FBeacon iOS-SDK Interface Document

Corresponding SDK version	Compilation date	Author
2.0	April 11, 2024	Chen Changhua
2.1	November 21, 2024	Chen Changhua
2.1.2.4	December 17, 2025	Chen Changhua

1 The first category FBBluetooth Browser

It involves understanding the FBBluetooth Browser and FBFilter classes.

1.1 Initialize FBBluetooth Browser object

```
/// 初始化
/// - Parameter type: 设备数组'peripheralItems'中只放置一种类型的设备
- (instancetype)initWithType:(const FBBluetoothBrowserType)type;
```

Explanation 1: Regarding the selection of the type parameter

```
typedef enum __FBBluetoothBrowserType : char {
    FBBluetoothBrowserTypeBeacon, // 标准Beacon广播（iBeacon、UID、URL、TLM、ATLBeacon）
    FBBluetoothBrowserTypeSensor, // 自定义广播（e.g 传感器数据）
    FBBluetoothBrowserTypeSetting // Beacon模块
} FBBluetoothBrowserType;
```

【1】 If FBBluetoothBrowserTypeBeacon is selected, only devices that have broadcasted standard Beacon broadcasts (iBeacon, UID, URL, TLM, ATLBecon) will be added to the peripheral array (peripheralItems), and iBeacon broadcasts will also be added as FBPeripheralItem instance objects to the peripheral array (peripheralItems).
【2】 If FBBluetooth Browser TypeSensor is selected, only devices that have broadcasted custom broadcasts (e.g. sensor data) will be added to the peripheral array (peripheralItems).
【3】 If FBBluetooth Browser TypeSetting is selected, only the Beacon device will be added to the peripheral array (peripheralItems).

1.2 Configure delegate for FBBluetooth Browser object

```
/// 委托对象
@property (nonatomic, weak, nullable) id<FBBluetoothBrowserDelegate>
delegate;
```

Explanation 1: Understanding FBBluetooth Browser Delegate

```
@protocol FBBluetoothBrowserDelegate <NSObject>

@required
/// 已经完成新外设的添加（外设数组'peripheralItems'中已经添加此外设）
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didAddPeripheralItem:(FBPeripheralItem *)peripheralItem;
/// 外设数组'peripheralItems'已更新
- (void)bluetoothBrowserDidUpdatePeripheralItems:(FBBluetoothBrowser
*)bluetoothBrowser;

@optional
/// 中心蓝牙状态发生改变
- (void)bluetoothBrowserDidChangeState:(FBBluetoothBrowser
*)bluetoothBrowser;
/// 是否添加新外设'peripheralItem'
- (BOOL)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
shouldAddPeripheralItem:(FBPeripheralItem *)peripheralItem;
/// 外设数组'peripheralItems'中的某个外设'peripheralItem'的本地名字发生改变
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didUpdatePeripheralItemName:(FBPeripheralItem *)peripheralItem;
/// 外设数组'peripheralItems'中的某个外设'peripheralItem'广播包里的名字发生改变
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didUpdatePeripheralItemAdvertisingName:(FBPeripheralItem *)peripheralItem;
/// 外设数组'peripheralItems'中的某个外设'peripheralItem'信号强度发生改变
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didUpdatePeripheralItemRSSI:(FBPeripheralItem *)peripheralItem;
@end
```

1.3 Scan configuration for FBBluetoothBrowser objects

After initializing the FBBluetooth Browser object, you can obtain the following properties under the object:

```
/// 外设数组'peripheralItems'中只放置一种类型的设备
@property (nonatomic, readonly) FBBluetoothBrowserType type;
/// 当前的过滤器
@property (nonatomic, readonly, strong, nonnull) FBFilter *filter;
/// 中心蓝牙的状态
@property (nonatomic, readonly, assign) CBManagerState state;
/// 外设数组（调用startScanning、startScanningWithServices:方法后，此属性值会清空）
@property (nonatomic, readonly, strong, nonnull) NSArray<FBPeripheralItem *>
*peripheralItems;
/// 已建立蓝牙连接的外设（调用startScanning、startScanningWithServices:方法后，此属
性值会清空）
@property (nonatomic, readonly, strong, nonnull) NSArray<FBPeripheralItem *>
*connectedPeripheralItems;
/// 委托对象
@property (nonatomic, weak, nullable) id<FBBluetoothBrowserDelegate>
delegate;
```

Explanation 1: Type and delegate are assigned values during initialization and configuration of the proxy.
Explanation 2: PeripheralItems and connectedPeripheralItems are currently definitely empty arrays because there have been no scanning or connection operations so far.
Explanation 3: Before planning to perform a scan operation, it is necessary to determine whether scanning can be performed based on the state attribute value. The enumeration values are as follows:

```
typedef NS_ENUM(NSInteger, CBManagerState) {
    CBManagerStateUnknown = 0,
    CBManagerStateResetting,
    CBManagerStateUnsupported,
    CBManagerStateUnauthorized,
    CBManagerStatePoweredOff,
    CBManagerStatePoweredOn,
} NS_ENUM_AVAILABLE(10_13, 10_0);
```

It is recommended to perform the scan operation only for CBManagerState PoweredOn, which can track changes in the state attribute value in a timely manner based on the delegate method of the FBBluetooth Browser object:

```
/// 中心蓝牙状态发生改变
- (void)bluetoothBrowserDidChangeState:(FBBluetoothBrowser
*)bluetoothBrowser;
```

Explanation 4: The class structure definition for the filter attribute is as follows

```objc
@interface FBFilter : NSObject

/// 是否开启「名称过滤」
@property (nonatomic, assign) BOOL filterByNameEnabled;
/// 筛选的「名称」
@property (nonatomic, strong) NSString *filterName;
/// 筛选的「RSSI最弱值」
@property (nonatomic, assign) CGFloat minimumRSSI;

/// 保存当前各属性值到缓存中，初始化FBBluetoothBrowser对象时默认使用上一次保存的FBFilter各属性值
- (void)saveData;

@end
```

After obtaining the filter object, configure the filtering conditions before performing the scan operation. Otherwise, use the previously saved FBFilter attribute values.

1.4 Start scanning peripheral broadcasts

```objc
/// 开始扫描外设
- (BOOL)startScanning;
```

OR

```objc
/// 开始对广播包中包含特定UUID的外设进行扫描
- (BOOL)startScanningWithServices:(nullable NSArray<NSString *> *)serviceUUIDs;
```

Explanation 1: Based on the FBBluetooth BrowserType enumeration value of the initialization configuration, whenever a new peripheral is discovered, the agent is asked if it should be added to the peripheral array peripheralItems.

```objc
/// 是否添加新外设'peripheralItem'
- (BOOL)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser shouldAddPeripheralItem:(FBPeripheralItem *)peripheralItem;
```

Explanation 2: After completing the addition of new peripherals, the agent will be informed

```objc
/// 已经完成新外设的添加（外设数组'peripheralItems'中已经添加此外设）
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser didAddPeripheralItem:(FBPeripheralItem *)peripheralItem;
```

1.5 Update of peripheral information in peripheral array

The Bluetooth module will continuously emit broadcasts, including updates to standard Beacon broadcast data, customized broadcasts such as temperature and humidity data, RSSI value updates, etc., which will be fed back to the delegate object through proxy methods. The App layer can refresh UI information based on such callbacks:

```
/// 外设数组'peripheralItems'中的某个外设'peripheralItem'的本地名字发生改变
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didUpdatePeripheralItemName:(FBPeripheralItem *)peripheralItem;
/// 外设数组'peripheralItems'中的某个外设'peripheralItem'广播包里的名字发生改变
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didUpdatePeripheralItemAdvertisingName:(FBPeripheralItem *)peripheralItem;
/// 外设数组'peripheralItems'中的某个外设'peripheralItem'信号强度发生改变
- (void)bluetoothBrowser:(FBBluetoothBrowser *)bluetoothBrowser
didUpdatePeripheralItemRSSI:(FBPeripheralItem *)peripheralItem;
```

Explanation 1: Although there is no specific callback method to inform the agent of updates to standard Beacon broadcast data, custom broadcasts such as temperature and humidity data, the latest other latest broadcast data can be obtained using the properties in FBPeripheralItem for UI refresh when the signal strength of a certain peripheral 'peripheralItem' in the peripheral array 'peripheralItems' changes.

1.6 Peripheral array 'peripheralItems' sorted by RSSI from strong to weak

By default, objects in the peripheral array 'peripheralItems' are added in the order they are scanned. In practical applications, it is often necessary to sort the peripheral array' peripheralItems' in RSSI from strong to weak, using the following method:

```
/// 将外设数组'peripheralItems'按RSSI从强到弱排序
- (void)sortByRSSI;
```

Explanation 1: After the SDK completes data rearrangement, it will notify the delegate through proxy, and the App layer can refresh the UI at this time:

```
/// 外设数组'peripheralItems'整体发生改变（比如调用sortByRSSI、startScanning、
startScanningWithServices:方法）
- (void)bluetoothBrowserDidUpdatePeripheralItems:(FBBluetoothBrowser
*)bluetoothBrowser;
```

1.7 Stop scanning peripherals

```
/// 停止扫描外设
- (void)stopScanning;
```

2 The second category of FBPeripheralItems

Understanding FBPeripheralItem and FBBeacon classes.

2.1 Device Information Structure

Through the operation steps of the first category, the device FBPeripheralItem object can be obtained. The information structure contained in this object is as follows:

```
/// 搜索到的蓝牙外设对象
@interface FBPeripheralItem : NSObject

#pragma mark - 设备基本信息
/// 是否可建立蓝牙连接
@property (nonatomic, assign, readonly, getter = isConnectable) BOOL connectable;
/// 本地名字
@property (nonatomic, strong, readonly, nullable) NSString *name;
/// 唯一标识
@property (nonatomic, strong, readonly, nullable) NSString *UUID;
/// 信号强度
@property (nonatomic, assign, readonly) int RSSI;
/// 广播包里面的名字
@property (nonatomic, strong, readonly, nullable) NSString *advertisingName;
/// 广播包里面的服务UUID
@property (nonatomic, strong, readonly, nullable) NSArray *serviceUUIDs;
/// 最新广播的时间戳
@property (nonatomic, assign, readonly) NSTimeInterval advertisementTimestamp;
/// 广播间隔时间数组
@property (nonatomic, strong, readonly, nullable) NSMutableArray *advertisementSpaceTimestampArray;
/// 广播频率（可根据广播间隔时间数组‘advertisementSpaceTimestampArray’按照App层算法另行估算）
@property (nonatomic, assign, readonly) NSTimeInterval advertisementRate;
/// 用于显示的名字，优先取advertisingName，若advertisingName长度为0则取name
@property (nonatomic, copy, readonly) NSString *displayedName;

#pragma mark - Beacon设备信息
/// 设备是否为Beacon设备
@property (nonatomic, assign, readonly, getter = isBeaconMoudle) BOOL beaconMoudle;
/// 设备模块类型编号
@property (nonatomic, assign, readonly) int modelIndex;
/// 设备MAC地址
@property (nonatomic, strong, readonly, nullable) NSString *macAddress;
/// 取MAC地址后6位
@property (nonatomic, strong, readonly, nullable) NSString *nameSuffix;
/// 设备固件版本号
@property (nonatomic, strong, readonly, nullable) NSString *firmwareVersion;
/// 设备电量
@property (nonatomic, assign, readonly) int quantityOfElectricity;
/// 设备是否包含PIN码（可连接但需要密码）
@property (nonatomic, assign, readonly) BOOL hasPINCode;
/// 设备是否包含NFC
@property (nonatomic, assign, readonly) BOOL hasNFC;
/// 设备是否包含LongRange125kbps广播
@property (nonatomic, assign, readonly) BOOL hasLongRange;
/// 设备是否包含LED
@property (nonatomic, assign, readonly) BOOL hasLED;
/// 设备是否包含蜂鸣器
```

```objc
@property (nonatomic, assign, readonly) BOOL hasBuzzer;
/// 设备是否包含Gsensor
@property (nonatomic, assign, readonly) BOOL hasGsensor;
/// 设备是否包含按键
@property (nonatomic, assign, readonly) BOOL hasKey;

#pragma mark - 标准Beacon广播
/// 标准Beacon广播（iBeacon、UID、URL、TLM、ATLBeacon）数组
@property (nonatomic, strong, readonly, nullable) NSMutableArray<FBBeacon *> *beacons;
/// 最新的标准Beacon广播（iBeacon、UID、URL、TLM、ATLBeacon）
@property (nonatomic, strong, readonly, nullable) FBBeacon *lastestBeacon;

#pragma mark - 自定义广播（e.g 传感器信息）
/// 包含温湿度广播
@property (nonatomic, assign, readonly, getter = isSensor) BOOL sensor;
/// 温度
@property (nonatomic, strong, readonly) NSString *temperature;
/// 湿度
@property (nonatomic, strong, readonly) NSString *humidity;

#pragma mark - 通讯模块
/// 是否打开，即是否可读写
@property (nonatomic, assign, readonly, getter = isOpened) BOOL opened;
/// 写数据时每包数据的间隔，单位为秒，支持的最小值为0.01
@property (nonatomic, assign) NSTimeInterval writeInterval;
/// 指定发送数据的特征UUID
@property (nonatomic, strong) NSString *writeCharacteristicUUID;
/// 发送数据带不带响应
@property (nonatomic, assign) BOOL writeWithoutResponse;

@end
```

Explanation 1: The information structure is mainly divided into four parts. The validity of data in the information structure is determined by the FBBluetooth Browser Type enumeration value passed in when initializing the FBBluetooth Browser object, and the corresponding relationship is as follows:

```
FBBluetoothBrowserTypeBeacon：
#pragma mark - 设备基本信息 && #pragma mark - Beacon设备信息 && #pragma mark - 标准Beacon广播

FBBluetoothBrowserTypeSensor：
#pragma mark - 设备基本信息 && #pragma mark - Beacon设备信息 && #pragma mark - 自定义广播（e.g 传感器信息）

FBBluetoothBrowserTypeSetting：
#pragma mark - 设备基本信息 && #pragma mark - Beacon设备信息

```

2.2 Standard Beacon Broadcast

At present, iBeacon, UID, URL, TLM, ATLBeacon broadcast data are supported, encapsulated by the FBBeacon class, and the data structure is as follows:

```
typedef enum __FBBeaconType : char {
    FBBeaconTypeUnknown,
    FBBeaconTypeIBeacon,
    FBBeaconTypeURL,
    FBBeaconTypeUID,
    FBBeaconTypeTLM,
    FBBeaconTypeAltBeacon
} FBBeaconType;

typedef enum __FBProximityType: NSUInteger {
    FBProximityTypeUnknown,
    FBProximityTypeImmediate,
    FBProximityTypeNear,
    FBProximityTypeFar
} FBProximityType;

/// 标准Beacon广播（iBeacon、UID、URL、TLM、ATLBeacon）
@interface FBBeacon : NSObject

/// 类型（iBeacon、URL、UID、TLM、AltBeacon）
@property (nonatomic, assign, readonly) FBBeaconType type;
/// 两个 Beacon 的相等判断
- (BOOL)isEqual:(FBBeacon *)otherBeacon;

#pragma mark - iBeacon

/// iBeacon
@property (nonatomic, strong, readonly, nullable) NSString *proximityUUID;
/// iBeacon
@property (nonatomic, assign, readonly) int major;
/// iBeacon
@property (nonatomic, assign, readonly) int minor;
/// iBeacon
@property (nonatomic, assign, readonly) FBProximityType proximityType;
/// iBeacon
@property (nonatomic, assign, readonly) double accuracy;

#pragma mark - UID<0x00>

/// UID<0x00>
@property (nonatomic, assign, readonly) NSInteger calibratedTxPowerAt0m_uid;
/// UID<0x00>
@property (nonatomic, strong, readonly, nullable) NSString *namespaceString;
/// UID<0x00>
@property (nonatomic, strong, readonly, nullable) NSString *instanceString;
/// UID<0x00>
@property (nonatomic, strong, readonly, nullable) NSString *reservedString;

#pragma mark - URL<0x10>

/// URL<0x10>
@property (nonatomic, assign, readonly) NSInteger calibratedTxPowerAt0m_url;
```

```
/// URL<0x10>
@property (nonatomic, strong, readonly, nullable) NSString *URLString;

#pragma mark - TLM<0x20>

/// TLM<0x20>
@property (nonatomic, assign, readonly) NSInteger tlmVersion;
/// TLM<0x20>
@property (nonatomic, assign, readonly) NSInteger batteryVoltage;
/// TLM<0x20>
@property (nonatomic, assign, readonly) NSInteger beaconTemperature;
/// TLM<0x20>
@property (nonatomic, assign, readonly) NSInteger advertisingPDUCount;
/// TLM<0x20>
@property (nonatomic, assign, readonly) NSInteger timeSincePowerOnOrReboot;

#pragma mark - AltBeacon<0xBEAC>

/// AltBeacon<0xBEAC>
@property (nonatomic, strong, readonly, nullable) NSString *manufacturerID;
/// AltBeacon<0xBEAC>
@property (nonatomic, strong, readonly, nullable) NSString *IDString;
/// AltBeacon<0xBEAC>
@property (nonatomic, assign, readonly) NSInteger calibratedTxPowerAt1m_alt;
/// AltBeacon<0xBEAC>
@property (nonatomic, strong, readonly, nullable) NSString
*manufacturerReservedString;

@end
```

Explanation 1: It is necessary to determine the FBBeaconType type and use the corresponding valid data.

3 The third major category FBPeripheralManager

Understanding FBPeripheralManager, FBConfiguration, FBMutableBeacon, and FBSession classes.

3.1 Logic summary

If you need to configure the FBPeripheralItem object, you will need to use the third category. The general operating logic is as follows:

(1) List of configurable parameters
(2) Obtain the current configuration value of the parameter
(3) Obtain the configuration range of parameter values
(4) Set and save new configuration values

3.2 Initialize FBPeripheralManager object

```
/// 初始化
- (instancetype)initWithPeripheralItem:(FBPeripheralItem *)peripheralItem
PINCode:(nullable NSString *)PINCode;
```

Explanation 1: After initialization is completed, the following objects can be obtained through the FBPeripheralManager object:

```
/// 外设
@property (nonatomic, strong, readonly) FBPeripheralItem *peripheralItem;
/// 通信会话层
@property (nonatomic, strong, readonly) FBSession *session;
```

At the same time, it is possible to configure the "Bluetooth disconnection callback" attribute:

```
/// 蓝牙断开的回调
@property (nonatomic, copy) void(^closeHandler)(NSError * _Nullable);
```

3.3 List of configurable parameters and configuration range of parameter values

The following methods and properties of FBPeripheralManager can be used to obtain a list of configurable parameters and a range of parameter values for the current FBPeripheralManager object:

```
/// （外路操作）加载型号、参数取值范围等
- (void)loadConfigurationWithCompletionHandler:(void (^)(FBConfiguration *
_Nullable))completionHandler;
/// 配置参数取值范围
@property (nonatomic, strong, readonly) FBConfiguration *configuration;
/// 可配置参数的名称
@property (nonatomic, strong, readonly) NSArray *paramKeys;
```

Note: In the FBPeripheralManager interface file, it will be found that there are 4 operations that belong to "out of line operations". Before calling such operation methods, it is necessary to determine whether the "operation is busy" based on the following methods. Only when the "handleBusy" property value is NO can it be executed.

```
/// 外路操作是否占线
@property (nonatomic, assign, readonly, getter=isHandleBusy) BOOL handleBusy;
```

3.4 Obtain the current configuration value of the parameter

```objc
/// （外路操作）从设备读取所有参数的当前配置值
- (void)loadParametersWithCompletionHandler:(void (^)(NSError *
_Nullable))completionHandler;
/// 配置参数的值
@property (nonatomic, strong, readonly) NSArray *paramValues;
/// 标准Beacon广播（iBeacon、UID、URL、TLM、ATLBeacon）数组
@property (nonatomic, strong, readonly) NSMutableArray<FBMutableBeacon *>
*beacons;
/// （闭路操作）查询参数
- (id)valueForName:(NSString *)name;
```

3.5 Set and save new configuration values

```objc
/// （闭路操作）更新参数的配置值
- (void)setValue:(id)value forName:(NSString *)name;
/// （外路操作）向设备写入所有参数（包括标准Beacon广播的配置）
- (void)saveParametersWithCompletionHandler:(void (^)(NSError *
_Nullable))completionHandler;
```

Note 1: To modify the Beacon broadcast content, it is necessary to modify the objects in the "Beacons" array properties.

3.6 Reset/Restore Default Factory Settings

```objc
/// （外路操作）重置设备
- (void)restoreWithCompletionHandler:(void (^)(NSError *
_Nullable))completionHandler;
```

4 The fourth category FBUpgradeManager

Understanding the FBUpgradeManager class.

4.1 Initialization

```objc
/// 初始化
- (instancetype)initWithPeripheralItem:(FBPeripheralItem *)peripheralItem
PINCode:(nullable NSString *)PINCode;
```

Explanation 1: After initialization is completed, the communication session layer object can be obtained:

```objc
/// 通讯会话层
@property (nonatomic, strong, readonly) FBSession *session;
```

4.2 Parsing upgrade files

```
/// 解析升级文件信息
/// @param data 文件数据
/// @param complete 完成回调
/// @param faile 失败回调
- (void)infoFromData:(NSData *)data complete:(void(^)(NSString *bootloader, NSString *binCrc, NSInteger length, NSString *versionRange, NSString *_Nullable modelType, NSInteger modelTypeNum, NSString *uploadModel, NSString *crc))complete faile:(void (^)(void))faile;
```

Explanation 1: This method will verify the legality of the upgrade file, and after passing the legality, the App application layer will decide whether to use the file for firmware upgrade.

4.3 Air upgrade

```
/// （外路操作）升级
/// @param path 固件路径
/// @param restore 是否恢复出厂设置
/// @param infoHandler 当前模块的信息（模块型号、固件版本、BT版本）
/// @param progressHandler 升级进度
/// @param completionHandler 升级操作结果
- (void)upgradeWithFilePath:(NSString *)path restore:(BOOL)restore infoHandler:(void (^)(NSDictionary * _Nullable))infoHandler progressHandler:(void (^)(CGFloat))progressHandler completionHandler:(void (^)(NSError *_Nullable))completionHandler;
```

Explanation 1: Aerial upgrades belong to "out of line operations", but considering that after initializing the FBUpgradeManager object, the session object is not shared with the session object in the FBPeripheralManager object, so there is no need to consider whether the operation is busy.

5 The fifth category of FBSessions

This class is an intermediate session layer for apps to read and write data to devices, involving Bluetooth connection and device authentication. After the initialization of the FBPeripheralManager and FBUpgradeManager classes, a corresponding FBSession object will be initialized internally, and the delegate of the FBSession object is held by the FBPeripheralManager and FBUpgradeManager objects. Therefore, during the device configuration and in flight upgrade process, the App application layer does not need to directly manage the FBSession. After the device configuration and in flight upgrade are completed, the SDK layer will close this intermediate session and disconnect the Bluetooth connection. In the App application layer, there is also a need to actively close sessions. Therefore, when configuring devices and upgrading in the air, the App layer should use the following methods to close sessions at appropriate times:

```
/// 关闭会话
- (void)closeSession;
```

Of course, according to the specific usage scenario of the App application layer, if the App layer really needs to hold the delegate of the FBSession object to complete communication operations. There are two situations:

1. If you still want to use the current FBSession object to complete the device configuration and in flight upgrade process, please return the delegate to the FBPeripheralManager and FBUpgradeManager objects at the appropriate time.
2. In non first scenario, the App layer can complete free data communication by initializing the FBSession object.

5.1 Initialization

```
/// 初始化
- (instancetype)initWithPeripheralItem:(FBPeripheralItem *)peripheralItem pinCode:(NSString *)pinCode;
/// 代理
@property (nonatomic, weak) id<FBSessionDelegate> delegate;
```

5.2 Open Session

```
/// 打开会话
- (void)openSession;
```

Explanation 1: Determine the state change of the session layer session based on the proxy callback:

```
@protocol FBSessionDelegate <NSObject>
@optional
/// 通话已打开
- (void)sessionDidOpen:(FBSession *)session;
/// 通话已结束
- (void)sessionDidClose:(FBSession *)session error:(NSError *)error;
/// 通话数据写入完成
- (void)sessionDidWrite:(FBSession *)session;
/// 通话接收数据
- (void)session:(FBSession *)session didReceiveData:(NSData *)data;
@end
```

5.3 Write data to/receive data from the device

Write

```
/// 通过会话写数据
- (BOOL)writeDataInSession:(NSData *)data;
```

Receive

```
/// 通话接收数据
- (void)session:(FBSession *)session didReceiveData:(NSData *)data;
```

5.4 Close session

```
/// 关闭会话
- (void)closeSession;
```

6 Category 6 Suota Upgrade

6.1 The module to be upgraded has completed the initialization of the Parameters object

Assign partial attribute values to the Parameters singleton class of the module to be upgraded, where the Paramters class data structure is as follows:

```
@interface Parameters : NSObject

+ (Parameters*) instance;

@property SuotaManager* suotaManager;
@property CBPeripheral* peripheral;
@property NSString *pinCode;
@property NSString *macAddress;
@property CBCentralManager *centralManager;

@end
```

Example:

```
Parameters.instance.peripheral = _peripheralItem.peripheral;
Parameters.instance.pinCode = _pinCode;
Parameters.instance.macAddress = _peripheralItem.macAddress;
```

6.2 OTA upgrade management class SuotaManager

（1）Initialize the SuotaManager object using some information from the Parameters singleton class, as shown in the following example:

```
strongSelf.suotaManager = [[SuotaManager alloc]
initWithPeripheral:strongSelf.peripheral suotaManagerDelegate:
(DeviceNavigationController*)strongSelf.parentViewController
pinCode:parameters.pinCode macAddress:parameters.macAddress];
```

（2）Then save the SuotaManager object to the Parameters singleton class for global management. The example code is as follows:

```
parameters.suotaManager = strongSelf.suotaManager;
```

(3) Establish a Bluetooth connection with the module to be upgraded using the SuotaManager object

```
if (strongSelf.suotaManager.state == DEVICE_DISCONNECTED) {
    [strongSelf.suotaManager connect];
}
```

(4) Obtain firmware version information of the module to be upgraded

Information is returned through callback methods, including:

```
- (void) onCharacteristicRead:(CBUUID*)uuid value:(NSString*)value {
    if ([uuid isEqual:SuotaProfile.CHARACTERISTIC_MANUFACTURER_NAME_STRING])
{
        containerView.manufacturerNameTextLabel.text = [value
isEqualToString:@"Dialog Semi"] ? @"Dialog Semiconductor" : value;
        SuotaLog(TAG, @"Manufacturer: %@", value);
        return;
    }

    if ([uuid isEqual:SuotaProfile.CHARACTERISTIC_MODEL_NUMBER_STRING]) {
        containerView.modelNumberTextLabel.text = value;
        SuotaLog(TAG, @"Model Number: %@", value);
        return;
    }

    if ([uuid isEqual:SuotaProfile.CHARACTERISTIC_FIRMWARE_REVISION_STRING])
{
        containerView.firmwareRevisionTextLabel.text = value;
        SuotaLog(TAG, @"Firmware Revision: %@", value);
        return;
    }

    if ([uuid isEqual:SuotaProfile.CHARACTERISTIC_SOFTWARE_REVISION_STRING])
{
        containerView.softwareRevisionTextLabel.text = value;
        SuotaLog(TAG, @"Software Revision: %@", value);
        return;
    }

    SuotaLog(TAG, @"Unknown info: %@", value);
}
```

(5) Select OTA firmware file

There are two steps that need to be executed in this stage. The first step is to convert the OTA firmware file into a SuotaFile object representation through parsing and other methods. By default, the valid filtered OTA firmware files in the Documents directory of the application are converted into a SuotaFile object array for presentation, as shown below:

```
@property NSArray<SuotaFile*>* fileArray;
self.fileArray = [SuotaFile listFilesWithHeaderInfo];
```

The second step is to assign the selected SuoFile object to the suotaFile property of the SuotaManager object. The relevant code example is as follows:

```
self.suotaManager.suotaFile = [[SuotaFile alloc] initWithURL:url];
```

(6) OTA upgrade Gpio pin configuration

The default pin configuration during the Dialogue OTA upgrade process should be as follows:

```
Memory type: SPI
MISO GPIO: P0_3
MOSI GPIO: P0_0
CS GPIO: P0_1
SCK GPIO:   P0_4
Image bank: 2
Block size: 240
```

The corresponding related codes are as follows:

```
[self.suotaManager initializeSuota:blockSize misoGpio:spiMISO
mosiGpio:spiMOSI csGpio:spiCS sckGpio:spiSCK imageBank:imageBank];
```

(7) Start OTA upgrade

```
[self.suotaManager startUpdate];
```

The various callbacks related to the upgrade process refer to the SuotaManagerial Delegate protocol method.

6.3 与升级过程相关的各种回调是指SuotaManagementDelegate协议方法。

```
@protocol SuotaManagerDelegate <NSObject>

@required

/*!
 * @method onConnectionStateChange:
 *
 * @param newStatus New connection status. The status code can be found in
SuotaManagerial Status.
 *
```

```
 * @discussion This method is triggered every time the connection status changes.
 */
- (void) onConnectionStateChange:(enum SuotaManagerStatus)newStatus;

@required

/*!
 * @method onServicesDiscovered
 *
 * @discussion Trigger this method upon service discovery.
 */
- (void) onServicesDiscovered;

@required

/*!
 * @method onCharacteristicRead:characteristic:
 *
 * @param characteristicGroup The current feature group. The feature group can be found in the CharacteristicGroup.
 * @param characteristic The features read.
 *
 * @discussion Trigger this method when reading features.
 */
- (void) onCharacteristicRead:(enum CharacteristicGroup)characteristicGroup characteristic:(CBCharacteristic*)characteristic;

@required

/*!
 * @method onDeviceInfoReadCompleted:
 *
 * @param status DeviceInfoReadStatusStatus. Indicates whether device information has been successfully read: Success indicates success, and NOVNet INFO indicates no readable device information.
 *
 * @discussion This method is triggered when all device information has been read.
 */
- (void) onDeviceInfoReadCompleted:(enum DeviceInfoReadStatus)status;

@required

/*!
 * @method onDeviceReady
 *
 * @discussion This method is triggered when all available SUOTA information has been read. If AUTOVNet INFO is set to<code>true</code>, it means that the device information has also been read.
 */
- (void) onDeviceReady;

@required

/*!
 * @method onSuotaLog:type:log:
 *
```

```
 * @param state current SuotaProtocolState。
 * @param type Log type.
 * @param log Associated status update message.
 *
 * @discussion If NOTIFY_SUOTA_STATUS为<code>true</code>，This method will be triggered.
 */
- (void) onSuotaLog:(enum SuotaProtocolState)state type:(enum SuotaLogType)type log:(NSString*)log;

@required

/*!
 * @method onChunkSend:totalChunks:chunk:block:blockChunks:totalBlocks:
 *
 * @param chunkCount The number of data blocks in the current block.
 * @param totalChunks The total number of data blocks.
 * @param chunk The number of data blocks in the current block.
 * @param block The current data block.
 * @param blockChunks The current data block.
 * @param totalBlocks The total number of data blocks.
 *
 * @discussion If NOTIFY_CHUNK_SEND为<code>true</code>，This method is triggered when sending data blocks.
 */
- (void) onChunkSend:(int)chunkCount totalChunks:(int)totalChunks chunk:(int)chunk block:(int)block blockChunks:(int)blockChunks totalBlocks:(int)totalBlocks;

@required

/*!
 * @method onBlockSent:
 *
 * @param block The current number of data blocks.
 * @param totalBlocks The total number of data blocks.
 *
 * @discussion Trigger this method after successfully transmitting the data block.
 */
- (void) onBlockSent:(int)block totalBlocks:(int)totalBlocks;

@required

/*!
 * @method updateSpeedStatistics:max:min:avg:
 *
 * @param current The transmission speed of the current block (Bps).
 * @param max The transmission speed of the current block (Bps).
 * @param min Minimum transmission speed (Bps).
 * @param avg Average transmission speed (Bps).
 *
 * @discussion If CALCULATE_STATISTICS为<code>true</code>，Then trigger this method every 500ms.
 */
- (void) updateSpeedStatistics:(double)current max:(double)max min:(double)min avg:(double)avg;

```

```
@required

/*!
 * @method updateCurrentSpeed:
 *
 * @param currentSpeed The number of bytes sent per second.
 *
 * @discussion If CALCULATE_STATISTICS为<code>true</code>，Then trigger this method every 1000ms. This represents the current number of bytes sent per second, not the average.
 */
- (void) updateCurrentSpeed:(double)currentSpeed;

@required

/*!
 * @method onUploadProgress:
 *
 * @param percent The percentage of firmware files that have been sent to the device.
 *
 * @discussion If NOTIFY_UPLOAD_PROGRESS为<code>true</code>，This method is triggered when uploading progress updates.
 */
- (void) onUploadProgress:(float)percent;

@required

/*!
 * @method onSuccess:imageUploadElapsedSeconds:
 *
 * @param totalElapsedSeconds This method is triggered when uploading progress updates.
 * @param imageUploadElapsedSeconds The time spent during image upload.
 *
 * @discussion Trigger this method after successful completion of the SUOTA process. If the time consumption cannot be calculated, the time parameter value is<code>-1</code>。
 */
- (void) onSuccess:(double)totalElapsedSeconds imageUploadElapsedSeconds:(double)imageUploadElapsedSeconds;

@required

/*!
 * @method onFailure:
 *
 * @param errorCode The reason for the failure. The error code can be found in SuotaErrors and applicationErrors.
 *
 * @discussion Trigger this method when an unexpected event occurs.
 */
- (void) onFailure:(int)errorCode;

@required

/*!
 * @method onRebootSent
```

```
 *
 * @discussion This method is triggered when a restart signal is sent to
the device.
 */
- (void) onRebootSent;

@end
```