- 📌 Version History
- Enum: `QCVolumeMode`
- Enum: `QCOperatorDeviceMode`
- Enum: `QGAISpeakMode`
- Class: `QCVolumeInfoModel`
- Development Guide
  - 1 Environment configuration
  - 1.1 Add framework
  - 1.2 Configure Bluetooth permissions
  - 1.3 Configure Wi-Fi permissions
  - 2 Usage
    - 2.1 Service UUIDs supported by the device
    - 2.2 import framework library
    - 2.3 Scan Glasses
    - 2.4 Cancel the scan of the Glasses
    - 2.5 Connect the Glasses
    - 2.6 Disconnect
- Class: `QCSDKCmdCreator`
  - Device Mode and Wi-Fi
  - Video and Audio Configuration
  - Media Management
  - Battery and Version Info
  - DFU Firmware Upgrade Methods
  - OTA and Other Features

QCSDKManager

- Overview
- QCSDKManagerDelegate Protocol
  - Battery Updates
  - Media Updates
  - Wi-Fi Firmware Upgrade
  - AI Chat Responses
- QCSDKManager Class
  - Properties
  - Singleton Instance
  - Peripheral Management
  - AI Functions
  - Wi-Fi Functions
- ❗ Error Codes (`QCErrorCode`)

📌 Version History

Version	Release Date	Change Log
v1.0.0	2025-07-23	Initial release.
v1.0.0	2025-08-12	Add download media resources

Enum: `QCVolumeMode`

Defines volume control modes used across the SDK.

Value	Description
`QCVolumeModeMusic`	Music volume mode
`QCVolumeModeCall`	Call volume mode
`QCVolumeModeSystem`	System volume mode

Enum: `QCOperatorDeviceMode`

Device operating modes representing the current functional state of the device.

Enum Value	Description
`QCOperatorDeviceModeUnkown` (0x00)	Unknown mode
`QCOperatorDeviceModePhoto` (0x01)	Photo mode (taking pictures)
`QCOperatorDeviceModeVideo`	Video recording mode
`QCOperatorDeviceModeVideoStop`	Stop video recording
`QCOperatorDeviceModeTransfer`	Data transfer mode
`QCOperatorDeviceModeOTA`	OTA (firmware update) mode
`QCOperatorDeviceModeAIPhoto`	AI-powered photo mode
`QCOperatorDeviceModeSpeechRecognition`	Speech recognition mode
`QCOperatorDeviceModeAuido`	Audio recording mode
`QCOperatorDeviceModeTransferStop`	Stop data transfer (Bluetooth off)
`QCOperatorDeviceModeFactoryReset`	Factory reset mode
`QCOperatorDeviceModeSpeechRecognitionStop`	Stop speech recognition
`QCOperatorDeviceModeAudioStop`	Stop audio recording
`QCOperatorDeviceModeFindDevice`	Find device mode
`QCOperatorDeviceModeRestart`	Restart device
`QCOperatorDeviceModeNoPowerP2P`	Restart P2P without power off
`QCOperatorDeviceModeSpeakStart`	Voice playback start
`QCOperatorDeviceModeSpeakStop`	Voice playback stop
`QCOperatorDeviceModeTranslateStart`	Translation start
`QCOperatorDeviceModeTranslateStop`	Translation stop

Enum: `QGAISpeakMode`

AI speaking modes indicating the speaking state of the device.

Enum Value	Description
`QGAISpeakModeStart` (0x01)	Start speaking
`QGAISpeakModeHold`	Pause speaking (hold)
`QGAISpeakModeStop`	Stop speaking
`QGAISpeakModeThinkingStart`	Start thinking (processing)
`QGAISpeakModeThinkingHold`	Hold thinking (processing)
`QGAISpeakModeThinkingStop`	Stop thinking (processing)
`QGAISpeakModeNoNet` (0xf1)	No network available

Class: `QCVolumeInfoModel`

Represents volume configuration for music, call, and system modes.

Property	Type	Description
`musicMin`	NSInteger	Minimum volume for music
`musicMax`	NSInteger	Maximum volume for music
`musicCurrent`	NSInteger	Current volume for music
`callMin`	NSInteger	Minimum volume for calls
`callMax`	NSInteger	Maximum volume for calls
`callCurrent`	NSInteger	Current volume for calls
`systemMin`	NSInteger	Minimum volume for system
`systemMax`	NSInteger	Maximum volume for system
`systemCurrent`	NSInteger	Current volume for system
`mode`	`QCVolumeMode`	Active volume mode (music/call/system)

Development Guide

1 Environment configuration

1.1 Add framework

Add `QCSDk.framework` to the project, the framework supports iOS 13.0 and above

Note: Because the classification is used in the framework, you need to add settings to the project

Target->Build Settings -> Other Linker Flags add **-ObjC**

1.2 Configure Bluetooth permissions

Configure bluetooth permissions in info.plist file

```
<key>NSBluetoothAlwaysUsageDescription</key>
<string>App needs to use your bluetooth device</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>App needs to use your bluetooth device</string>
```

1.3 Configure Wi-Fi permissions

1. Configure local network permissions `NSLocalNetworkUsageDescription` in info.plist file

   ```
   <key>NSLocalNetworkUsageDescription</key>
   <string>App needs to use your local network.</string>
   ```

2. Configure local network permissions `App Transport Security Settings` in info.plist file
3. Configure local network permissions `Bonjour services` in info.plist file
4. Add `Hotspot` in `Signing & Capabilities`

2 Usage

2.1 Service UUIDs supported by the device

Defined in `QCSDKManager.h`, the service UUID supported by the device:

```
extern NSString *const QCSDKSERVERUUID1;
extern NSString *const QCSDKSERVERUUID2;
```

2.2 import framework library

Introduce the framework library into the code

```
#import <QCSDK/QCSDKManager.h>
#import <QCSDK/QCSDKCmdCreator.h>
```

Initialize `[QCSDKManager shareInstance]` with a singleton

> `QCSDKManager`:Peripherals for joining connections
>
> `QCSDKCmdCreator`:Used to send commands to peripherals

2.3 Scan Glasses

initialization

Scanning can only be started when permissions are allowed and Bluetooth is turned on.

Import Apple's CoreBluetooth library and follow two protocols `<CBCentralManagerDelegate, CBPeripheralDelegate>`

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

Declare central and peripheral roles

```
/*Central Role,app*/
@property (strong, nonatomic) CBCentralManager *centerManager;

/*Peripheral role, scanned peripherals*/
@property (strong, nonatomic) NSMutableArray<CBPeripheral *> *peripherals;

/*Connected peripheral role*/
@property (strong, nonatomic) CBPeripheral *connectedPeripheral;
```

Instantiate the central role

```
self.centerManager = [[CBCentralManager alloc] initWithDelegate:self queue:nil];
```

Scan Glasses

Using Scan Peripherals

```
NSArray *serviceUUIDStrings = @[QCBANDSDKSERVERUUID1,QCBANDSDKSERVERUUID2];

NSMutableArray *uuids = [NSMutableArray array];
for (id obj in serviceUUIDStrings) {
    if ([obj isKindOfClass:[NSString class]]) {
        CBUUID *uuid = [CBUUID UUIDWithString:obj];
        [uuids addObject:uuid];
    }
}

NSDictionary *option = @{CBCentralManagerScanOptionAllowDuplicatesKey : [NSNumber
numberWithBool:NO]};
[self.centerManager scanForPeripheralsWithServices:uuids options:option];
```

Note: To obtain the scanned peripheral devices in the agent, you can perform secondary filtering through the device name and other related information.

```
- (void)centralManager:(CBCentralManager *)central didDiscoverPeripheral:(CBPeripheral
*)peripheral advertisementData:(NSDictionary<NSString *,id> *)advertisementData RSSI:
(NSNumber *)RSSI {
    if (peripheral.name.length > 0) {
        [self.peripherals addObject:peripheral];
        [self.deviceList reloadData];
    }
}
```

2.4 Cancel the scan of the Glasses

Call the interface of the central role to stop scanning

```
[self.centerManager stopScan];
```

2.5 Connect the Glasses

start connecting

```
self.connectedPeripheral = self.peripherals[indexPath.row];
[self.centerManager connectPeripheral:self.connectedPeripheral options:nil];
```

After the connection is successful, pass in the peripheral device to the SDK

```
- (void)centralManager:(CBCentralManager *)central didConnectPeripheral:(CBPeripheral
*)peripheral {
    [[QCSDKManager shareInstance] addPeripheral:peripheral];
}
```

2.6 Disconnect

```
[self.centerManager cancelPeripheralConnection:self.connectedPeripheral];
```

After disconnecting, remove peripherals

```
- (void)centralManager:(CBCentralManager *)central didDisconnectPeripheral:(CBPeripheral
*)peripheral error:(nullable NSError *)error {
    [[QCSDKManager shareInstance] removePeripheral:peripheral];
}
```

Class: `QCSDKCmdCreator`

`QCSDKCmdCreator` provides static methods to control the device, manage media, perform firmware updates, and configure various device features.

Device Mode and Wi-Fi

Method	Description
`setDeviceMode:success:fail:`	Set the device's current operation mode.
`openWifiWithMode:success:fail:`	Open device Wi-Fi with specified mode, returns SSID and password.

Video and Audio Configuration

Method	Description
`setVideoInfo:duration:success:fail:`	Set video recording parameters such as angle and duration.
`getVideoInfoSuccess:fail:`	Retrieve current video recording configuration.
`setAudioInfo:duration:success:fail:`	Set audio recording parameters such as angle and duration.
`getAudioInfoSuccess:fail:`	Retrieve current audio recording configuration.

Media Management

Method	Description
`getDeviceMedia:fail:`	Retrieve counts and sizes of media files on the device.
`deleleteAllMediasSuccess:fail:`	Delete all media files on the device.
`deleleteMedia:success:fail:`	Delete a specified media file by name.
`getThumbnail:success:fail:`	Retrieve thumbnail image data from a specified media pocket.

Battery and Version Info

Method	Description
`getDeviceBattery:fail:`	Get battery percentage and charging status.
`getDeviceVersionInfoSuccess:fail:`	Get hardware and firmware version details.

DFU Firmware Upgrade Methods

Method	Description
`switchToDFU:`	Switch device to DFU (Device Firmware Update) mode.
`initDFUFirmwareType:binFileSize:checkSum:crc16:finished:`	Initialize DFU process with firmware metadata.
`sendFilePacketData:serialNumber:finished:`	Send a single packet of firmware data with serial number.
`checkMyFirmwareWithData:finished:`	Verify sent firmware data.
`finishDFU:`	Complete the DFU upgrade process.
`checkCurrentStatusWithData:finished:`	Get current DFU process status from device.
`getDFUBandTypeInfoSuccess:fail:`	Retrieve DFU band type and status (single/dual band).
`switchToOneBandDFU:`	Switch to single-band DFU mode.

OTA and Other Features

Method	Description
`sendOTAFileLink:finished:`	Send OTA firmware download URL to device.
`setupDeviceDateTime:`	Synchronize device date and time.
`sendVoiceHeartbeatWithFinished:`	Send heartbeat signal to maintain voice feature connection.
`getVoiceWakeupWithFinished:`	Get voice wakeup status.
`setVoiceWakeup:finished:`	Enable or disable voice wakeup feature.
`getWearingDetectionWithFinished:`	Get wearing detection status.
`setWearingDetection:finished:`	Enable or disable wearing detection.
`getDeviceConfigWithFinished:`	Retrieve device configuration.
`setAISpeekModel:finished:`	Set AI speaking mode.
`getVolumeWithFinished:`	Get current volume settings.
`setVolume:finished:`	Set volume settings using `QCVolumeInfoModel`.
`setBTStatus:finished:`	Enable or disable Bluetooth.
`getBTStatusWithFinished:`	Get Bluetooth status.

QCSDKManager

Overview

`QCSDKManager` is the main interface for managing connections with peripheral devices, handling media resources, and integrating AI chat functionality.
It provides APIs to add/remove peripherals, download resources via Wi-Fi, and receive AI-generated content such as text, images, and voice.

QCSDKManagerDelegate Protocol

The `QCSDKManagerDelegate` protocol defines optional callback methods for device status updates, media synchronization, firmware upgrade progress, and AI chat responses.

Battery Updates

```
- (void)didUpdateBatteryLevel:(NSInteger)battery charging:(BOOL)charging;
```

- **battery**: Current battery percentage (0–100).
- **charging**: `YES` if charging, otherwise `NO`.

Media Updates

```
- (void)didUpdateMediaWithPhotoCount:(NSInteger)photo
                          videoCount:(NSInteger)video
                          audioCount:(NSInteger)audio
                                type:(NSInteger)type;
```

- **photo**: Number of photo files.
- **video**: Number of video files.
- **audio**: Number of audio files.
- **type**: Media update type identifier.

Wi-Fi Firmware Upgrade

```
- (void)didUpdateWiFiUpgradeProgressWithDownload:(NSInteger)download
                                        upgrade1:(NSInteger)upgrade1
                                        upgrade2:(NSInteger)upgrade2;
```

Reports upgrade progress.

```
- (void)didReceiveWiFiUpgradeResult:(BOOL)success;
```

Reports whether the firmware upgrade succeeded.

AI Chat Responses

- **Image Response**

```
- (void)didReceiveAIChatImageData:(NSData *)imageData;
```

Raw image data (PNG, JPEG).

- **Voice Response**

```
- (void)didReceiveAIChatVoiceData:(NSData *)pcmData;
```

PCM audio (16kHz, 16-bit, mono).

- **Text Response**(developering)

```
- (void)didReceiveAIChatTextMessage:(NSString *)message;
```

Recognized or generated AI chat text.

QCSDKManager Class

The main SDK manager class for handling device connections and AI/Wi-Fi operations.

Properties

```
@property(nonatomic, assign) BOOL debug;
@property(nonatomic, weak) id<QCSDKManagerDelegate> delegate;
```

- **debug**: Enable or disable debug logging.
- **delegate**: The object that receives SDK event callbacks.

Singleton Instance

```
+ (instancetype)shareInstance;
```

Returns the shared singleton instance of `QCSDKManager`.

Peripheral Management

```
- (void)addPeripheral:(CBPeripheral *)peripheral finished:(void (^)(BOOL))finished;
```

Adds a peripheral device.

```
- (void)removePeripheral:(CBPeripheral *)peripheral;
```

Removes a specific peripheral.

```
- (void)removeAllPeripheral;
```

Removes all peripherals.

AI Functions

```
- (void)stopAIChat;
```

Stops AI chat processing.

Wi-Fi Functions

```
- (void)startToDownloadMediaResourceWithProgress:(void (^)(NSInteger receivedSize,
                                                           NSInteger expectedSize,
                                                           CGFloat progress))progress
                                      completion:(void (^)(NSString * _Nullable filePath,
                                                           NSError * _Nullable error,
                                                           NSInteger totalFiles,
                                                           NSInteger
successCount))completion;
```

Downloads media resources over Wi-Fi.

- **progress**: Reports download progress.
- **completion**: Reports final result, including file path, error info, total count, and success count.

❗ Error Codes (`QCErrorCode`)

The `QCErrorCode` enumeration defines common error codes that can occur during device communication and resource operations.

Error Code	Value	Description
`QCErrorCodeInvalidWifiOrPassword`	2000	WiFi SSID or password is empty or invalid.
`QCErrorCodeFailedToGetGlassesIP`	2001	Failed to obtain the glasses' IP address.
`QCErrorCodeFailedToGetAppIP`	2002	Failed to obtain the app's IP address.
`QCErrorCodeLocalNetworkNotAuthorized`	2003	Local network access is not authorized by the user.
`QCErrorCodeDownloadConfigFileFailed`	2004	Failed to download the configuration file.
`QCErrorCodeDownloadFileFailed`	2005	Failed to download the media file.
`QCErrorCodeFileListEmpty`	2006	The file list retrieved from the device is empty.