

Vispek SDK iOS 端使用说明

本文档说明如何在 iOS App 中集成

libWSBKSDK.framework，并通过蓝牙连接设备、读取设备信息和获取光谱数据。若需要完整示例，请参考 ios_SDK/SDKDemo。

1. SDK 包内容

```
ios_SDK
├── Bluetooth_Communication_Protocol_CN.xlsx #
├── SDK
│   ├── iphoneos/libWSBKSDK.framework # iOS framework
│   └── SDKDemo # Objective-C
├── SDK
├── Vispek SDK IOS .md # Markdown
├── Vispek SDK IOS .pdf # PDF
└── *.png # Xcode
```

核心示例文件：

- SDKDemo/SDKDemo/ViewController.m：代理设置、扫描、连接、发送指令和解析回调。
- SDK/iphoneos/libWSBKSDK.framework/Headers/WSBKBlueToothSDK.h：公开 API。
- SDK/iphoneos/libWSBKSDK.framework/Headers/WSBKPeripheralInfoModel.h：扫描设备模型。

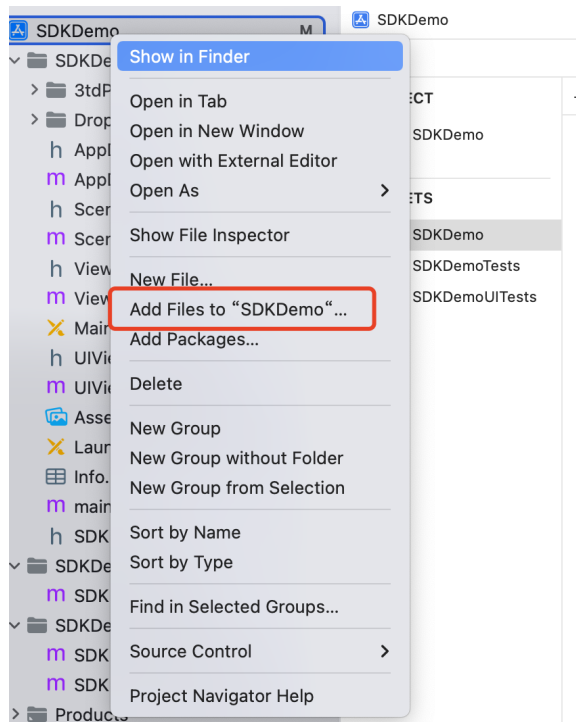
2. 环境要求

- iOS 10.0 或以上版本。
- Xcode。
- 当前交付目录提供 iphoneos 真机包，主要用于真机调试和发布。
- 如技术支持另行提供 simulator 包，仅用于开发调试；正式发布时请使用真机包，避免 App Store 审核或架构校验问题。

3. 集成 framework

3.1 添加 framework 文件

- 将 libWSBKSDK.framework 拷贝到客户项目目录。
- 打开 Xcode，选择 Add Files to " "，把 framework 添加到工程。



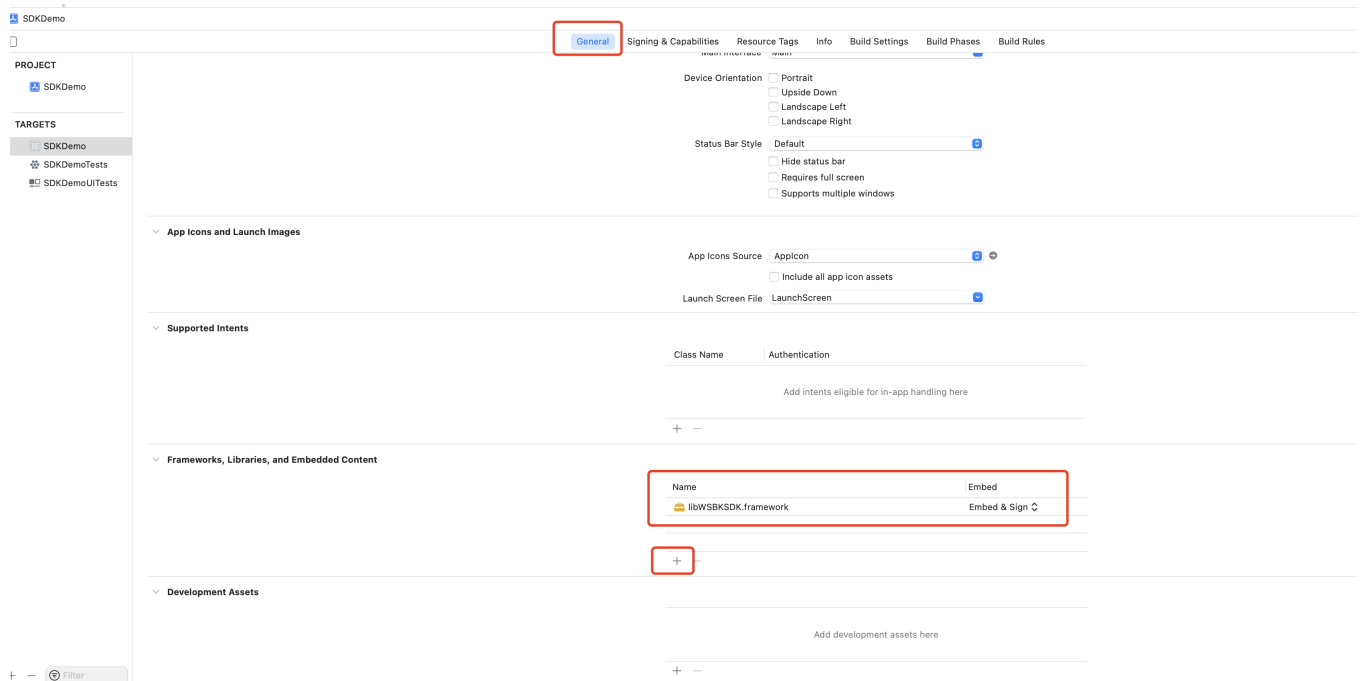
add_frameworks.png

3.2 设置 Embed & Sign

打开 Xcode, 选择:

TARGETS -> General -> Frameworks, Libraries, and Embedded Content

添加 libWSBKSDK.framework, 并将 Embed 设置为 Embed & Sign。



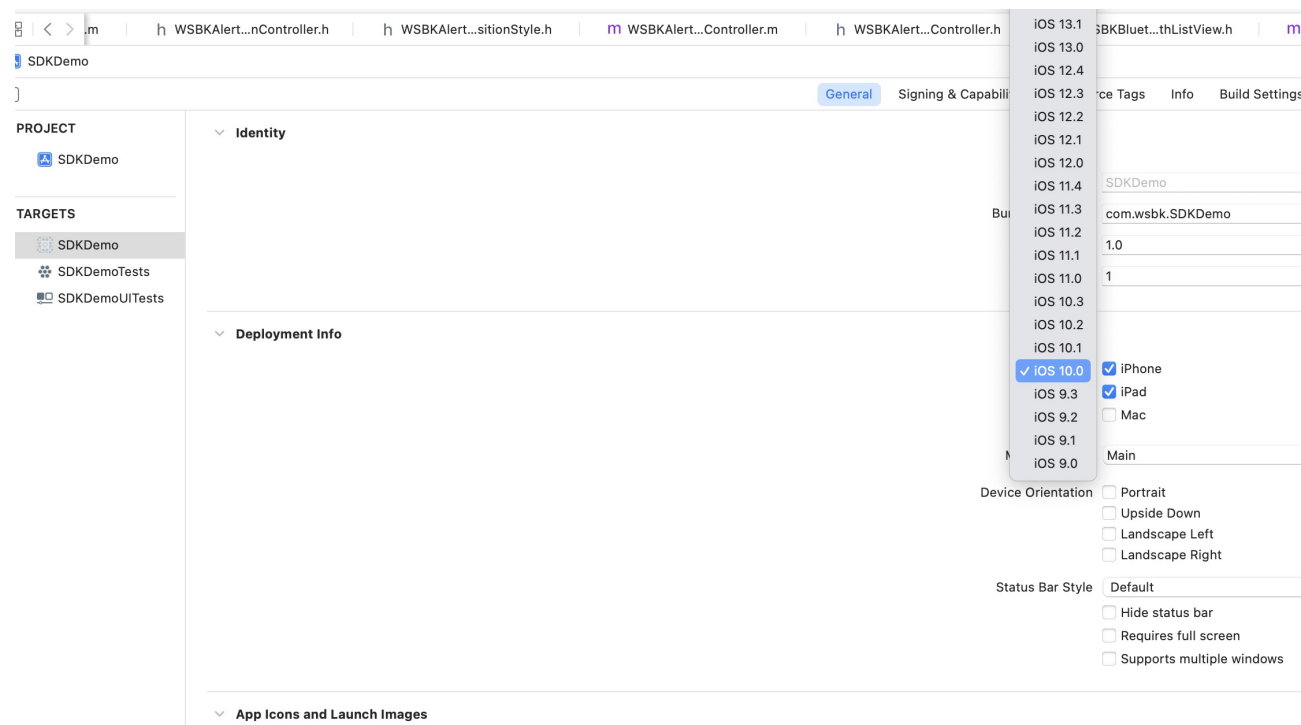
set_embed_sign.png

3.3 设置 Deployment Target

打开 Xcode, 选择:

TARGETS -> General -> Deployment Info -> iOS Deployment Target

设置为 10.0 或以上版本。



set_app_ios_version.png

3.4 配置蓝牙权限

在 Info.plist 中添加蓝牙权限说明:

```

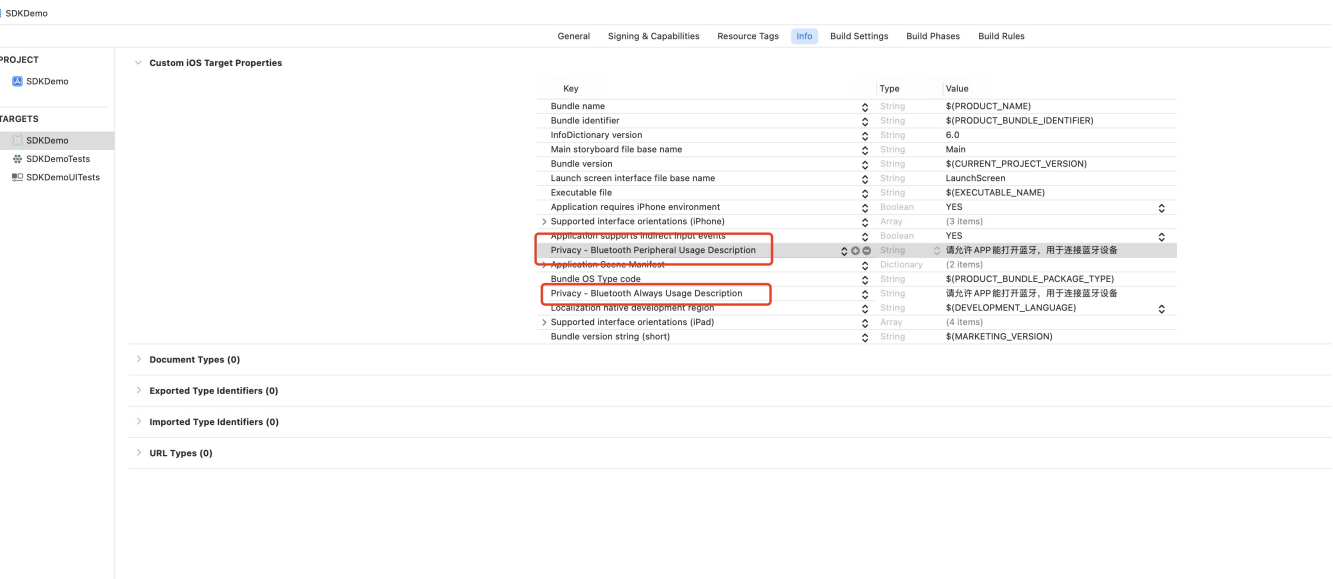
<key>NSBluetoothAlwaysUsageDescription</key>
<string>    App        Vispek    </string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>    App        Vispek    </string>

```

如果客户 App 自身还使用定位能力，再按业务需要添加定位权限说明。

Xcode 可视化配置位置:

TARGETS -> Info -> Custom iOS Target Properties



add_authorization.png

4. 导入 SDK

在需要使用 SDK 的 Objective-C 文件中导入：

```
#import <libWSBKSDK/libWSBKSDK.h>
```

控制器遵循 WSBKBluetoothSDKDelegate：

```
@interface ViewController () <WSBKBluetoothSDKDelegate>
@end
```

在合适的生命周期中设置代理：

```
- (void)viewDidLoad {
    [super viewDidLoad];
    [WSBKBluetoothSDK shared].delegate = self;
}
```

5. 扫描蓝牙设备

开始扫描：

```
[[WSBKBluetoothSDK shared] startScanPeripheral];
```

停止扫描：

```
[[WSBKBluetoothSDK shared] stopScanPeripheral];
```

扫描结果通过代理返回：

```
- (void)getScanResultPeripherals:(NSArray *)peripheralInfoArr {
    // peripheralInfoArr      WSBKPeripheralInfoModel
}
```

WSBKPeripheralInfoModel 常用字段：

```
@property (nonatomic, strong) NSNumber *RSSI;
@property (nonatomic, strong) CBPeripheral *peripheral;
@property (nonatomic, strong) NSDictionary *advertisementData;
```

客户 UI 可展示设备名称、信号强度等信息，选择设备后使用 peripheral 连接。

6. 连接和断开设备

连接客户选择的设备：

```
[[WSBKBluetoothSDK shared] connectPeripheral:model.peripheral];
```

连接成功：

```
- (void)connectSuccess {
    // UI      “ ”
}
```

连接失败：

```
- (void)connectFailed {
    //
}
```

设备断开：

```
- (void)disconnectPeripheral:(CBPeripheral *)peripheral {  
    //    UI    “    ”  
}
```

断开全部设备：

```
[[WSBKBlueToothSDK shared] disconnectAllPeripherals];
```

断开指定设备：

```
[[WSBKBlueToothSDK shared] disconnectLastPeripheral:peripheral];
```

7. 蓝牙状态代理

SDK 提供以下可选代理方法：

```
- (void)systemBluetoothClose {  
    //  
}  
  
- (void)systemBluetoothOpen {  
    //          SDK          sysytemBluetoothOpen  
}  
  
- (void)getScanResultPeripherals:(NSArray *)peripheralInfoArr {  
    //  
}  
  
- (void)connectSuccess {  
    //  
}  
  
- (void)connectFailed {  
    //  
}  
  
- (void)disconnectPeripheral:(CBPeripheral *)peripheral {  
    //  
}
```

8. 发送指令获取数据

SDK 支持的指令类型：

```
typedef NS_ENUM(NSUInteger, WSBKBlueToothSDKCommandType) {  
    WSBKBlueToothSDKBatteryCommandType, //  
    WSBKBlueToothSDKSpectralCommandType, //  
    WSBKBlueToothSDKTemperatureCommandType, //  
    WSBKBlueToothSDKDeviceVersionCommandType, //  
    WSBKBlueToothSDKDeviceSNCommandType //  
};
```

发送指令：

```
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKBatteryCommandType  
getDeviceData:^(NSString *response, NSString *response2) {  
    // response  
} failed:^(NSString *failed) {  
    // failed  
}];
```

常用示例：

```
//
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKBatteryCommandType
getData:^(NSString *response, NSString *response2) {
self.deviceBatteryPowerLab.text = response;
} failed:^(NSString *failed) {
NSLog(@"failed: %@", failed);
}];

//
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKTemperatureCommandType
getData:^(NSString *response, NSString *response2) {
self.temperatureLab.text = response;
} failed:^(NSString *failed) {
NSLog(@"failed: %@", failed);
}];

//
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKDeviceVersionCommandType
getData:^(NSString *response, NSString *response2) {
self.deviceVersionLab.text = response;
} failed:^(NSString *failed) {
NSLog(@"failed: %@", failed);
}];

//
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKDeviceSNCommandType
getData:^(NSString *response, NSString *response2) {
self.deviceCodeLab.text = response;
} failed:^(NSString *failed) {
NSLog(@"failed: %@", failed);
}];

//
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKSpectralCommandType
getData:^(NSString *response, NSString *response2) {
self.SpectralDataLab.text = response; //
self.highVoltageLab.text = response2; //
} failed:^(NSString *failed) {
NSLog(@"failed: %@", failed);
}];
```

返回值说明：

- 电量、温度、硬件版本、设备编码：结果在 response，response2 通常为空。
- 光谱：response 为低电压光谱值，response2 为高电压光谱值。

9. 其它可用 API

```
//      peripherals
- (NSArray *)findConnectedPeripherals;

//      peripheralName      peripheral
- (CBPeripheral *)findConnectedPeripheral:(NSString *)peripheralName;

//
- (void)searchServerAndCharacteristicUUID;

//
- (void)AutoReconnect:(CBPeripheral *)peripheral;

//
- (void)AutoReconnectCancel:(CBPeripheral *)peripheral;

//      UUID
- (CBPeripheral *)retrievePeripheralWithUUIDString:(NSString *)UUIDString;
```

10. 光谱数据规则

iOS 光谱指令返回两组光谱值。参数名沿用 SDK 现有命名，但按协议含义应理解为高/低电压光谱值：

- response：低电压光谱值。
- response2：高电压光谱值。

如需查看字节级协议字段、命令帧和 CRC 规则，请参考本 SDK 包内的 Bluetooth_Communication_Protocol_CN.xlsx。

如果客户不调用 Vispek 后台接口，而是自建算法平台：

- 可按自建算法平台的输入要求自行组织高电压和低电压光谱值。
- 建议在算法侧明确两组数组与 LED/通道序号的对应关系，便于后续计算和数据追溯。

如果客户调用 Vispek 后台接口：

- 必须按 Vispek 指定顺序拼接完整光谱数据提交。
- 指定顺序为“低电压光谱值在前，高电压光谱值在后”： $l1, l2, \dots, l22, h1, h2, \dots, h22$ 。

11. App Store 审核注意事项

App 使用蓝牙能力时，建议在审核备注中说明蓝牙用途，例如“用于连接 Vispek 硬件设备并读取检测数据”。如审核需要，可提供 App 操作视频链接，展示设备连接和数据读取流程。

发布前请确认：

- 使用的是 iPhone OS 真机 framework。
- Info.plist 中已添加蓝牙权限说明。
- 真机上已验证扫描、连接、断开和数据读取流程。

12. 常见问题

12.1 无法扫描到设备

优先检查：

- 手机蓝牙是否打开。
- App 是否已获得蓝牙权限。
- 当前是否在真机运行。
- 设备是否通电并处于可连接状态。
- 设备是否已被其它 App 或系统连接。

12.2 连接失败

建议检查：

- 是否使用扫描结果中的 CBPeripheral 调用 connectPeripheral:。
- 是否在连接前设置 [WSBKBlueToothSDK shared].delegate = self。
- 是否在连接弹窗取消时调用了 stopScanPeripheral。
- 是否重复连接同一设备。

12.3 发送指令失败或没有返回数据

建议检查：

- 是否已经收到 connectSuccess。
- 是否在连接成功后再发送指令。
- 是否在失败回调中输出 failed 信息。
- 光谱指令返回需要同时读取 response 中的低电压光谱值和 response2 中的高电压光谱值。

13. AI 接入速查

给其它 AI 或开发助手的快速规则：

- iOS SDK 包路径: `ios_SDK/SDK/iphoneos/libWSBKSDK.framework`
- Demo 工程路径: `ios_SDK/SDKDemo`
- 主要示例文件: `ios_SDK/SDKDemo/SDKDemo/ViewController.m`
- 公开头文件: `ios_SDK/SDK/iphoneos/libWSBKSDK.framework/Headers/WSBKBlueToothSDK.h`
- 蓝牙协议表: `ios_SDK/Bluetooth_Communication_Protocol_CN.xlsx`
- 导入方式: `#import <libWSBKSDK/libWSBKSDK.h>`
- 单例入口: `[WSBKBlueToothSDK shared]`
- 代理协议: `WSBKBlueToothSDKDelegate`
- 扫描入口: `startScanPeripheral`
- 连接入口: `connectPeripheral:`
- 发送指令入口: `sendCommandType:getDeviceData:failed:`
- 断开入口: `disconnectAllPeripherals` 或 `disconnectLastPeripheral:`

AI 修改或生成客户接入代码时必须保留以下顺序：

- 集成 `libWSBKSDK.framework` 并设置 Embed & Sign。
- 在 `Info.plist` 中添加蓝牙权限说明。
- 导入 `<libWSBKSDK/libWSBKSDK.h>`。
- 控制器遵循 `WSBKBlueToothSDKDelegate`。
- 设置 `[WSBKBlueToothSDK shared].delegate = self`。
- 调用 `startScanPeripheral` 扫描设备。
- 从 `getScanResultPeripherals:` 返回的模型中取 `CBPeripheral`。
- 调用 `connectPeripheral:` 连接设备。
- 收到 `connectSuccess` 后发送指令。
- 在 `getDeviceData` 成功回调中读取 `response` 和 `response2`。
- 页面退出或客户主动断开时调用断开 API。

不要做的事：

- 不要修改 framework 二进制文件。
- 不要把公开 API 方法名 `sysytemBluetoothOpen` 改成 `systemBluetoothOpen`，该拼写必须以 SDK 头文件为准。
- 不要跳过 SDK 直接使用 `CoreBluetooth` 重写协议，除非客户明确要求绕过 SDK。
- 不要忽略 `response2` 中的高电压光谱值。
- 不要使用 simulator 包发布 App。当前目录提供的是 `iphoneos` 真机包。