GizWifi SDK iOS 参考手册

修订记录

修改时间	修改内容	版本	修改人	备注
2016.5.10	更新目录	1.0.0	Pomia	
2016.9.7	增加定时任务接口	1.0.1	Pomia	
2016.9.27	增加新接口说明	1.0.2	Pomia	
2016.10.19	启动接口中增加域名、pk 过滤参数 设备配置模组类型增加一个自定义枚举值 旧的启动接口仍然兼容，但不推荐使用 旧的切换域名接口仍然兼容，但不推荐使用 定时任务接口已废弃，不推荐使用	1.0.3	Pomia	
2016.11.7	启动接口参数使用变更 增加设备全球域名部署接口	1.0.4	Pomia	
2016.11.30	startWithAppID 接口增加开启设备域名自动设置参数 setDeviceServerInfo 接口 mac 参数使用变更	1.0.5	Pomia	
2017.1.25	增加新的设备定时任务接口 增加设备分享接口 增加一些枚举定义	1.0.6	Pomia	
2017.4.13	增加新的添加子设备接口	1.0.7	Pomia	
2017.8.14	增加用户反馈接口	1.0.8	Pomia	
2017.8.15	修改启动接口功能描述	1.0.9	Pomia	
2017.9.30	增加新的启动接口 新增设备 OTA 接口、中控分组、场景接口 补充错误码、枚举定义	1.1.0	Pomia	
2017.10.13	补充定时任务 attrs 说明	1.1.1	Pomia	
2017.12.22	修改启动接口说明 新增配网接口 新增停止配网接口 设备类增加 rootDevice 变量 修改错误码说明	1.1.2	Pomia	
2018.4.16	修改启动接口说明 修改配网接口说明 远端设备绑定接口增加参数 新增设备 QRCode 绑定接口 新增分享权限转移接口 设备类增加 attrStatus 状态缓存变量	1.1.3	Pomia	

	错误码增加 8319 模组类型增加芯海模组			
2018.7.2	设备类增加数据点内容变量 设备类增加固件版本号变量 增加设备网络类型枚举 邮箱注册增加邮箱激活支持	1.1.4	Pomia	
2018.8.21	新增中控联动接口 新增设备的安全注册接口 新增轻网关子设备接口 新增相关错误码和枚举定义	1.1.5	Pomia	
2019.4.19	新增推送绑定解绑接口 模组类型增加汉枫 V8 模组模式增加多设备 AirLink 配置模式 更新定时任务 attrs 说明 新增相关错误码	1.1.6	翁丹丽	
2019.6.21	新增两个订阅接口	1.1.7	翁丹丽	
2019.9.3	新增 mesh 组网相关接口	1.1.8	翁丹丽	
2019.11.1	新增蓝牙配网枚举 新增注销账户接口	1.1.9	翁丹丽	
2021.1.18	新增双通道设备对象 新增蓝牙设备安全注册接口 新增双通道设备发现回调	1.2.0	翁丹丽	
2022.2.10	增加蓝牙配网示例 增加加密注册相关接口	1.2.1	翁丹丽	
2022.2.25	增加 OTA、蓝牙本地控制文档	1.2.2	翁丹丽	
2022.5.5	增加蓝牙设备配网状态字段，配网前缀允许传数组，用来定向配网蓝牙设备	1.2.3	翁丹丽	

1. GizWifiSDK 类

1.1. 简介

机智云 Wifi SDK 的基础类。该类提供了初始化、基本设置、用户管理、设备管理的基本接口。继承 NSObject。遵循 Cocoa 的规则，方法前面是加号（+）的，则为静态接口，方法前面是减号（–）的，则为实例接口。

1.2. 属性变量

属性	描述
delegate	使用委托获取对应事件。GizWifiSDK 对应的回调接口在 GizWifiSDKDelegate 定义。需要用到哪个接口，回调即可。
deviceList	NSArray 类型，为 GizWifiDevice 对象数组。设备列表缓存，APP 访问该变量即可得到当前 GizWifiSDK 发现的设备列表。

1.3. 回调接口

以下是 GizWifiSDK 提供的所有回调接口，将在在后续 API 定义中详细介绍:

- didNotifyEvent：SDK 系统事件通知
- didGetCurrentCloudService：服务域名独立部署的回调接口
- didSetDeviceOnboarding：设备配置结果的回调接口
- didDiscovered：设备列表上报的回调接口
- didBindDevice：设备绑定结果的回调接口
- didUnbindDevice：设备解除绑定结果的回调接口
- didRegisterUser：用户注册结果的回调接口
- didUserLogin：用户登录结果的回调接口
- didTransAnonymousUser：匿名用户转换的回调接口
- didChangeUserPassword：更换用户密码结果的回调接口
- didGetUserInfo：获取用户信息的回调接口

- didChangeUserInfo：修改用户信息结果的回调接口
- didGetCaptchaCode：获取图片验证码的回调接口
- didRequestSendPhoneSMSCode：请求手机短信验证码的回调接口
- didVerifyPhoneSMSCode：验证手机短信验证码的回调接口
- didDisableLAN：小循环是否禁用的回调接口
- didGetSSIDList：获取设备周围 Wi-Fi 热点列表的回调接口
- didChannelIDBind：绑定推送 ID 的回调接口
- didChannelIDUnBind：解绑推送 ID 的回调接口
- didDiscoveredMeshDevices:获取蓝牙 Mesh 设备的回调接口
- didChangeDeviceMesh：切换蓝牙 Mesh 设备组网的回调
- didRestoreDeviceFactorySetting：恢复蓝牙 Mesh 设备出厂设置的回调
- didAddDevicesToGroup：添加蓝牙 Mesh 设备到分组的回调
- didDeleteDevicesToGroup：将蓝牙 mesh 设备从指定分组删除的回调
- didGetDeviceGroups：获取设备所在分组集合的回调
- didRegisterBleDevice：注册蓝牙设备回调接口
- didDiscoverBleDevice：本地双通道设备发现回调

1.4. API 定义

【sharedInstance】

定义	+ (instancetype)sharedInstance;
功能描述	获取 GizWifiSDK 单例的实例。
返回值	返回初始化后 SDK 唯一的实例。SDK 未初始化，或者初始化失败，返回 nil。
代码示例	GizWifiSDK mGizWifiSDKInstance = [GizWifiSDK sharedInstance];

【startWithAppInfo】

定义	+ (void)startWithAppInfo:(NSDictionary*)appInfo productInfo:(NSArray *)productInfo cloudServiceInfo:(NSDictionary*)cloudSeviceInfo autoSetDeviceDomain:(BOOL)autoSetDeviceDomain;
功能描述	启动 SDK。该接口执行成功后，其他接口功能才能正常执行。该接口执行结果由回调 didNotifyEvent 通知 App，回调参数 eventID 为 8316 表示 SDK 启动成功。 如果 App 要做域名切换和设备类型过滤，则要在此时就指定好域名 cloudSeviceInfo 和产品信息 productInfo。 SDK 启动成功后，如果 App 已经设置了 delegate，SDK 会立即通过 didDiscovered 上报发现的设备。

参数	appInfo	应用信息，格式：{"appId": "xxx", "appSecret": "xxx"}。 此参数不能填 nil，appId 和 appSecret 必须为有效值。 在机智云开发者中心 dev.gizwits.com 中，每个注册的设备在对应的“应用配置”中，都能够查到对应的 appId 和 appSecret
	productInfo	此参数为选填。 产品信息数组，格式：[{"productKey": "xxx", "productSecret": "xxx", "usingAdapter": "xxx"}]，其中 usingAdapter 选填，可选项见 GizAdapterType。 当 usingAdapter 为 GizAdapterWifiBle 时，用来标识模组支持蓝牙近场控制，SDK 会自动发现这个产品。 如果填写了此参数，需保证 productKey 和 productSecret 都为有效值，否则会被忽略。SDK 会根据此参数过滤设备列表
	cloudServiceInfo	服务器域名信息。 如果使用机智云全球部署域名，此参数填 nil，此时 SDK 将自动匹配对应的域名服务。 独立部署时此参数必须指定域名信息，形如：xxx.gizwits.com 如果要使用特殊端口号，可指定 Http 端口：xxx.gizwits.com:81，或同时指定 Http 和 Https 端口：xxx.gizwits.com:81&8443。 参数为字典格式： { "openAPIInfo": "xxx", // NSString类型，api服务域名，必填 "siteInfo": "xxx" // NSString类型，site服务域名，可不填 "pushInfo": "xxx" // NSString类型，推送服务域名，可不填 }
	autoSetDeviceDomain	此参数仅用于全球部署，而设备全球部署已由 SDK 在设备配网时自动完成，因此该参数不再生效。
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didNotifyEvent:(GizEventType)eventType eventSource:(id)eventSource eventID:(GizWifiErrorCode)eventID eventMessage:(NSString *)eventMessage;	
回调说明	当发生 GizEventType 中列举的事件类型时，SDK 会主动触发该回调，该回调通知的主要是发生的异常事件	
回调参数	wifiSDK	回调的 GizWifiSDK 单例

	eventType	事件类型。指明发生了哪一类的事件，详细见 GizEventType 枚举定义
	eventSource	事件源，指是谁触发的事件。如果 eventType 是 GizEventSDK，eventSource 为 nil；如果是 GizEventDevice，eventSource 需要强制转换为 GizWifiDevice 类型再使用；如果是 GizEventM2Mservice 或者 GizEventToken，eventSource 需要强制转换为 NSString 类型再使用
	eventID	事件 ID。代表事件编号，详细见 GizWifiErrorCode 枚举定义。该参数指出 eventSource 发生了什么事
	eventMessage	事件 ID 的消息描述

代码示例

```
// 设置 SDK 委托
[GizWifiSDK sharedInstance].delegate = self;
// 设置 AppInfo
NSDictionary* appInfo = @{@"appId": @"your_app_id", @"appSecret":
@"your_app_secret"};

// 设置要过滤的设备 productKey 列表。不过滤则直接传 nil
NSArray *productInfo = [NSArray arrayWithObjects: @{@"productKey":
@"your_product_key", @"productSecret": @"your_product_secret"}, nil];
// 指定要切换的域名信息。使用机智云生产环境则传 nil
// NSDictionary* cloudServiceInfo = @{@"openAPIInfo":
@"your_api_domain"};
// 调用 SDK 的启动接口
[GizWifiSDK startWithAppInfo:appInfo productInfo:productInfo
cloudServiceInfo:nil autoSetDeviceDomain:NO];

// 实现系统事件通知回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK
didNotifyEvent:(GizEventType)eventType eventSource:(id)eventSource
eventID:(GizWifiErrorCode)eventID eventMessage:(NSString *)eventMessage
{
    if(eventType == GizEventSDK) {
        // SDK的通知
        NSLog(@"SDK event happened: [%@] = %@", @(eventID), eventMessage);
        if(result.code == GIZ_SDK_START_SUCCESS) {
          // SDK启动成功
        }
    } else if(eventType == GizEventDevice) {
        // 设备连接断开时可能产生的通知
        GizWifiDevice* mDevice = (GizWifiDevice*)eventSource;
```

```
        NSLog(@"device mac %@ disconnect caused by %@",
        mDevice.macAddress, eventMessage);
    } else if(eventType == GizEventM2MService) {
        // M2M服务返回的异常通知
        NSLog(@"M2M domain %@ exception happened: [%@] = %@",
        (NSString*)eventSource, @(eventID), eventMessage);
    } else if(eventType == GizEventToken) {
        // token失效通知
        NSLog(@"token %@ expired: %@", (NSString*)eventSource,
        eventMessage);
    }
}
```

【getVersion】

定义	+ (NSString *)getVersion;
功能描述	获取 SDK 版本号。
返回值	返回当前 SDK 的版本号码
代码示例	[[GizWifiSDK sharedInstance] getVersion];

【userFeedback】

定义	+ (void)userFeedback:(NSString*)contactInfo feedbackInfo:(NSString*)feedbackInfo sendLog:(BOOL)sendLog;	
功能描述	用户问题日志反馈接口。此接口无回调，调用后就会上传信息。目前信息上传后，需要联系机智云 FAE 查看	
参数	contactInfo	用户的联系方式。此参数为选填
	feedbackInfo	用户反馈的信息。此参数为选填
	sendLog	是否发送问题日志。如果前面两个参数都没填，则默认发送问题日志
代码示例	[GizWifiSDK userFeedback:@"your_phone" feedbackInfo:@"your_message" sendLog:YES];	

【getCurrentCloudService】

定义	+ (void) getCurrentCloudService
功能描述	查询当前使用的云服务域名信息
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK

	didGetCurrentCloudService:(NSError *)result cloudServiceInfo:(NSDictionary *)cloudServiceInfo;	
回调说明	查询结果	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，cloudServiceInfo 为 nil
	cloudServiceInfo	当前域名信息，字典{key: value}格式： { "openAPIDomain" : "xxx", // NSString类型 "openAPIPort" : xxx, // int类型 "siteDomain" : "xxx", // NSString类型 "sitePort" : xxx, // int类型 }
代码示例	[GizWifiSDK sharedInstance].delegate = self; [GizWifiSDK getCurrentCloudService]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetCurrentCloudService:(NSError *)result cloudServiceInfo:(NSDictionary *)cloudServiceInfo { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }	

【setDeviceOnboardingDeploy】

定义	- (void)setDeviceOnboardingDeploy:(NSString *)ssid key:(NSString *)key configMode:(GizWifiConfigureMode)mode softAPSSIDPrefix:(NSString *)softAPSSIDPrefix timeout:(int)timeout wifiGAgentType:(NSArray *)types bind:(BOOL)bind;
功能描述	设备配网接口。配网时可自动完成设备域名部署，此接口对模组固件版本向前兼容。 把设备配置到局域网 wifi 时，如果选择了自动绑定，则设备 wifi 固件必须支持域名上报，并且 App 要在用户登录成功后再调用此接口。 设备处于 softap 模式时，会产生一个热点名称，手机 wifi 连接此热点后就可以配置了。

	如果是机智云提供的设备 wifi 固件,热点名称前缀为"XPG-GAgent-",密码为"123456789"或无密码。设备处于 airlink 模式时，手机随时都可以开始配置	
参数	ssid	待配置的路由 SSID 名
	key	待配置的路由密码
	mode	配置模式，详细见 GizWifiConfigureMode 枚举定义。
	softAPSSIDPrefix	SoftAPMode 模式下 SoftAP 的 SSID 前缀或全名，默认前缀为：XPG-GAgent-。SDK 根据该参数判断手机当前是否连上了设备的 SoftAP 热点。AirLink 模式时传 nil 即可
	timeout	配置的超时时间。SDK 默认执行的最小超时时间为 30 秒。在超时时间内如果无法配置和绑定会回调配网失败
	wifiGAgentType	待配置的模组类型，是 GizWifiGAgentType 枚举数组。GizWifiGAgentType 定义了 SDK 支持的所有模组类型，还定义了一个 GizGAgentOther 枚举值，用于开发者使用自己的配置库进行设备配置，此时参数传 GizGAgentOther 即可。 此参数传 nil、空数组或无有效枚举值，则默认为乐鑫模组
	bind	配网时是否自动绑定，YES 为自动绑定，NO 为不自动绑定。请注意，自动绑定要求先完成用户登录
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didSetDeviceOnboarding:(NSError *)result device:(GizWifiDevice *)device;	
回调说明	注意：如果调用 startWithAppInfo 接口时指定了 productInfo，但设备不在此指定范围内，则允许被配置成功，但不会出现在设备列表中。	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	配置成功或失败。如果配置失败，其他参数为 nil
	device	配网成功的设备
代码示例	[GizWifiSDK sharedInstance].delegate = self; // airlink 配置 [[GizWifiSDK sharedInstance] setDeviceOnboardingDeploy:@"your_ssid" key:@"your_key" mode:GizWifiAirLink softAPSSIDPrefix:nil timeout:60 wifiGAgentType:[NSArray arrayWithObjects: @(GizGAgentESP), nil] bind: NO]; // softap 配置 [[GizWifiSDK sharedInstance] setDeviceOnboardingDeploy:@"your_ssid" key:@"your_key" mode:GizWifiSoftAP softAPSSIDPrefix: @"your_gagent_hotspot_prefix" timeout:60 wifiGAgentType:nil] bind:NO];	

```
// 蓝牙配置
[[GizWifiSDK sharedInstance] setDeviceOnboardingDeploy:@"your_ssid"
key:@"your_key" mode:GizWifiBleLink softAPSSIDPrefix:
@"your_gagent_hotspot_prefix" timeout:60 wifiGAgentType:nil] bind:NO];


// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didSetDeviceOnboarding:(NSError
*)result device:(GizWifiDevice *)device {
       if(result.code == GIZ_SDK_SUCCESS) {
           // 配置成功
       } else {
           // 配置失败
       }
}
```

【setDeviceOnboardingDeploy】

重载方法，支持传入数组类型的 softAPSSIDPrefix

定义	`- (void)setDeviceOnboardingDeploy:(NSString * _Nonnull)ssid key:(NSString * _Nullable)key configMode:(GizWifiConfigureMode)mode softAPSSIDPrefixs:(NSArray<NSString *> * _Nullable)softAPSSIDPrefixs timeout:(int)timeout wifiGAgentType:(NSArray * _Nullable)types bind:(BOOL)bind;`

【stopDeviceOnboarding】

定义	`- (void)stopDeviceOnboarding;`	
功能描述	停止设备配网，停止后 `didSetDeviceOnboarding` 回调中返回的错误码为 `GIZ_SDK_ONBOARDING_STOPPED`	
回调	`- (void)wifiSDK:(GizWifiSDK *)wifiSDK didSetDeviceOnboarding:(NSError *)result device:(GizWifiDevice *)device;`	
回调说明	注意：如果调用 `startWithAppInfo` 接口时指定了待筛选的 `productInfo` 集合，如果设备被成功配置到路由上了，会返回配置成功，但不会出现在设备列表中。	
回调参数	`wifiSDK`	回调的 `GizWifiSDK` 单例
	`result`	配置成功或失败。如果配置失败，其他参数为 `nil`
	`device`	配网成功的设备
代码示例	`[GizWifiSDK sharedInstance].delegate = self;` `[[GizWifiSDK sharedInstance] stopDeviceOnboarding];`	

```
// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didSetDeviceOnboarding:(NSError *)result device:(GizWifiDevice *)device {
       if(result.code == GIZ_SDK_ONBOARDING_STOPPED) {
           // 配网终止
       }
}
```

【getBoundDevices】

定义	- (void)getBoundDevices:(NSString *)uid token:(NSString *)token;	
功能描述	获取绑定设备列表。在不同的网络环境下，有不同的处理： 当手机能访问外网时，该接口会向云端发起获取绑定设备列表请求； 当手机不能访问外网时，局域网设备是实时发现的，但会保留之前已经获取过的绑定设备； 手机处于无网模式时，局域网未绑定设备会消失，但会保留之前已经获取过的绑定设备；	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscovered:(NSError *)result deviceList:(NSArray *)deviceList;	
回调说明	该回调接口，在不调用 getBoundDevices 时也可能会由 SDK 主动触发，主动触发是由于 SDK 发现设备列表发生了变化，此时错误码 GIZ_SDK_SUCCESS； getBoundDevices 接口调用时会触发该回调，错误码代表云端请求状态，设备列表是绑定设备与局域网设备合并之后的集合；	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，deviceList 为非 nil 集合
	deviceList	GizWifiDevice 实例组成的数组，该参数将只返回根据指定 productKey 筛选过的设备集合。productKey 在 getBoundDevices 接口调用时指定

代码示例

```
[GizWifiSDK sharedInstance].delegate = self;
[[GizWifiSDK sharedInstance] getBoundDevices:@"your_uid" token:@"your_token"];

// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscovered:(NSError *)result deviceList:(NSArray *)deviceList {
       // 提示错误原因
       if(result.code != GIZ_SDK_SUCCESS) {
```

```
            NSLog(@"result: %@", result.localizedDescription);
        }
        // 显示设备列表
        NSLog(@"discovered deviceList: %@", deviceList);
}
```

【getBoundBleDevice】

定义	- (NSArray <GizWifiBleDevice *>* _Nullable)getBoundBleDevice;	
功能描述	触发搜索，并返回本地 BLE 设备列表。 需要初始化 SDK 时传入这款设备的 PK,并且 GizAdapterType 设置成 GizAdapterWifiBle	
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscoverBleDevice:(NSError *)result deviceList:(NSArray<GizWifiBleDevice *> *)deviceList;	
回调说明	该回调接口，在不调用 getBoundBleDevices 时也可能会由 SDK 主动触发，主动触发是由于 SDK 发现设备列表发生了变化，此时错误码 GIZ_SDK_SUCCESS；	
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，deviceList 为非 nil 集合
	deviceList	GizWifiBleDevice 实例组成的数组

代码示例

```
[GizWifiSDK sharedInstance].delegate = self;
[[GizWifiSDK sharedInstance] getBoundBleDevice];

// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscoverBleDevice:(NSError *)result deviceList:(NSArray<GizWifiBleDevice *> *)deviceList {

}
```

【bindRemoteDevice】

定义	- (void)bindRemoteDevice:(NSString *)uid token: (NSString *)token mac:(NSString *)mac productKey:(NSString *)productKey productSecret:(NSString *)productSecret beOwner:(BOOL)beOwner;	
功能描述	绑定远端设备到服务器	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
	mac	待绑定设备的 mac
	productKey	待绑定设备的 productKey
	productSecret	待绑定设备的 productSecret
	beOwner	是否以 owner 身份绑定设备。此参数只对首次绑定的用户有效

回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didBindDevice:(NSError *)result did:(NSString *)did;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	did	绑定成功的设备 did
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] bindRemoteDevice:@"your_uid" token:@"your_token" mac:@"your_mac" productKey:@"your_product_key" productSecret:@"your_product_secret" beOwner:NO]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didBindDevice:(NSError *)result did:(NSString *)did { if(result.code == GIZ_SDK_SUCCESS) { // 绑定成功 } else { // 绑定失败 } }	

【bindDeviceByQRCode】

定义	- (void)bindDeviceByQRCode:(NSString*)uid token:(NSString *)token QRContent:(NSString *)QRContent beOwner:(BOOL)beOwner;	
功能描述	根据二维码绑定设备到服务器	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
	QRContent	二维码内容。二维码需联系机智云 FAE 提供
	beOwner	是否以 owner 身份绑定设备。此参数只对首次绑定的用户有效
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didBindDevice:(NSError *)result did:(NSString *)did;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	did	绑定成功的设备 did
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] bindDeviceByQRCode:@"your_uid"	

	token:@"your_token" QRContent:@"your_QRContent" beOwner:NO]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didBindDevice:(NSError *)result did:(NSString *)did { if(result.code == GIZ_SDK_SUCCESS) { // 绑定成功 } else { // 绑定失败 } }

【unbindDevice】

定义	- (void)unbindDevice:(NSString *)uid token:(NSString *)token did:(NSString *)did;	
功能描述	从服务器解绑设备	
参数	uid	用户登录或注册时得到的 uid
	token	用户登录或注册时得到的 token
	did	待解绑设备的 did
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUnbindDevice:(NSError *)result did:(NSString *)did;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	did	已解绑的设备 did
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] unbindDevice:@"your_uid" token:@"your_token" did:@"your_did"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didUnbindDevice:(NSError *)result did:(NSString *)did { if(result.code == GIZ_SDK_SUCCESS) { // 解绑成功 } else { // 解绑失败 }	

	}

【deviceSafetyRegister】

定义	+ (void)deviceSafetyRegister:(GizWifiDevice*)gateway productKey:(NSString*)productKey devicesInfo:(NSArray*)devicesInfo;	
功能描述	设备安全注册接口。向云端加密注册设备，注册成功时返回设备 did，同时如果用户已登录则会自动绑定已注册成功的设备，绑定成功的设备会主动触发设备列表更新。需注意，安全注册需要 productKey 和 productSecret，这两个信息应在 startWithAppInfo 接口参数 productInfo 的指定范围内	
参数	gateway	设备的代理网关，此参数选填。若要注册的设备不需要代理网关，此参数可传 nil
	productKey	设备的产品类型识别码，此参数必填。若填入的 productKey 不在启动接口参数 productInfo 的指定范围将不会向云端注册
	devicesInfo	要注册的设备信息，可同时传多组设备信息，NSDictionary 数组，格式如下：[{mac:"xxx", meshID:"xxx", alias:"xxx", authCode:"xxx"}, ...] mac 设备物理唯一标识，最大 32 字符长度，NSString 类型。必填 meshID 设备组网 ID，最大 256 字符长度，NSString 类型。必填 alias 设备别名，最大 128 字符长度，NSString 类型。选填 authCode 设备注册的授权码，32 字符长度，由开发者自定义生成，NSString 类型。选填
回调	- (void)wifiSDK:(GizWifiSDK*)wifiSDK didDeviceSafetyRegister:(NSArray*)successDevices failedDevices:(NSArray*)failedDevices;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	successDevices	注册成功的设备信息，NSDictionary 数组，nil 表示无注册成功的设备。格式如下：[{mac:"xxx", productKey:"xxx", did:"xxx"}, ...] mac 注册成功的设备 mac，NSString 类型 productKey 注册成功的设备产品类型标识，NSString 类型 did 注册成功的设备唯一标识，NSString 类型
	failedDevices	注册失败的设备信息，NSDictionary 数组，nil 表示无注册失败的设备。格式如下：[{mac:"xxx", productKey:"xxx", errorCode:"xxx"}, ...] mac 注册失败的设备 mac，NSString 类型 productKey 注册失败的设备产品类型标识，NSString 类型 errorCode 失败的错误码，NSNumber 类型。错误码见枚举定义 GizWifiErrorCode

代码示例	// 设置委托 [GizWifiSDK sharedInstance].delegate = self; NSDictionary *deviceInfo = @{@"mac":device.macAddress, @"meshID":@"your_meshID", @"alias":@"your_device_alias", @"authCode":@"your_authCode" }; [GizWifiSDK deviceSafetyRegister:device productKey:@"your_productKey" devicesInfo:@[deviceInfo]]; //监听回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didDeviceSafetyRegister:(NSArray *)successDevices failedDevices:(NSArray *)failedDevices { for (NSDictionary *successDevice in successDevices) { // 处理安全注册成功的设备 } for (NSDictionary *failedDevice in failedDevices) { // 获取安全注册失败的设备和原因 } }

【deviceSafetyUnbind】

定义	+ (void)deviceSafetyUnbind:(NSDictionary*)devicesInfo;	
功能描述	设备安全解绑接口。此接口会在云端把设备的所有关联用户都解绑，可同时解绑多个相同产品类型的设备。但如果设备的产品类型（productKey）不一致将不会解绑任何设备	
参数	devicesInfo	要解绑的设备信息，格式：[{"device": device, "authCode": "xxx"}]，device 为 GizWifiDevice 对象，authCode 为授权码。authCode 不是必填参数，若没有授权码则不需要填写此字段
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDeviceSafetyUnbind:(NSArray *)failedDevices;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	failedDevices	解绑失败的设备，NSDictionary 数组，nil 表示全部解绑成功。字典格式如下：[{device:xxx, errorCode:xxx}, ...] device 解绑失败的设备对象，GizWifiDevice 类型 errorCode 失败的错误码，NSNumber 类型，见 GizWifiErrorCode 枚举定义
代码示例	// 设置委托	

```
[GizWifiSDK sharedInstance].delegate = self;
NSDictionary *deviceInfo = @{@"device":device,
                             @"authCode":@"your_authCode"};
[GizWifiSDK deviceSafetyUnbind:@[deviceInfo]];

//监听回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDeviceSafetyUnbind:(NSArray
*)failedDevices {
    for (NSDictionary *failedDevice in failedDevices) {
        //处理安全解绑失败的设备
    }
}
```

【registerBleDevice】

定义	- (void)registerBleDevice: (NSString *_Nonnull)mac;	
功能描述	注册蓝牙设备。向云端注册该设备，并且绑定到当前用户下	
参数	mac	设备的 mac 地址
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didRegisterBleDevice:(NSError * _Nullable)result mac:(NSString * _Nullable)mac productKey:(NSString * _Nullable)productKey;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	注册成功或失败，详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示注册成功，其他为失败
	mac	注册成功的设备 mac，NSString 类型， 失败为 nil
	productKey	注册成功的设备产品类型标识，NSString 类型， 失败为 nil
代码示例	// 设置委托 [GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] registerBleDevice:mac]; //监听回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didRegisterBleDevice:(NSError *)result mac:(NSString *)mac productKey:(NSString *)productKey { if (result.code == GIZ_SDK_SUCCESS) { // 注册设备成功，获取 mac 和 productKey } }	

【didDiscoverBleDevice】

回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didDiscoverBleDevice:(NSError * _Nullable)result deviceList:(NSArray<NSDictionary *> * _Nullable)deviceList;	
功能描述	本地缓存的双通道设备发生变化便会回调一次	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。
	deviceList	返回最新的双通道设备列表
代码示例	// 设置委托 [GizWifiSDK sharedInstance].delegate = self; //监听回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscoverBleDevice:(NSError *)result deviceList:(NSArray<NSDictionary *> *)deviceList { if (result.code == GIZ_SDK_SUCCESS) { // 注册设备成功，获取 mac 和 productKey } }	

【registerUser】

定义	- (void)registerUser:(NSString *)username password:(NSString *)password verifyCode:(NSString *)code accountType:(GizUserAccountType)accountType;	
功能描述	用户注册。需指定用户类型注册。手机用户的用户名是手机号，邮箱用户的用户名是邮箱、普通用户的用户名可以是普通用户名 注意：如果邮箱注册启用了邮箱激活，会返回注册成功，但不会返回 uid、token	
参数	username	注册用户名（可以是手机号、邮箱或普通用户名）
	password	注册密码
	code	手机短信验证码。短信验证码注册后就失效了，不能被再次使用
	accountType	用户类型，详细见 GizUserAccountType 枚举定义。注册手机号时，此

		参数指定为手机用户，注册邮箱时，此参数指定为邮箱用户，注册普通用户名时，此参数指定为普通用户
	encrypt	报文内容是否需要加密
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didRegisterUser:(NSError *)result uid:(NSString *)uid token:(NSString *)token;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	uid	注册成功后得到的 uid
	token	注册成功后得到的 token
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] registerUser:@"your_phone_number" password:@"your_password" verifyCode:@"your_verify_code" accountType:GizUserPhone]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didRegisterUser:(NSError *)result uid:(NSString *)uid token:(NSString *)token { if(result.code == GIZ_SDK_SUCCESS) { // 注册成功 } else { // 注册失败 } }	

【userLogin】

定义	- (void)userLogin:(NSString *)username password:(NSString *)password;	
功能描述	用户登录。需使用注册成功的用户名、密码进行登录，可以是手机用户名、邮箱用户名或普通用户名	
参数	username	注册成功的用户名
	password	注册成功的用户密码
	encrypt	报文内容是否需要加密
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其

		他为失败。失败时，uid、token 为 nil
	uid	登录成功后得到的 uid
	token	登录成功后得到的 token
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] userLogin:@"your_user_name" password:@"your_user_password"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token { if(result.code == GIZ_SDK_SUCCESS) { // 登录成功 } else { // 登录失败 } }	

【dynamicLogin】

定义	- (void)dynamicLogin:(NSString *)phone code:(NSString *)code;	
功能描述	动态验证码登录。登录用户名为手机号，以手机收到的登录验证码登录	
参数	phone	注册用户名（可以是手机号、邮箱或普通用户名）
	code	手机短信验证码。短信验证码注册后就失效了，不能被再次使用
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	uid	注册成功后得到的 uid
	token	注册成功后得到的 token
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] dynamicLogin:@"your_phone_number" code:@"your_verify_code"; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didRegisterUser:(NSError *)result uid:(NSString *)uid token:(NSString *)token { if(result.code == GIZ_SDK_SUCCESS) {	

```
        // 注册成功
    } else {
        // 注册失败
    }
}
```

【userLoginAnonymous】

定义	- (void)userLoginAnonymous;	
功能描述	匿名登录。匿名方式登录，不需要注册用户账号。	
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didRegisterUser:(NSError *)result uid:(NSString *)uid token:(NSString *)token;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	uid	注册成功后得到的 uid
	token	注册成功后得到的 token

代码示例

```
[GizWifiSDK sharedInstance].delegate = self;
[[GizWifiSDK sharedInstance] userLoginAnonymous];

// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 登录成功
    } else {
        // 登录失败
    }
}
```

【userLoginWithThirdAccount】

定义	- (void)userLoginWithThirdAccount:(GizThirdAccountType)thirdAccountType uid:(NSString *)uid token:(NSString *)token tokenSecret:(NSString *) tokenSecret;	
功能描述	第三方账号登录（第三方接口登录方式）	
参数	thirdAccountType	第三方账号类型，详细见 GizThirdAccountType 枚举定义
	uid	通过第三方平台 api 方式登录后得到的 uid

	token	通过第三方平台 api 方式 登录后得到的 token
	tokenSecret	推特账号登录时需要通过推特平台 api 方式得到此参数，其他第三方账号此参数可传 nil
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	uid	登录成功后得到的 uid
	token	登录成功后得到的 token
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] userLoginWithThirdAccount:GizThirdBAIDU uid:@"your_third_uid" token:@"your_third_token" tokenSecret:nil]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token { if(result.code == GIZ_SDK_SUCCESS) { // 登录成功 } else { // 登录失败 } }	

【transAnonymousUser】

定义	- (void)transAnonymousUser:(NSString *)token username:(NSString *)username password:(NSString *)password verifyCode:(NSString *)code accountType:(GizUserAccountType)accountType;	
功能描述	匿名用户转换，可转换为手机用户或者普通用户。注意，待转换的帐号必须是还未注册过的	
参数	token	用户登录或注册时得到的 token
	username	待转换的普通账号或手机号
	password	转换后的帐号密码
	code	转换为手机用户时要使用的手机短信验证码
	accountType	用户类型，详细见 GizThirdAccountType 枚举定义。待转换的用户名是手机号时，此参数指定为 GizUserPhone，待转换用户名是普通账号时，此参数指定为 GizUserNormal

	encrypt	报文内容是否需要加密
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didTransAnonymousUser:(NSError *)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] transAnonymousUser:@"your_token" username:@"your_phone_number" password:@"your_password" verifyCode:@"your_verify_code" accountType:GizUserPhone]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didTransAnonymousUser:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 转换成功 } else { // 转换失败 } }	

【userLogout】

定义	- (void)userLogout;	
功能描述	注销用户，会将用户已经订阅的设备取消订阅，并删除远程设备列表	
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didUserLogout: (NSError * _Nullable)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，uid、token 为 nil
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] userLogout]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogout:(NSError *)result { if (result.code == GIZ_SDK_SUCCESS) { // 注销成功	

	} }

【changeUserPassword】

定义	- (void)changeUserPassword:(NSString *)token oldPassword:(NSString *)oldPassword newPassword:(NSString *)newPassword;	
功能描述	修改用户密码	
参数	token	用户登录或注册时得到的 token
	oldPassword	旧密码
	newPassword	新密码
	encrypt	报文内容是否需要加密
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserPassword:(NSError *)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] changeUserPassword:@"your_token" oldPassword:@"your_old_password" newPassword:@"your_new_password"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserPassword:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 修改成功 } else { // 修改失败 } }	

【resetPassword】

定义	- (void)resetPassword:(NSString *)username verifyCode:(NSString *)code newPassword:(NSString *)newPassword accountType:(GizUserAccountType)accountType;
功能描述	重置密码。手机号重置密码时通过手机短信验证码重置，邮箱重置密码时需通过邮箱密码重置链接重置

参数	username	待重置密码的手机号或邮箱
	code	重置手机用户密码时需要使用手机短信验证码（通过 requestSendPhoneSMSCode 方法获取）
	newPassword	新密码。邮箱重置密码时不需要填充密码，可指定为 nil
	accountType	用户类型，详细见 GizThirdAccountType 枚举定义。待重置密码的用户名是手机号时，此参数指定为手机用户，待重置密码的用户名是邮箱时，此参数指定为邮箱用户
	encrypt	报文内容是否需要加密
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserPassword:(NSError *)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] resetPassword:@"your_phone_number" verifyCode:@"your_verify_code" newPassword:@"your_new_password" accountType:GizUserPhone]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserPassword:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 修改成功 } else { // 修改失败 } }	

【getUserInfo】

定义	- (void)getUserInfo:(NSString *)token;	
功能描述	获取用户信息	
参数	token	用户登录或注册时得到的 token
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetUserInfo:(NSError *)result userInfo:(GizUserInfo*)userInfo;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示

		成功，其他为失败
	userInfo	用户信息，详细见 GizUserInfo 类
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] getUserInfo:@"your_token"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetUserInfo:(NSError *)result userInfo:(GizUserInfo *)userInfo { if(result.code == GIZ_SDK_SUCCESS) { // 获取成功 } else { // 获取失败 } }	

【changeUserInfo】

定义	- (void)changeUserInfo:(NSString *)token username:(NSString *)username SMSVerifyCode:(NSString *)code accountType:(GizUserAccountType)accountType additionalInfo:(GizUserInfo *)additionalInfo;	
功能描述	修改用户信息，包括用户名和个人信息。用户名只支持修改手机号或邮箱，手机号或邮箱必须是没有注册过的。该接口用于以下场景：只修改手机号、只修改邮箱、只修改普通用户的个人信息、同时修改手机号和补充信息、同时修改邮箱和补充信息。只修改个人信息时，accountType 可以指定为 GizUserNormal；修改手机号要指定为 GizUserPhone；修改邮箱要指定为 GizUserEmail	
参数	token	用户登录或注册时得到的 token
	username	待修改的手机号或邮箱
	code	修改手机号时要使用的手机短信验证码
	accountType	用户类型，详细见 GizThirdAccountType 枚举定义。修改手机号时，accountType 传 GizUserPhone；修改普通用户名时，accountType 传 GizUserEmail；只修改个人信息时，accountType 传 GizUserNormal；同时修改用户名和个人信息时，可根据待修改的是手机号还是邮箱来指定。
	additionalInfo	待修改的个人信息，详细见 GizUserInfo 类定义。如果只修改个人信息，需要指定 token，username、code 填 nil
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserInfo:(NSError *)result;	

回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 修改手机号 [[GizWifiSDK sharedInstance] changeUserInfo:@"your_token" username:@"your_phone_number" SMSVerifyCode:@"your_verify_code" userType:GizUserPhone additionalInfo:nil]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserInfo:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 修改成功 } else { // 修改失败 } }	

【getCaptchaCode】

定义	- (void)getCaptchaCode:(NSString *)appSecret;	
功能描述	获取图片验证码。开发者登录 site.gizwits.com，在自己账户下的应用管理中可以得到 App Secret，通过应用的 App Secret 才能获取到图片验证码。	
参数	appSecret	应用的 secret 信息，从 site.gizwits.com 中可以看到
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetCaptchaCode:(NSError *)result token:(NSString *)token captchaId:(NSString *)captchaId captchaURL:(NSString *)captchaURL;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	token	图片验证码 token。图片验证码 token 在 1 小时后过期
	captchaId	图片验证码 id。图片验证码 5 分钟后过期
	captchaURL	图片验证码网址。图片验证码 url 在使用后过期
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] getCaptchaCode:@"your_app_secret"];	

```
// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetCaptchaCode:(NSError
*)result token:(NSString *)token captchaId:(NSString *)captchaId
captchaURL:(NSString *)captchaURL {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 获取成功
    } else {
        // 获取失败
    }
}
```

【requestSendPhoneSMSCode】

定义	- (void)requestSendPhoneSMSCode:(NSString *)token captchaId:(NSString *)captchaId captchaCode:(NSString *)captchaCode phone:(NSString *)phone;	
功能描述	通过图形验证码获取手机短信验证码	
参数	token	通过 getCaptchaCode 获取到的 token
	captchaId	通过 getCaptchaCode 获取到的 captchaId
	captchaCode	图片验证码的内容
	phone	手机号
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didRequestSendPhoneSMSCode:(NSError *)result token:(NSString *)token;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	token	请求短信验证码的 token

代码示例

```
[GizWifiSDK sharedInstance].delegate = self;
[[GizWifiSDK sharedInstance] requestSendPhoneSMSCode:@"your_token"
captchaId:@"your_captcha_id" captchaCode:@"your_captcha_code"
phone:@"your_phone_number"];

// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK
didRequestSendPhoneSMSCode:(NSError *)result token:(NSString *)token {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 获取成功
    } else {
```

```
        // 获取失败
    }
}
```

【requestSendPhoneSMSCode】

定义	- (void)requestSendPhoneSMSCode:(NSString *)appSecret phone:(NSString *)phone;	
功能描述	通过手机号请求短信验证码	
参数	appSecret	应用的 secret 信息，从 site.gizwits.com 中可以看到
	phone	手机号
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didRequestSendPhoneSMSCode:(NSError *)result token:(NSString *)token	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
	token	请求短信验证码时得到的 token
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] requestSendPhoneSMSCode:@"your_app_secret" phone:@"your_phone_number"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didRequestSendPhoneSMSCode:(NSError *)result token:(NSString *)token { if(result.code == GIZ_SDK_SUCCESS) { // 请求成功 } else { // 请求失败 } }	

【verifyPhoneSMSCode】

定义	- (void)verifyPhoneSMSCode:(NSString *)token verifyCode:(NSString *)code phone:(NSString *)phone;	
功能描述	验证手机短信验证码。注意，验证短信验证码后，验证码就失效了，无法再用于手机号注册	
参数	token	验证码的 token，通过 getCaptchaCode 获取

	code	手机短信验证码
	phone	手机号码
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didVerifyPhoneSMSCode:(NSError *)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] verifyPhoneSMSCode:@"your_token" verifyCode:@"your_verify_code" phone:@"your_phone_number"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didVerifyPhoneSMSCode:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 验证成功 } else { // 验证失败 } }	

【getDevicesToSetServerInfo】

定义	+ (void)getDevicesToSetServerInfo;	
功能描述	获取可以设置域名的设备列表。该接口返回支持域名设置功能的设备信息列表，App 可以在给设备设置域名前，先调用该接口查看有哪些设备可以设置域名。	
回调	- (void)wifiSDK:(GizWifiSDK*)wifiSDK didGetDevicesToSetServerInfo:(NSError*)result devices:(NSArray*)devices;	
回调说明	该回调接口只返回设备的 mac、productKey、domain 这三个信息，不返回设备对象	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	获取成功或失败。如果获取失败，其他参数为 nil
	devices	设备信息字典组成的数组。设备信息的字典格式如下： { “mac”: “xxx” // 设备 mac 地址 “productKey”: “xxx” // 设备的 productKey “domain”: “xxx” // 设备的域名信息

		}
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 获取可设置域名的设备列表 [[GizWifiSDK sharedInstance] getDevicesToSetServerInfo]; // 实现回调 - (void)wifiSDK:(GizWifiSDK*)wifiSDK didGetDevicesToSetServerInfo:(NSError*)result devices:(NSArray*)devices { if(result.code == GIZ_SDK_SUCCESS) { // 获取成功 } else { // 获取失败 } }	

【setDeviceServerInfo】

定义	+ (void)setDeviceServerInfo:(NSString*)domain mac:(NSString*)mac;	
功能描述	此接口为手动设置设备域名接口，可为设备设置对应的云服务域名。 设备和手机都连接到同一个 wifi 路由器后，可以设置设备要连接的云服务域名。可以设置当前已上线的所有小循环设备的域名。也可以单独设置某个设备的域名。如果不知道设备的 MAC 地址，可以先调用 getDevicesToSetServerInfo 接口查看有哪些设备可以设置域名，再调用该接口进行设置。 注意： 1、只支持可设置域名的设备 2、调用该接口将关闭已开启的设备域名自动设置功能	
参数	domain	待设置的域名。若该参数为 nil，SDK 将根据用户手机的地理位置信息为设备设置机智云统一部署的云服务域名。 若要让设备连接独立部署的私有云域名，该参数为对应的私有云域名字符串，格式为：api.xxxxxx.com。这里需保证传入的域名是有效的，否则可能导致设备无法正常工作
	mac	待设置的设备 mac。默认参数为 nil，即所有已发现的小循环设备都会被修改域名。如果只设置特定设备的域名，需指定 mac 地址
回调	- (void)wifiSDK:(GizWifiSDK*)wifiSDK didSetDeviceServerInfo:(NSError*)result mac:(NSString*)mac;	

回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	mac	设置域名的设备 mac
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 给设备设置域名 [[GizWifiSDK sharedInstance] setDeviceServerInfo:nil mac :@"your_device_mac"]; // 实现回调 - (void)wifiSDK:(GizWifiSDK*)wifiSDK didSetDeviceServerInfo:(NSError*)result mac:(NSString*)mac { if(result.code == GIZ_SDK_SUCCESS) { // 设置成功 } else { // 设置失败 } }	

【disableLAN】

定义	+ (void)disableLAN:(BOOL)disabled	
功能描述	设置是否禁用小循环功能	
参数	disabled	禁用或启用小循环
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDisableLAN:(NSError *)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; [GizWifiSDK disableLAN: YES]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didDisableLAN:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 成功	

```
    } else {
        // 失败
    }
};
```

【getSSIDList】

定义	- (void)getSSIDList;	
功能描述	在 Soft-AP 模式时，获得设备的 SSID 列表。SSID 列表通过异步回调方式返回	
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetSSIDList:(NSError *)result ssidList:(NSArray *)ssidList;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，ssidList 为 nil
	ssidList	为若干 GizWifiSSID 实例组成的 SSID 信号列表
代码示例	[GizWifiSDK sharedInstance].delegate = self; [[GizWifiSDK sharedInstance] getSSIDList]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetSSIDList:(NSError *)result ssidList:(NSArray *)ssidList { if(result.code == GIZ_SDK_SUCCESS) { // 获取成功 } else { // 获取失败 } }	

【setLogLevel】

定义	+ (void)setLogLevel:(GizLogPrintLevel)logPrintLevel;	
功能描述	设置日志输出级别。该级别指日志在调试终端的输出级别，默认是全部输出的。 日志输出级别不影响日志文件的输出，无论日志输出级别设成什么，SDK 都会将运行日志写入文件。日志文件存放在 Documents 目录下：GizWifiSDK/GizSDKLog/	
参数	logLevel	日志输出级别，参考 GizLogPrintLevel 定义
代码示例	[[GizWifiSDK sharedInstance] setLogLevel: GizLogPrintAll];	

【channelIDBind】

定义	- (void)channelIDBind:(NSString *)token channelID:(NSString *)channelID alias:(NSString *)alias pushType:(GizPushType)pushType;	
功能描述	绑定推送的 ID 到当前用户下，以接收设备的通知事件	
参数	token	用户登录或注册时得到的 token
	channelID	从第三方推送平台获取的推送 ID
	alias	推送别名，只在极光推送中产生作用
	pushType	推送类型，详细见 GizPushType 枚举定义
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChannelIDBind:(NSError *)result;	
回调参数	wifiSDK	wifiSDK 为回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。result.code 为 GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 绑定推送 ID [[GizWifiSDK sharedInstance] channelIDBind:@"your_user_token" channelID:@"your_push_channelID" alias:nil pushType:GizPushJiGuang]; // 绑定推送回调 - (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didChannelIDBind:(NSError * _Nonnull)result { if (result.code == GIZ_SDK_SUCCESS) { // 绑定推送 ID 成功 } else { // 绑定推送 ID 失败 } }	

【channelIDUnBind】

定义	- (void)channelIDUnBind:(NSString *)token channelID:(NSString *)channelID;
功能描述	解绑推送 ID 与当前用户的绑定关系，以解除接收设备的通知事件

参数	token	用户登录或注册时得到的 token
	channelID	从第三方推送平台获取的推送 ID
回调	- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChannelIDUnBind:(NSError *)result;	
回调参数	wifiSDK	wifiSDK 为回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。result.code 为 GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，其他回调参数为 nil
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 解绑推送 ID [[GizWifiSDK sharedInstance] channelIDUnBind:@"your_user_token" channelID:@"your_push_channelID"]; // 解绑推送 ID 回调 - (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didChannelIDUnBind:(NSError * _Nonnull)result { if (result.code == GIZ_SDK_SUCCESS) { //解绑推送引擎成功 } else { //解绑推送引擎失败 } }	

【setUserMeshName】

定义	+ (void)setUserMeshName:(NSString *)meshName password:(NSString *)password uuidInfo:(NSDictionary *)uuidInfo meshLTK:(NSData *)meshLTK meshVendor:(GizMeshVerdor)meshVendor;	
功能描述	设置 Mesh 组网信息，在用户登录成功之后调用	
参数	meshName	当前登录用户所在组网名称
	password	当前登录用户所在组网密码
	uuidInfo	服务和角色特征值，格式：{"serviceUUID": "xxx", "pairUUID": "xxx", "commandUUID": "xxx", "notifyUUID": "xxx”}, 这几个 key 分别对应为：服务特征值、登录配对角色特征值、控制角色特征值、通知角色特征值。此参数不能为 nil，且每个 key 都必传

	meshLTK	mesh 设备通信密钥。此参数不能为 nil， 且范围在 1~16 个字节之间
	meshVendor	GizMeshVerdor 类型，表示 Mesh 设备所属厂商
代码示例	NSDictionary *uuidInfo = @{@"serviceUUID”:@“device_serviceUUID”, @"pairUUID”:@“device_pairUUID”, @"commandUUID”:@“device_commandUUID”, @"notifyUUID”:@“device_notifyUUID” }; Byte ltkBuffer[16]; memset(ltkBuffer, 0, 16); for (int j=0; j<16; j++) { if (j<8) { ltkBuffer[j] = 0xc0+j; }else{ ltkBuffer[j] = 0xd0+j; } } NSData *meshLTK = [NSData dataWithBytes:ltkBuffer length:16]; [GizWifiSDK setUserMeshName:@"your_user_mesh_name" password:@"your_user_mesh_password" uuidInfo:uuidInfo meshLTK:meshLTK meshVendor:GizMeshTelink];	

【searchMeshDevice】

定义	+(void)searchMeshDevice:(NSString * _Nullable)meshName timeout:(int)timeout;	
功能描述	搜索 mesh 组网设备。每次搜索指定超时时间，时间到则停止搜索，在搜索期间每发现一台新设备会触发一次 didDiscoveredMeshDevices 回调，若在查找期间没发现任何设备，在查找结束时会返回查找停止	
参数	meshName	指定 mesh 网络名称。如果为 nil，则搜索所有 mesh 网络设备；不为 nil，则只搜索该 mesh 网络设备
	timeout	搜索的超时时间（以秒为单位）
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didDiscoveredMeshDevices:(NSError * _Nullable)result meshDeviceList:(NSArray * _Nonnull)meshDeviceList;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，deviceList 大小为 0

	meshDeviceList	mesh 设备列表，NSDictionary 数组。格式：[{"mac":"xxx", "meshID": "xxx", "advData":"xxx"}]
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 查找组网 meshName 下的蓝牙设备 [GizWifiSDK searchMeshDevice:@"meshName" timeout:timeout]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscoveredMeshDevices:(NSError *)result meshDeviceList:(NSArray *)meshDeviceList { if (result.code == GIZ_SDK_SUCCESS) { // 发现新设备 for (NSDictionary *meshDevice in meshDeviceList) { // 对新设备做进一步操作 } } else { // 发现设备失败 } }	

【changeDeviceMesh】

定义	+ (void)changeDeviceMesh:(NSDictionary * _Nonnull)meshDeviceInfo currentMesh:(NSDictionary * _Nonnull)currentMeshInfo newMeshID:(NSUInteger)newMeshID;	
功能描述	修改 Mesh 设备组网，即将设备从 A 组网切换到 B 组网，通常是调用这个接口将设备从发现的组网切换到当前用户的组网下；搜索到新的 mesh 设备后，需要先调用这个接口切网，再去做安全注册。	
参数	meshDeviceInfo	mesh 设备信息 NSDictionary 类型 {"mac":"xxx", "meshID": "xxx"}。此参数不能为 nil
	currentMeshInfo	设备当前所在组网信息,格式:{"meshName": "xxx","password": "xxx"}，不能为空
	newMeshID	即将为该设备分配的新 MeshID，取值范围 1~255,需要传一个在当前组网未被使用过的 meshID
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK	

	didChangeDeviceMesh:(NSDictionary * _Nonnull)meshDeviceInfo result:(NSError * _Nullable)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	meshDeviceInfo	切网成功，返回新的设备信息,切网失败，返回原来的设备信息，NSDictionary 类型 {"mac":"xxx", "meshID": "xxx"}
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; // 切网设备的信息 NSDictionary *meshDeviceInfo = @{@"mac": @"FFFF83AB2317", @"meshID": @"17"}; // 设备当前所在组网信息 NSDictionary *currentMeshInfo = @{@"meshName": @"device_meshName", @"password": @"device_meshPassword"}; // 切网后为设备重新分配的 MeshID int newMeshID = 1; // 去切换设备组网 [GizWifiSDK changeDeviceMesh:meshDeviceInfo currentMesh:currentMeshInfo newMeshID:newMeshID]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeDeviceMesh:(NSDictionary *)meshDeviceInfo result:(NSError *)result { if (result.code == GIZ_SDK_SUCCESS) { // 切网成功 } else { // 切网失败 } }	

【restoreDeviceFactorySetting】

定义	+ (void)restoreDeviceFactorySetting:(GizLiteGWSubDevice * _Nonnull)meshDevice;
功能描述	恢复 Mesh 设备出厂设置，只能对属于当前用户组网的设备调用这个接口

参数	meshDevice	要恢复出厂设置的设备对象。此参数不能为 nil
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didRestoreDeviceFactorySetting:(NSString * _Nullable)mac result:(NSError * _Nullable)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	mac	返回恢复出厂设置成功的设备 mac 地址
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; //即将恢复出厂设置的设备 GizLiteGWSubDevice *meshDevice = deviceList.firstObject; //恢复设备出厂设置 [GizWifiSDK restoreDeviceFactorySetting:meshDevice]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didRestoreDeviceFactorySetting:(NSString *)mac result:(NSError *)result { if (result.code == GIZ_SDK_SUCCESS) { //恢复出厂设置成功 } else { //恢复出厂设置失败 } }	

【addDevices】

定义	+ (void)addDevices:(NSArray<GizLiteGWSubDevice *> * _Nonnull)meshDevices toGroup:(uint8_t)groupID;	
功能描述	将所给设备添加到给定分组中	
参数	groupID	分组 ID 范围 [1, 254]， 一台设备最多只能创建 8 个分组
	meshDevices	mesh 设备列表，列表元素是 GizLiteGWSubDevice 类型
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didAddDevicesToGroup:(NSArray<GizLiteGWSubDevice *> * _Nullable)successMeshDevice result:(NSError * _Nullable)result;	

回调参数	wifiSDK	回调的 GizWifiSDK 单例
	successMeshDevice	成功添加到指定分组的设备集合
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; NSMutableArray<GizLiteGWSubDevice *> *meshDeviceList = deviceList; //将设备集合 meshDeviceList 添加到分组 1 中 [GizWifiSDK addDevices:meshDeviceList toGroup:1]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didAddDevicesToGroup:(NSArray<GizLiteGWSubDevice *> *)successMeshDevice result:(NSError *)result { if (result.code == GIZ_SDK_SUCCESS) { for (GizWifiDevice *device in successMeshDevice) { //遍历获取分组成功的设备 } } }	

【deleteDevices】

定义	+ (void)deleteDevices:(NSArray<GizLiteGWSubDevice *> * _Nonnull)meshDevices fromGroup:(uint8_t)groupID;	
功能描述	将所给设备从给定分组删除	
参数	groupID	分组 ID 范围 [1, 254]
	meshDevices	mesh 设备列表，列表元素是 GizLiteGWSubDevice 类型
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didDeleteDevicesFromGroup:(NSArray<GizLiteGWSubDevice *> * _Nullable)successMeshDevice result:(NSError * _Nullable)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	successMeshDevice	删除分组成功的设备列表
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例	[GizWifiSDK sharedInstance].delegate = self; NSMutableArray<GizLiteGWSubDevice *> *meshDeviceList = deviceList; //将设备集合 meshDeviceList 从分组 1 删除 [GizWifiSDK deleteDevices:meshDeviceList fromGroup:1]; // 实现回调 - (void)wifiSDK:(GizWifiSDK *)wifiSDK didDeleteDevicesFromGroup:(NSArray<GizLiteGWSubDevice *> *)successMeshDevice result:(NSError *)result { if (result.code == GIZ_SDK_SUCCESS) { for (GizLiteGWSubDevice *device in successMeshDevice) { //遍历从分组移除成功的设备 } } }

【getDeviceGroups】

定义	+ (void)getDeviceGroups:(GizLiteGWSubDevice * _Nullable)meshDevice;	
功能描述	获取设备所在的分组集合	
参数	meshDevice	GizLiteGWSubDevice 类型的设备对象
回调	- (void)wifiSDK:(GizWifiSDK * _Nonnull)wifiSDK didGetDeviceGroups: (NSArray<NSNumber *> * _Nonnull)groupIDList device:(GizLiteGWSubDevice * _Nonnull)meshDevice result:(NSError * _Nullable)result;	
回调参数	wifiSDK	回调的 GizWifiSDK 单例
	groupIDList	设备所在分组的 ID 列表
	meshDevice	GizLiteGWSubDevice 类型的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	[GizWifiSDK sharedInstance].delegate = self; GizLiteGWSubDevice *meshDevice = deviceList.firstObject;	

```
    //获取设备所在分组列表
    [GizWifiSDK getDeviceGroups:meshDevice]

// 实现回调
- (void)wifiSDK:(GizWifiSDK *)wifiSDK
didGetDeviceGroups:(NSArray<NSNumber *> *)groupIDList
device:(GizLiteGWSubDevice *)meshDevice result:(NSError *)result{
    if (result.code == GIZ_SDK_SUCCESS) {
        //成功获取到设备的分组列表 groupIDList
    }
    else {
        //获取分组失败
    }
}
```

2. GizWifiDevice 类

2.1. 简介

机智云 Wifi 的设备类。GizWifiDevice 类为 APP 开发者提供设备订阅、设备数据通知、设备实时状态通知，例如热水器的水温等功能。该设备实例是通过 GizWifiDevice 类分配出来的，不能自行创建。

2.2. 属性访问

属性	描述
delegate	使用委托获取对应事件。GizWifiDevice 对应的回调接口在 GizWifiDeviceDelegate 定义。需要用到哪个接口，回调即可。
macAddress	NSString 类型。设备的物理地址，如果是 VIRTUAL:SITE，则是虚拟设备
did	NSString 类型。设备云端身份标识 DID
ipAddress	NSString 类型。设备的 ip 地址，大循环设备的 ip 地址为云端服务器域名
productKey	NSString 类型。设备的产品类型识别码
productName	NSString 类型。设备的产品名称
productType	GizWifiDeviceType 类型。设备分类，是中控设备还是普通设备
remark	NSString 类型。设备的备注信息，设备绑定后可以修改，默认为空
alias	NSString 类型。设备的别名，设备绑定后可以修改，默认为空

属性	描述
isLAN	BOOL 类型。设备是否为小循环
isBind	BOOL 类型。设备是否已绑定
isDisabled	BOOL 类型。判断设备是否已在云端注销
isSubscribed	BOOL 类型。设备是否已订阅
isProductDefined	BOOL 类型。设备是否定义了产品数据点
netStatus	GizWifiDeviceNetStatus 类型。设备的网络状态，表示在线、离线或是否就绪
netType	GizDeviceNetType 类。设备的网络类型，表示设备是否为 WiFi、NB 等类型
sharingRole	GizDeviceSharingUserRole 类型。表示绑定设备的用户具有的权限
rootDevice	GizWifiDevice 类型。设备的根设备。子设备的根设备为网关，无根设备时为 nil
productUI	NSString 类型，设备数据点定义的 Json 字符串
deviceModuleFirmwareVer	NSString 类型，设备的 WiFi 固件版本号
deviceMcuFirmwareVer	NSString 类型，设备的 MCU 固件版本号

2.3. 回调接口

以下是 GizWifiDevice 类提供的所有回调接口，将在在后续 API 定义中详细介绍:

- didUpdateNetStatus：设备网络状态变化通知
- didReceiveData：控制指令回复或设备状态上报的回调
- didSetSubscribe：设备订阅或解除订阅的回调
- didGetHardwareInfo：设备硬件信息的回调
- didSetCustomInfo：设置设备绑定信息的回调
- didExitProductionTesting：设备退出产测的回调

下面是对其中主动上报回调的具体说明:

【didUpdateNetStatus】

回调	- (void)device:(GizWifiDevice *)device didUpdateNetStatus:(GizWifiDeviceNetStatus)netStatus;	
回调说明	该回调主动上报设备的网络状态变化，当设备重上电、断电或可控时会触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	netStatus	设备是离线、在线还是可控状态
代码示例	// mDevice为从设备列表中取到的设备对象	

	mDevice.delegate = self; // 实现回调 - (void)device:(GizWifiDevice *)device didUpdateNetStatus:(GizWifiDeviceNetStatus)netStatus { }

【didReceiveData】

回调	- (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)dataMap withSN:(NSNumber *)sn;	
回调说明	该回调主动上报设备的状态变化，设备收到控制指令后可能引起的状态变化或设备主动上报状态时会触发该回调	
回调参数	device	回复状态的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，dataMap 为空字典
	data	设备上报的数据内容，字典格式： { "data": [value], // value 为 NSDictionary 类型，内容为设备状态键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致 "alerts": [value], // value 为 NSDictionary 类型，内容为设备报警键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致 "faults": [value], // value 为 NSDictionary 类型，内容为设备故障键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致 "binary": [value], // value 为 NSData 类型，内容为二进制数据，指没有在 site 上定义数据点的需要透传的数据 }
	sn	控制指令的应答序号，此应答序号与 APP 发送控制指令的序号一致。设备主动上报数据和回复状态查询时，序号为 0
代码示例	// mDevice为从设备列表中取到的设备对象 mDevice.delegate = self; // 实现回调 - (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)data withSN:(NSNumber *)sn { if(result.code == GIZ_SDK_SUCCESS) {	

	}

2.4. API

【setSubscribe:autoGetDeviceStatus:】

定义	- (void)setSubscribe:(BOOL)subscribed autoGetDeviceStatus:(BOOL)autoGetDeviceStatus;	
功能描述	设备订阅或解除订阅。订阅了设备，表示使用者关心这个设备的消息推送。解除订阅，表示使用者不关心这个设备的消息推送。订阅设备后，SDK 将自动登录和自动绑定设备。解除订阅后，设备连接将自动断开，但不会自动解绑。一般来说，设备订阅都会成功的，SDK 会记住设备是否被订阅了	
参数	subscribed	订阅或者解除订阅。YES 表示订阅，NO 表示解除订阅
	autoGetDeviceStatus	是否自动去获取一次设备状态。YES 表示在与设备建立连接成功之后，SDK 会主动去获取一次设备状态； NO 表示 SDK 不会主动获取设备状态
回调	- (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed;	
回调参数	device	回调的 GizWifiDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，设备的订阅状态无变化
	isSubscribed	设备是被订阅了还是被取消订阅了。YES 表示被订阅，NO 表示被解除订阅
代码示例	// mDevice 为从设备列表中取到的设备对象 mDevice.delegate = self; // 订阅设备并自动获取一次设备状态 [mDevice setSubscribe:YES autoGetDeviceStatus:YES]; // 订阅设备不自动获取状态 [mDevice setSubscribe:YES autoGetDeviceStatus:NO]; // 解除订阅 [mDevice setSubscribe:NO autoGetDeviceStatus:NO]; // 实现回调 - (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed { if(result.code == GIZ_SDK_SUCCESS) { // 订阅或解除订阅成功 } else {	

```
            // 操作失败
        }
    }
```

【setSubscribe:subscribed:】

定义	- (void)setSubscribe:(NSString *)productSecret subscribed:(BOOL)subscribed;	
功能描述	设备订阅或解除订阅。订阅了设备，表示使用者关心这个设备的消息推送。解除订阅，表示使用者不关心这个设备的消息推送。订阅设备后，SDK 将自动登录和自动绑定设备，并且与设备建立连接成功之后会去自动获取一次设备状态。解除订阅后，设备连接将自动断开，但不会自动解绑。一般来说，设备订阅都会成功的，SDK 会记住设备是否被订阅了	
参数	productSecret	设备所属产品的秘钥
	subscribed	订阅或者解除订阅。YES 表示订阅，NO 表示解除订阅
回调	- (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed;	
回调参数	device	回调的 GizWifiDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，设备的订阅状态无变化
	isSubscribed	设备是被订阅了还是被取消订阅了。YES 表示被订阅，NO 表示被解除订阅

代码示例

```
// mDevice 为从设备列表中取到的设备对象
mDevice.delegate = self;
[mDevice setSubscribe:@"your_product_secret" subscribed:YES]; // 订阅设备
[mDevice setSubscribe:@"your_product_secret" subscribed:NO]; // 解除订阅

// 实现回调
- (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 订阅或解除订阅成功
    } else {
        // 操作失败
    }
}
```

【setSubscribe】

定义	- (void)setSubscribe:(BOOL)subscribed;	
功能描述	设备订阅或解除订阅。订阅了设备，表示使用者关心这个设备的消息推送。解除订阅，表示使用者不关心这个设备的消息推送。订阅设备后，SDK 将自动登录和自动绑定设备，并且与设备建立连接成功之后会去自动获取一次设备状态。解除订阅后，设备连接将自动断开，但不会自动解绑。一般来说，设备订阅都会成功的，SDK 会记住设备是否被订阅了	
参数	subscribed	订阅或者解除订阅。YES 表示订阅，NO 表示解除订阅
回调	- (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed;	
回调参数	device	回调的 GizWifiDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，设备的订阅状态无变化
	isSubscribed	设备是被订阅了还是被取消订阅了。YES 表示被订阅，NO 表示被解除订阅
代码示例	// mDevice 为从设备列表中取到的设备对象 mDevice.delegate = self; // 订阅设备 [mDevice setSubscribe:YES]; // 解除订阅 [mDevice setSubscribe:NO]; // 实现回调 - (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed { if(result.code == GIZ_SDK_SUCCESS) { // 订阅或解除订阅成功 } else { // 操作失败 } }	

【getDeviceStatus】

定义	- (void)getDeviceStatus:(NSArray*) attrs;	
功能描述	获取设备状态。已订阅的设备变为可控状态后才能获取到状态。如果设备是变长数据点类型，则可查询指定的数据点状态	
参数	attrs	要查询状态的数据点名称，为 NSString 类型数组。此参数默认值为 nil。SDK

		默认返回设备的所有数据点状态。若要查询某些数据点的状态，参数应指定为要查询的数据点名称数组
回调	- (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)dataMap withSN:(NSNumber *)sn;	
回调说明	设备回复或上报的数据中，当 SDK 遇到无法解析的数据时，会作为透传数据处理，此时错误码为 GIZ_SDK_SUCCESS。	
回调参数	device	回复状态的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，dataMap 为空字典
	data	设备上报的数据内容，字典格式： { "data": [value], // value 为 NSDictionary 类型，内容为设备状态键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致 "alerts": [value], // value 为 NSDictionary 类型，内容为设备报警键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致 "faults": [value], // value 为 NSDictionary 类型，内容为设备故障键值对，[数据点标识名：数据点值]，数据点值的类型与 site 上的定义一致 "binary": [value], // value 为 NSData 类型，内容为二进制数据，指没有在 site 上定义数据点的需要透传的数据 }
	sn	控制指令的应答序号，此应答序号与 APP 发送控制指令的序号一致。设备主动上报数据和回复状态查询时，序号为0
代码示例	// mDevice为从设备列表中取到的设备对象 mDevice.delegate = self; [mDevice getDeviceStatus:nil]; // 实现回调 - (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)data withSN:(NSNumber *)sn { if(result.code == GIZ_SDK_SUCCESS) { // 查询成功 } else { // 查询失败 } }	

【write】

定义	- (void)write:(NSDictionary *)data withSN:(int)sn;	
功能描述	给设备发送控制指令。已订阅的设备变为可控状态后才能发送控制指令	
参数	data	该参数为要发给设备的操作指令。为字典格式，字典键值对可按以下方式填充： 1、如果设备有数据点定义，操作指令一次可以下发多个数据点。字典中的 key 为数据点名称，value 为数据点的值。value 类型要与数据点定义一致： （1）如果数据点为布尔类型，则 value 为 NSNumber 类型； （2）如果数据点为数值类型，则 value 为 NSNumber 类型； （3）如果数据点为枚举类型，则 value 为枚举序号（NSNumber 类型）或者枚举字符串（NSString 类型）； 如果数据点为扩展类型，则 value 为 NSData 类型； 2、如果设备操作采用透传方式，透传指令一次只能下发一条。透传数据的 key 为”binary”，value 为 NSData 类型
	sn	控制命令序号，用于对应控制命令应答数据。控制确认回调时会返回这个 sn
回调	- (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)dataMap withSN:(NSNumber *)sn;	
回调说明	设备回复或上报的数据中，当 SDK 遇到无法解析的数据时，会作为透传数据处理，此时错误码为 GIZ_SDK_SUCCESS。	
回调参数	device	回复状态的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，dataMap 为空字典
	data	设备上报的数据内容，字典格式： { "data": [value], // value 为 NSDictionary 类型，内容为设备状态键值对，[数据点标识名: 数据点值]，数据点值的类型与 site 上的定义一致 "alerts": [value], // value 为 NSDictionary 类型，内容为设备报警键值对，[数据点标识名: 数据点值]，数据点值的类型与 site 上的定义一致 "faults": [value], // value 为 NSDictionary 类型，内容为设备故障键值对，[数据点标识名: 数据点值]，数据点值的类型与 site 上的定义一致 "binary": [value], // value 为 NSData 类型，内容为二进制数据，指没有在 site 上定义数据点的需要透传的数据 }
	sn	控制指令的应答序号，此应答序号与 APP 发送控制指令的序号一致。设备主动上报数据和回复状态查询时，序号为 0

代码示例	// mDevice 为从设备列表中取到的设备对象 mDevice.delegate = self; // 开灯 int sn = 5; [mDevice write: @{@"LED_OnOff": @(YES)} sn:@(sn)]; // 实现回调 - (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)data withSN:(NSNumber *)sn { if(result.code == GIZ_SDK_SUCCESS) { if (sn == 5) { // 命令序号相符，开灯指令执行成功 } else { // 其他命令的 ack 或者数据上报 } } else { // 执行失败 } }

【setCustomInfo】

定义	- (void)setCustomInfo:(NSString *)remark alias:(NSString *)alias;	
功能描述	修改设备的备注和别名。设备绑定后才能修改。 请注意，remark 和 alias 内容中若有以下左边的 5 个字符，将被转义为右边的内容： & --> & < --> < > --> > " --> " ' --> &#39	
参数	remark	待修改的备注信息。传 nil 表示不修改，传@""则会覆盖为空串
	alias	待修改的设备别名。传 nil 表示不修改，传@""则会覆盖为空串
回调	- (void)device:(GizWifiDevice *)device didSetCustomInfo:(NSError *)result;	
回调参数	device	修改备注和别名的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	// mDevice为从设备列表中取到的设备对象	

```
mDevice.delegate = self;
[mDevice setCustomInfo:@"your_remark" alias:@"your_alias"];

// 实现回调
- (void)device:(GizWifiDevice *)device didSetCustomInfo:(NSError
*)result {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 修改成功
    } else {
        // 修改失败
    }
}
```

【getHardwareInfo】

定义	- (void) getHardwareInfo;	
功能描述	获取硬件信息	
回调	- (void)device:(GizWifiDevice *)device didGetHardwareInfo:(NSError *)result hardwareInfo:(NSDictionary *)hardwareInfo;	
回调参数	device	返回硬件信息的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，hardwareInfo 为 nil
	hardwareInfo	硬件信息。对应的硬件信息键值对有： { "wifiHardVersion": [value], // value 为 NSString 类型，设备的 Wifi 模组硬件版本号 "wifiSoftVersion": [value], // value 为 NSString 类型，设备的 Wifi 模组软件版本号 "wifiFirmwareId": [value], // value 为 NSString 类型，设备的 Wifi 固件 ID "wifiFirmwareVer": [value], // value 为 NSString 类型，设备的 Wifi 固件版本 "mcuHardVersion": [value], // value 为 NSString 类型，设备的硬件版本号 "mcuSoftVersion": [value], // value 为 NSString 类型，设备的软件版本号 "productKey": [value], // value 为 NSString 类型，设备的产品唯一标识码 }

代码示例	// mDevice为从设备列表中取到的设备对象 mDevice.delegate = self; [mDevice getHardwareInfo]; // 实现回调 - (void)device:(GizWifiDevice *)device didGetHardwareInfo:(NSError *)result hardwareInfo:(NSDictionary *)hardwareInfo { if(result.code == GIZ_SDK_SUCCESS) { // 获取成功 } else { // 获取失败 } }

【exitProductionTesting】

定义	- (void) exitProductionTesting;	
功能描述	退出产测模式。不订阅设备就可以调用此接口，设备进入产测模式后会响应	
回调	- (void)device:(GizWifiDevice *)device didExitProductionTesting:(NSError *)result;	
回调参数	device	退出产测的设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	// mDevice为从设备列表中取到的设备对象 mDevice.delegate = self; [mDevice exitProductionTesting]; // 实现回调 - (void)device:(GizWifiDevice *)device didExitProductionTesting:(NSError *)result { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }	

3. GizLiteGWDevice 类

3.1. 简介

GizLiteGWDevice 类为 APP 开发者提供轻网关子设备操作，可读取设备组网标识。

该类继承自 GizWifiDevice 类，除下列属性外，也具备 GizWifiDevice 类的所有属性和方法。

3.2. 属性访问

属性	描述
meshId	NSString 类型，只读。设备组网标识

4. GizWifiBleDevice 类

4.1. 简介

GizWifiBleDevice 类为 APP 开发者提供双通道设备，双通道设备即既可以走 WiFi 控制也可以走蓝牙控制的设备；当 WiFi 控制可行时，走 WiFi 控制；当 WiFi 离线时，蓝牙通道被开启，走蓝牙控制。

该类继承自 GizWifiDevice 类，除下列属性外，也具备 GizWifiDevice 类的所有属性和方法。

4.2. 属性访问

属性	描述
isBlueLocal	标志设备当前的状态；true：表示当前使用的是蓝牙通道
bleWorkStatus	标志当前设备工作状态，参考 **GizBleWorkStatusType**

4.3. API

【connectBle】

定义	- (void)connectBle: (bleCallback)callback;	
功能描述	当设备处于蓝牙通道时，即 isBlueLocal 为 true 时，调用此接口可以与设备建立蓝牙连接	
回调	回调原型：typedef void (^bleCallback)(GizWifiErrorCode errorCode);	
回调参数	errorCode	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示连接

		成功，其他为失败。
代码示例	// mDevice 为从设备列表中取到的设备对象 mDevice.delegate = self; // 订阅设备 if ([mDevice.device isMemberOfClass:[GizWifiBleDevice class]]) { GizWifiBleDevice *bleDevice = (GizWifiBleDevice *)mDevice; if (bleDevice.netStatus != GizDeviceControlled && bleDevice.isBlueLocal) { [bleDevice connectBle:^(GizWifiErrorCode errorCode) { if (errorCode == GIZ_SDK_SUCCESS) { // 连接设备成功 } }]; } }	

【disconnectBle】

定义	- (void)disconnectBle: (bleCallback)callback;	
功能描述	当设备处于蓝牙通道时，即 isBlueLocal 为 true 时，调用此接口可以与设备断开蓝牙连接	
回调	回调原型：typedef void (^bleCallback)(GizWifiErrorCode errorCode);	
回调参数	errorCode	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示断连成功，其他为失败。
代码示例	// mDevice 为从设备列表中取到的设备对象 mDevice.delegate = self; // 订阅设备 if ([mDevice.device isMemberOfClass:[GizWifiBleDevice class]]) { GizWifiBleDevice *bleDevice = (GizWifiBleDevice *)mDevice; [bleDevice disconnectBle:^(GizWifiErrorCode errorCode) { }] }	

【checkUpdate】

定义	- (void)checkUpdate:(GizOTAFirmwareType)firmwareType completion:(void(^)(NSError * _Nonnull result , NSString *lastVersion, NSString *currentVersion))completion;	
参数	firmwareType	详细见 GizOTAFirmwareType
	listener	回调

功能描述	检查设备是否可以 OTA	
回调	void(^)(NSError * _Nonnull result , NSString *lastVersion, NSString *currentVersion)	
回调参数	result	result.code 为 GizWifiErrorCode。详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示 check 成功，其他为失败。
	lastVersion	云端可更新固件的版本号
	currentVersion	当前设备版本号
代码示例	[device checkUpdate:enumType completion:^(NSError * _Nonnull result, NSString * _Nonnull lastVersion, NSString * _Nonnull currentVersion) { if (result.code == GIZ_SDK_SUCCESS) { // 检查通过后就可以调用 OTA 接口了 } else { } }];	

【startUpgrade】

定义	- (void)startUpgrade:(GizOTAFirmwareType)firmwareType listener:(void (^)(GizOTAEventType type, NSError *result))listener;	
参数	firmwareType	详细见 GizOTAFirmwareType
	listener	回调
功能描述	开始 OTA	
回调	void (^)(GizOTAEventType type, NSError *result)	
回调参数	type	OTA 事件，见 GizOTAEventType
	result	result.code 为 GizWifiErrorCode。详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 以及 type 为 GizOTAEventFinish 表示成功，其他为失败。
代码示例	[device startUpgrade:enumType listener:^(GizOTAEventType type, NSError * _Nonnull result) { // 处理回调的事件，提示用户 }];	

5. GizWifiCentralControlDevice 类

5.1. 简介

GizWifiCentralControlDevice 类为 APP 开发者提供中控子设备操作，包括获取子设备列表、添加子设备、删除子设备等功能。

该类继承自 GizWifiDevice 类，除下列属性和方法外，也具备 GizWifiDevice 类的所有属性和方法。

中控子设备用 GizWifiDevice 对象表示。开发者得到中控设备的子设备列表时，应使用 GizWifiDevice 类处理。

5.2. 属性访问

属性	描述
subDeviceList	NSArray 类型，GizWifiDevice 对象数组，只读。中控子设备列表

5.3. 回调接口

以下是 GizWifiCentralControlDevice 类提供的所有回调接口，将在后续 API 定义中详细介绍：

- didUpdateSubDevices：中控子设备列表回调

【didUpdateSubDevices】

定义	- (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList;	
功能描述	子设备列表回调接口。添加、删除、同步更新子设备列表以及子设备列表变化上报都使用该回调接口。	
回调参数	device	触发回调的 GizWifiCentralControlDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 subDeviceList 为中控当前的子设备列表，其他为失败，此时 subDeviceList 大小为 0。 子设备列表主动上报时该参数为 GIZ_SDK_SUCCESS，子设备添加、删除、同步更新时该参数是 GIZ_SDK_SUCCESS 或其他错误码
	subDeviceList	子设备列表。GizWifiDevice 对象数组
代码示例	// mDevice为从设备列表中取到的中控设备对象 mDevice.delegate = self; // 实现回调 - (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device	

	result:(NSError*)result subDeviceList:(NSArray *)subDeviceList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的子设备列表 } else { // 失败处理 } }

5.4. API

【addSubDevice】

定义	- (void)addSubDevice:(NSArray*)deviceMacs;	
功能描述	添加子设备。该接口让中控处于组网模式，等待子设备入网。只有中控设备可控后才能执行此操作。该接口会向中控设备发送添加子设备请求，中控设备将添加后的子设备列表通过回调返回	
参数	deviceMacs	要添加的子设备 mac 地址数组，NSString 数组，默认为 nil。默认时中控添加所有能够加入中控的子设备，若指定 mac 地址则中控只添加这些指定的子设备
代码示例	// mDevice为从设备列表中取到的中控设备对象 mDevice.delegate = self; [mDevice addSubDevice:nil]; // 实现回调 - (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList { if(result.code == GIZ_SDK_SUCCESS) { // 添加成功 } else { // 添加失败 } }	

【deleteSubDevice】

定义	- (void)deleteSubDevice:(GizWifiDevice *)device;	
功能描述	删除子设备，只有中控设备可控后才能执行此操作。该接口会向中控设备发送删除子设备请求，中控设备将删除后的子设备列表通过回调返回	
参数	device	待删除的子设备对象。在中控设备的子设备列表中找到子设备，设备对

		象传入该参数。此参数不能为 nil
代码示例		// mDevice为从设备列表中取到的中控设备对象 mDevice.delegate = self; // mSubDevice是从子设备列表中获取到的要删除的设备实体对象 [mDevice deleteSubDevice: mSubDevice]; // 实现回调 - (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList { if(result.code == GIZ_SDK_SUCCESS) { // 删除成功 } else { // 删除失败 } }

【updateSubDevices】

定义	- (void)updateSubDevices;
功能描述	同步更新子设备列表。只有中控设备可控后才能执行此操作。该接口会向中控设备发送获取子设备列表请求，中控设备将子设备列表通过回调返回
代码示例	// mDevice为从设备列表中取到的中控设备对象 mDevice.delegate = self; [mDevice updateSubDevices]; // 实现回调 - (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList { if(result.code == GIZ_SDK_SUCCESS) { // 获取成功 } else { // 获取失败 } }

6. GizUserInfo 类

6.1. 简介

GizUserInfo 类为开发者提供用户信息存取属性。

6.2. 属性

属性	描述
uid	NSString 类型。用户登录后得到的 uid，提供 get 方法
username	NSString 类型。用户名：手机号或者邮箱，只读不可写
email	NSString 类型。用户邮箱，只读不可写
phone	NSString 类型。用户手机号，只读不可写
isAnonymous	BOOL 类型。是否为匿名用户，只读不可写
lang	NSString 类型。用户的语言环境，只读不可写
name	NSString 类型。用户昵称，可写
userGender	GizUserGenderType 类型。用户性别，可写
birthday	NSString 类型。用户生日，可写
address	NSString 类型。用户家庭住址，可写
remark	NSString 类型。用户的备注信息，可写
deviceBindTime	NSString 类型。此变量只用于表示用户绑定设备的时间

7. GizWifiSSID 类

7.1. 简介

路由的 SSID 信息类，包括 SSID 名和信号强度。

7.2. 属性

属性	描述
ssid	SSID 名。我们连接一个 Wi-Fi 热点时，可以搜索到的名字
rssi	热点对应的信号强度。取值范围 0-100

8. GizDeviceSharing 类

8.1. 简介

GizDeviceSharing 类为 APP 开发者提供设备分享功能，用户绑定设备后，其他人可以通过设备分享的方式使用设备。与设备分享的有关的用户分为四类：normal、specail、owner、guest，下面简单介绍这几类用户的权限：

normal：设备没有被分享过时，任何已绑定的用户都是 normal 用户，设备仍然可以被其他用户绑定；

special：只有第一个绑定设备的用户才可以分享设备，并成为 owner

owner：用户有 owner 后，其他用户不可以再绑定设备，只能通过分享的方式使用设备。owner 用户可以解绑所有其他已绑定用户

guest：接受分享邀请的用户是 guest 用户

8.2. 回调接口

以下是 GizDeviceSharing 类提供的所有回调接口，将在后续 API 定义中详细介绍：

- didGetBindingUsers：获取设备已绑定用户的回调
- didUnbindUser：解绑设备已绑用户的回调
- didGetDeviceSharingInfos：获取分享邀请列表的回调
- didSharingDevice：创建分享邀请的回调
- didRevokeDeviceSharing：撤回分享邀请的回调
- didAcceptDeviceSharing：接受分享邀请的回调
- didCheckDeviceSharingInfoByQRCode：查看二维码邀请信息的回调
- didAcceptDeviceSharingByQRCode：扫码接受分享邀请的回调
- didModifySharingInfo：修改分享别名的回调
- didQueryMessageList：查询消息列表的回调
- didMarkMessageStatus：标记或删除消息的回调

8.3. API

【setDelegate】

定义	+ (void)setDelegate:(id <GizDeviceSharingDelegate>)delegate;	
功能描述	设置设备分享委托	
参数	delegate	设备分享的委托
代码示例	[GizDeviceSharing setDelegate:self];	

【getBindingUsers】

定义	+ (void)getBindingUsers:(NSString*)token deviceID:(NSString*)deviceID;	
功能描述	查询设备的已绑定用户列表。只有 owner 用户才能查询设备的已绑用户	
参数	token	用户 token
	deviceID	要查询的设备 did
回调	- (void)didGetBindingUsers:(NSError*)result deviceID:(NSString*)deviceID bindUsers:(NSArray*)bindUsers	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时，bindUsers 回调参数为 nil
	deviceID	发起查询的设备 ID
	bindUsers	NSString 类型数组，设备的已绑定用户列表。失败时为 nil
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 查询设备的已绑定用户列表 [GizDeviceSharing getBindingUsers:@"your_token" deviceID:@"your_device_id"]; // 实现回调 - (void)didGetBindingUsers:(NSError*)result deviceID:(NSString*)deviceID bindUsers:(NSArray*)bindUsers { if(result.code == GIZ_SDK_SUCCESS) { // 查询成功 } else { // 查询失败 } }	

【unbindUser】

定义	+ (void)unbindUser:(NSString*)token deviceID:(NSString*)deviceID guestUID:(NSString*)guestUID;	
功能描述	解绑设备的已绑定用户。只有 owner 才能解绑其他已绑用户	
参数	token	用户 token
	deviceID	要解绑用户的设备 ID

	guestUID	要解绑的用户 ID
回调	- (void)didUnbindUser:(NSError*)result deviceID:(NSString *)deviceID guestUID:(NSString*)guestUID	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	deviceID	解绑用户的设备 ID
	guestUID	解绑的用户 ID
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 解绑其他用户 [GizDeviceSharing unbindUser:@"your_token" deviceID: @"your_device_id" guestUID:@"guest_uid_to_unbind"]; // 实现回调 - (void)didUnbindUser:(NSError*)result deviceID:(NSString *)deviceID guestUID:(NSString*)guestUID { if(result.code == GIZ_SDK_SUCCESS) { // 解绑成功 } else { // 解绑失败 } }	

【sharingOwnerTransfer】

定义	+ (void)sharingOwnerTransfer:(NSString*)token deviceID:(NSString*)deviceID guestUID:(NSString*)guestUID;	
功能描述	设备分享权限转移。只有 owner 才能转移分享权限	
参数	token	用户 token
	deviceID	要转移权限的设备 ID
	guestUID	guest 用户 ID。owner 把分享权限转移给 guest 用户
回调	- (void)didSharingOwnerTransfer:(NSError * _Nonnull)result deviceID:(NSString * _Nullable)deviceID guestUID:(NSString * _Nullable)guestUID	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

	deviceID	转移权限的设备 ID
	guestUID	接收权限的 guest 用户 ID
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 把权限转移给 guest 用户 [GizDeviceSharing sharingOwnerTransfer:@"your_token" deviceID: @"your_device_id" guestUID:@"guest_uid_to_transfer"]; // 实现回调 - (void)didSharingOwnerTransfer:(NSError*)result deviceID:(NSString *)deviceID guestUID:(NSString*)guestUID { if(result.code == GIZ_SDK_SUCCESS) { // 权限转移成功 } else { // 权限转移失败 } }	

【getDeviceSharingInfos】

定义	+ (void)getDeviceSharingInfos:(NSString*)token sharingType:(GizDeviceSharingType)sharingType deviceID:(NSString*)deviceID;	
功能描述	查询设备的分享邀请列表。可以查询自己发起的分享邀请，或者查询分享给自己的分享邀请，owner 和 guest 用户都可以查询	
参数	token	用户 token
	sharingType	要查询的分享邀请类型是分享给自己的还是自己分享给别人的，见枚举定义 GizDeviceSharingType
	deviceID	查询分享邀请的设备 ID
回调	- (void)didGetDeviceSharingInfos:(NSError*)result deviceID:(NSString*)deviceID deviceSharingInfos:(NSArray*)deviceSharingInfos	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	deviceID	查询分享邀请的设备 ID
	deviceSharingInfos	GizDeviceSharingInfo 类对象数组，分享邀请列表。如果失败，此参数为 nil

代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 查询自己发出的分享邀请 [GizDeviceSharing getDeviceSharingInfos:@"your_token" sharingType: GizDeviceSharingByMe deviceID: @"your_device_id"]; // 实现回调 - (void)didGetDeviceSharingInfos:(NSError*)result deviceID:(NSString*)deviceID deviceSharingInfos:(NSArray*)deviceSharingInfos { if(result.code == GIZ_SDK_SUCCESS) { // 查询成功 } else { // 查询失败 } }

【sharingDevice】

定义	+ (void)sharingDevice:(NSString*)token deviceID:(NSString*)deviceID sharingWay:(GizDeviceSharingWay)sharingWay guestUser:(NSString*)guestUser guestUserType:(GizUserAccountType)guestUserType;	
功能描述	创建分享邀请。special 和 owner 用户可以通过账号分享或二维码分享的方式分享设备。账号分享邀请 24 小时后失效，二维码邀请 15 分钟后失效	
参数	token	用户 token
	deviceID	创建分享邀请的设备 ID
	sharingWay	分享邀请是通过账号分享还是二维码分享，见 GizDeviceSharingWay 枚举定义
	guestUser	如果是账号分享，要指定用户名，用户名可以是普通用户名、手机号、邮箱、用户的 uid。如果是二维码分享，该参数可传 nil
	guestUserType	账号分享时，该参数需要指定用户名是哪种类型，见 GizUserAccountType 枚举定义。如果是通过用户的 uid 分享的，此变量应为 GizUserOther，其他按照对应的用户类型传值
回调	- (void)didSharingDevice:(NSError*)result deviceID:(NSString*)deviceID sharingID:(NSInteger)sharingID QRCodeImage:(UIImage*)QRCodeImage	

回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	deviceID	创建分享邀请的设备 ID
	sharingID	分享邀请创建成功时被分配的 ID。失败时该参数为 nil
	QRCodeImage	二维码图片内容。二维码邀请创建失败或者账号分享时，该参数为 nil
代码示例		

```
// 设置设备分享的委托
[GizDeviceSharing setDelegate:self];

// 通过手机号分享设备
[GizDeviceSharing sharingDevice:@"your_token" deviceID:
@"your_device_id" sharingWay:GizDeviceSharingByNormal
guestUser:@"guest_phone_number" guestUserType:GizUserPhone];

// 实现回调
- (void)didSharingDevice:(NSError*)result
deviceID:(NSString*)deviceID sharingID:(NSInteger)sharingID
QRCodeImage:(UIImage*)QRCodeImage {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 分享邀请创建成功
    } else {
        // 创建失败
    }
}
```

【revokeDeviceSharing】

定义	+ (void)revokeDeviceSharing:(NSString*)token sharingID:(NSInteger)sharingID;	
功能描述	撤回分享邀请。只有 owner 才能撤回自己的分享邀请，已经发出的分享邀请，可以随时撤回。一旦撤回成功，guest 用户会被解绑不能使用该设备	
参数	token	用户 token
	sharingID	要撤回的分享邀请 ID
回调	- (void)didRevokeDeviceSharing:(NSError*)result sharingID:(NSInteger)sharingID	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sharingID	撤回的分享邀请 ID

代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 撤回分享邀请 [GizDeviceSharing revokeDeviceSharing:@"your_token" sharingID: your_sharing_id]; // 实现回调 - (void)didRevokeDeviceSharing:(NSError*)result sharingID:(NSInteger)sharingID { if(result.code == GIZ_SDK_SUCCESS) { // 撤回成功 } else { // 撤回失败 } }

【acceptDeviceSharing】

定义	+ (void)acceptDeviceSharing:(NSString*)token sharingID:(NSInteger)sharingID accept:(BOOL)accept;	
功能描述	接受分享邀请。owner 用户以账号方式分享设备后，guest 账号可以接受或拒绝邀请	
参数	token	用户 token
	sharingID	要接受的分享邀请 ID
	accept	接受或拒绝邀请。YES 表示接受，NO 表示拒绝
回调	- (void)didAcceptDeviceSharing:(NSError*)result sharingID:(NSInteger)sharingID	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sharingID	接受或拒绝的邀请 ID
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 接受邀请 [GizDeviceSharing acceptDeviceSharing:@"your_token" sharingID: your_sharing_id accept:YES]; // 实现回调	

	- (void)didAcceptDeviceSharing:(NSError*)result sharingID:(NSInteger)sharingID { if(result.code == GIZ_SDK_SUCCESS) { // 接受成功 } else { // 接受失败 } }

【checkDeviceSharingInfoByQRCode】

定义	+ (void)checkDeviceSharingInfoByQRCode:(NSString*)token QRCode:(NSString*)QRCode;	
功能描述	查看二维码邀请信息。owner 用户不能查看二维码邀请信息	
参数	token	用户 token
	QRCode	二维码邀请内容。App 扫描邀请二维码时，按照以下格式解析出 type 和 code 内容：type=share&code=xxxxxxxxxx。把解析出来的 code 内容传入此参数
回调	- (void)didCheckDeviceSharingInfoByQRCode:(NSError*)result userName:(NSString*)userName productName:(NSString*)productName deviceAlias:(NSString*)deviceAlias expiredAt:(NSString*)expiredAt	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	userName	创建分享邀请的 owner 用户名
	productName	设备的产品名称
	deviceAlias	设备的别名
	expiredAt	分享邀请的过期时间
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 查看扫码邀请信息 [GizDeviceSharing checkDeviceSharingInfoByQRCode:@"your_token" QRCode:@"your_sharing_code"]; // 实现回调 - (void)didCheckDeviceSharingInfoByQRCode:(NSError*)result userName:(NSString*)userName productName:(NSString*)productName deviceAlias:(NSString*)deviceAlias expiredAt:(NSString*)expiredAt {	

	if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }

【acceptDeviceSharingByQRCode】

定义	+ (void)acceptDeviceSharingByQRCode:(NSString*)token QRCode:(NSString*)QRCode;	
功能描述	接受二维码分享邀请。owner 用户不能调用此接口	
参数	token	用户 token
	QRCode	二维码邀请内容。App 扫描邀请二维码时，按照以下格式解析出 type 和 code 内容：type=share&code=xxxxxxxxxx。把解析出来的 code 内容传入此参数
回调	- (void)didAcceptDeviceSharingByQRCode:(NSError*)result	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 接受二维码分享邀请 [GizDeviceSharing acceptDeviceSharingByQRCode:@"your_token" QRCode:@"your_sharing_code"]; // 实现回调 - (void)didAcceptDeviceSharingByQRCode:(NSError*)result { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }	

【modifySharingInfo】

定义	+ (void)modifySharingInfo:(NSString*)token sharingID:(NSInteger)sharingID sharingAlias:(NSString*)sharingAlias;

功能描述	修改分享邀请别名	
参数	token	用户 token
	sharingID	要修改的分享邀请 ID
	sharingAlias	要修改的分享邀请别名
回调	- (void)didModifySharingInfo:(NSError*)result sharingID:(NSInteger)sharingID	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sharingID	修改别名的分享邀请 ID
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 修改分享邀请别名 [GizDeviceSharing modifySharingInfo:@"your_token" sharingID:your_sharing_id sharingAlias:@"your_sharing_alias"]; // 实现回调 - (void)didModifySharingInfo:(NSError*)result sharingID:(NSInteger)sharingID { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }	

【queryMessageList】

定义	+ (void)queryMessageList:(NSString*)token messageType:(GizMessageType)messageType;	
功能描述	查询消息列表。可查询分享消息	
参数	token	用户 token
	messageType	要查询的消息类型，见 GizMessageType 枚举定义
回调	- (void)didQueryMessageList:(NSError*)result messageList:(NSArray*)messageList	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

	messageList	查询的消息列表
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 查询消息列表 [GizDeviceSharing queryMessageList:@"your_token" messageType: GizMessageSharing]; // 实现回调 - (void)didQueryMessageList:(NSError*)result messageList:(NSArray*)messageList { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }	

【markMessageStatus】

定义	+ (void)markMessageStatus:(NSString*)token messageID:(NSString*)messageID messageStatus:(GizMessageStatus)messageStatus;	
功能描述	标记消息已读或删除	
参数	token	用户 token
	messageID	要标记或删除的消息 ID
	messageStatus	标记为已读或者删除，见 GizMessageStatus 枚举定义
回调	- (void)didMarkMessageStatus:(NSError*)result messageID:(NSString*)messageID	
回调参数	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	messageID	标记已读或删除的消息 ID
代码示例	// 设置设备分享的委托 [GizDeviceSharing setDelegate:self]; // 标记已读 [GizDeviceSharing markMessageStatus:@"your_token" messageID : @"your_message_id" messageType: GizMessageRead];	

```
// 实现回调
- (void)didMarkMessageStatus:(NSError*)result
messageID:(NSString*)messageID {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 成功
    } else {
        // 失败
    }
}
```

9. GizDeviceSharingInfo 类

9.1. 简介

GizDeviceSharingInfo 类是设备分享信息类。

9.2. 属性

属性	描述
id	NSInteger 类型，只读不可写。设备分享 ID，设备分享创建成功时会被分配一个 ID
deviceID	NSString 类型，只读不可写。设备 ID
productName	NSString 类型，只读不可写。设备的产品名称
deviceAlias	NSString 类型，只读不可写。设备别名
userInfo	GizUserInfo 类对象，只读不可写。这条分享邀请的账号信息，分享者或者被分享者的账号信息
alias	NSString 类型，只读不可写。这条分享邀请的别名
way	GizDeviceSharingWay 枚举类型，只读不可写。分享邀请是账号分享还是二维码分享
status	GizDeviceSharingStatus 枚举类型，只读不可写。分享邀请的状态，是被接受还是被拒绝的，或者还未接受
createdAt	NSString 类型，只读不可写。分享邀请的创建时间
updatedAt	NSString 类型，只读不可写。分享邀请的更新时间
expiredAt	NSString 类型，只读不可写。分享邀请的超时时间

10. GizMessage 类

10.1. 简介

GizMessage 类是机智云消息类。

10.2. 属性

属性	描述
id	NSString 类型，只读不可写。消息 ID
type	GizMessageType 枚举类型，只读不可写。消息类型，是系统消息还是分享消息
status	GizMessageStatus 枚举类型，只读不可写。消息状态，是否是已读、未读或已删除消息
createdAt	NSString 类型，只读不可写。消息生成时间
updatedAt	NSString 类型，只读不可写。消息更新时间
content	NSString 类型，只读不可写。消息内容

11. GizDeviceOTA 类

11.1. 简介

GizDeviceOTA 类提供设备固件升级功能。可升级设备的 wifi 模组固件以及 mcu 固件。

11.2. 属性访问

以下是 GizDeviceOTA 类提供的所有属性变量：

属性	描述
delegate	GizDeviceOTADelegate 委托

11.3. 回调接口

以下是 GizDeviceOTA 类提供的所有回调接口：

- didCheckDeviceUpdate：检查固件更新的回调
- didUpgradeDevice：固件开始升级的回调
- didNotifyDeviceUpdate：设备固件有更新的主动通知
- didNotifyDeviceUpgradeStatus：固件升级状态的主动通知

【didCheckDeviceUpdate】

回调	- (void)didCheckDeviceUpdate:(GizWifiDevice *)device result:(NSError*)result wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion;	
回调说明	检查设备更新的回调，调用检查更新接口checkDeviceUpdate时触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	result	接口执行结果，见 GizWifiErrorCode 定义。GIZ_SDK_SUCCESS 表示成功，其他为失败。失败时参数 wifiVersion 和 mcuVersion 值为 nil
	wifiVersion	模组固件版本，字典格式：{"latest":"xxx","current":"xxx"}。若此参数为 nil，表示没有检查到模组固件更新信息
	mcuVersion	mcu 固件版本，字典格式：{"latest":"xxx","current":"xxx"}。若此参数为 nil，表示没有检查到 mcu 固件更新信息
代码示例	GizDeviceOTA.delegate = self; // 实现回调 - (void)didCheckDeviceUpdate:(GizWifiDevice *)device result:(NSError*)result wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion { if(result.code == GIZ_SDK_SUCCESS) { // 成功处理，取最新版本号 } else { // 失败处理 } }	

【didUpgradeDevice】

回调	- (void)didUpgradeDevice:(GizWifiDevice*)device result:(NSError*)result firmwareType:(GizOTAFirmwareType)firmwareType;	
回调说明	设备开始升级的回调，调用开始升级接口upgradeDevice时触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	result	接口执行结果，见 GizWifiErrorCode 定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	firmwareType	正在升级的固件类型

代码示例	GizDeviceOTA.delegate = self; // 实现回调 - (void)didUpgradeDevice:(GizWifiDevice*)device result:(NSError*)result firmwareType:(GizOTAFirmwareType)firmwareType { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }

【didNotifyDeviceUpdate】

回调	- (void)didNotifyDeviceUpdate:(GizWifiDevice*)device wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion;	
回调说明	设备固件有更新的主动通知。设备固件有新版本时触发该回调	
回调参数	device	回调的 GizWifiDevice 对象
	wifiVersion	模组固件版本，字典格式：{"latest":"xxx","current":"xxx"}。 若此参数为 nil，表示没有检查到模组固件更新信息
	mcuVersion	mcu 固件版本，字典格式：{"latest":"xxx","current":"xxx"}。 若此参数为 nil，表示没有检查到 mcu 固件更新信息
代码示例	GizDeviceOTA.delegate = self; // 实现回调 - (void)didNotifyDeviceUpdate:(GizWifiDevice*)device wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion { }	

【didNotifyDeviceUpgradeStatus】

回调	- (void)didNotifyDeviceUpgradeStatus:(GizWifiDevice*)device firmwareType:(GizOTAFirmwareType)firmwareType upgradeStatus:(NSError*)upgradeStatus;
回调说明	设备升级状态的主动通知。设备在升级过程中会主动上报升级状态，此时会触发该回调

回调参数	device	回调的 GizWifiDevice 对象
	firmwareType	正在升级的固件类型
	upgradeStatus	设备升级状态，见 GizWifiErrorCode 定义中枚举值范围[8350, 8360]
代码示例	GizDeviceOTA.delegate = self; // 实现回调 - (void)didNotifyDeviceUpdate:(GizWifiDevice*)device wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion { }	

11.4. API

【checkDeviceUpdate】

定义	+ (void)checkDeviceUpdate:(NSString*)uid token:(NSString*)token device:(GizWifiDevice*)device;	
功能描述	检查固件是否有更新	
参数	uid	用户 uid
	token	用户 token
	device	待检查固件版本的设备
代码示例	// 设置 OTA 委托 GizDeviceOTA.delegate = self; // 检查固件版本是否有更新。mDevice 为从设备列表中取到的待升级的设备 [GizDeviceOTA checkDeviceUpdate:@"your_uid" token: @"your_token" device:mDevice]; // 实现回调 - (void)didCheckDeviceUpdate:(GizWifiDevice *)device result:(NSError*)result wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion { if(result.code == GIZ_SDK_SUCCESS) { // 成功，比较固件版本号是否有更新 } else { // 失败	

	} }

【upgradeDevice】

定义	+ (void)upgradeDevice:(NSString *)uid token:(NSString*)token device:(GizWifiDevice*)device firmwareType:(GizOTAFirmwareType)firmwareType;	
功能描述	开始升级	
参数	uid	用户 uid
	token	用户 token
	device	要升级的设备
	firmwareType	要升级的固件类型，见 GizOTAFirmwareType 枚举定义
代码示例	// 设置 OTA 委托 [GizDeviceOTA setDelegate:self]; // 开始升级。mDevice 为刚检查过版本信息待升级的设备 [GizDeviceOTA upgradeDevice:@"your_uid" token: @"your_token" device:mDevice firmwareType: GizOTAFirmareModule]; // 实现回调 - (void)didCheckDeviceUpdate:(GizWifiDevice *)device result:(NSError*)result wifiVersion:(NSDictionary*)wifiVersion mcuVersion:(NSDictionary*)mcuVersion { if(result.code == GIZ_SDK_SUCCESS) { // 成功 } else { // 失败 } }	

12. GizDeviceGroupCenter 类

12.1. 简介

GizDeviceGroupCenter 类为 APP 开发者提供中控子设备分组操作，包括创建设备分组、删除、更新设备分组列表等功能。

12.2. 属性访问

以下是 GizDeviceGroupCenter 类提供的所有属性变量访问：

属性	描述
delegate	GizDeviceGroupCenterDelegate 委托

【getGroupListGateway】

定义	+ (NSArray*)getGroupListGateway:(GizWifiDevice*)groupOwner;	
功能描述	获取 SDK 本地缓存的指定网关设备上的分组列表	
参数	groupOwner	中控设备对象，此参数不能填 nil
返回值	组列表，GizDeviceGroup 对象数组	
代码示例	// mDevice是在设备列表中得到的网关设备对象， NSArray* groupList = [GizDeviceGroupCenter getGroupListGateway: mDevice];	

12.3. 回调接口

以下是 GizDeviceGroupCenter 类提供的所有回调接口：

- didUpdateSubDevices：中控子设备列表回调

【didUpdateGroups】

定义	- (void)didUpdateGroups:(GizWifiDevice*)groupOwner result:(NSError*)result groupList:(NSArray*)groupList;	
功能描述	组列表回调接口。调用添加组接口 addGroup、删除组接口 removeGroup、同步更新组列表接口 updateGroups、组列表变化上报时触发该回调	
回调参数	groupOwner	触发回调的 GizWifiCentralControlDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 subDeviceList 为中控当前的子设备列表，其他为失败，此时 subDeviceList 大小为 0。 子设备列表主动上报时该参数为 GIZ_SDK_SUCCESS，子设备添加、删除、同步更新时该参数是 GIZ_SDK_SUCCESS 或其他错误码
	groupList	组列表。GizDeviceGroup 对象数组
代码示例	// mDevice为从设备列表中取到的中控设备对象 mDevice.delegate = self;	

```
// 实现回调
- (void)didUpdateGroups:(GizWifiDevice*)groupOwner
result:(NSError*)result groupList:(NSArray*)groupList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的组列表
    } else {
        // 失败处理
    }
}
```

12.4. API

【addGroup】

定义	+ (void)addGroup:(GizWifiDevice*)groupOwner groupType:(NSString*)groupType groupName:(NSString*)groupName groupDevices:(NSArray*)groupDevices;	
功能描述	添加分组。添加成功后会被分配一个组 ID，同时返回最新的分组列表，添加失败时返回错误信息	
参数	groupOwner	管理分组的设备，此参数必填，填 nil 或无效无法添加分组
	groupType	分组类型，即设备的产品唯一标识 productKey。此参数必填，填 nil 或无效无法添加分组
	groupName	组名称。此参数可选填，App 可以在成功创建组以后再修改组名称
	groupDevices	组设备列表，是 GizWifiDevice 对象数组。此参数可选填， App 可以在添加组以后再添加组设备

代码示例

```
// mOwner为中控设备，mDevice为中控子设备列表中要加入到分组中的设备对象
GizDeviceGroupCenter.delegate = self;
NSArray *list= [[NSArray alloc]initWithObjects:mdevice,nil];
[GizDeviceGroupCenter addGroup:mOwner groupType:@"your_product_key"
groupName:@ "your_group_name" groupDevices:list];

// 实现回调
- (void)didUpdateGroups:(GizWifiDevice*)groupOwner
result:(NSError*)result groupList:(NSArray*)groupList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的组列表
    } else {
        // 失败处理
```

	} }

【removeGroup】

定义	+ (void)removeGroup:(GizWifiDevice*)groupOwner group:(GizDeviceGroup*)group;	
功能描述	删除设备分组。删除成功时返回最新的组列表，删除失败时返回错误信息	
参数	groupOwner	管理分组的设备，此参数必填，填 nil 或无效无法删除分组
	group	待删除的组。此参数不能填 nil
代码示例	// mOwner为中控设备，mGroup为从组列表中取到的组对象 GizDeviceGroupCenter.delegate = self; [GizDeviceGroupCenter removeGroup:mOwner group:mGroup]; // 实现回调 - (void)didUpdateGroups:(GizWifiDevice*)groupOwner result:(NSError*)result groupList:(NSArray*)groupList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组列表 } else { // 失败处理 } }	

【updateGroups】

定义	+ (void)updateGroups:(GizWifiDevice*)groupOwner;	
功能描述	更新分组列表。更新成功时返回最新的组列表，更新失败时返回错误信息	
参数	groupOwner	管理分组的设备，此参数必填，填 nil 或无效无法更新分组列表
代码示例	// mOwner为中控设备 GizDeviceGroupCenter.delegate = self; [GizDeviceGroupCenter updateGroups:mOwner]; // 实现回调 - (void)didUpdateGroups:(GizWifiDevice*)groupOwner result:(NSError*)result groupList:(NSArray*)groupList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组列表 } else {	

```
        // 失败处理
    }
}
```

13. GizDeviceGroup 类

13.1. 简介

GizDeviceGroup 类提供设备分组控制、编辑组设备、修改组信息等功能。

13.2. 属性访问

以下是 GizDeviceGroup 类提供的所有属性变量:

属性	描述
delegate	使用委托获取对应事件。GizDeviceGroup 对应的回调接口在 GizDeviceGroupDelegate 定义，需要用到哪个接口，实现相应的回调即可。此变量可写
groupID	NSString 类型。组 ID，是 groupOwner 创建设备分组时分配的唯一标识。此变量只读
groupOwner	GizWifiDevice 类型。管理组的设备，用来创建、删除、维护组信息。目前 groupOwner 只支持中控设备。此变量只读
groupType	NSString 类型。组类型，即设备的 productKey。由于组是由同类型设备组成，组类型就是设备的产品类别唯一标识。此变量只读
groupName	NSString 类型。组名称。此变量只读
groupDeviceList	NSArray 类型，GizWifiDevice 对象数组。这是组设备列表，缓存了添加到组里的设备。此变量只读

13.3. 回调接口

以下是 GizDeviceGroupCenter 类提供的所有回调接口,将在后续 API 定义中详细介绍:

- didUpdateGroupInfo：组信息更新回调
- didUpdateGroupDevices：组设备列表更新回调
- didWrite：组操作回调

【didUpdateGroupInfo】

定义	- (void)group:(GizDeviceGroup*)group didUpdateGroupInfo:(NSError*)result;

功能描述	分组信息更新回调。调用修改组信息接口 editGroupInfo、组信息变化上报时触发该回调	
回调参数	group	触发回调的组对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	// mGroup为从组列表中取到的分组对象 mGroup.delegate = self; // 实现回调 - (void)group:(GizDeviceGroup*)group didUpdateGroupInfo:(NSError*)result { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组信息 } else { // 失败处理 } }	

【didUpdateGroupDevices】

定义	- (void)group:(GizDeviceGroup*)group didUpdateGroupDevices:(NSError*)result groupDeviceList:(NSArray*)groupDeviceList;	
功能描述	组设备列表更新回调。调用添加组设备接口 addGroupDevice、删除组设备接口 removeGroupDevice、更新组设备接口 updateGroupDevices、组设备变化上报时触发该回调	
回调参数	group	触发回调的组对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
代码示例	// mGroup为从组列表中取到的分组对象 mGroup.delegate = self; // 实现回调 - (void)group:(GizDeviceGroup*)group didUpdateGroupInfo:(NSError*)result { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组列表 } else { // 失败处理 }	

	}

【didWrite】

定义	- (void)group:(GizDeviceGroup*)group didWrite:(NSError*)result sn:(int)sn;	
功能描述	分组控制的回调接口。调用分组控制接口 write 时触发该回调	
回调参数	group	执行控制命令的组对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sn	控制命令的序号
代码示例	// mGroup为从组列表中取到的组对象 mGroup.delegate = self; // 实现回调 - (void)group:(GizDeviceGroup*)group didWrite:(NSError*)result sn:(int)sn { if(result.code == GIZ_SDK_SUCCESS) { // 组操作成功 } else { // 失败处理 } }	

13.4. API

【editGroupInfo】

定义	- (void)editGroupInfo:(NSString*)groupName;	
功能描述	修改设备分组信息。修改成功时返回最新的组信息，修改失败时返回错误信息	
参数	groupName	待修改的组名称。此参数不能填 nil
代码示例	// mGroup为从组列表中取到的组对象 mGroup.delegate = self; [mGroup editGroupInfo: @"your_group_name"]; // 实现回调 - (void)group:(GizDeviceGroup*)group	

	didUpdateGroupInfo:(NSError*)result { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组信息 } else { // 失败处理 } }

【addGroupDevice】

定义	- (void)addGroupDevice:(NSArray*)groupDevices;	
功能描述	添加组设备。添加成功时返回最新的组设备列表，添加失败时返回错误信息	
参数	groupDevices	待添加的组设备，为 GizWifiDevice 对象数组。此参数不能填 nil 或空数组
代码示例	// mGroup为从组列表中取到的组对象, mDevice为从中控设备子列表中取到的设备对象 mGroup.delegate = self; NSArray *list= [[NSArray alloc]initWithObjects:mdevice,nil]; [mGroup addGroupDevice:list]; // 实现回调 - (void)group:(GizDeviceGroup*)group didUpdateGroupDevices:(NSError*)result groupDeviceList:(NSArray*)groupDeviceList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组设备列表 } else { // 失败处理 } }	

【removeGroupDevice】

定义	- (void)removeGroupDevice:(NSArray*)groupDevices;	
功能描述	删除设备分组。删除成功时返回最新的组列表，删除失败时返回错误信息	
参数	groupDevices	待删除的组设备，为 GizWifiDevice 对象数组。此参数不能填 nil 或空数组
代码示例	// mGroup为从组列表中取到的组对象, mDevice为从组设备列表中取到的设备对象 mGroup.delegate = self; NSArray *list= [[NSArray alloc]initWithObjects:mdevice,nil];	

```
[mGroup removeGroupDevice:list];

// 实现回调
- (void)group:(GizDeviceGroup*)group
didUpdateGroupDevices:(NSError*)result groupDeviceList:(NSArray
*)groupDeviceList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的组设备列表
    } else {
        // 失败处理
    }
}
```

【updateGroupDevices】

定义	+ (void) updateGroupDevices;
功能描述	更新分组列表。更新成功时返回最新的组列表，更新失败时返回错误信息
代码示例	// mGroup为从组列表中取到的组对象 mGroup.delegate = self; [mGroup updateGroupDevices]; // 实现回调 - (void)didUpdateGroupDevices:(GizWifiDevice*)groupOwner result:(NSError*)result groupList:(NSArray*)groupList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的组设备列表 } else { // 失败处理 } }

【write】

定义	- (void)write:(NSDictionary*)data sn:(int)sn;	
功能描述	执行组操作	
参数	data	待执行的组操作，数据点键值对字典，键值对为{数据点名称：数据点值}。数据点名称和值的类型要符合设备的数据点定义，透传数据的数据点名称要填"binary"，数据点值的类型要用NSData，不符合格式的数据点参数可能无法下发。此参数不能填 nil 或空字典

	sn	控制命令序号。用于对应控制命令应答数据，控制确认回调时会返回这个 sn。如果 App 需要对应控制命令的执行顺序，sn 就要指定为一个正整数。如果 App 不关心操作执行顺序，sn 填 0。负数默认按照 0 处理

代码示例

```
// mGroup为从组列表中取到的组对象
mGroup.delegate = self;
[mGroup write:@{@"LED_OnOff": @(YES)} sn:@0];

// 实现回调
- (void)group:(GizDeviceGroup*)group didWrite:(NSError*)result
sn:(int)sn {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 组操作成功
    } else {
        // 失败处理
    }
}
```

14. GizDeviceSceneCenter 类

14.1. 简介

GizDeviceSceneCenter 类为 APP 开发者提供中控子设备分组操作，包括创建设备分组、删除、更新设备分组列表等功能。

14.2. 属性访问

以下是 GizDeviceSceneCenter 类提供的属性变量：

属性	描述
delegate	GizDeviceSceneCenterDelegate 委托

【getSceneListGateway】

定义	+ (NSArray*)getSceneListGateway:(GizWifiDevice*)sceneOwner;	
功能描述	获取 SDK 本地缓存的指定网关设备上的场景列表	
参数	sceneOwner	中控设备对象，此参数不能填 nil
返回值	场景列表，GizDeviceScene 对象数组	

代码示例

```
// mOwner是在设备列表中得到的中控设备对象，
NSArray* sceneList = [GizDeviceSceneCenter getSceneListGateway:
```

	mOwner];

14.3. 回调接口

以下是 GizDeviceSceneCenter 类提供的所有回调接口：

- didUpdateScenes：中控场景设备列表回调

【didUpdateScenes】

定义	- (void)didUpdateScenes:(GizWifiDevice*)sceneOwner result:(NSError*)result sceneList:(NSArray*)sceneList;	
功能描述	场景列表回调接口。调用添加场景接口 addScene、删除场景接口 removeScene、同步更新场景列表接口 updateScenes、场景列表变化上报时触发该回调	
回调参数	sceneOwner	触发回调的 GizWifiCentralControlDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 subDeviceList 为中控当前的子设备列表，其他为失败，此时 subDeviceList 大小为 0。 子设备列表主动上报时该参数为 GIZ_SDK_SUCCESS，子设备添加、删除、同步更新时该参数是 GIZ_SDK_SUCCESS 或其他错误码
	sceneList	场景列表。GizDeviceScene 对象数组
代码示例	GizDeviceSceneCenter.delegate = self; // 实现回调 - (void)didUpdateScenes:(GizWifiDevice*)sceneOwner result:(NSError*)result sceneList:(NSArray*)sceneList; { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的场景列表 } else { // 失败处理 } }	

14.4. API

【addScene】

定义	+ (void)addScene:(GizWifiDevice*)sceneOwner sceneName:(NSString*)sceneName sceneItems:(NSArray*)sceneItems;
功能描述	添加场景。添加成功后会被分配一个场景 ID，此时会回调最新的场景列表，失败时回调错误信

	息	
参数	sceneOwner	管理场景的设备，此参数必填，填 nil 或无效无法添加分组
	sceneName	场景名称。此参数为选填参数，如果不指定场景名称此参数填 nil，App 可以在成功添加场景以后再修改场景名称
	sceneItems	场景项列表，是 GizDeviceSceneItem 对象数组。此参数为选填参数，如果不指定场景项此参数填 nil 或空数组，App 可以在成功添加场景以后再添加场景项
代码示例	// mOwner为中控设备，mDevice为中控子设备列表中要加入到场景项中的设备对象 GizDeviceSceneCenter.delegate = self; GizDeviceSceneItem* item = [GizDeviceSceneItem sceneItemWithDevice:mDevice data:@{@"LED_OnOff": @(YES)}]; NSArray *list= [[NSArray alloc]initWithObjects:item,nil]; [GizDeviceSceneCenter addScene: mOwner sceneName:@"your_scene_name" sceneItems:list]; // 实现回调 - (void)didUpdateScenes:(GizWifiDevice*)sceneOwner result:(NSError*)result sceneList:(NSArray*)sceneList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的场景列表 } else { // 失败处理 } }	

【removeScene】

定义	+ (void)removeScene:(GizWifiDevice*)sceneOwner scene:(GizDeviceScene*)scene;	
功能描述	删除场景。删除成功时返回最新的场景列表，删除失败时返回错误信息	
参数	sceneOwner	管理场景的设备，此参数必填，填 nil 或无效无法删除场景
	scene	待删除的场景。此参数不能填 nil
代码示例	// mOwner为中控设备，mScene为场景列表中待移除的场景对象 GizDeviceSceneCenter.delegate = self; [GizDeviceSceneCenter removeScene:mOwner scene:mScene]; // 实现回调 - (void)didUpdateScenes:(GizWifiDevice*)sceneOwner result:(NSError*)result sceneList:(NSArray*)sceneList {	

```
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的场景列表
    } else {
        // 失败处理
    }
}
```

【updateScenes】

定义	+ (void)updateScenes:(GizWifiDevice*)sceneOwner;	
功能描述	更新场景列表。更新成功时返回最新的场景列表，更新失败时返回错误信息	
参数	sceneOwner	管理场景的设备，此参数必填，填 nil 或无效无法更新场景列表
代码示例	// mOwner为中控设备 GizDeviceGroupCenter.delegate = self; [GizDeviceSceneCenter updateScenes: mOwner]; // 实现回调 - (void)didUpdateScenes:(GizWifiDevice*)sceneOwner result:(NSError*)result sceneList:(NSArray*)sceneList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的场景列表 } else { // 失败处理 } }	

15. GizDeviceScene 类

15.1. 简介

GizDeviceScene 类提供场景执行、场景编辑等功能。

15.2. 属性访问

以下是 GizDeviceScene 类提供的所有属性变量：

属性	描述
delegate	使用委托获取对应事件。GizDeviceScene 对应的回调接口在 GizDeviceSceneDelegate 定义，需要用到哪个接口，实现相应的回调即可。此变量可写

属性	描述
sceneID	NSString 类型。场景 ID，是 sceneOwner 添加场景时分配的唯一标识。此变量只读
sceneOwner	GizWifiDevice 类型。管理场景的设备，用来添加、删除、维护场景信息。目前 sceneOwner 只支持中控设备。此变量只读
sceneName	NSString 类型。场景名称。此变量只读
sceneItemList	NSArray 类型，GizWifiDevice 对象数组。这是场景项列表，缓存了添加到场景里的场景项。此变量只读

15.3. 回调接口

以下是 GizDeviceScene 类提供的所有回调接口，将在后续 API 定义中详细介绍:

- didUpdateSceneInfo：场景信息更新回调
- didUpdateSceneItems：场景项列表更新回调
- didUpdateSceneStatus：场景状态更新回调
- didExecuteScene：场景状态更新回调

【didUpdateSceneInfo】

定义	- (void)scene:(GizDeviceScene*)scene didUpdateSceneInfo:(NSError*)result;	
功能描述	场景信息更新回调。调用编辑场景信息接口 editSceneInfo、场景信息变化上报时使用该回调	
回调参数	scene	触发回调的场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败

代码示例

```
// mScene为从场景列表中取到的场景对象
mScene.delegate = self;

// 实现回调
- (void)scene:(GizDeviceScene*)scene
didUpdateSceneInfo:(NSError*)result {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的场景信息
    } else {
        // 失败处理
    }
}
```

【didUpdateSceneItems】

定义	- (void)scene:(GizDeviceScene*)scene didUpdateSceneItems:(NSError*)result sceneItemList:(NSArray*)sceneItemList;	
功能描述	场景项列表更新回调。调用编辑场景项接口 editSceneItems、场景项列表变化上报时触发该回调	
回调参数	scene	触发回调的场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功同时 sceneItemList 为最新的场景项列表，其他为失败
	sceneItemList	场景项列表，为 GizDeviceSceneItem 对象数组

代码示例

```
// mScene为从场景列表中取到的场景对象
mScene.delegate = self;

// 实现回调
- (void)scene:(GizDeviceScene*)scene
didUpdateSceneItems:(NSError*)result
sceneItemList:(NSArray*)sceneItemList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的场景项列表
    } else {
        // 失败处理
    }
}
```

【didUpdateSceneStatus】

定义	- (void)scene:(GizDeviceScene*)scene didUpdateSceneStatus:(NSError*)result status:(GizDeviceSceneStatus)status;	
功能描述	场景状态更新回调。调用接口 updateSceneStatus、场景执行状态更新上报时触发该回调	
回调参数	scene	触发回调的场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	status	场景的执行状态

代码示例

```
// mScene为从场景列表中取到的场景对象
mScene.delegate = self;
```

```
// 实现回调
- (void)scene:(GizDeviceScene*)scene
didUpdateSceneStatus:(NSError*)result
status:(GizDeviceSceneStatus)status {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收变更的场景状态
    } else {
        // 失败处理
    }
}
```

【didExecuteScene】

定义	- (void)scene:(GizDeviceScene*)scene didExecuteScene:(NSError*)result sn:(int)sn;	
功能描述	场景执行回调。调用场景执行接口 executeScene 时触发该回调	
回调参数	scene	场景对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，其他为失败
	sn	执行序号

代码示例

```
// mScene为从场景列表中取到的场景对象
mScene.delegate = self;

// 实现回调
- (void)scene:(GizDeviceScene*)scene didExecuteScene:(NSError*)result
sn:(int)sn {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收执行结果
    } else {
        // 失败处理
    }
}
```

15.4. API

【editSceneInfo】

定义	- (void)editSceneInfo:(NSString*)sceneName;
功能描述	编辑场景信息。编辑成功时返回最新的场景信息，编辑失败时返回错误信息

参数	sceneName	待修改的场景名称。此参数不能填 nil
代码示例	// mScene为从场景列表中取到的场景对象 mScene.delegate = self; [mScene editSceneInfo: @"your_scene_name"]; // 实现回调 - (void)scene:(GizDeviceScene*)scene didUpdateSceneInfo:(NSError*)result { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的场景信息 } else { // 失败处理 } }	

【editSceneItems】

定义	- (void)editSceneItems:(NSArray*)sceneItems;	
功能描述	编辑场景项列表。编辑成功时返回最新的场景项列表，失败时返回错误信息	
参数	sceneItems	待编辑的场景项列表，包含全部场景项列表，GizDeviceSceneItem 对象数组。这个列表必须能符合预期的修改结果，列表中应包括新添加的、待修改的、不修改的，如果有待删除的要移除掉。此参数不能填 nil 或空数组
代码示例	// mScene为从场景列表中取到的场景对象, mDevice为中控设备子列表中取到的设备对象 mScene.delegate = self; GizDeviceSceneItem* item = [GizDeviceSceneItem sceneItemWithDevice:mDevice data:@{@"LED_OnOff": @(YES)}]; NSArray *list= [[NSArray alloc]initWithObjects:item,nil]; [mScene editSceneItems:list]; // 实现回调 - (void)scene:(GizDeviceScene*)scene didUpdateSceneItems:(NSError*)result sceneItemList:(NSArray*)sceneItemList { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的场景项列表 } else { // 失败处理 } }	

【executeScene】

定义	- (void)executeScene:(BOOL)startup sn:(int)sn;	
功能描述	启动或取消场景执行	
参数	startup	执行或取消执行。YES 为执行，NO 为取消执行
	sn	执行序号。用于对应执行应答数据，执行确认回调时会返回这个 sn。如果 App 需要对应执行的执行顺序，sn 就要指定为一个正整数。如果 App 不关心操作执行顺序，sn 填 0。负数默认按照 0 处理
代码示例	// mScene为从场景列表中取到的场景对象 mScene.delegate = self; [mScene executeScene:@(YES) sn:@0]; // 实现回调 - (void)scene:(GizDeviceScene*)scene didExecuteScene:(NSError*)result sn:(int)sn { if(result.code == GIZ_SDK_SUCCESS) { // 接收执行结果 } else { // 失败处理 } }	

【updateSceneStatus】

定义	- (void)updateSceneStatus;
功能描述	更新场景的执行状态
代码示例	// mScene为从场景列表中取到的场景对象 mScene.delegate = self; [mScene updateSceneStatus]; // 实现回调 - (void)scene:(GizDeviceScene*)scene didUpdateSceneStatus:(NSError*)result status:(GizDeviceSceneStatus)status { if(result.code == GIZ_SDK_SUCCESS) { // 接收变更的场景状态 } else { // 失败处理 }

	}

16. GizDeviceSceneItem 类

16.1. 简介

GizDeviceSceneItem 提供场景项信息。场景项分为设备场景项、组场景项和延时场景项三种类型，设备场景项和组场景项必须要设置数据点操作，延时场景项只需要设置延时时间不需要设置数据点操作

16.2. 属性访问

以下是 GizDeviceSceneItem 类提供的所有属性变量:

属性	描述
device	NSString 类型。备场景项的中控子设备，如果 sceneItemType 是设备场景项，需要关心此变量。此变量只读
group	GizWifiDevice 类型。组场景项的中控子设备，如果 sceneItemType 是组场景项，需要关心此变量。此变量只读
data	NSDictionary 类型。设备或组要执行的操作键值对字典：{操作名字: 操作值}，如果 sceneItemType 是设备场景项或组场景项，需要关心此变量此变量只读
delayTime	延时的毫秒数，如果 sceneItemType 是延时场景项，需要关心此变量。此变量只读
sceneItemType	场景项类型，见 GizSceneItemType 枚举定义。此变量只读

16.3. API

【sceneItemWithDevice】

定义	+ (id)sceneItemWithDevice:(GizWifiDevice*)device data:(NSDictionary*)data;
功能描述	构造函数，用于创建设备场景项对象
代码示例	// mDevice为中控子设备列表中要加入到场景项中的设备对象 GizDeviceSceneItem* item = [GizDeviceSceneItem sceneItemWithDevice:mDevice data:@{@"LED_OnOff": @(YES)}];

【sceneItemWithGroup】

定义	+ (id)sceneItemWithGroup:(GizDeviceGroup*)group data:(NSDictionary*)data;
功能描述	构造函数，用于创建组场景项对象
代码示例	// mGroup为中控组列表中要加入到场景项中的组对象 GizDeviceSceneItem* item = [GizDeviceSceneItem sceneItemWithGroup:mGroup data:@{@"LED_OnOff": @(YES)}];

【sceneItemWithDelay】

定义	+ (id)sceneItemWithDelay:(unsigned int)delayTime;
功能描述	构造函数，用于创建延时场景项对象
代码示例	GizDeviceSceneItem* item = [GizDeviceSceneItem sceneItemWithDelay:10];

17. GizDeviceSchedulerCenter 类

17.1. 简介

GizDeviceSchedulerCenter 类为 APP 开发者提供设备定时任务管理功能，管理用户在设备上设置的定时任务。

17.2. 属性访问

以下是 GizDeviceSchedulerCenter 类提供的属性变量：

属性	描述
delegate	GizDeviceSchedulerCenterDelegate 委托

【getSchedulerListCloud】

定义	+ (NSArray*)getSchedulerListCloud:(GizWifiDevice*)schedulerOwner;	
功能描述	获取 SDK 本地缓存的云端定时任务列表，这个列表缓存了 GizDeviceScheduler 对象	
参数	schedulerOwner	执行定时任务的设备，此参数不能填 nil
返回值	定时任务列表。参数值为 nil 或无效设备时返回空数组	
代码示例	// schedulerOwner是在设备列表中得到的设备对象， NSArray* schedulerList = [GizDeviceSchedulerCenter	

	getSchedulerListCloud:schedulerOwner];

17.3. 回调接口

以下是 GizDeviceSchedulerCenter 类提供的所有回调接口：

- didUpdateSchedulers：定时任务列表回调

【didUpdateSchedulers】

定义	- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList;	
功能描述	定时任务列表回调接口。调用创建定时任务接口 createScheduler、删除定时任务接口 deleteScheduler、同步更新定时任务列表接口 updateSchedulers、定时任务列表变化上报时该回调	
回调参数	schedulerOwner	触发回调的 GizWifiDevice 设备对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 schedulerList 为定时任务列表；其他为失败，此时 schedulerList 大小为 0
	schedulerList	设备定时任务列表，GizDeviceSchedulerSuper 对象数组，GizDeviceSchedulerSuper 是 GizDeviceScheduler 的父类。处理云端定时任务时，App 可以用 GizDeviceScheduler 类对象处理定时任务列表
代码示例	// mDevice为从设备列表中取到的设备对象 mDevice.delegate = self; // 实现回调 - (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList { if(result.code == GIZ_SDK_SUCCESS) { // 接收上报的定时任务列表 for (GizDeviceScheduler *scheduler in schedulerList) { if ([scheduler isKindOfClass:[GizDeviceScheduler class]]) { // 处理云端定时任务 } } } else { // 失败处理 } }	

17.4. API

【createScheduler】

定义	+ (void)createScheduler:(NSString*)uid token(NSString*)token schedulerOwner:(GizWifiDevice*)schedulerOwner scheduler:(GizDeviceSchedulerSuper*)scheduler schedulerTasks:(NSArray*)schedulerTasks;	
功能描述	创建定时任务。创建成功会被分配一个定时任务 ID，同时通过回调接口 didUpdateSchedulers 给 App 返回定时任务列表，创建失败时返回错误信息	
参数	uid	用户 uid。创建云端定时任务时此参数必填，空串和无效值无法创建云端定时任务
	token	用户 token。创建云端定时任务时此参数必填，空串和无效值无法创建云端定时任务
	schedulerOwner	管理定时任务的设备，此参数必填，填 nil 无法创建定时任务
	scheduler	定时任务对象，用于填写定时任务内容的，此参数不能填 nil。 云端定时任务对象通过 GizDeviceScheduler 类的构造函数创建，GizDeviceScheduler 类提供三种构造方法，App 根据需要选择对应的方法创建对象
	schedulerTasks	创建云端定时任务时不需要填写此参数
代码示例	// 一次性定时任务，在 2017 年 1 月 16 日早上 6 点 30 分开灯 GizDeviceScheduler *scheduler = [[GizDeviceScheduler schedulerOneTime: @{@"LED_OnOff": @YES} date: @"2017-01-16" time:@"06:30" enabled:@YES remark: @"开灯任务"]; // 设置定时任务委托 [GizDeviceSchedulerCenter setDelegate:self]; // 创建云端定时任务，mDevice 为从设备列表中取到的设备对象 [GizDeviceSchedulerCenter createScheduler:@"your_uid" token:@"your_token" schedulerOwner:mDevice scheduler:scheduler schedulerTasks:nil]; // 实现回调 - (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList { if(result.code == GIZ_SDK_SUCCESS) { // 定时任务创建成功，参考 didUpdateSchedulers 接口代码示例 } else {	

```
        // 创建失败
    }
}
```

【deleteScheduler】

定义	+ (void)deleteScheduler:(NSString *)uid token:(NSString *)token schedulerOwner:(GizWifiDevice *)schedulerOwner scheduler:(GizDeviceSchedulerSuper*)scheduler;	
功能描述	删除定时任务。删除成功时通过回调接口 didUpdateSchedulers 给 App 返回定时任务列表，删除失败时返回错误信息	
参数	uid	用户 uid。删除云端定时任务时此参数必填，空串和无效值无法删除云端定时任务
	token	用户 token。删除云端定时任务时此参数必填，空串和无效值无法删除云端定时任务
	schedulerOwner	管理定时任务的设备，此参数必填，填 nil 无法删除定时任务
	scheduler	待删除的定时任务对象，是从定时任务列表中得到的 GizDeviceScheduler 对象，此参数必填，不能填 nil

代码示例

```
// 设置定时任务委托
[GizDeviceSchedulerCenter setDelegate:self];

// 删除设备的定时任务，mDevice为从设备列表中取到的要删除定时任务的设备对象
// mScheduler为从定时任务列表中得到的待删除的定时任务对象
[GizDeviceSchedulerCenter deleteScheduler:@"your_uid"
token:@"your_token" schedulerOwner:mDevice scheduler:mScheduler];

// 实现回调
- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner
result:(NSError*)result schedulerList:(NSArray*)schedulerList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 定时任务删除成功，参考didUpdateSchedulers接口代码示例
    } else {
        // 删除失败
    }
}
```

【updateSchedulers】

定义	+ (void)updateSchedulers:(NSString *)uid token:(NSString *)token

	schedulerOwner:(GizWifiDevice *)schedulerOwner;	
功能描述	同步更新设备定时任务列表。成功时通过回调接口 didUpdateSchedulers 返回同步更新结果，失败时返回错误信息	
参数	uid	用户 uid。同步更新云端定时任务时此参数必填
	token	用户 token。同步更新云端定时任务时此参数必填
	schedulerOwner	管理定时任务的设备，此参数必填
代码示例	// 设置定时任务委托 [GizDeviceSchedulerCenter setDelegate:self]; // 与设备的同步定时任务，mDevice为从设备列表中取到设备对象 [GizDeviceSchedulerCenter updateSchedulers:@"your_uid" token:@"your_token" schedulerOwner:mDevice]; // 实现回调 - (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList { if(result.code == GIZ_SDK_SUCCESS) { // 定时任务同步更新成功，参考didUpdateSchedulers接口代码示例 } else { // 同步失败 } }	

18. GizDeviceScheduler 类

18.1. 简介

GizDeviceScheduler 类是云端定时任务类，可设置一次性定时任务、按月重复的定时任务、按周重复的定时任务。一次性定时任务是指只执行一次定时任务，按月重复定时任务是指在每月特定日期执行定时任务，按周重复定时任务是指在每周特定时间执行定时任务。

该类继承自父类 GizDeviceSchedulerSuper，GizDeviceSchedulerCenter 类回调定时任务列表时用 GizDeviceSchedulerSuper 对象做回调。App 处理定时任务列表时需要判断数组内的对象类型是否为 GizDeviceScheduler 类。

18.2. 属性访问

属性	描述
delegate	使用委托获取对应事件。GizDeviceScheduler 对应的回调接口在

属性	描述
	GizDeviceSchedulerDelegate 定义，需要用到哪个接口，实现相应的回调即可。此变量可写
schedulerID	NSString 类型，只读。定时任务 ID，是定时任务创建时被分配的唯一标识
schedulerOwner	GizWifiDevice 类型，只读。管理定时任务的设备，是用于创建、删除、维护定时任务信息的
date	NSString 类型，可读写。定时任务的预设日期，格式形如：1990-10-03。定时任务会在预设日期这一天到达时执行
time	NSString 类型，可读写。定时任务的预设时间，24 小时制，格式形如：07:08。定时任务会在预设时间到达时执行
weekdays	GizScheduleWeekday 枚举数组，可读写。按周重复，定时任务会每周重复执行
monthDays	NSNumber 类型数组，可读写。按天重复，范围是 1～31。定时任务会每月重复执行
enabled	BOOL 类型，可读写。定时任务是否开启。YES 表示开启，NO 表示关闭
remark	NSString 类型，可读写。云端定时任务的备注信息
attrs	NSDictionary 类型，可读写。定时任务要执行的动作，数据点键值对字典，键值对为{数据点名称：数据点值}。数据点名称和值的类型要符合设备的数据点定义： （1）如果数据点为布尔类型，则 value 为 NSNumber 类型； （2）如果数据点为数值类型，则 value 为 NSNumber 类型； （3）如果数据点为枚举类型，则 value 为枚举序号（NSNumber 类型）或者枚举字符串（NSString 类型）； （4） 如果数据点为扩展类型，则 value 为 NSData 类型；
schedulerType	云端定时任务有三种类型：一次性任务、按周重复、按天重复，是根据时间重复类型区分的。一次性任务需要同时设置 date、time，按周重复需要设置 time 和 weekdays，按天需要设置 time 和 monthDays；
startDate	NSString 类型，可读写。云端定时任务预设的起始时间，格式与 date 相同
endDate	NSString 类型，可读写。云端定时任务预设的结束时间，格式与 date 相同
createdDateTime	NSString 类型，只读不可写。定时任务的创建时间

18.3. 回调接口

以下是 GizDeviceSchedulerCenter 类提供的所有回调接口，将在后续 API 定义中详细介绍：

- didUpdateSchedulerInfo：定时任务列表回调

【didUpdateSchedulerInfo】

定义	- (void)scheduler:(GizDeviceScheduler*)scheduler

	didUpdateSchedulerInfo:(NSError*)result;	
功能描述	定时任务信息更新回调。调用修改定时任务信息接口 editSchedulerInfo、定时任务信息变化上报时触发该回调	
回调参数	scheduler	触发回调的云端定时任务对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功，此时 schedulerList 为定时任务列表；其他为失败，此时 schedulerList 大小为 0
代码示例	// mDevice为从设备列表中取到的设备对象 mDevice.delegate = self; // 实现回调 - (void)scheduler:(GizDeviceScheduler*)scheduler didUpdateSchedulerInfo:(NSError*)result { if(result.code == GIZ_SDK_SUCCESS) { // 接收上报的定时任务信息变更 } else { // 失败处理 } }	

18.4. API

【schedulerOneTime】

定义	+ (id)schedulerOneTime:(NSDictionary*)attrs date:(NSString*)date time:(NSString*)time enabled:(BOOL)enabled remark:(NSString*)remark;	
功能描述	创建一次性云端定时任务时使用此构造方法	
参数	attrs	定时任务操作键值对字典：{操作名字: 操作值}，请注意不支持透传数据。此参数不能填 nil 或空字典
	date	定时任务的预设日期，格式形如：1990-10-03。定时任务将在预设日期这一天到达时执行。此参数不能填 nil 或空串，如果填写了过去日期或者不符合约定格式，无法在云端创建定时任务
	time	定时任务的预设时间，24 小时制，格式形如：07:08。定时任务将在预设时间到达时执行。此参数不能填 nil 或空串，必须符合约定格式，否则无法在云端创建定时任务
	enabled	定时任务是否开启。true 表示开启，false 表示不开启
	remark	定时任务备注信息。此参数可选填，可填 nil

代码示例	// 一次性定时任务，在 2017 年 1 月 16 日早上 6 点 30 分开灯 GizDeviceScheduler *scheduler = [[GizDeviceScheduler schedulerOneTime:@{@"LED_OnOff": @YES} date: @"2017-01-16" time:@"06:30" enabled:@YES remark: @"开灯任务"]; // 设置定时任务委托 [GizDeviceSchedulerCenter setDelegate:self]; // 创建云端定时任务，mDevice 为从设备列表中取到的设备对象 [GizDeviceSchedulerCenter createScheduler:@"your_uid" token:@"your_token" schedulerOwner:mDevice scheduler:scheduler schedulerTasks:nil]; // 实现回调 - (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList { if(result.code == GIZ_SDK_SUCCESS) { // 定时任务创建成功，参考 didUpdateSchedulers 接口代码示例 } else { // 创建失败 } }

【schedulerWeekRepeat】

定义	+ (id)schedulerWeekRepeat:(NSDictionary*)attrs time:(NSString*)time weekDays:(NSArray*)weekDays enabled:(BOOL)enabled remark:(NSString*)remark;	
功能描述	创建按周重复的云端定时任务时使用此构造方法	
参数	attrs	定时任务操作键值对字典：{操作名字: 操作值}，请注意不支持透传数据。此参数不能填 nil 或空字典
	time	定时任务的预设时间，24 小时制，格式形如：07:08。此参数不能填 nil 或空串，必须符合约定格式。定时任务将在预设时间到达时执行
	weekDays	按周重复，GizScheduleWeekday 数组。定时任务可以预设为每周的某几天重复执行。此参数不能填 nil 或空数组，数组中重复的值会被合并，无效值会被滤除，如果滤除后数组大小为 0 按空数组处理
	enabled	定时任务是否开启。true 表示开启，false 表示不开启
	remark	定时任务备注信息。此参数可选填，可填 nil
代码示例	// 按周重复定时任务，每周一和周五早上 6 点 30 分开灯	

```
GizDeviceScheduler *scheduler = [[GizDeviceScheduler schedulerOneTime:
@{@"LED_OnOff": @YES} time:@"06:30" weekDays:[NSArray
arrayWithObjects: @(GizScheduleMonday), @(GizScheduleFriday), nil]
enabled:@YES remark: @"开灯任务"];

// 设置定时任务委托
[GizDeviceSchedulerCenter setDelegate:self];

// 创建云端定时任务，mDevice 为从设备列表中取到的设备对象
[GizDeviceSchedulerCenter createScheduler:@"your_uid"
token:@"your_token" schedulerOwner:mDevice scheduler:scheduler
schedulerTasks:nil];

// 实现回调
- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner
result:(NSError*)result schedulerList:(NSArray*)schedulerList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 定时任务创建成功，参考 didUpdateSchedulers 接口代码示例
    } else {
        // 创建失败
    }
}
```

【schedulerDayRepeat】

定义	+ (id) schedulerDayRepeat:(NSDictionary*)attrs time:(NSString*)time monthDays:(NSArray*)monthDays enabled:(BOOL)enabled remark:(NSString*)remark;	
功能描述	创建按周重复的云端定时任务时使用此构造方法	
参数	attrs	定时任务操作键值对字典：{操作名字: 操作值}，请注意不支持透传数据。此参数不能填 nil 或空字典
	time	定时任务的预设时间，24 小时制，格式形如：07:08。此参数不能填 nil 或空串，必须符合约定格式。定时任务将在预设时间到达时执行
	monthDays	按天重复，整数数组。定时任务可以预设为每周的某几天重复执行。此参数不能填 nil 或空数组，数组中重复的值会被合并，无效值会被滤除，如果滤除后数组大小为 0 按空数组处理
	enabled	定时任务是否开启。true 表示开启，false 表示不开启
	remark	定时任务备注信息。此参数可选填，可填 nil
代码示例	// 按天重复定时任务，每月 1 号和 15 号早上 6 点 30 分开灯	

	GizDeviceScheduler *scheduler = [[GizDeviceScheduler schedulerOneTime: @{@"LED_OnOff": @YES} time:@"06:30" weekDays:[NSArray arrayWithObjects: @1, @15, nil] enabled:@YES remark: @"开灯任务"]; // 设置定时任务委托 [GizDeviceSchedulerCenter setDelegate:self]; // 创建云端定时任务，mDevice 为从设备列表中取到的设备对象 [GizDeviceSchedulerCenter createScheduler:@"your_uid" token:@"your_token" schedulerOwner:mDevice scheduler:scheduler schedulerTasks:nil]; // 实现回调 - (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList { if(result.code == GIZ_SDK_SUCCESS) { // 定时任务创建成功，参考 didUpdateSchedulers 接口代码示例 } else { // 创建失败 } }

【editSchedulerInfo】

定义	- (void)editSchedulerInfo:(NSString*)uid token(NSString*)token type:(GizSchedulerType)type;	
功能描述	修改定时任务信息，此接口可用于修改云端或中控的定时任务信息。请注意，必须要先修改对应的变量值，然后再调用此接口完成修改。修改成功时返回最新的定时任务信息，修改失败时返回错误信息	
参数	uid	用户 uid。此参数必填，不能填空串或无效值
	token	用户 token。此参数必填，不能填空串或无效值
	schedulerType	定时任务类型，GizSchedulerType 枚举。此参数不能填无效值
代码示例	// 把之前创建好的一次性定时任务修改成每月 1 号和 15 号重复执行的定时任务，scheduler 是定时任务列表中要修改的定时任务对象 scheduler.date = @"2017-01-16"; scheduler.time = @"06:30"; scheduler.remark = @"开灯任务"; scheduler.attrs = @{@"LED_OnOff": @YES}; scheduler.monthDays = @[@1, @15];	

```
// 设置定时任务委托
[scheduler setDelegate:self];

// 修改设备的定时任务
[scheduler editScheduler:@"your_uid" token:@"your_token"
type:GizSchedulerDayRepeat];

// 实现回调
- (void)scheduler:(GizDeviceScheduler*)scheduler
didUpdateSchedulerInfo:(NSError*)result {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 接收定时任务信息变更
    } else {
        // 失败处理
    }
}
```

19. GizDeviceJointActionCenter 类

19.1. 简介

GizDeviceJointActionCenter 类为 APP 开发者提供网关联动功能，管理用户在网关上设置的联动操作。

19.2. 属性访问

以下是 GizDeviceJointActionCenter 类提供的属性变量：

属性	描述
delegate	GizDeviceJointActionCenter 委托

【getJointActionListGateway】

定义	+(NSArray*)getJointActionListGateway:(GizWifiDevice*)jointActionOwner;	
功能描述	获取 SDK 本地缓存的中控设备下的设备联动列表，这个联动列表缓存了 GizDeviceJointAction 对象	
参数	jointActionOwn	中控设备，此参数不能填 nil

	er	
返回值	联动列表。参数为 nil 或无效设备时返回空数组	
代码示例	// mOwner是在设备列表中得到的中控设备对象， NSArray* jointActionList = [GizDeviceJointActionCenter getJointActionListGateway: mOwner];	

19.3. 回调接口

以下是 GizDeviceJointActionCenter 类提供的所有回调接口：

- didUpdateJointActions：联动列表更新回调

【didUpdateJointActions】

定义	- (void)didUpdateJointActions:(GizWifiDevice*)jointActionOwner result:(NSError*)result jointActionList:(NSArray*)jointActionList;	
功能描述	联动列表回调。调用添加联动接口 addJointAction、删除联动接口 removeJointAction、更新联动列表接口 updateJointActions、联动列表变化上报时都使用该回调接口。	
回调参数	jointActionOwner	触发回调的 GizWifiDevice 对象
	result	详细见 GizWifiErrorCode 枚举定义。GIZ_SDK_SUCCESS 表示成功同时 jointActionList 为最新的联动列表，其他为失败并且 jointActionList 大小为 0
	jointActionList	联动列表，是 GizDeviceJointAction 对象数组
代码示例	//设置联动的委托 GizDeviceJointActionCenter.delegate = self; // 实现回调 - (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner result:(NSError *)result jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList { if (result.code == GIZ_SDK_SUCCESS) { // 接收上报的联动列表 for (GizDeviceJointAction *jointAction in jointActionList) { if ([jointAction isKindOfClass:[GizDeviceJointAction class]]) { // 处理联动规则 } } } else { //失败处理	

	} }

19.4. API

【addJointAction】

定义	+ (void)addJointAction:(GizWifiDevice*)jointActionOwner jointActionName:(NSString*)jointActionName enabled:(BOOL)enabled jointActionRule:(GizDeviceJointActionRule*)jointActionRule;	
功能描述	添加联动。添加成功后会被分配一个联动 ID，同时通过回调接口 didUpdateJointActions 给 APP 返回最新的联动列表，失败时返回错误信息	
参数	jointActionOwner	联动的管理者设备。此参数不能填 nil
	jointActionName	联动名称。此参数为选填参数，如果不指定名称此参数填 nil，App 可以在成功添加联动以后再修改名称
	enabled	是否开启联动。true 为开启，false 为关闭
	jointActionRule	联动规则,是 GizDeviceJointActionRule 对象。此参数不能填 nil
代码示例	// 创建联动触发条件: 以子设备 subDevice1 灯被打开为条件 GizDeviceJointActionRuleCondition *condition = [GizDeviceJointActionRuleCondition jointActionRuleCondition:subDevice1 data:@{@"LED_OnOff": @YES} conditionOperator:GizConditionEqual]; // 创建联动触发结果: 当联动条件被触发时，打开子设备 subDevice2 的灯 GizDeviceJointActionRuleResultEvent *resultEvent = [GizDeviceJointActionRuleResultEvent jointActionRuleResultEventByDevice:subDevice2 data:@{@"LED_OnOff": @YES}]; // 创建联动规则对象 GizDeviceJointActionRule *rule = [GizDeviceJointActionRule jointActionRule:@[condition] conditionCombType:GizLogicalOr resultEvents:@[resultEvent]]; // 设置联动的委托 GizDeviceJointActionCenter.delegate = self;	

```
// 添加联动规则到中控中，centralDevice 从设备列表中取到的中控设备，
// subDevice1 和 subDevice2 是中控 centralDevice 下的子设备
 [GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    }
    else {
        // 添加失败
    }
}
```

【removeJointAction】

定义	+ (void)removeJointAction:(GizWifiDevice*)jointActionOwner jointAction:(GizDeviceJointAction*)jointAction;	
功能描述	删除联动。删除成功时通过回调接口 didUpdateJointActions 给 APP 返回最新的联动列表，删除失败时返回错误信息。	
参数	jointActionOwner	联动的管理者，参见 GizDeviceJointAction 类中 jointActionOwner 变量的描述。此参数不能填 nil
	jointAction	待删除的联动。参见 GizDeviceJointAction 类中 jointAction 变量的描述。此参数不能填 nil

代码示例

```
// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 删除设备的联动规则: centralDevice 为从设备列表中取到的中控设备，jointAction 为从 centralDevice 的联动列表中取到的联动动作对象
[GizDeviceJointActionCenter removeJointAction:centralDevice
jointAction:jointAction];

// 删除联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
```

	`` if (result.code == GIZ_SDK_SUCCESS) { // 联动删除成功，参考 didUpdateJointActions 接口代码示例 } else { // 删除失败 } } ```

【updateJointActions】

定义	+ (void)updateJointActions:(GizWifiDevice*)jointActionOwner;	
功能描述	更新联动列表。更新成功时通过回调接口 didUpdateJointActions 给 APP 返回最新的联动列表，更新失败时返回错误信息。	
参数	jointActionOwner	联动的管理者，参见 GizDeviceJointAction 类中 jointActionOwner 变量的描述。此参数不能填 nil。
代码示例	``` // 设置联动的委托 GizDeviceJointActionCenter.delegate = self; // 更新联动列表: centralDevice 为从设备列表中取到的中控设备 [GizDeviceJointActionCenter updateJointActions:centralDevice]; // 更新联动回调 - (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner result:(NSError *)result jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList { if (result.code == GIZ_SDK_SUCCESS) { // 联动列表更新成功，参考 didUpdateJointActions 接口代码示例 } else { // 更新失败 } } ```	

20.GizDeviceJointAction 类

20.1. 简介

GizDeviceJointAction 类是中控联动类，可设置联动规则。

20.2. 属性访问

属性	描述
JointActionID	只读，NSString 类型。联动 ID，是联动创建时被分配的唯一标识
jointActionOwner	只读，GizWifiDevice 类型。管理联动的设备，是用于创建、删除、维护联动信息的
jointActionName	只读，NSString 类型。联动名称
jointActionRule	只读，GizDeviceJointActionRule 对象。联动规则
enabled	只读，BOOL 类型。联动是否开启

21. GizDeviceJointActionRule 类

21.1. 简介

GizDeviceJointActionRule 类是联动规则。只有中控才支持联动规则设置。设定联动规则后，当触发条件满足时会触发结果事件的执行。触发条件即当设备某个数据点达到了某个值的范围，结果事件即让某个设备、组执行某个动作或开启场景、定时任务。触发条件是条件集合，结果事件是事件集合。App 可以让条件集合中所有条件同时满足时触发某些动作的执行，也可以当任一条件满足时触发某些动作的执行。

21.2. 属性访问

属性	描述
conditionList	只读，NSArray 类型。设备联动规则的触发条件，GizDeviceJointActionRuleCondition 对象数组
conditionCombType	只读，GizLogicalOperator 枚举值。此变量是联动规则的触发条件组合类型，GizLogicalAnd 类型代表并列条件，即 conditionList 中所有条件必须同时满足；GizLogicalOr 类型代表任意条件，即 conditionList 中有一个条件满足即可；如果 conditionList 中只有一个条件，可忽略此变量
resultEventList	只读，NSArray 类型。设备联动规则的结果事件，GizDeviceJointActionRuleResultEvent 对象数组

21.3. API

【jointActionRule】

定义	+ (id)jointActionRule:(NSArray*)conditions conditionCombType:(GizLogicalOperator)conditionCombType resultEvents:(NSArray*)resultEvents;

功能描述	构造函数，用于创建联动规则对象，目前只支持中控。	
参数	conditions	条件列表，GizDeviceJointActionRuleCondition 对象数组。此参数不能填 nil 或空数组，可以设置一个或多个条件
	conditionCombType	条件组合类型。conditions 中只有一个条件时，忽略此变量。详细见类中的只读变量 conditionCombType 说明。
	resultEvents	结果事件集合，GizDeviceJointActionRuleResultEvent 对象数组。此参数不能填 nil 或空数组，可以填一个或多个事件。
代码示例		

```
// 创建联动触发条件: 以子设备 subDevice1 灯被打开为条件
GizDeviceJointActionRuleCondition *condition =
[GizDeviceJointActionRuleCondition
jointActionRuleCondition:subDevice1 data:@{@"LED_OnOff": @YES}
conditionOperator:GizConditionEqual];

// 创建联动触发结果: 当联动条件被触发时，打开子设备 subDevice2 的灯
GizDeviceJointActionRuleResultEvent *resultEvent =
[GizDeviceJointActionRuleResultEvent
jointActionRuleResultEventByDevice:subDevice2 data:@{@"LED_OnOff":
@YES}];

// 创建联动规则对象
 GizDeviceJointActionRule *rule = [GizDeviceJointActionRule
jointActionRule:@[condition] conditionCombType:GizLogicalOr
resultEvents:@[resultEvent]];

// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 添加联动规则到中控中，centralDevice 是从设备列表中取到的中控设备,
// subDevice1 和 subDevice2 是中控 centralDevice 下的子设备
 [GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    } else {
        // 添加失败
```

	} }

22.GizDeviceJointActionRuleConditio n类

22.1. 简介

GizDeviceJointActionRuleCondition 类是联动规则中的条件。

22.2. 属性访问

属性	描述
device	只读，GizWifiDevice 类型。联动规则中的中控子设备
data	只读，NSDictionary 类型。数据点键值对{key: value}，key 为要判断的状态名称，value 要判断的状态阀值。注意只能设置一个数据点
conditionOperator	只读，GizConditionOperator 枚举值。此变量提供 6 种运算方法，用于把条件设定值与设备实际状态值做比较

22.3. API

【jointActionRuleCondition】

定义	+ (id)jointActionRuleCondition:(GizWifiDevice*)device data:(NSDictionary*)data conditionOperator:(GizConditionOperator)conditionOperator;	
功能描述	构造函数，用于创建联动规则的触发条件。目前只支持中控。	
参数	device	触发条件中的中控子设备。此参数不能填 nil。
	data	触发条件中的设备状态阀值：{操作名字: 操作值}，此参数不能填 nil 或空字典。请注意：无效数据点无法触发结果事件的执行
	conditionOperator	条件运算符，用于中控判断子设备状态是否满足条件。此参数必须填有效范围内的值
代码示例	// 创建联动触发条件: 以子设备 subDevice1 灯被打开为条件 GizDeviceJointActionRuleCondition *condition = [GizDeviceJointActionRuleCondition jointActionRuleCondition:subDevice1 data:@{@"LED_OnOff": @YES}	

```
conditionOperator:GizConditionEqual];

// 创建联动触发结果: 当联动条件被触发时，打开子设备 subDevice2 的灯
GizDeviceJointActionRuleResultEvent *resultEvent =
[GizDeviceJointActionRuleResultEvent
jointActionRuleResultEventByDevice:subDevice2 data:@{@"LED_OnOff":
@YES}];

// 创建联动规则对象
 GizDeviceJointActionRule *rule = [GizDeviceJointActionRule
jointActionRule:@[condition] conditionCombType:GizLogicalOr
resultEvents:@[resultEvent]];

// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 添加联动规则到中控中，centralDevice 是从设备列表中取到的中控设备,
// subDevice1 和 subDevice2 是中控 centralDevice 下的子设备
 [GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    }
    else {
        // 添加失败
    }
}
```

23.GizDeviceJointActionRuleResultEvent 类

23.1. 简介

GizDeviceJointActionRuleResultEvent 类是联动规则中要触发的结果。

23.2. 属性访问

属性	描述
device	只读，GizWifiDevice 类型。联动规则结果中的中控子设备
group	只读,GizDeviceGroup 类型。联动规则结果中的中控分组,如果 resultEventType 是组事件，需要关心此变量
scene	只读，GizDeviceScene 类型。联动规则结果中的场景，如果 resultEventType 是场景事件，需要关心此变量
scheduler	只读，GizDeviceSchedulerSuper 类型。联动规则结果中的定时任务，如果 resultEventType 是定时任务事件，需要关心此变量
data	只读，NSDictionary 类型。设备或组要执行的操作键值对字典：{操作名字: 操作值}，resultEventType 是设备或组事件需要关心此变量
enabled	只读，BOOL 类型。场景或定时任务事件是否开启
resultEventType	只读， GizJointActionRuleEventType 枚举值。设备事件只需要关心 device、data；组事件只需要关心 group、data

23.3. API

【jointActionRuleResultEventByDevice】

定义	+ (id)jointActionRuleResultEventByDevice:(GizWifiDevice*)device data:(NSDictionary*)data;	
功能描述	构造函数，用于创建联动规则要执行的设备事件。目前只支持中控。	
参数	device	设备事件的中控子设备。此参数不能填 nil
	data	要执行的操作键值对字典：{操作名字: 操作值}，此参数不能填 nil 或空字典。请注意：无效数据点无法让设备执行命令。
代码示例	// 创建联动触发条件: 以子设备 subDevice1 灯被打开为条件 GizDeviceJointActionRuleCondition *condition = [GizDeviceJointActionRuleCondition jointActionRuleCondition:subDevice1 data:@{@"LED_OnOff": @YES} conditionOperator:GizConditionEqual]; // 创建联动触发结果: 当联动条件被触发时，打开子设备 subDevice2 的灯 GizDeviceJointActionRuleResultEvent *resultEvent = [GizDeviceJointActionRuleResultEvent jointActionRuleResultEventByDevice:subDevice2 data:@{@"LED_OnOff": @YES}];	

```
// 创建联动规则对象
 GizDeviceJointActionRule *rule = [GizDeviceJointActionRule
jointActionRule:@[condition] conditionCombType:GizLogicalOr
resultEvents:@[resultEvent]];

// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 添加联动规则到中控中，centralDevice 是从设备列表中取到的中控设备，
// subDevice1 和 subDevice2 是中控 centralDevice 下的子设备
 [GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    } else {
        // 添加失败
    }
}
```

【jointActionRuleResultEventByGroup】

定义	+ (id)jointActionRuleResultEventByGroup:(GizDeviceGroup*)group data:(NSDictionary*)data;	
功能描述	构造函数，用于创建联动规则要执行的组事件。目前只支持中控。	
参数	group	组事件的中控分组。此参数不能填 nil。
	data	要执行的操作键值对字典：{操作名字: 操作值}，此参数不能填 nil 或空字典。请注意：无效数据点无法执行联动规则
代码示例	// 创建联动触发条件: 以子设备 subDevice 灯被打开为条件 GizDeviceJointActionRuleCondition *condition = [GizDeviceJointActionRuleCondition jointActionRuleCondition:subDevice data:@{@"LED_OnOff": @YES} conditionOperator:GizConditionEqual]; // 创建联动触发结果: 当联动条件被触发时，打开分组 group 中所有子设备的灯 GizDeviceJointActionRuleResultEvent *resultEvent =	

```
[GizDeviceJointActionRuleResultEvent
jointActionRuleResultEventByGroup:group data:@{@"LED_OnOff": @YES}];

// 创建联动规则对象
GizDeviceJointActionRule *rule = [GizDeviceJointActionRule
jointActionRule:@[condition] conditionCombType:GizLogicalOr
resultEvents:@[resultEvent]];

// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 添加联动规则到中控中，centralDevice 从设备列表中取到的中控设备，group 是中控
centralDevice 下的某一个分组
[GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    } else {
        // 添加失败
    }
}
```

【jointActionRuleResultEventByScene】

定义	+ (id)jointActionRuleResultEventByScene:(GizDeviceScene*)scene enabled:(BOOL)enabled;	
功能描述	构造函数，用于创建联动规则要执行的场景事件。目前只支持中控。	
参数	scene	场景事件。此参数不能填 nil
	enabled	是否开启
代码示例	// 创建联动触发条件: 以子设备 subDevice 灯被打开为条件 GizDeviceJointActionRuleCondition *condition = [GizDeviceJointActionRuleCondition jointActionRuleCondition:subDevice data:@{@"LED_OnOff": @YES} conditionOperator:GizConditionEqual];	

```
// 创建联动触发结果: 当联动条件被触发时, 执行场景 scene
GizDeviceJointActionRuleResultEvent *resultEvent =
[GizDeviceJointActionRuleResultEvent
jointActionRuleResultEventByScene:scene enabled:YES];

// 创建联动规则对象
GizDeviceJointActionRule *rule = [GizDeviceJointActionRule
jointActionRule:@[condition] conditionCombType:GizLogicalOr
resultEvents:@[resultEvent]];

// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 添加联动规则到中控中，centralDevice 从设备列表中取到的中控设备, scene 是中控下的某一个有效场景
[GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    } else {
        // 添加失败
    }
}
```

【jointActionRuleResultEventByScheduler】

定义	+ (id)jointActionRuleResultEventByScheduler:(GizDeviceSchedulerSuper*)scheduler enabled:(BOOL)enabled;	
功能描述	构造函数，用于创建联动规则要执行的定时任务事件。目前只支持中控。	
参数	scheduler	定时任务事件的中控定时任务。此参数不能填 nil。
	enabled	是否开启
代码示例	// 创建联动触发条件: 以子设备 subDevice 灯被打开为条件 GizDeviceJointActionRuleCondition *condition =	

```
[GizDeviceJointActionRuleCondition jointActionRuleCondition:subDevice
data:@{@"LED_OnOff": @YES} conditionOperator:GizConditionEqual];

// 创建联动触发结果: 当联动条件被触发时, 执行定时任务 scheduler
GizDeviceJointActionRuleResultEvent *resultEvent =
[GizDeviceJointActionRuleResultEvent
jointActionRuleResultEventByScheduler:scheduler enabled:YES];

// 创建联动规则对象
GizDeviceJointActionRule *rule = [GizDeviceJointActionRule
jointActionRule:@[condition] conditionCombType:GizLogicalOr
resultEvents:@[resultEvent]];

// 设置联动的委托
GizDeviceJointActionCenter.delegate = self;

// 添加联动规则到中控中，centralDevice 从设备列表中取到的中控设备, scheduler
是中控下的某一个有效定时任务
[GizDeviceJointActionCenter addJointAction:centralDevice
jointActionName:@"联动名称" enabled:YES jointActionRule:rule];

// 添加联动回调
- (void)didUpdateJointActions:(GizWifiDevice *)jointActionOwner
result:(NSError *)result
jointActionList:(NSArray<GizDeviceJointAction *> *)jointActionList {
    if (result.code == GIZ_SDK_SUCCESS) {
        // 联动添加成功，参考 didUpdateJointActions 接口代码示例
    } else {
        // 添加失败
    }
}
```

24. 枚举定义

24.1. 简介

本节说明 GizWifiSDK 中使用的所有枚举定义。

24.2. 定义

【GizLogPrintLevel】

功能描述：日志打印级别。

枚举 ID	枚举定义	描述
0	GizLogPrintNone	不输出任何日志
1	GizLogPrintI	输出错误日志
2	GizLogPrintII	输出调试日志
3	GizLogPrintAll	输出数据日志

【GizEventType】

功能描述：事件通知类型。

枚举 ID	枚举定义	描述
0	GizEventSDK	SDK 系统事件
1	GizEventDevice	设备异常事件
2	GizEventM2MService	M2M 异常事件
5	GizEventToken	Token 失效事件

【GizWifiConfigureMode】

功能描述：设备配置模式。

枚举 ID	枚举定义	描述
0	GizWifiSoftAP	SoftAP 配置模式
1	GizWifiAirLink	AirLink 配置模式
2	GizWifiAirLinkMulti	支持多设备进行 AirLink 配置
3	GizWifiBleLink	蓝牙配置模式
4	GizWifiBleLinkMulti	支持多设备同时进行蓝牙配置

【GizWifiDeviceType】

功能描述：设备类型。

枚举 ID	枚举定义	描述
0	GizDeviceNormal	普通设备

枚举 ID	枚举定义	描述
1	GizDeviceCenterControl	中控设备
2	GizDeviceSub	子设备

【GizAdapterType】

功能描述：转换器声明

枚举 ID	枚举定义	描述
0	GizAdapterNon	默认值，普通产品不需要特殊声明
4	GizAdapterWifiBle	模组支持近场蓝牙控制

【GizBleWorkStatusType】

功能描述：蓝牙工作状态

枚举 ID	枚举定义	描述
0	GizBleWorkStatusTypeNormal	普通工作状态
1	GizBleWorkStatusTypeOnBording	配网状态

【GizThirdAccountType】

功能描述：第三方账号类型。

枚举 ID	枚举定义	描述
0	GizThirdBAIDU	百度账号
1	GizThirdSINA	新浪账号
2	GizThirdQQ	QQ 账号
3	GizThirdWeChat	微信账号
4	GizThirdFacebook	Facebook 账号
5	GizThirdTwitter	推特账号

【GizUserAccountType】

功能描述：机智云用户类型。

枚举 ID	枚举定义	描述
0	GizUserNormal	普通用户

枚举 ID	枚举定义	描述
1	GizUserPhone	手机用户
2	GizUserEmail	电子邮箱用户
3	GizUserOther	其他用户类型（包括匿名用户）

【GizDeviceNetType】

功能描述：设备网络类型。

枚举 ID	枚举定义	描述
0	GizDeviceNetWifi	WiFi 类型
1	GizDeviceNetNB	NB 类型
2	GizDeviceNetBT	BT 类型
3	GizDeviceNetBLE	BLE 类型

【GizWifiDeviceNetStatus】

功能描述：设备网络状态类型。

枚举 ID	枚举定义	描述
0	GizDeviceOffline	离线状态
1	GizDeviceOnline	在线状态
2	GizDeviceControlled	可控状态

【GizWifiGAgentType】

功能描述：模组类型。

枚举 ID	枚举定义	描述
0	GizGAgentMXCHIP	庆科 3162
1	GizGAgentHF	汉枫模组
2	GizGAgentRTK	睿昱模组
3	GizGAgentWM	联盛德模组
4	GizGAgentESP	乐鑫模组
5	GizGAgentQCA	高通模组
6	GizGAgentTI	TI 模组
7	GizGAgentFSK	语音天下模组

枚举 ID	枚举定义	描述
8	GizGAgentMXCHIP3	庆科 mico
9	GizGAgentBL	古北模组
10	GizGAgentAtmelEE	Atmel 模组
11	GizGAgentOther	其他模组
12	GizGAgentFlyLink	芯海模组
13	GizGAgentMxchipAWS	庆科 AWS
14	GizGAgentHFV8	汉枫 v8 模组
15	GizGAgentESPBroadcast	乐鑫模组广播模式

【GizPushType】

枚举 ID	枚举定义	描述
0	GizPushBaiDu	百度推送
1	GizPushJiGuang	极光推送
2	GizPushAWS	亚马逊推送
3	GizPushXinGe	信鸽推送

【GizUserGenderType】

功能描述：用户性别。

枚举 ID	枚举定义	描述
0	GizUserGenderMale	男
1	GizUserGenderFemale	女
2	GizUserGenderUnknown	其他

【GizDeviceSharingUserRole】

功能描述：不同的用户角色设备分享时具有不同的设备绑定权限

枚举 ID	枚举定义	描述
0	GizDeviceSharingNormal	普通绑定用户
1	GizDeviceSharingSpecial	潜在 Owner 用户（第一个绑定了设备还未进行分享的用户）

枚举 ID	枚举定义	描述
2	GizDeviceSharingOwner	Owner 用户
3	GizDeviceSharingGuest	Guest 用户

【GizDeviceSharingType】

功能描述：设备分享类型

枚举 ID	枚举定义	描述
0	GizDeviceSharingByMe	自己发出的分享邀请
1	GizDeviceSharingToMe	分享给自己的邀请

【GizDeviceSharingWay】

功能描述：分享方式

枚举 ID	枚举定义	描述
0	GizDeviceSharingByNormal	账号分享
1	GizDeviceSharingByQRCode	二维码分享

【GizDeviceSharingStatus】

功能描述：设备分享状态

枚举 ID	枚举定义	描述
0	GizDeviceSharingNotAccepted	未接受
1	GizDeviceSharingAccepted	已接受
2	GizDeviceSharingRefused	已拒绝
3	GizDeviceSharingCancelled	已取消

【GizMessageType】

功能描述：消息类型

枚举 ID	枚举定义	描述
0	GizMessageSystem	系统消息
1	GizMessageSharing	分享消息

【GizMessageStatus】

功能描述：消息状态

枚举 ID	枚举定义	描述
0	GizMessageUnread	未读消息
1	GizMessageRead	已读消息
2	GizMessageDeleted	已删除消息

【GizOTAEventType】

功能描述：OTA 事件类型

枚举 ID	枚举定义	描述
0	GizOTAEventPretreatment	开始预处理，检查版本等动作
1	GizOTAEventDownload	开始下载固件中
2	GizOTAEventTransmit	开始发送固件
3	GizOTAEventReboot	设备接收完固件，准备重启
4	GizOTAEventFinish	OTA 完成

【GizOTAFirmwareType】

功能描述：固件升级类型

枚举 ID	枚举定义	描述
0	GizOTAFirmareModule	设备的模组固件
1	GizOTAFirmareMcu	设备的 mcu 固件

【GizSceneItemType】

功能描述：场景项类型

枚举 ID	枚举定义	描述
0	GizSceneItemDevice	设备场景项
1	GizSceneItemGroup	组场景项
2	GizSceneItemDelay	延时场景项

【GizSchedulerType】

功能描述：定时任务类型

枚举 ID	枚举定义	描述
0	GizSchedulerDelay	延时任务
1	GizSchedulerOneTime	一次性定时任务
2	GizSchedulerWeekRepeat	按周重复定时任务
3	GizSchedulerDayRepeat	按天重复定时任务

【GizScheduleStatus】

功能描述：定时任务执行状态

枚举 ID	枚举定义	描述
0	GizScheduleSucceed	执行成功
1	GizScheduleFailed	执行失败
2	GizScheduleNotDone	执行中

【GizScheduleWeekday】

功能描述：按周重复

枚举 ID	枚举定义	描述
0	GizScheduleSunday	周日
1	GizScheduleMonday	周一
2	GizScheduleTuesday	周二
3	GizScheduleWednesday	周三
4	GizScheduleThursday	周四
5	GizScheduleFriday	周五
6	GizScheduleSaturday	周六

【GizLogicalOperator】

功能描述：条件组合类型

枚举 ID	枚举定义	描述
0	GizLogicalAnd	逻辑与

枚举 ID	枚举定义	描述
1	GizLogicalOr	逻辑或

【GizConditionOperator】

功能描述：条件运算符

枚举 ID	枚举定义	描述
0	GizConditionEqual	相等
1	GizConditionLessThan	小于
2	GizConditionMoreThan	大于
3	GizConditionLessThanOrEqual	小于等于
4	GizConditionMoreThanOrEqual	大于等于
5	GizConditionNotEqual	不相等

【GizJointActionRulerEventType】

功能描述：联动规则结果事件类型

枚举 ID	枚举定义	描述
0	GizJointActionRulerEventDevice	设备事件
1	GizJointActionRulerEventGroup	组事件
2	GizJointActionRulerEventScene	场景事件
3	GizJointActionRulerEventScheduler	定时任务事件

【GizMeshVerdor】

功能描述：mesh 设备厂商

枚举 ID	枚举定义	描述
0	GizMeshTelink	泰凌微
1	GizMeshJingXun	晶讯

【GizLocalMeshType】

功能描述：蓝牙本地控制类型（即蓝牙低功耗协议通讯）

枚举 ID	枚举定义	描述
0	GizLocalMeshUnSupport	不支持蓝牙本地控制

枚举 ID	枚举定义	描述
2	GizLocalMeshSub	标识该 productKey 的子设备支持蓝牙本地控制
3	GizLocalMeshGateway	标识该 productKey 的网关子设备支持蓝牙本地控制

【GizWifErrorCode】

功能描述：错误码定义。

枚举 ID	枚举定义	描述
0	GIZ_SDK_SUCCESS	执行成功
8001	GIZ_SDK_PARAM_FORM_INVALID	SDK 内部通讯数据格式无效
8002	GIZ_SDK_CLIENT_NOT_AUTHEN	SDK 未启动
8003	GIZ_SDK_CLIENT_VERSION_INVALID	无效的 SDK 版本号
8004	GIZ_SDK_UDP_PORT_BIND_FAILED	udp 端口绑定失败
8005	GIZ_SDK_DAEMON_EXCEPTION	SDK 后台服务发生异常
8006	GIZ_SDK_PARAM_INVALID	接口参数无效
8007	GIZ_SDK_APPID_LENGTH_ERROR	AppID 长度错误
8008	GIZ_SDK_LOG_PATH_INVALID	SDK 日志路径无效
8009	GIZ_SDK_LOG_LEVEL_INVALID	日志级别无效
8010	GIZ_SDK_UID_INVALID	uid 参数无效
8011	GIZ_SDK_TOKEN_INVALID	token 参数无效
8012	GIZ_SDK_USER_NOT_LOGIN	用户未登录
8013	GIZ_SDK_APPID_INVALID	AppID 无效
8014	GIZ_SDK_APP_SECRET_INVALID	App secret 无效
8015	GIZ_SDK_PRODUCT_KEY_INVALID	产品类型识别码无效
8016	GIZ_SDK_PRODUCT_SECRET_INVALID	产品秘钥无效
8017	GIZ_SDK_DEVICE_NOT_IN_LAN	设备不在局域网
8018	GIZ_SDK_PRODUCTKEY_NOT_IN_SPECIAL_LIST	设备不在指定的产品类型识别码内
8019	GIZ_SDK_PRODUCTKEY_NOT_RELATED_WITH_APPID	设备的产品类型识别码跟当前 AppID 未关联
8020	GIZ_SDK_NO_AVAILABLE_DEVICE	批量设置设备域名信息时没有可用设备
8021	GIZ_SDK_DEVICE_CONFIG_SEND_FAILED	设备配置信息发送失败
8022	GIZ_SDK_DEVICE_CONFIG_IS_RUNNING	设备正在配网
8023	GIZ_SDK_DEVICE_CONFIG_TIMEOUT	设备配置超时

枚举 ID	枚举定义	描述
8024	GIZ_SDK_DEVICE_DID_INVALID	设备 did 无效
8025	GIZ_SDK_DEVICE_MAC_INVALID	设备 mac 无效
8026	GIZ_SDK_SUBDEVICE_DID_INVALID	子设备无效
8027	GIZ_SDK_DEVICE_PASSCODE_INVALID	设备鉴权码无效
8028	GIZ_SDK_DEVICE_NOT_CENTERCONTROL	不是中控设备
8029	GIZ_SDK_DEVICE_NOT_SUBSCRIBED	设备还未订阅
8030	GIZ_SDK_DEVICE_NO_RESPONSE	设备未响应
8031	GIZ_SDK_DEVICE_NOT_READY	设备还未就绪
8032	GIZ_SDK_DEVICE_NOT_BINDED	设备还未绑定
8033	GIZ_SDK_DEVICE_CONTROL_WITH_INVALID_COMMAND	设备控制指令中包含无效指令
8034	GIZ_SDK_DEVICE_CONTROL_FAILED	设备控制失败
8035	GIZ_SDK_DEVICE_GET_STATUS_FAILED	设备状态查询失败
8036	GIZ_SDK_DEVICE_CONTROL_VALUE_TYPE_ERROR	设备控制指令参数类型错误
8037	GIZ_SDK_DEVICE_CONTROL_VALUE_OUT_OF_RANGE	设备控制指令参数值不在有效范围内
8038	GIZ_SDK_DEVICE_CONTROL_NOT_WRITABLE_COMMAND	设备控制指令中包含不可写指令
8039	GIZ_SDK_BIND_DEVICE_FAILED	设备绑定失败
8040	GIZ_SDK_UNBIND_DEVICE_FAILED	设备解绑失败
8041	GIZ_SDK_DNS_FAILED	域名解析失败
8042	GIZ_SDK_M2M_CONNECTION_SUCCESS	M2M 服务器连接成功
8043	GIZ_SDK_SET_SOCKET_NON_BLOCK_FAILED	Socket 非阻塞设置失败
8044	GIZ_SDK_CONNECTION_TIMEOUT	连接超时
8045	GIZ_SDK_CONNECTION_REFUSED	连接被拒绝
8046	GIZ_SDK_CONNECTION_ERROR	发生网络异常
8047	GIZ_SDK_CONNECTION_CLOSED	连接被对端关闭
8048	GIZ_SDK_SSL_HANDSHAKE_FAILED	SSL 握手失败
8049	GIZ_SDK_DEVICE_LOGIN_VERIFY_FAILED	设备登录验证失败
8050	GIZ_SDK_INTERNET_NOT_REACHABLE	当前外网无法访问
8051	GIZ_SDK_M2M_CONNECTION_FAILED	M2M 服务器连接失败
8061	GIZ_SDK_GET_DEVICE_LOG_STOPPED	停止接收设备日志
8062	GIZ_SDK_NO_AVAILABLE_DEVICE_TO_GET_DEVICE_LOG	没有可以获取设备日志的设备

枚举 ID	枚举定义	描述
8063	GIZ_SDK_IS_GETTING_DEVICE_LOG	正在获取设备日志
8095	GIZ_SDK_HTTP_SERVER_NOT_SUPPORT_API	云服务不支持此 API
8096	GIZ_SDK_HTTP_ANSWER_FORMAT_ERROR	HTTP 应答格式错
8097	GIZ_SDK_HTTP_ANSWER_PARAM_ERROR	HTTP 应答参数错误
8098	GIZ_SDK_HTTP_SERVER_NO_ANSWER	HTTP 服务无响应
8099	GIZ_SDK_HTTP_REQUEST_FAILED	HTTP 请求失败
8100	GIZ_SDK_OTHERWISE	保留的错误字
8101	GIZ_SDK_MEMORY_MALLOC_FAILED	内存分配失败
8102	GIZ_SDK_THREAD_CREATE_FAILED	线程创建失败
8103	GIZ_SDK_AES_ENCRYPT_FAILED	数据 AES 加密失败
8104	GIZ_SDK_AES_DECRYPT_FAILED	数据 AES 解密失败
8105	GIZ_SDK_JSON_OBJECT_CREATE_FAILED	Json 对象创建失败
8106	GIZ_SDK_JSON_PARSE_FAILED	Json 解析失败
8107	GIZ_SDK_JSON_UNFORMAT_FAILED	Json 去格式化失败
8108	GIZ_SDK_JSON_DUPLICATE_FAILED	Json 拷贝失败
8120	GIZ_SDK_SCHEDULER_CREATE_FAILED	定时任务创建失败
8121	GIZ_SDK_SCHEDULER_DELETE_FAILED	定时任务删除失败
8122	GIZ_SDK_SCHEDULER_EDIT_FAILED	定时任务信息编辑失败
8123	GIZ_SDK_SCHEDULER_LIST_UPDATE_FAILED	定时任务列表更新失败
8124	GIZ_SDK_SCHEDULER_TASK_EDIT_FAILED	定时任务的任务项编辑失败
8125	GIZ_SDK_SCHEDULER_TASK_LIST_UPDATE_FAILED	定时任务的任务项列表更新失败
8126	GIZ_SDK_SCHEDULER_ID_INVALID	定时任务 ID 无效
8127	GIZ_SDK_SCHEDULER_ENABLE_DISABLE_FAILED	定时任务开启或关闭失败
8128	GIZ_SDK_SCHEDULER_STATUS_UPDATE_FAILED	定时任务状态更新失败
8140	GIZ_SDK_SUBDEVICE_ADD_FAILED	子设备添加失败
8141	GIZ_SDK_SUBDEVICE_DELETE_FAILED	子设备删除失败
8142	GIZ_SDK_SUBDEVICE_LIST_UPDATE_FAILED	子设备列表更新失败

枚举 ID	枚举定义	描述
8150	GIZ_SDK_GROUP_ID_INVALID	组 ID 无效
8151	GIZ_SDK_GROUP_PRODUCTKEY_INVALID	组类型无效
8152	GIZ_SDK_GROUP_FAILED_DELETE_DEVICE	删除组设备失败
8153	GIZ_SDK_GROUP_FAILED_ADD_DEVICE	添加组设备失败
8154	GIZ_SDK_GROUP_GET_DEVICE_FAILED	组设备列表更新失败
8155	GIZ_SDK_GROUP_CREATE_FAILED	创建组失败
8156	GIZ_SDK_GROUP_DELETE_FAILED	删除组失败
8157	GIZ_SDK_GROUP_EDIT_FAILED	编辑组失败
8158	GIZ_SDK_GROUP_LIST_UPDATE_FAILED	组列表更新失败
8159	GIZ_SDK_GROUP_COMMAND_WRITE_FAILED	组控制失败
8170	GIZ_SDK_SCENE_CREATE_FAILED	场景创建失败
8171	GIZ_SDK_SCENE_DELETE_FAILED	场景删除失败
8172	GIZ_SDK_SCENE_EDIT_FAILED	场景编辑失败
8173	GIZ_SDK_SCENE_LIST_UPDATE_FAILED	场景列表更新失败
8174	GIZ_SDK_SCENE_ITEM_LIST_EDIT_FAILED	场景项列表编辑失败
8175	GIZ_SDK_SCENE_ITEM_LIST_UPDATE_FAILED	场景项列表更新失败
8176	GIZ_SDK_SCENE_ID_INVALID	场景 ID 无效
8177	GIZ_SDK_SCENE_RUN_FAILED	场景执行失败
8178	GIZ_SDK_SCENE_STATUS_UPDATE_FAILED	场景状态更新失败
8190	GIZ_SDK_PARAM_GATEWAY_OWNER_REQUIRED	管理者设备必须是一个网关
8191	GIZ_SDK_PARAM_NO_DEVICE_OWNER	没有指定管理者设备
8201	GIZ_SDK_DATAPOINT_NOT_DOWNLOAD	设备数据点文件还未下载
8202	GIZ_SDK_DATAPOINT_SERVICE_UNAVAILABLE	设备数据点文件下载服务不可用
8203	GIZ_SDK_DATAPOINT_PARSE_FAILED	设备数据点解析失败
8204	GIZ_SDK_DATAPOINT_NO_MODIFIED	配置文件无变化

枚举 ID	枚举定义	描述
8221	GIZ_SDK_JOINT_ACTION_CREATE_FAILED	联动创建失败
8222	GIZ_SDK_JOINT_ACTION_DELETE_FAILED	联动删除失败
8223	GIZ_SDK_JOINT_ACTION_VER_UNSUPPORTED	不支持的联动版本
8224	GIZ_SDK_JOINT_ACTION_CONDITION_COMBI_INVALID	联动条件组合无效
8225	GIZ_SDK_JOINT_ACTION_CONDITION_OPERATOR_INVALID	联动条件运算符无效
8226	GIZ_SDK_JOINT_ACTION_RESULT_TYPE_INVALID	联动结果类型无效
8227	GIZ_SDK_PARAM_JOINT_ACTION_REQUIRE_RULE	必须设置联动规则
8228	GIZ_SDK_PARAM_JOINT_ACTION_REQUIRE_CONDITION	必须设置联动条件
8229	GIZ_SDK_PARAM_JOINT_ACTION_CONDITION_REQUIRE_DEVICE	必须设置联动条件中的设备
8230	GIZ_SDK_PARAM_JOINT_ACTION_CONDITION_REQUIRE_DATA	必须设置联动条件中的动作
8231	GIZ_SDK_PARAM_JOINT_ACTION_CONDITION_DATA_ONLY_SUPPORT_ONE	联动条件中的设备只能设置一个动作
8232	GIZ_SDK_PARAM_JOINT_ACTION_CONDITION_DATA_INVALID	联动条件中的设备动作无效
8233	GIZ_SDK_PARAM_JOINT_ACTION_REQUIRE_RESULT	必须设置联动的期望结果
8234	GIZ_SDK_PARAM_JOINT_ACTION_RESULT_REQUIRE_DATA	必须设置联动要触发的动作
8235	GIZ_SDK_PARAM_JOINT_ACTION_RESULT_REQUIRE_DEVICE	必须设置触发联动动作的设备
8236	GIZ_SDK_PARAM_JOINT_ACTION_RESULT_REQUIRE_GROUP	必须设置触发联动的组
8237	GIZ_SDK_PARAM_JOINT_ACTION_RESULT_REQUIRE_SCENE	必须设置触发联动的场景
8238	GIZ_SDK_PARAM_JOINT_ACTION_RESULT_REQUIRE_SCHEDULER	必须设置触发联动的定时任务
8250	GIZ_SDK_PRODUCTKEY_NOT_IN_SPECIFY	产品类型不在指定范围内
8251	GIZ_SDK_DEVICE_PRODUCTKEY_DIFFERENT	设备产品类型不同，无法同时安全解绑
8252	GIZ_SDK_DEVICE_MESHID_EMPTY	设备 meshId 不能为空
8253	GIZ_SDK_DEVICE_MESHID_INVALID	设备 meshId 无效
8254	GIZ_SDK_DEVICE_MESHID_IS_NOT_NUMBER	设备 meshId 应为数字
8255	GIZ_SDK_DEVICE_MESH_GATEWAY_UNKNOWN	无法识别 mesh 网关
8256	GIZ_SDK_DEVICE_MESH_GATEWAY_NOT_READY	mesh 网关未就绪

枚举 ID	枚举定义	描述
8257	GIZ_SDK_DEVICE_CONTROL_NEED_MESH_GATEWAY	需要有 mesh 网关才能执行控制指令
8258	GIZ_SDK_DEVICE_MAC_LENGTH_INVALID	设备 mac 长度无效
8280	GIZ_SDK_BLE_DEVICE_CONNECT_FAILED	蓝牙设备连接失败
8281	GIZ_SDK_BLE_BLUETOOTH_FUNCTION_NOT_TURNED_ON	蓝牙功能没打开
8282	GIZ_SDK_BLE_PARAM_UUID_INFO_REQUIRED	服务角色特征值不能为空
8283	GIZ_SDK_BLE_PARAM_LTK_REQUIRED	通信密钥 LTK 不能为空
8284	GIZ_SDK_BLE_UNFIND_DEVICE_PERIPHERAL	没有找到蓝牙设备对应的外设
8285	GIZ_SDK_BLE_LOGIN_FAILED	登录蓝牙设备失败
8286	GIZ_SDK_BLE_SEARCH_DEVICE_STOPPED	搜索蓝牙设备操作已经停止
8287	GIZ_SDK_BLE_CANNOT_FIND_DEVICE_SERVER_OR_CHARACTERISTICS	查找不到设备的服务和角色特征值
8288	GIZ_SDK_BLE_MESHNAME_AND_PASSWORD_CANNOT_BE_EMPTY	当前用户的组网名称或密码不能为空
8289	GIZ_SDK_BLE_ADD_DEVICE_TO_GROUND_FAILED	添加分组失败
8290	GIZ_SDK_BLE_HAS_CONFLICT_OPERATION_IS_ONGOING	当前有设备正在做切网，分组或者恢复出厂设置，不能同时再进行相关操作
8291	GIZ_SDK_BLE_CURRENT_USER_MESHNAME_OR_PASSWORD_INVALID	当前用户的组网名称或密码无效
8292	GIZ_SDK_BLE_DEVICE_IS_DISCONNECTED	设备处于断开连接状态
8300	GIZ_SDK_SDK_NOT_INITIALIZED	SDK 还未初始化
8301	GIZ_SDK_APK_CONTEXT_IS_NULL	Android context 无效，无法启动 SDK
8302	GIZ_SDK_APK_PERMISSION_NOT_SET	还没有设置必须的访问权限
8303	GIZ_SDK_CHMOD_DAEMON_REFUSED	无法修改 SDK 后台服务的执行权限
8304	GIZ_SDK_EXEC_DAEMON_FAILED	SDK 后台服务启动失败
8305	GIZ_SDK_EXEC_CATCH_EXCEPTION	SDK 后台服务启动发生异常
8306	GIZ_SDK_APPID_IS_EMPTY	AppID 为空，无法使用 SDK
8307	GIZ_SDK_UNSUPPORTED_API	此 API 已废弃，不再提供支持
8308	GIZ_SDK_REQUEST_TIMEOUT	执行超时
8309	GIZ_SDK_DAEMON_VERSION_INVALID	SDK 后台服务版本无效
8310	GIZ_SDK_PHONE_NOT_CONNECT_TO_SOFTAP_SSID	手机没有连接设备热点

枚举 ID	枚举定义	描述
8311	GIZ_SDK_DEVICE_CONFIG_SSID_NOT_MATCHED	手机当前 Wifi 与设备配网 SSID 不匹配，无法完成设备配网
8312	GIZ_SDK_NOT_IN_SOFTAPMODE	设备没有在 softap 配网模式下
8313	GIZ_SDK_CONFIG_NO_AVAILABLE_WIFI	手机当前不是 Wifi 网络
8314	GIZ_SDK_RAW_DATA_TRANSMIT	当前为原始数据透传方式
8315	GIZ_SDK_PRODUCT_IS_DOWNLOADING	正在下载设备数据点文件
8316	GIZ_SDK_START_SUCCESS	SDK 启动成功
8317	GIZ_SDK_NEED_UPDATE_TO_LATEST	SDK 需要升级到最新版本
8318	GIZ_SDK_ONBOARDING_STOPPED	设备配网被中断
8319	GIZ_SDK_ONBOARDING_WIFI_IS_5G	当前配网路由是 5G
8350	GIZ_SDK_OTA_FIRMWARE_IS_LATEST	当前固件是最新版本，不需要升级
8351	GIZ_SDK_OTA_FIRMWARE_CHECK_UPDATE_FAILED	固件检查更新失败
8352	GIZ_SDK_OTA_FIRMWARE_DOWNLOAD_OK	固件下载成功
8353	GIZ_SDK_OTA_FIRMWARE_DOWNLOAD_FAILED	固件下载失败
8354	GIZ_SDK_OTA_DEVICE_BUSY_IN_UPGRADE	设备忙，固件正在升级
8355	GIZ_SDK_OTA_PUSH_FAILED	固件推送失败
8356	GIZ_SDK_OTA_FIRMWARE_VERSION_TOO_LOW	固件版本过低
8357	GIZ_SDK_OTA_FIRMWARE_CHECK_FAILED	固件校验失败
8358	GIZ_SDK_OTA_UPGRADE_FAILED	固件升级失败
8359	GIZ_SDK_OTA_FIRMWARE_VERIFY_SUCCESS	固件校验成功
8360	GIZ_SDK_OTA_DEVICE_NOT_SUPPORT	设备不支持手机 OTA 升级
8400	GIZ_SDK_WS_HANDSHAKE_FAILED	websocket 握手失败
8401	GIZ_SDK_WS_LOGIN_FAILED	websocket 登录失败
8402	GIZ_SDK_WS_DEVICE_SUBSCRIBE_FAILED	websocket 设备订阅失败
8403	GIZ_SDK_WS_DEVICE_UNSUBSCRIBE_FAILED	websocket 设备解除订阅失败
9001	GIZ_OPENAPI_MAC_ALREADY_REGISTERED	mac already registered!
9002	GIZ_OPENAPI_PRODUCT_KEY_INVALID	product_key invalid
9003	GIZ_OPENAPI_APPID_INVALID	appid invalid

枚举 ID	枚举定义	描述
9004	GIZ_OPENAPI_TOKEN_INVALID	token invalid
9005	GIZ_OPENAPI_USER_NOT_EXIST	user not exist
9006	GIZ_OPENAPI_TOKEN_EXPIRED	token expired
9007	GIZ_OPENAPI_M2M_ID_INVALID	m2m_id invalid
9008	GIZ_OPENAPI_SERVER_ERROR	server error
9009	GIZ_OPENAPI_CODE_EXPIRED	code expired
9010	GIZ_OPENAPI_CODE_INVALID	code invalid
9011	GIZ_OPENAPI_SANDBOX_SCALE_QUOTA_EXHAUSTED	sandbox scale quota exhausted!
9012	GIZ_OPENAPI_PRODUCTION_SCALE_QUOTA_EXHAUSTED	production scale quota exhausted!
9013	GIZ_OPENAPI_PRODUCT_HAS_NO_REQUEST_SCALE	product has no request scale!
9014	GIZ_OPENAPI_DEVICE_NOT_FOUND	device not found!
9015	GIZ_OPENAPI_FORM_INVALID	form invalid!
9016	GIZ_OPENAPI_DID_PASSCODE_INVALID	did or passcode invalid!
9017	GIZ_OPENAPI_DEVICE_NOT_BOUND	device not bound!
9018	GIZ_OPENAPI_PHONE_UNAVALIABLE	phone unavailable!
9019	GIZ_OPENAPI_USERNAME_UNAVALIABLE	username unavailable!
9020	GIZ_OPENAPI_USERNAME_PASSWORD_ERROR	username or password error!
9021	GIZ_OPENAPI_SEND_COMMAND_FAILED	send command failed!
9022	GIZ_OPENAPI_EMAIL_UNAVALIABLE	email unavailable!
9023	GIZ_OPENAPI_DEVICE_DISABLED	device is disabled!
9024	GIZ_OPENAPI_FAILED_NOTIFY_M2M	fail to notify m2m!
9025	GIZ_OPENAPI_ATTR_INVALID	attr invalid!
9026	GIZ_OPENAPI_USER_INVALID	user invalid!
9027	GIZ_OPENAPI_FIRMWARE_NOT_FOUND	firmware not found!
9028	GIZ_OPENAPI_JD_PRODUCT_NOT_FOUND	JD product info not found!
9029	GIZ_OPENAPI_DATAPOINT_DATA_NOT_FOUND	datapoint data not found!
9030	GIZ_OPENAPI_SCHEDULER_NOT_FOUND	scheduler not found!
9031	GIZ_OPENAPI_QQ_OAUTH_KEY_INVALID	qq oauth key invalid!
9032	GIZ_OPENAPI_OTA_SERVICE_OK_BUT_IN_IDLE	ota upgrade service OK, but in idle or disable!
9033	GIZ_OPENAPI_BT_FIRMWARE_UNVERIFIED	bt firmware unverified, except verify

枚举 ID	枚举定义	描述
		device!
9034	GIZ_OPENAPI_BT_FIRMWARE_NOTHING_TO_UPGRADE	bt firmware is OK, but nothing to upgrade!
9035	GIZ_OPENAPI_SAVE_KAIROSDB_ERROR	Save kairosdb error!
9036	GIZ_OPENAPI_EVENT_NOT_DEFINED	event not defined!
9037	GIZ_OPENAPI_SEND_SMS_FAILED	send sms failed!
9038	GIZ_OPENAPI_APPLICATION_AUTH_INVALID	X-Gizwits-Application-Auth invalid!
9039	GIZ_OPENAPI_NOT_ALLOWED_CALL_API	Not allowed to call deprecated API!
9040	GIZ_OPENAPI_BAD_QRCODE_CONTENT	bad qrcode content!
9041	GIZ_OPENAPI_REQUEST_THROTTLED	request was throttled
9042	GIZ_OPENAPI_DEVICE_OFFLINE	device offline!
9043	GIZ_OPENAPI_TIMESTAMP_INVALID	'X-Gizwits-Timestamp invalid!
9044	GIZ_OPENAPI_SIGNATURE_INVALID	X-Gizwits-Signature invalid!
9045	GIZ_OPENAPI_DEPRECATED_API	API deprecated!
9046	GIZ_OPENAPI_REGISTER_IS_BUSY	Register already in progress!
9056	GIZ_OPENAPI_ALTER_PASSWORD_FAILED	alter password error
9065	GIZ_OPENAPI_APPID_PK_NOT_RELATION	appid has no relation with pk!
9066	GIZ_OPENAPI_CALL_INNER_FAILED	call innerapi failed!
9068	GIZ_OPENAPI_DEVICE_SHARING_NOT_ENABLED	Device share not enabled for this product!
9069	GIZ_OPENAPI_NOT_FIRST_USER_OF_DEVICE	You are not the first user of this device!
9072	GIZ_OPENAPI_PRODUCT_KEY_AUTHEN_FAULT	App auth key invalid!
9073	GIZ_OPENAPI_BUSY_NOW	operation in process, please try again later.
9074	GIZ_OPENAPI_TWITTER_CONSUMER_KEY_INVALID	App twitter consumer_key or consumer_secret not valid.
9075	GIZ_OPENAPI_NOT_ALLOW_WEEK_PASSWORD	weak password not allowed.
9076	GIZ_OPENAPI_CODE_NOT_EXIST	activation code does not exist!
9077	GIZ_OPENAPI_EMAIL_NOT_ACTIVE	email already exists but not activate!
9078	GIZ_OPENAPI_EMAIL_NOT_ENABLE	activation email not enable!
9079	GIZ_OPENAPI_DEVICE_REGISTER_NOT_FOUND	no device register info found!

枚举 ID	枚举定义	描述
9080	GIZ_OPENAPI_CANNOT_SHARE_TO_SELF	can not share device to self!
9081	GIZ_OPENAPI_ONLY_OWNER_CAN_SHARE	guest or normal user can not share device!
9082	GIZ_OPENAPI_NOT_FOUND_GUEST	guest user not found!
9083	GIZ_OPENAPI_GUEST_ALREADY_BOUND	guest user alread bound!
9084	GIZ_OPENAPI_NOT_FOUND_SHARING_INFO	sharing record not found!
9085	GIZ_OPENAPI_NOT_FOUND_THE_MESSAGE	message record not found!
9087	GIZ_OPENAPI_SHARING_IS_WAITING_FOR_ACCEPT	sharing alread created, waiting for the guest to accept!
9088	GIZ_OPENAPI_SHARING_IS_EXPIRED	sharing record expired!
9089	GIZ_OPENAPI_SHARING_IS_COMPLETED	sharing record status is not unaccept!
9090	GIZ_OPENAPI_INVALID_SHARING_BECAUSE_UNBINDING	owner binding disabled!
9092	GIZ_OPENAPI_ONLY_OWNER_CAN_BIND	owner exist, guest can not bind!
9093	GIZ_OPENAPI_ONLY_OWNER_CAN_OPERATE	permission denied, you are not owner!
9094	GIZ_OPENAPI_SHARING_ALREADY_CANCELLED	sharing already canceled!
9095	GIZ_OPENAPI_OWNER_CANNOT_UNBIND_SELF	can not unbind self!
9096	GIZ_OPENAPI_ONLY_GUEST_CAN_CHECK_QRCODE	permission denied, you are not guest!
9098	GIZ_OPENAPI_MESSAGE_ALREADY_DELETED	notify delele binding failed!
9099	GIZ_OPENAPI_BINDING_NOTIFY_FAILED	notify delele binding failed!
9100	GIZ_OPENAPI_ONLY_SELF_CAN_MODIFY_ALIAS	permission denied, you are not owner or guest!
9101	GIZ_OPENAPI_ONLY_RECEIVER_CAN_MARK_MESSAGE	permission denied, you are not the receiver!
9102	GIZ_OPENAPI_GUEST_NOT_BIND	guest not bind
9103	GIZ_OPENAPI_CANNOT_TRANSFER_OWNER_TO_SELF	can not transfer owner privilege to self!
9104	GIZ_OPENAPI_TRANSFER_OWNER_TO_LIMIT_GUEST	can not transfer owner privilege to a time limited guest!
9502	GIZ_OPENAPI_DEVICE_ALREADY_UPGRADE	push rule not in effect or device already upgrade
9999	GIZ_OPENAPI_RESERVED	reserved
10003	GIZ_SITE_PRODUCTKEY_INVALID	产品标识码无效

枚举 ID	枚举定义	描述
10010	GIZ_SITE_DATAPOINTS_NOT_DEFINED	数据点未定义
10011	GIZ_SITE_DATAPOINTS_NOT_MALFORME	数据点异常