FeasyBlue iOS-SDK Interface Guide

Table of contents

[中文版]

Corresponding SDK Version	Doc. Version	Author
FeasyBlue SDK iOS 1.0.4.zip	1.0.4.0	Chen Changhua

Chapter 1

1. Access method

1. Unzip the SDK and drag all the extracted files into the project.
2. Set linker flags，TARGETS -> Build Settings -> Linking -> Other Linker Flags，Add-ObjC

Chapter 2

2. Search (FEBluetoothSDK)

2.1 2.1 Introduction

Searching for BLE Devices

2.2 2.2 Method

```
/// Search for common communication devices
/// @param scanBlock callback (peripheral:Searched Devices)
- (void)scan:(void(^)(FEPeripheral* peripheral, BOOL␣
↪isNew))scanBlock;
```

2.3 2.3 Search Example

```
[[FEBluetoothSDK sharedFEBluetoothSDK] scan:^(FEPeripheral␣
↪*peripheral, BOOL isNew) {
            if (isNew){
                // peripheral Is a new device
            }else{
                // peripheral Not a new device. It is recommended␣
↪to refresh the device information
            }
        }];
```

2.4 2.4 Stop searching

```
[[FEBluetoothSDK sharedFEBluetoothSDK] stopScan];
```

2.5 2.4 Search filter device

(If you need to filter, set the filter conditions before searching. After setting the filter conditions, scan Peripherals will change, please refresh the list in a timely manner)

2.5.1 2.4.1 Filtering signal range equipment

parame-ter	describe
isFil-terRSSI	Filter by signal value
minRSSI	Filter the minimum signal value and do not call back if this signal is not reached. The setting range is -100 ~ 0，default-100
maxRSSI	Filter the maximum signal value and do not call back if it exceeds this signal. The setting range is -100 ~ 0，default0
Filtering example	

```
// Enable filtering
[FEBluetoothSDK sharedFEBluetoothSDK].filter.isFilterRSSI = YES;
// Equipment with filtration less than - 60
[FEBluetoothSDK sharedFEBluetoothSDK].filter.minRSSI = -60;
// Filter devices greater than 0
[FEBluetoothSDK sharedFEBluetoothSDK].filter.maxRSSI = 0;
```

2.5.2 2.4.2 Filter Device Name

parameter	describe
isFilterName	Filter by device name
filtrationName	Filter device name (only callbacks containing this name)
Filtering example	

```
// Enable filtering
[FEBluetoothSDK sharedFEBluetoothSDK].filter.isFilterName = YES;
// Filtering devices that do not contain Feasycom is case␣
↪insensitive
[FEBluetoothSDK sharedFEBluetoothSDK].filter.filtrationName = @
↪"Feasycom";
```

Chapter 3

3. Connect (FEBluetoothSDK)

3.1 3.1 Introduction:

Connecting Devices

3.2 3.2 Method

```
/// Connecting the device into normal communication mode
/// @param peripheral device
/// @param isConnectedBlock Whether the connection was successful␣
↪or not is called back. Disconnection is also called back from here
- (void)connect:(FEPeripheral *)peripheral connectState:(void(^
↪)(FEPeripheral *peripheral, CONNECTSTATE␣
↪connectState))connectStateBlock;
```

Note:

- In order to ensure effective communication and high-speed transmission, our company has developed the FACP communication protocol itself. It is recommended to enable this protocol. Currently, there are two versions of FACP: FSCFacpType_ 2_ 0 and FSCFacp-Type_ 2_ 1, where FSCFacpType_ 2_ 1. Support data retransmission.
- The module needs to be able to support FSCFacpType for data retransmission_ 2_ 1 function module firmware. If the firmware does not support it, even if the APP chooses to enable FSCFacpType_ 2_ 1. The actual value is FSCFacpType_ 2_ 0.

- To successfully enable the FACP communication protocol, you must configure this attribute (the FACpType attribute of the FEPeripheral class) before connecting to the device, as follows:

```
peripheral.facpTpye = FSCFacpType_2_1;
```

3.3 3.3 Connection example

```
[[FEBluetoothSDK sharedFEBluetoothSDK] connect:peripheral␣
↪isConnected:^(FEPeripheral *peripheral, CONNECTSTATE␣
↪connectState) {
                    // Determine the connection result based on the␣
↪connectState
                }];
```

3.4 3.4 Disconnect

```
[[FEBluetoothSDK sharedFEBluetoothSDK] disconnect:peripheral␣
↪complete:^(FEPeripheral * _Nonnull peripheral) {
        // Disconnection complete
    }];
```

Chapter 4

4. Callback, including received data (FEPeripheral)

4.1 4.1 Introduction

Collecting Multiple Callbacks

4.2 4.2 Method

```
/// Callback
/// @param connectStatusBlock Connection Status
/// @param deviceInfoBlock Read module information
/// @param rssiBlock Signal value
/// @param sendBlock Send a callback (error: Send an error message, sending success is nil. pause: Indicates whether to pause. finish: Indicates whether the sending is complete)
/// @param receiveBlock receive data
- (void)callBackConnectStatus:(void(^ _Nullable)(FEPeripheral *peripheral, CBPeripheralState state))connectStatusBlock
      deviceInfo:(void(^ _Nullable)(FEPeripheral *peripheral, FEDeviceInfo * _Nullable deviceInfo))deviceInfoBlock
            RSSI:(void(^ _Nullable)(FEPeripheral *peripheral, NSNumber* _Nullable RSSI, NSError* _Nullable error))rssiBlock
            send:(void(^ _Nullable)(FEPeripheral *peripheral,
```

(continues on next page)

(continued from previous page)

```
↪NSError* _Nullable error, BOOL pause, BOOL finish))sendBlock
        receive:(void(^ _Nullable)(FEPeripheral *peripheral,␣
↪CBCharacteristic* characteristic, NSError* _Nullable␣
↪error))receiveBlock;
```

4.3 4.3 Callback example

Note: The data currently received is characteristic.value. `peripheral.reData` is spliced data that automatically clears the previous data after reaching a certain amount.


```
[self.peripheral callBackConnectStatus:^(FEPeripheral * _Nonnull␣
↪peripheral, CBPeripheralState state) {
        // Connection status callback
    } deviceInfo:^(FEPeripheral * _Nonnull peripheral, FEDeviceInfo␣
↪* _Nullable deviceInfo) {
        // Device information callback (module support required)
    } RSSI:^(FEPeripheral * _Nonnull peripheral, NSNumber * _
↪Nullable RSSI, NSError * _Nullable error) {
        // Signal value callback
    } send:^(FEPeripheral * _Nonnull peripheral, NSError * _
↪Nullable error, BOOL pause, BOOL finish) {
        // Send completion callback. A pause value of YES indicates␣
↪a pause, and a finish value of YES indicates that all data has␣
↪been sent
    } receive:^(FEPeripheral * _Nonnull peripheral,␣
↪CBCharacteristic * _Nonnull characteristic, NSError * _Nullable␣
↪error) {
        // Received data (currently received data is characteristic.
↪ value)
    }];
```

Chapter 5

5. Send data (FEPeripheral)

5.1 5.1 Introduction

Directly transmit all data without considering subcontracting, and internally automatically subcontract in the most appropriate manner to achieve the highest transmission speed.

5.2 5.2 Method

```
/// SendData
/// @param data Data sent
/// @param isResponse With or without feedback
- (void)send:(NSData * _Nullable)data withResponse:(BOOL)isResponse;
```

5.3 5.3 Send Example

```
[self.peripheral send:data withResponse:NO];
```

5.4 5.4 Stop Send

```
[self.peripheral stopSend];
```

Chapter 6

6. Data verification (FEPeripheral)

6.1 6.1 Introduction

Including quantity, CRC 32, and sending time.

6.2 6.2 Attribute Introduction

attribute	describe
receiveCount	Received Quantity
sendCount	Send Quantity
sendTimeInterval	Send Time
crc32_re	CRC32 receiving data
crc32_se	CRC32 for sending data
other	Auxiliary variables (user can customize and use them at will)

Chapter 7

7. Other

Switch SDK Log: isShowLog。（FEBluetoothSDK）

```
// Open Log
[FEBluetoothSDK sharedFEBluetoothSDK].isShowLog  = YES;
```

Chapter 8

8. Appendix

8.1 Download PDF

- FeasyBlue iOS-SDK Interface Guide (PDF)