数据长度	1	通信数据域的字节数，取值范围为1~255
通信数据域	n	通信数据内容
校验	1	包括"收发地址"、"数据长度"、"通信数据域"在内的所有字节按无符号8位累加结果

BLE 模式状态查询请求示例：

手机发送：0x05 0xAA 0x12 0x01 0x14 0x27(0x05 是蓝牙通讯数据总长度)

手机接收：0x06 0xAA 0x21 0x02 0x15 0x00 0x38(0x06 是蓝牙通讯数据总长度)

BLE mode status query request example:

Mobile phone sends: 0x05 0xAA 0x12 0x01 0x14 0x27 (0x05 is the total length of the Bluetooth communication data)

Mobile phone receives: 0x06 0xAA 0x21 0x02 0x15 0x00 0x38 (0x06 is the total length of the Bluetooth communication data)

充电器蓝牙名称规则：

ISDT 1 CM1620XX cm1620XXXXXXXXXX

ISDT 0 CM1620XX cm1620XXXXXXXXXX

说明:

ISDT-制造商(占 4 个字节)

1-绑定状态　0-默认状态(占 1 个字节)

CM1620XX-设备名称（占 8 个字节）

cm1620XXXXXXXXXX-用户自定义的设备名称（占 16 个字节）

Charger Bluetooth name rules:

ISDT 1 CM1620XX cm1620XXXXXXXXXX

ISDT 0 CM1620XX cm1620XXXXXXXXXX

Description:

ISDT-Manufacturer (occupies 4 bytes)

1-Binding state 0-Default state (occupies 1 byte)

CM1620XX-device name (occupies 8 bytes)

cm1620XXXXXXXXXX-User-defined device name (occupies 16 bytes)

2、搜索设备

2. Search for equipment

2.1 创建 BLEManager.swift 并引入用到的 Framework

2.1 Create BLEManager.swift and introduce the Framework used

```
import UIKit

import CoreBluetooth


class BLEManager: NSObject, CBPeripheralDelegate,
CBCentralManagerDelegate {


var centralManager : CBCentralManager?

var peripheral: CBPeripheral? = nil

var characteristic: Characteristic? = nil

Var uuid : String
let UUID_ISDT_BLE_DATA_CHANNEL_FFF7 = "FFF7" //BLE Feature

var bleAvailable:Bool = false
static let sharedInstance = BleManager()

typealias ScanClbkType = (_ device: CBPeripheral, _ name: String)->()
var scanClbk: ScanClbkType?

private override init(var peripheral : CBPeripheral?
) {
initBluetooth()

}
}
```

2.2 搜索设备调用 centralManager!.scanForPeripherals 函数

2.2 Search for devices and call the function centralManager!.scanForPeripherals

```
//Initial centralManager
func initBluetooth() {
        centralManager = CBCentralManager.init(delegate: self, queue: nil)
}
```

```
}
```

```
//Initial centralManager callback
func centralManagerDidUpdateState(_ central: CBCentralManager) {
        switch central.state{
        case .unknown:
            bleAvailable = false
            Logger.debug("unknown")
        case .resetting:
            bleAvailable = false
            Logger.debug("resetting")
        case .unsupported:
            bleAvailable = false
            Logger.debug("unsupported")
        case .unauthorized:
            bleAvailable = false
            Logger.debug("unauthorized")
        case .poweredOff:
            bleAvailable = false
            Logger.debug("poweredOff")
        case .poweredOn:
            bleAvailable = true
            Logger.debug("poweredOn")
        }
        btState = central.state
    }
```

```
//Search device
func startScanDevice() {

if !bleAvailable {
            return
}
    centralManager!.scanForPeripherals(withServices: nil, options: nil)
}
```

2.3 发现外围设备进行过滤

2.3 Discovery of peripheral devices and filtering

```
//Callback when peripheral devices are found
func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {

if nil != peripheral.name {

Logger.debug("Name=\(peripheral.name!) UUID=\(peripheral.identifier)")
```

```
var peripheralName = peripheral.name!
 if let nameTmp = advertisementDaa["kCBAdvDataLocalName"] as? String {
    peripheralName = nameTmp
}

if currentDeviceScan != nil {
if  .orderedSame==currentDeviceScan!.identifier.caseInsensitiveCompare(peripheral.i
dentifier.uuidString) {
    isdtScanner.found = true
    peripheral = peripheral
    stopScanDevice()
}

} else if peripheralName.hasPrefix("ISDT1") {//Filter unbound devices
if scanClbk != nil {
Logger.debug("peripheralName=\(peripheralName)UUID=\(peripheral.identifier)")
scanClbk!(peripheral, peripheralName)
            }
          }
      }
}

func stopScanDevice() {
      if !btAvailable {
          return
```

```
        }
        if centralManager!.isScanning {
            Logger.debug("Stop scan")
            // © = nil
            centralManager!.stopScan()
        }
    }
```

2.4 创建 ScanViewController 并引用 BLEManager class 得到搜索到的设备

2.4 Create ScanViewController and reference the BLEManager class to get the discovered device

```
let bleManager = BleManager.sharedInstance

bleMng.scanClbk = {
                (_ device: CBPeripheral, _ name: String)->() in
                if nil == device.name {
                    return
                }
                if name.count < 13{
                    return
                }

                if !name.hasPrefix(“ISDT1”) {
                    return
                }

               Logger.debug("ret ",name)
                      let typeIndexStart = name.index(name.startIndex, offsetBy: ISDT_WIRELESS_DEVICE_IN_BIND_PREFIX.count)
                 let typeIndexEnd = name.index(name.startIndex, offsetBy: ISDT_WIRELESS_DEVICE_TAG_SIZE)

    let deviceName    = String(name[typeIndexStart..<typeIndexEnd])
let nameIndexStart= name.index(name.startIndex, offsetBy: 13)
let userDeviceName = String(name[nameIndexStart...])
let devicePeripheral = device
}
```

2.5 连接设备调用 centralManager!.connect 函数

2.5 Connect the device to call the centralManager!.connect function

```
func connectDevice(peripheral: CBPeripheral) {
            centralManager!.connect(peripheral, options: nil)
}
  //Bluetooth connection success callback
func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
            Logger.debug("connected OK")
            stopScanDevice()
            peripheral.delegate = self
            peripheral.discoverServices(nil)
}

//Bluetooth connection failure callback
func centralManager(_ central: CBCentralManager, didFailToConnect peripheral: CBPeripheral, error: Error?) {
                                 Logger.debug("connect failed, device\(peripheral.name!),reason\(String(describing: error))")
      }
```

2.6 发现服务

2.6 Discover service

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
      if nil != error {
         Logger.debug("discover error: \(String(describing: error))")
         return
      }
      if peripheral.services?.count == 0 {
         Logger.debug("No services")
      }
      for service: CBService in peripheral.services! {
         Logger.debug("serviceUUID=\(service.uuid)")
         peripheral.discoverCharacteristics(nil, for: service)
      }
  }
```

2.7 发现特征并申请 FFF7 特征

2.7 Discover features and apply for FFF7 features

```
func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
        if nil != error {
            Logger.debug("discover chara error: \(String(describing: error))")
        } else {
                didFindCharacteristic(chaArray: service.characteristics!, peripheral: peripheral)
            findDeviceTest = true
        }
}
func didFindCharacteristic(chaArray: [CBCharacteristic], peripheral: CBPeripheral) -> Void {
        Logger.debug("didFind")

        characteristic = nil

for characteristic: CBCharacteristic in chaArray {

If
characteristic.uuid.uuidString.caseInsensitiveCompare(BleManager.UUID_ISDT_BLE_
DATA_CHANNEL_FFF7) == .orderedSame {

                Logger.debug("find: \(characteristic.uuid)")

                uuid = peripheral.identifier.uuidString

                characteristic = characteristic

                peripheral.setNotifyValue(true, for: characteristic)

            }

}

          isdtConnecter.connected = true

        }


//Update notification status callback

func peripheral(_ peripheral: CBPeripheral, didUpdateNotificationStateFor
characteristic: CBCharacteristic, error: Error?) {

        if nil != error {

                        Logger.debug("update value didUpdateNotificationStateFor:
\(String(describing: error))")

        }

}
```

3、绑定设备

注：设备必须为绑定状态

3. Device binding

Note: The device must be in the binding state.

3.1 创建 IsdtPackBase.swift

3.1 Create IsdtPackBase.swift

```
import Foundation

class IsdtPackBase {
    static let SYNC_WORD:UInt8 = 0xAA
    static let ADDRESS:UInt8 = 0x12
    static let ADDRESS_RECV:UInt8 = 0x21
    static let MAX_ASSEMBLE_SEND_PACK_NUM = 200
    static let MAX_BLE_LEN_BIG: Int = 140
static let MAX_BLE_LEN_FFF8: Int = 150

    //Filter out more than two consecutive 0xAA
    func assembleByte(cmd: inout [UInt8]?, byte: UInt8) {
        if nil == cmd {
            return
        }

        if IsdtPackBase.SYNC_WORD == byte {
            cmd?.append(byte)
        }
        cmd?.append(byte)
    }

    func getByte(cmd: [UInt8]?, index: inout Int) -> UInt8 {
        if nil == cmd {
            return UInt8(0)
        }
        if index < cmd!.count {
            let ret = cmd![index]
            index += 1
            return ret
        } else {
            return UInt8(0)
        }
}

func setCmdWord(cmdWord: UInt8) {
        self.cmdWord = cmdWord
}

func parse(cmd: [UInt8]?) {

}

func assemble() -> [UInt8]? {
        return nil
}

static func createInstance(_ cmd: [UInt8]) -> IsdtPackBase? {
        var isdtPackRet: IsdtPackBase? = nil

        switch cmd[0] {
        case 0x19:
            isdtPackRet = IsdtPackBleBind()
        case 0xE1:
                isdtPackRet = IsdtPackBleHardInfo()
            default:
                isdtPackRet = nil
            }

            if nil != isdtPackRet {
                                              isdtPackRet!.timeStamp      =
Date().timeIntervalSince1970
                isdtPackRet!.parse(cmd: cmd)
            }
            return isdtPackRet
        }

//Send data calibration
        class  func  assembleIsdtPackSend(isdtPackBase:  IsdtPackBase?,
maxLenMinus1: Int?) -> PackSend? {
            var packSendRet:PackSend? = nil
            let timeStampNow = Date().timeIntervalSince1970
            var maxLenMinus1Tmp = MAX_BLE_LEN_SMALL - 1
            if nil != maxLenMinus1 {
                maxLenMinus1Tmp = maxLenMinus1!
            }
            isdtPackSendListLock.lock()
            do {
                                    while    isdtPackSendList.count     >
MAX_ASSEMBLE_SEND_PACK_NUM {
                    isdtPackSendList.remove(at: 0)
                }
                for (i, packSendTmp) in isdtPackSendList.enumerated().reversed() {
                    if packSendTmp.sent {
                        isdtPackSendList.remove(at: i)
                    } else if timeStampNow - packSendTmp.timeStamp < 0 || timeStampNow - packSendTmp.timeStamp > 1 {
                        isdtPackSendList.remove(at: i)
                    }
                }

                if isdtPackSendList.count == 0 {
                    packSendRet = nil
                } else {
                    packSendRet = isdtPackSendList[0]
                }
                if nil == isdtPackBase {
                    return packSendRet
                }
                let cmdList = isdtPackBase!.assemble()
                if nil == cmdList {
                    return packSendRet
                }
                if cmdList![0] + 1 != cmdList!.count {
                    return packSendRet
                }
```

```
                  var sendCount = 0
                  let sendBytesAll = cmdList!.count - 1

                  for i in 0...(sendBytesAll/maxLenMinus1Tmp) {
                       if sendBytesAll - i * maxLenMinus1Tmp >= maxLenMinus1Tmp {
                            sendCount = maxLenMinus1Tmp
                       } else {
                            sendCount = sendBytesAll - i * maxLenMinus1Tmp
                       }
                       if 0 == sendCount {
                            break
                       }
                       var data:[UInt8]? = []
                       data!.append(UInt8(sendCount & 0xFF))
                       for j in 0..<sendCount {
                            data!.append(cmdList![1+i*maxLenMinus1Tmp+j])
                       }
                       isdtPackSendList.append(PackSend(content: data))
                  }
             } catch {

             }
```

```
            if nil == packSendRet && isdtPackSendList.count > 0 {
                  packSendRet = isdtPackSendList[0]
            }
            isdtPackSendListLock.unlock()

            return packSendRet
      }
//Receive data calibration
       class func parseIsdtPack(_ data: UnsafePointer<UInt8>?) ->
IsdtPackBase? {
            var findRead: Bool = false
            var isdtPackBaseTmp: IsdtPackBase? = nil
            isdtPackListLock.lock()
            do {
                  while isdtPackList.count > MAX_PARSE_PACK_NUM {
                        isdtPackList.removeFirst()
                  }
                                 for (i, isdtPackBaseTmp) in
isdtPackList.enumerated().reversed() {
                        if findRead {
                              isdtPackList.remove(at: i)
                             } else if isdtPackBaseTmp.readFlag ||
Date().timeIntervalSince1970 - isdtPackBaseTmp.timeStamp > 5 {
                              findRead = true
                              isdtPackList.remove(at: i)
                        }
                  }
```

```
if nil == data {
      if(isdtPackList.count == 0) {
            return nil
      } else {
            return isdtPackList[0]
      }
}
let bleLen = data![0]
if bleLen > MAX_BLE_LEN_BIG {
      if(isdtPackList.count == 0) {
            return nil
      } else {
            return isdtPackList[0]
      }
}

for i in 1...bleLen {
      if SYNC_WORD == data![Int(i)] {
            syncByteCount += 1
            if (syncByteCount & 0x01) == 0x01 {
                  continue
            }
      } else {
            if (syncByteCount & 0x01) == 0x01 {
                  parseState = parseStateWaitAddress
```

```
                    }
                    syncByteCount = 0
                }

                switch parseState {
                case parseStateWaitAddress:
                                Logger.debug("parse  address:
data[\(i)]=\(data![Int(i)])")
                    if ADDRESS_RECV == data![Int(i)] {
                        parseState = parseStateWaitLength
                         checkSum = Int32(data![Int(i)] &
0xFF)
                    } else {
                                Logger.debug("parse  address
error")
                        parseState = parseStateWaitSync
                    }
                case parseStateWaitLength:
                    Logger.debug("parse length")
                    dataLength = data![Int(i)]
                    dataCount = UInt32(data![Int(i)])
                    if dataCount == 0 {
                        break
                    }
                    checkSum += Int32(data![Int(i)] & 0xFF)
                    parseState = parseStateWaitData
                    buffer.removeAll()
                case parseStateWaitData:
                    // NSLog("parse data")
                    buffer.append(data![Int(i)])
                    checkSum += Int32(data![Int(i)] & 0xFF)
                    dataCount -= 1
                    if 0 == dataCount {
                        parseState = parseStateWaitChkSum
                    }
                case parseStateWaitChkSum:
                    Logger.debug("parse checksum")
                    parseState = parseStateWaitSync
                    if (data![Int(i)] & 0xFF) == (checkSum &
0xFF){
                        Logger.debug("parse checksum ok")
                                let newPack: IsdtPackBase? =
createInstance(buffer)
                        if 0 == isdtPackList.count {
                                if nil == newPack {
                                        isdtPackBaseTmp = nil
                                } else {
                                        isdtPackList.append(n
ewPack!)
                                                     isdtPackBaseTmp =
isdtPackList[0]
                                }
                        } else {
                                if nil != newPack {
                                        isdtPackList.append(n
ewPack!)
                                }
                                        isdtPackBaseTmp =
isdtPackList[0]
                        }
                    } else {
                        Logger.debug("parse checksum err:
cmd[\(i)]=\(data![Int(i)]),chk=\(checkSum&0xff)")
                    }

                default:
                    // var defaultString = "defaultString: "
                    /// NSLog(defaultString)
                    parseState = parseStateWaitSync
                    break
                }
            }
        } catch {
        }
        isdtPackListLock.unlock()
        return isdtPackBaseTmp
    }
     }
```

3.2 创建发送绑定信息包 IsdtPackBaseReq.swift

3.2 Create and send the binding information package IsdtPackBaseReq.swift

```
import Foundation

class IsdtPackBleBindReq: IsdtPackBase {
      private var cmd:[UInt8]?
      var uuid:String = ""

      override func assemble() -> [UInt8]? {
            if nil == cmd {
                  cmd = []
            }

            let uuidTmp = UUID.init(uuidString: uuid)
            if nil == uuidTmp {
                  return nil
            }

            var checksum:Int = 0
            cmd!.removeAll()
            cmd!.append(0)
            cmd!.append(IsdtPackBase.SYNC_WORD)
            assembleByte(cmd: &cmd, byte: IsdtPackBase.ADDRESS)
            checksum += (Int(IsdtPackBase.ADDRESS) & 0xFF)
            assembleByte(cmd: &cmd, byte: UInt8(0x11))
            checksum += 0x11
            assembleByte(cmd: &cmd, byte: UInt8(0x18))
```

```
checksum += 0x18



assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.0)
checksum += Int(uuidTmp!.uuid.0)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.1)
checksum += Int(uuidTmp!.uuid.1)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.2)
checksum += Int(uuidTmp!.uuid.2)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.3)
checksum += Int(uuidTmp!.uuid.3)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.4)
checksum += Int(uuidTmp!.uuid.4)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.5)
checksum += Int(uuidTmp!.uuid.5)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.6)
checksum += Int(uuidTmp!.uuid.6)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.7)
checksum += Int(uuidTmp!.uuid.7)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.8)
checksum += Int(uuidTmp!.uuid.8)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.9)
checksum += Int(uuidTmp!.uuid.9)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.10)
checksum += Int(uuidTmp!.uuid.10)
assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.11)
```

```
            checksum += Int(uuidTmp!.uuid.11)
            assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.12)
            checksum += Int(uuidTmp!.uuid.12)
            assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.13)
            checksum += Int(uuidTmp!.uuid.13)
            assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.14)
            checksum += Int(uuidTmp!.uuid.14)
            assembleByte(cmd: &cmd, byte: uuidTmp!.uuid.15)
            checksum += Int(uuidTmp!.uuid.15)

            assembleByte(cmd: &cmd, byte: (UInt8)(checksum & 0xFF))
            cmd![0] = UInt8((cmd!.count - 1) & 0xFF)
            return cmd

        }
}
```

3.3 创建接收绑定信息解析包 IsdtPackBaseBind.swift

3.3 Create a parsing package IsdtPackBaseBind.swift for receiving binding information

```
import Foundation

class IsdtPackBleBind: IsdtPackBase {
      var bound = false
      override func parse(cmd: [UInt8]?) {
            if nil == cmd {
                  return
            }
            var i:Int = 0
            var tmp:UInt8 = 0
            tmp = getByte(cmd: cmd, index: &i)
            setCmdWord(cmdWord: tmp)
            tmp = getByte(cmd: cmd, index: &i)
            bound = (tmp == UInt8(0))

      }
}
```

3.4 创建 PackSend.swift

3.4 Create PackSend.swift

```
import Foundation
```

```
class PackSend {
      var sent:Bool
      var sendContent:[UInt8]?
      var timeStamp:TimeInterval
      init(content: [UInt8]?) {
            sendContent = content
            sent = false
            timeStamp = Date().timeIntervalSince1970
      }
}
```

3.5 发送绑定请求

3.5 Send binding request

```
let   packSendTmp   =   IsdtPackBase.assembleIsdtPackSend(isdtPackBase:
isdtPackBleBindReq, maxLenMinus1: 139)
doPackSend(packSendTmp )
```

```
func doPackSend(packSend: PackSend?) {
            if let sendContent = packSend?.sendContent {
                  let data = Data.init(bytes: sendContent)
                  var s = ””
                  for i in 0..<sendContent.count {
```

```
                s += String(format: "%02x ", sendContent[i])
            }
            Logger.debug("start write: " + s)
            if characteristic != nil {
                Logger.debug("characteristic: \(String(describing: characteristic))")
                peripheral?.writeValue(data, for: characteristic!, type: .withResponse)
            }
        }
    }
```

3.6 接收设备发送的信息并判断

3.6 Receive the information sent by the device and make a judgement

```
func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {
        if nil != error {
            Logger.debug("update value error: \(String(describing: error))")
            return
        }

        let data = characteristic.value

        valNotifyTmp = [UInt8](data!)
        if nil != valNotifyTmp {
            Logger.debug("start recv: \(valNotifyTmp!)")
        }
        let isdtPackBaseTmp = IsdtPackBase.parseIsdtPack(valNotifyTmp)

        if nil != isdtPackBaseTmp {
            isdtPackBaseTmp!.readFlag = true
            if let isdtPackBind = isdtPackBaseTmp as? IsdtPackBleBind {
    //The binding request information responded by the device and make a judgment

                if isdtPackBind.bound {
                    Logger.debug("Bind ok");
                    isdtBinder.bound = true
                } else {
                    isdtBinder.boundInfoChanged = true
                    Logger.debug("Bind error")
                }
            } else if (isdtPackBaseTmp as? IsdtPackBleHardInfo) != nil {
                Logger.debug("Version query ok");
                isdtVersionQuerier.queried = true
                isdtVersionQuerier.isdtPackBaseRead = isdtPackBaseTmp
            }
```

}

3.7 绑定成功后请创建数据库保存设备一下信息

3.7 After successful binding, please create a database to save the device information

userDeviceName

deviceName

Mac

Uuid

3.8 当断开连接后重新连接请重复绑定请求，当设备回复请求是 0xFF 时表示设备已经被其他手机绑定。请删除保存的设备信息重新绑定。

为了确保手机与设备是否保持通讯必须要一条协议一直循环发送，发送时间间隔请在 50ms 以上。当设备在 5 秒内无回应表示断开连接。

3.8 When disconnecting and reconnecting, please repeat the binding request. If the device responds with 0xFF, it means that the device has been bound by other mobile phones. Please delete the saved device information and bind again.

To ensure that the mobile phone and the device maintain communication, a protocol must be sent cyclically with an interval greater than 50ms between each instance. If the device does not respond within 5 seconds, it means disconnection.

ISDT 深圳艾斯特创新科技有限公司