

Vispek SDK iOS User Guide

This guide explains how to integrate `libWSBKSDK.framework` into an iOS app, connect to a Vispek Bluetooth device, read device information, and collect spectral data. For a complete working reference, use `ios_SDK_en/SDKDemo`.

1. Package Contents

```
ios_SDK_en
|-- Bluetooth_Communication_Protocol_EN.xlsx # English BLE protocol table
|-- SDK
| `-- iphoneos/libWSBKSDK.framework # iOS device framework
|-- SDKDemo # Objective-C sample project
|-- SDK_Integration_Guide
|-- Vispek_SDK_iOS_User_Guide.md # English Markdown guide
|-- Vispek_SDK_iOS_User_Guide.pdf # English PDF guide
`-- *.png # Xcode setup screenshots
```

Key sample files:

- `SDKDemo/SDKDemo/ViewController.m`: delegate setup, scan, connect, command sending, and response parsing.
- `SDK/iphoneos/libWSBKSDK.framework/Headers/WSBKBluetoothSDK.h`: public SDK API.
- `SDK/iphoneos/libWSBKSDK.framework/Headers/WSBKPeripheralInfoModel.h`: scanned device model.

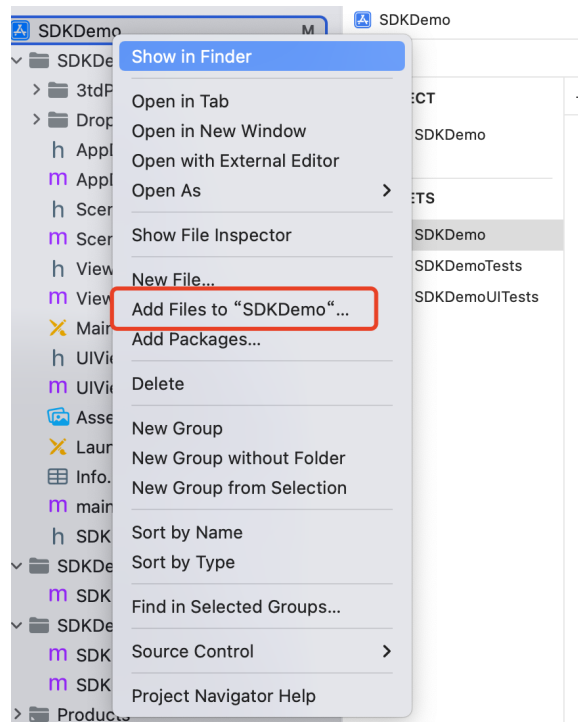
2. Requirements

- iOS 10.0 or later.
- Xcode.
- The delivered iphoneos framework is for device debugging and App Store release.
- If Vispek technical support provides a simulator framework separately, use it for development only. Switch back to the iphoneos framework before release.

3. Integrate the Framework

3.1 Add the Framework

- Copy `libWSBKSDK.framework` into the customer project directory.
- In Xcode, choose `Add Files` to "ProjectName" and add the framework to the project.



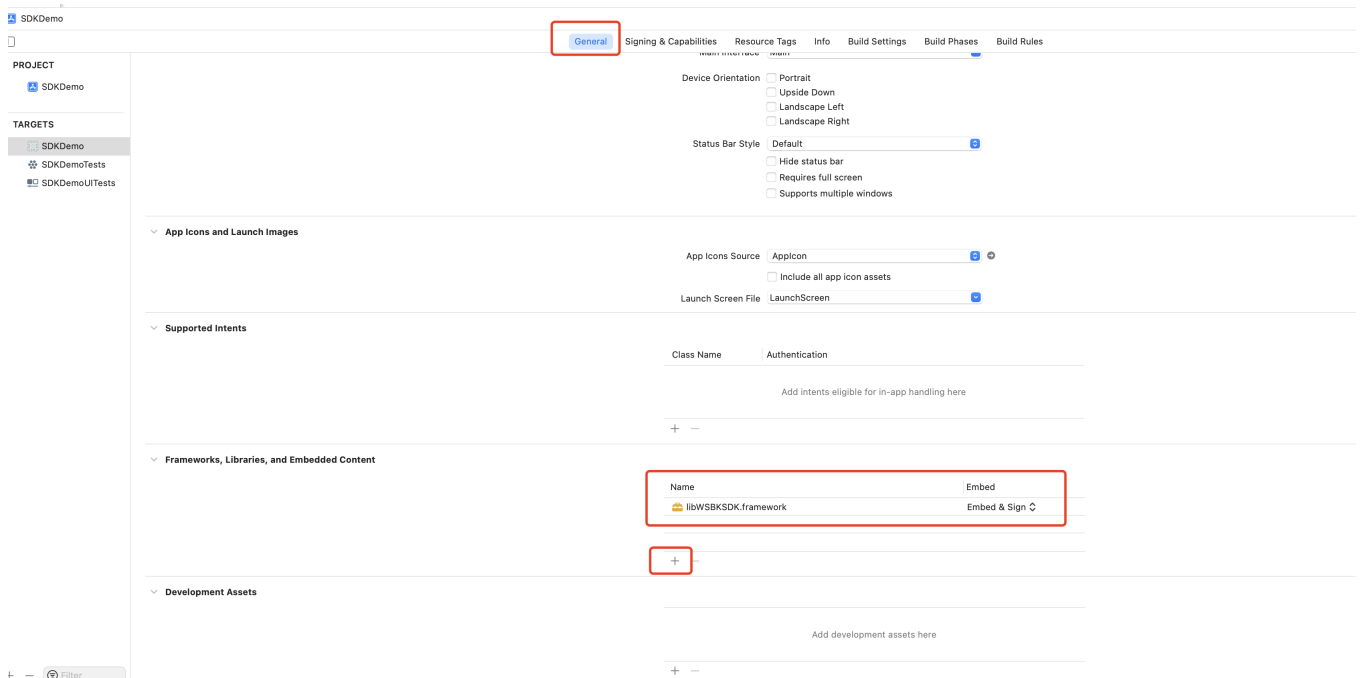
add_frameworks.png

3.2 Set Embed & Sign

In Xcode, open:

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

Add libWSBKSDK.framework and set Embed to Embed & Sign.



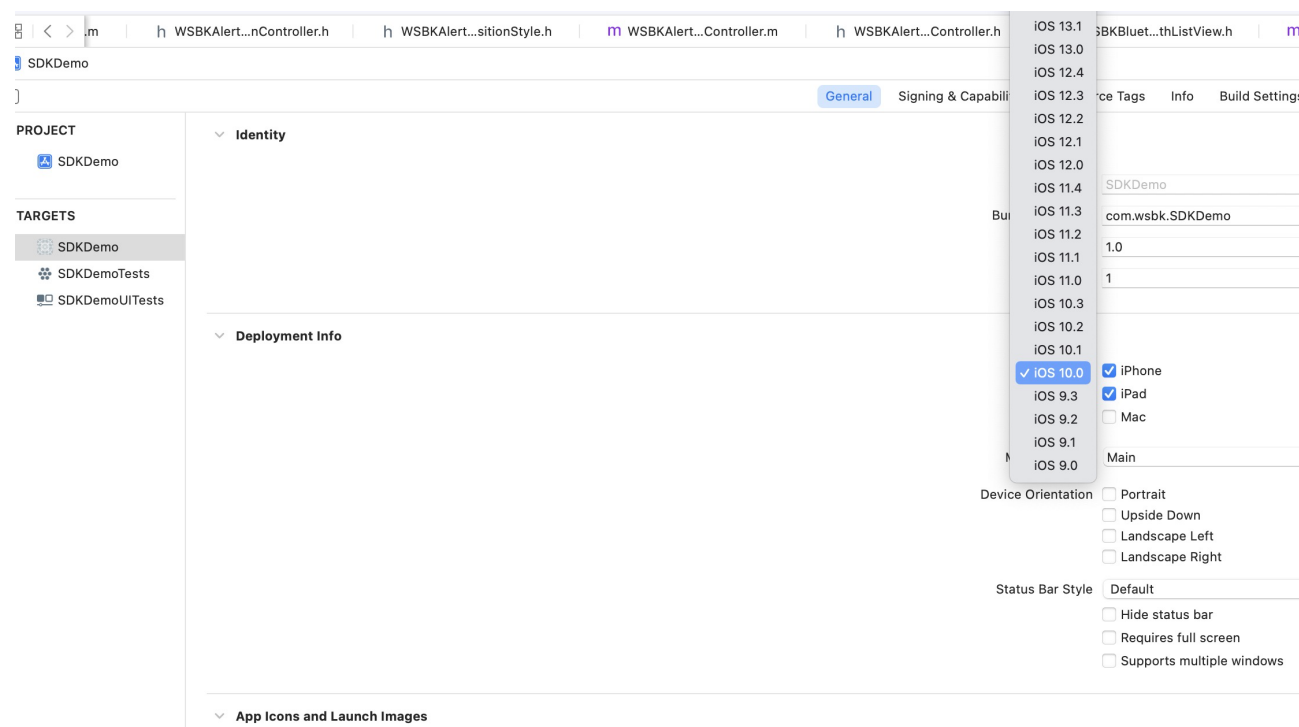
set_embed_sign.png

3.3 Set Deployment Target

In Xcode, open:

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

Set the deployment target to 10.0 or later.



set_app_ios_version.png

3.4 Configure Bluetooth Permissions

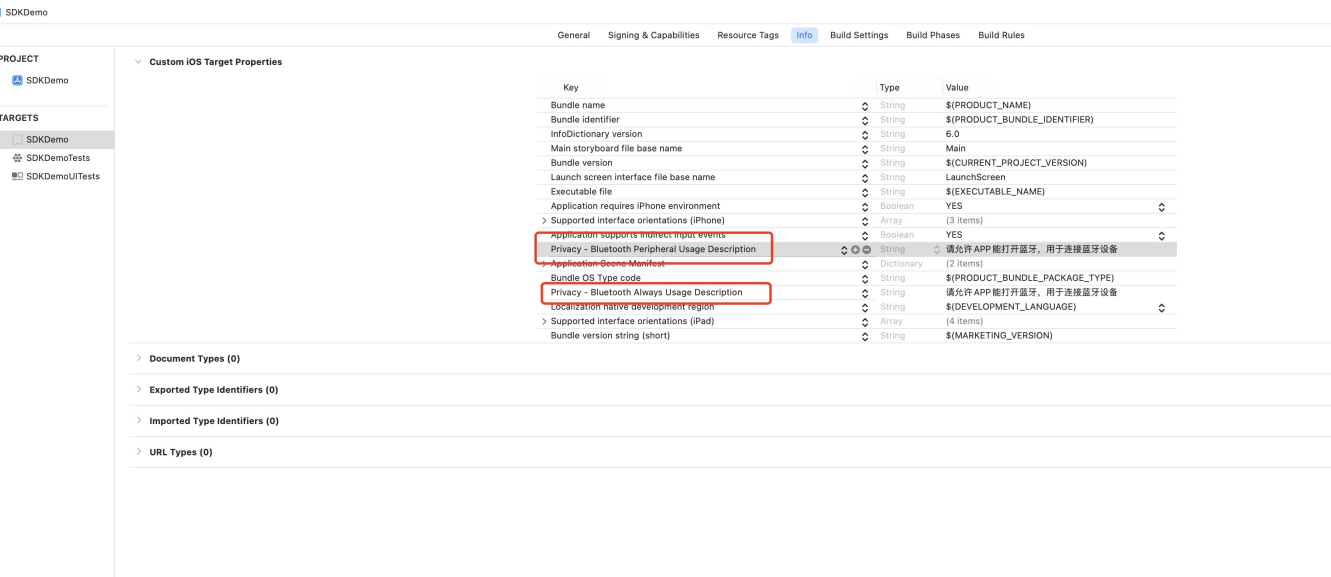
Add Bluetooth usage descriptions to Info.plist:

```
<key>NSBluetoothAlwaysUsageDescription</key>
<string>This app uses Bluetooth to connect to Vispek devices.</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>This app uses Bluetooth to connect to Vispek devices.</string>
```

If the customer app also uses location features, add location permission descriptions according to the app's own business flow.

Visual Xcode path:

```
TARGETS -> Info -> Custom iOS Target Properties
```



add_authorization.png

4. Import the SDK

Import the framework in the Objective-C file that uses the SDK:

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

Make the controller conform to WSBKBluetoothSDKDelegate:

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

Set the delegate in an appropriate lifecycle method:

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

5. Scan for Bluetooth Devices

Start scanning:

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

Stop scanning:

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

Scan results are returned through the delegate:

```
- (void)getScanResultPeripherals:(NSArray *)peripheralInfoArr {
    // Items in peripheralInfoArr are usually WSBKPeripheralInfoModel objects.
}
```

Common WSBKPeripheralInfoModel fields:

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

The customer UI can display device name and signal strength. After the user selects a device, connect with the selected peripheral.

6. Connect and Disconnect

Connect to the selected device:

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

Connection succeeded:

```
- (void)connectSuccess {
    // Update UI state, for example show "Disconnect".
}
```

Connection failed:

```
- (void)connectFailed {
    // Ask the user to retry or check the device state.
}
```

Device disconnected:

```
- (void)disconnectPeripheral:(CBPeripheral *)peripheral {  
    // Update UI state, for example show "Connect Bluetooth".  
}
```

Disconnect all devices:

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

Disconnect a specific device:

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

7. Bluetooth State Delegates

The SDK provides these optional delegate methods:

```
- (void)systemBluetoothClose {  
    // System Bluetooth is off. Ask the user to enable Bluetooth.  
}  
  
- (void)systemBluetoothOpen {  
    // System Bluetooth is on. Note: keep the SDK's public method spelling as  
    // systemBluetoothOpen.  
}  
  
- (void)getScanResultPeripherals:(NSArray *)peripheralInfoArr {  
    // Scanned Bluetooth device list.  
}  
  
- (void)connectSuccess {  
    // Connected successfully.  
}  
  
- (void)connectFailed {  
    // Connection failed.  
}  
  
- (void)disconnectPeripheral:(CBPeripheral *)peripheral {  
    // Current device disconnected.  
}
```

8. Send Commands

Supported command types:

```
typedef NS_ENUM(NSUInteger, WSBKBlueToothSDKCommandType) {  
    WSBKBlueToothSDKBatteryCommandType, // Battery level  
    WSBKBlueToothSDKSpectralCommandType, // Spectral data  
    WSBKBlueToothSDKTemperatureCommandType, // Temperature  
    WSBKBlueToothSDKDeviceVersionCommandType, // Hardware version  
    WSBKBlueToothSDKDeviceSNCommandType // Device serial number  
};
```

Send a command:

```
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKBatteryCommandType  
getDeviceData:^(NSString *response, NSString *response2) {  
    // response contains returned device data.  
} failed:^(NSString *failed) {  
    // failed contains the failure message.  
}];
```

Common examples:

```

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

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

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

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

// Spectral data
[[WSBKBlueToothSDK shared] sendCommandType:WSBKBlueToothSDKSpectralCommandType
getDeviceData:^(NSString *response, NSString *response2) {
self.SpectralDataLab.text = response; // Low-voltage spectral values
self.highVoltageLab.text = response2; // High-voltage spectral values
} failed:^(NSString *failed) {
NSLog(@"failed: %@", failed);
}];

```

Return value notes:

- Battery, temperature, hardware version, and serial number: the result is in response; response2 is usually empty.
- Spectral data: response contains low-voltage spectral values; response2 contains high-voltage spectral values.

9. Other Available APIs

```

// Gets the currently connected peripherals.
- (NSArray *)findConnectedPeripherals;

// Finds the currently connected peripheral by peripheral name.
- (CBPeripheral *)findConnectedPeripheral:(NSString *)peripheralName;

// Searches services and characteristics. Valid after connection succeeds.
- (void)searchServerAndCharacteristicUUID;

// Adds a peripheral to the auto-reconnect list.
- (void)AutoReconnect:(CBPeripheral *)peripheral;

// Removes a peripheral from the auto-reconnect list.
- (void)AutoReconnectCancel:(CBPeripheral *)peripheral;

// Retrieves a paired peripheral by UUID string. The returned peripheral can be connected
directly without scanning.
- (CBPeripheral *)retrievePeripheralWithUUIDString:(NSString *)UUIDString;

```

10. Spectral Data Rules

The iOS spectral command returns two spectral arrays. The parameter names are fixed by the SDK API, while the protocol-level meaning is high/low voltage:

- response: low-voltage spectral values.

- response2: high-voltage spectral values.

For byte-level protocol fields, command frames, and CRC rules, see `Bluetooth_Communication_Protocol_EN.xlsx` included in this SDK package.

If the customer uses their own algorithm platform instead of the Vispek backend:

- The customer may arrange high-voltage and low-voltage spectral values according to the input requirements of their own algorithm platform.
- Keep the LED/channel mapping between the two arrays clear for calculation and traceability.

If the customer calls the Vispek backend:

- Submit the complete spectral payload in the order required by Vispek.
- The required order is low-voltage values first, followed by high-voltage values: `l1, l2, . . . , l22, h1, h2, . . . , h22`.

11. App Store Review Notes

When the app uses Bluetooth, explain the Bluetooth purpose in App Review notes, for example: "The app uses Bluetooth to connect to Vispek hardware devices and read test data." If requested, provide a video link showing device connection and data reading.

Before release, confirm:

- The app uses the `iphoneos` framework.
- `Info.plist` contains Bluetooth usage descriptions.
- Scan, connect, disconnect, and data reading are verified on a real device.

12. Troubleshooting

No Devices Found

Check:

- Phone Bluetooth is enabled.
- The app has Bluetooth permission.
- The app is running on a real device.
- The device is powered on and connectable.
- The device is not already connected by another app or the system.

Connection Fails

Check:

- The `CBPeripheral` from scan results is passed to `connectPeripheral:`.
- `[WSBKBlueToothSDK shared].delegate = self` is set before connecting.
- `stopScanPeripheral` is called when the connection picker is canceled.
- The same device is not being connected repeatedly.

Command Fails or No Data Returns

Check:

- `connectSuccess` has been received.
- Commands are sent only after connection succeeds.
- The failed callback logs the failure message.

- Spectral commands read both low-voltage values in response and high-voltage values in response2.

13. AI Integration Quick Reference

Use these notes when another AI or developer assistant helps with integration:

- iOS SDK package: `ios_SDK_en/SDK/iphoneos/libWSBKSDK.framework`
- Demo project: `ios_SDK_en/SDKDemo`
- Main sample file: `ios_SDK_en/SDKDemo/SDKDemo/ViewController.m`
- Public header: `ios_SDK_en/SDK/iphoneos/libWSBKSDK.framework/Headers/WSBKBlueToothSDK.h`
- BLE protocol table: `ios_SDK_en/Bluetooth_Communication_Protocol_EN.xlsx`
- Import: `#import <libWSBKSDK/libWSBKSDK.h>`
- Singleton entry: `[WSBKBlueToothSDK shared]`
- Delegate protocol: `WSBKBlueToothSDKDelegate`
- Scan entry: `startScanPeripheral`
- Connect entry: `connectPeripheral:`
- Send command entry: `sendCommandType:getDeviceData:failed:`
- Disconnect entry: `disconnectAllPeripherals` or `disconnectLastPeripheral:`

Preserve this order when generating customer integration code:

- Integrate `libWSBKSDK.framework` and set Embed & Sign.
- Add Bluetooth usage descriptions to `Info.plist`.
- Import `<libWSBKSDK/libWSBKSDK.h>`.
- Make the controller conform to `WSBKBlueToothSDKDelegate`.
- Set `[WSBKBlueToothSDK shared].delegate = self`.
- Call `startScanPeripheral` to scan devices.
- Get `CBPeripheral` from the model returned by `getScanResultPeripherals:`.
- Call `connectPeripheral:` to connect.
- Send commands after receiving `connectSuccess`.
- Read response and response2 in the success callback.
- Disconnect when the page exits or the user disconnects manually.

Do not:

- Modify framework binary files.
- Rename the public API method `sysytemBluetoothOpen` to `systemBluetoothOpen`; keep the spelling from the SDK header.
- Skip the SDK and rewrite the BLE protocol with `CoreBluetooth` unless the customer explicitly requests it.
- Ignore the high-voltage spectral values in response2.
- Release the app with a simulator framework. This package provides the `iphoneos` device framework.