# 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<br>设备配置模组类型增加一个自定义枚举值<br>旧的启动接口仍然兼容，但不推荐使用<br>旧的切换域名接口仍然兼容，但不推荐使用<br>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<br>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<br>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<br>增加设备分享接口<br>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 | |

# 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
- didDisableLAN：小循环是否禁用的回调接口
- didDiscovered：设备列表上报的回调接口
- didGetSSIDList：获取设备周围 Wi-Fi 热点列表的回调接口
- didSetDeviceOnboarding：设备配置结果的回调接口
- 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
- didUpdateProduct：设备配置文件上报的回调接口
- 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
- 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
- didChangeUserInfo：修改用户信息结果的回调接口
- didGetUserInfo：获取用户信息的回调接口

# 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]; |

## 【startWithAppID】

| | | |
|---|---|---|
| 定义 | + (void)startWithAppID:(NSString *)appID appSecret:(NSString*)appSecret specialProductKeys:(NSArray *)specialProductKeys cloudServiceInfo:(NSDictionary *)cloudSeviceInfo autoSetDeviceDomain:(BOOL)autoSetDeviceDomain; | |
| 功能描述 | 初始化 SDK。该接口执行后，其他接口功能才能正常执行。如果已经设置了 delegate，SDK 会立即通过 didDiscovered 上报发现的设备。<br><br>如果 App 要做域名切换和设备的 productKey 过滤，建议在 SDK 初始化时就指定好要切换的域名和产品 productKey。<br><br>如果需要设置设备连接的云服务域名，可以在该接口调用时开启自动设置功能。这时 SDK 会让所有支持域名设置的设备都与 App 连接到同一个云服务域名上。但该接口默认是不开启此功能的。<br><br>注意：设备域名自动设置开启后会一直生效，但调用 setDeviceServerInfo 接口时将会终止自动设置 | |
| 参数 | appid | 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。此参数无默认值，开发者必须传入正确的 appID |
| | appSecret | 在机智云开发者中心 dev.gizwits.com 的“应用配置”中，可以看到与 App ID 对应的 App Secret。此参数无默认值，开发者必须传入正确的 appSecret |
| | specialProductKeys | 要过滤的设备产品类型 productKey 列表，为 NSString 数组。此参数默认值为 nil，此时 SDK 返回所有设备。若希望 SDK 只返回过滤后的设备，则参数应指定为需要的设备产品类型 productKey |
| | cloudServiceInfo | 要切换的服务器域名信息。此参数默认值为 nil，此时 SDK 将根据用户手机的地理位置信息为 App 设置机智云统一部署的云服务域名。<br>若 App 希望使用独立部署的私有云服务域名，需按照以下字典 |

| | | |
|---|---|---|
| | | {key: value}格式传值：<br>{<br>"openAPIInfo": "xxx", // NSString类型，api服务域名<br>"siteInfo": "xxx" // NSString类型，site服务域名<br>"pushInfo": "xxx" // NSString类型，推送服务域名<br>}<br>其中，openAPIInfo 和 siteInfo 必须传值，pushInfo 可选。<br><br>可以不指定端口号，SDK 会使用默认的服务端口。此时形如：api.gizwits.com<br>指定端口号时，需同时指定 Http 和 Https 端口。此时形如：xxx.gizwits.com:81&8443 |
| | autoSetDeviceDomain | 是否要开启设备域名的自动设置功能。此参数默认值为 NO，即不开启自动设置。<br>参数值传 YES，则开启设备域名的自动设置功能。如果开启了设备域名的自动设置，小循环设备将被连接到 App 当前使用的云服务域名上 |
| 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<br>[GizWifiSDK sharedInstance].delegate = self;<br>// 设置要过滤的设备 productKey 列表。不需要过滤则不用定义此变量直接传 nil<br>NSArray *specialProductKeys = [NSArray arrayWithObjects: | |

```
@"your_product_key", nil];
// 指定要切换的域名信息。使用机智云生产环境的 App 则传 nil
NSDictionary* cloudServiceInfo = @{@"openAPIInfo": @"your_api_domain",
@"siteInfo": @"your_site_domain"};
// 调用 SDK 的启动接口
[GizWifiSDK startWithAppID:@"your_appid" appSecret:@"your_app_secret"
specialProductKeys:specialProductKeys
cloudServiceInfo:cloudServiceInfo 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);
    } 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);
    }
}
```

## 【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<br>didGetCurrentCloudService:(NSError *)result<br>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：<br>{<br>"openAPIDomain" : "xxx", // NSString类型<br>"openAPIPort" : xxx, // int类型<br>"siteDomain" : "xxx", // NSString类型<br>"sitePort" : xxx, // int类型<br>} |
| 代码示例 | [GizWifiSDK sharedInstance].delegate = self;<br>[GizWifiSDK getCurrentCloudService];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK<br>didGetCurrentCloudService:(NSError *)result<br>cloudServiceInfo:(NSDictionary *)cloudServiceInfo {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 成功<br>} else {<br>// 失败<br>}<br>} | |

## 【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]; |

## 【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。<br>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]; | |

## 【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;<br>[GizWifiSDK disableLAN: YES];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDisableLAN:(NSError *)result {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 成功<br>  } else {<br>  // 失败<br>  }<br>}; | |

## 【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;<br>[[GizWifiSDK sharedInstance] getSSIDList];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetSSIDList:(NSError *)result ssidList:(NSArray *)ssidList {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 获取成功<br>  } else {<br>  // 获取失败<br>  }<br>} |

## 【setDeviceOnboarding】

| | | |
|---|---|---|
| 定义 | - (void)setDeviceOnboarding:(NSString *)ssid key:(NSString *)key configMode:(GizWifiConfigureMode)mode softAPSSIDPrefix:(NSString *)softAPSSIDPrefix timeout:(int)timeout wifiGAgentType:(NSArray *)types; | |
| 功能描述 | 把设备配置到局域网 wifi 上。设备处于 softap 模式时，模组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固件，模组热点名称前缀为 "XPG-GAgent-"，密码为"123456789"。设备处于 airlink 模式时，手机随时都可以开始配置。但无论哪种配置方式，设备上线时，手机要连接到配置的局域网 wifi 上，才能够确认设备已配置成功。<br>设备配置成功时，在回调中会返回设备 mac 地址。如果设备重置了，设备 did 可能要在设备搜索回调中才能获取。 | |
| 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 以此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 |
| | wifiGAgentType | 待配置的模组类型，是一个 GizWifiGAgentType 枚举数组。若不指定则默认配置乐鑫模组。GizWifiGAgentType 定义了 SDK 支持的所有模组类型。GizWifiGAgentType 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 |
| 回调 | - (void)wifiSDK:(GizWifiSDK *)wifiSDK didSetDeviceOnboarding:(NSError | |

| | | |
|---|---|---|
| | *)result mac:(NSString *)mac did:(NSString *)did productKey:(NSString *)productKey; | |
| 回调说明 | 注意：如果调用 getBoundDevices 接口时指定了待筛选的 productKey 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 |
| | mac | 设备 mac 地址 |
| | did | 设备 did。配置成功时，did 的值取决于设备是否有上报 |
| | productKey | 设备的产品类型标识 |
| 代码示例 | [GizWifiSDK sharedInstance].delegate = self;<br><br>// airlink 配置<br>[[GizWifiSDK sharedInstance] setDeviceOnboarding:@"your_ssid" key:@"your_key" mode:GizWifiAirLink softAPSSIDPrefix:nil timeout:60 wifiGAgentType:[NSArray arrayWithObjects: @(GizGAgentESP), nil]];<br><br>// softap 配置<br>[[GizWifiSDK sharedInstance] setDeviceOnboarding:@"your_ssid" key:@"your_key" mode:GizWifiSoftAP softAPSSIDPrefix:@"your_gagent_hotspot_prefix" timeout:60 wifiGAgentType:nil]];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didSetDeviceOnboarding:(NSError *)result mac:(NSString *)mac did:(NSString *)did productKey:(NSString *)productKey {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 配置成功<br>  } else {<br>  // 配置失败<br>  }<br>} | |

## 【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<br>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：<br>{<br>“mac”: “xxx” // 设备 mac 地址<br>“productKey”: “xxx” // 设备的 productKey<br>“domain”: “xxx” // 设备的域名信息<br>} |
| 代码示例 | [GizWifiSDK sharedInstance].delegate = self;<br><br>// 获取可设置域名的设备列表<br>[[GizWifiSDK sharedInstance] getDevicesToSetServerInfo];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK*)wifiSDK<br>didGetDevicesToSetServerInfo:(NSError*)result<br>devices:(NSArray*)devices {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 获取成功<br>} else {<br>// 获取失败<br>}<br>} | |

## 【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。<br><br>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。<br><br>注意：<br>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。<br>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
    }
}
```

## 【getBoundDevices】

| | |
|---|---|
| 定义 | - (void)getBoundDevices:(NSString *)uid token:(NSString *)token specialProductKeys:(NSArray *)specialProductKeys; |
| 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：<br>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；<br>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；<br>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；<br>请注意：此接口传入的 uid、token，如果长度错误，SDK 会继续使用之前的 uid、token 作 |

| | | |
|---|---|---|
| | 处理 | |
| 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 |
| | specialProductKeys | 指定要搜索的产品类型，为 NSString 数组。可以指定一个或多个产品类型，如果不指定则返回所有设备 |
| 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；<br>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;<br>[[GizWifiSDK sharedInstance] getBoundDevices:@"your_uid" token:@"your_token" specialProductKeys:[NSArray arrayWithObjects: @"your_product_key", nil]];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didDiscovered:(NSError *)result deviceList:(NSArray *)deviceList {<br>  // 提示错误原因<br>  if(result.code != GIZ_SDK_SUCCESS) {<br>  NSLog(@"result: %@", result.localizedDescription);<br>  }<br>  // 显示设备列表<br>  NSLog(@"discovered deviceList: %@", deviceList);<br>} | |

## 【bindRemoteDevice】

| | |
|---|---|
| 定义 | - (void)bindRemoteDevice:(NSString *)uid token: (NSString *)token mac:(NSString *)mac productKey:(NSString *)productKey productSecret:(NSString *)productSecret; |

| | | |
|---|---|---|
| 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 |
| 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;<br>[[GizWifiSDK sharedInstance] bindRemoteDevice:@"your_uid" token:@"your_token" mac:@"your_mac" productKey:@"your_product_key" productSecret:@"your_product_secret"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didBindDevice:(NSError *)result did:(NSString *)did {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 绑定成功<br>  } else {<br>  // 绑定失败<br>  }<br>} | |

## 【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;<br>[[GizWifiSDK sharedInstance] unbindDevice:@"your_uid" token:@"your_token" did:@"your_did"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUnbindDevice:(NSError *)result did:(NSString *)did {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 解绑成功<br>  } else {<br>  // 解绑失败<br>  }<br>} | |

## 【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;<br>[[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 *)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
    }
}
```

## 【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;<br>[[GizWifiSDK sharedInstance] requestSendPhoneSMSCode:@"your_token" captchaId:@"your_captcha_id" captchaCode:@"your_captcha_code" phone:@"your_phone_number"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didRequestSendPhoneSMSCode:(NSError *)result token:(NSString *)token {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 获取成功<br>  } else {<br>  // 获取失败<br>  }<br>} | |

## 【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;<br>[[GizWifiSDK sharedInstance] verifyPhoneSMSCode:@"your_token" verifyCode:@"your_verify_code" phone:@"your_phone_number"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didVerifyPhoneSMSCode:(NSError *)result {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 验证成功<br>} else {<br>// 验证失败<br>}<br>} | |

## 【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 | |
| 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 |
| 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;<br>[[GizWifiSDK sharedInstance] registerUser:@"your_phone_number" password:@"your_password" verifyCode:@"your_verify_code" accountType:GizUserPhone];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didRegisterUser:(NSError *)result uid:(NSString *)uid token:(NSString *)token {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 注册成功<br>  } else {<br>  // 注册失败<br>  }<br>} | |

## 【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;<br>[[GizWifiSDK sharedInstance] userLoginAnonymous];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 登录成功 | |

| | |
|---|---|
| | } else {<br>// 登录失败<br>}<br>} |

## 【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 |
| 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;<br>[[GizWifiSDK sharedInstance] userLogin:@"your_user_name" password:@"your_user_password"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 登录成功<br>} else {<br>// 登录失败<br>}<br>} | |

## 【userLoginWithThirdAccount】

| | |
|---|---|
| 定义 | - (void)userLoginWithThirdAccount:(GizThirdAccountType)thirdAccountType uid:(NSString *)uid token:(NSString *)token; |

| | | |
|---|---|---|
| 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 |
| 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;<br>[[GizWifiSDK sharedInstance] userLoginWithThirdAccount:GizThirdBAIDU uid:@"your_third_uid" token:@"your_third_token"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didUserLogin:(NSError *)result uid:(NSString *)uid token:(NSString *)token {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 登录成功<br>  } else {<br>  // 登录失败<br>  }<br>} | |

## 【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 |
| 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;<br>[[GizWifiSDK sharedInstance] changeUserPassword:@"your_token" oldPassword:@"your_old_password" newPassword:@"your_new_password"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserPassword:(NSError *)result {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 修改成功<br>  } else {<br>  // 修改失败<br>  }<br>} | |

## 【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 |
| 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;<br>[[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
    }
}
```

## 【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，并且手机号或邮箱必须是已经注册过的。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;<br><br>// 修改手机号<br>[[GizWifiSDK sharedInstance] changeUserInfo:@"your_token" username:@"your_phone_number" SMSVerifyCode:@"your_verify_code" userType:GizUserPhone additionalInfo:nil];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didChangeUserInfo:(NSError *)result {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 修改成功<br>  } else {<br>  // 修改失败<br>  }<br>} |

## 【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;<br>[[GizWifiSDK sharedInstance] getUserInfo:@"your_token"];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didGetUserInfo:(NSError *)result userInfo:(GizUserInfo *)userInfo {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 获取成功<br>  } else {<br>  // 获取失败<br>  } | |

| | |
|---|---|
| | } |

## 【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 |
| 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;<br>[[GizWifiSDK sharedInstance] transAnonymousUser:@"your_token" username:@"your_phone_number" password:@"your_password" verifyCode:@"your_verify_code" accountType:GizUserPhone];<br><br>// 实现回调<br>- (void)wifiSDK:(GizWifiSDK *)wifiSDK didTransAnonymousUser:(NSError *)result {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 转换成功<br>  } else {<br>  // 转换失败<br>  }<br>} | |

## 【userFeedback】

| | |
|---|---|
| 定义 | + (void)userFeedback:(NSString*)contactInfo |

| | | |
|---|---|---|
| | feedbackInfo:(NSString*)feedbackInfo sendLog:(BOOL)sendLog; | |
| 功能描述 | 用户反馈，此接口无回调。调用后就会上传信息 | |
| 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]; | |

# 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

| 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 |
| netStatus | GizWifiDeviceNetStatus 类型。设备的网络状态 |
| 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 |

| 属性 | 描述 |
| --- | --- |
| 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 |
| sharingRole | GizDeviceSharingUserRole 类型。表示绑定设备的用户具有的权限 |

## 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：

- 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
- didSetSubscribe：设备订阅或解除订阅的回调
- didUpdateNetStatus：设备网络状态变化通知
- didReceiveData：接收到设备状态上报的回调

## 2.4. API

### 【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<br>mDevice.delegate = self;<br><br>// 实现回调<br>- (void)device:(GizWifiDevice *)device didUpdateNetStatus:(GizWifiDeviceNetStatus)netStatus {<br>} | |

### 【setSubscribe】

| | |
| --- | --- |
| 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。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 | 设备的产品秘钥。在机智云开发者中心 dev.gizwits.com 的“产品信息”中，可以看到与 Product Key 对应的 Product Secret。此参数无默认值，开发者必须传入正确的 productSecret |
| | 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<br>mDevice.delegate = self;<br>[mDevice setSubscribe:@"your_product_secret" subscribed:YES]; // 订阅设备<br>[mDevice setSubscribe:@"your_product_secret" subscribed:NO]; // 解除订阅<br><br>// 实现回调<br>- (void)device:(GizWifiDevice *)device didSetSubscribe:(NSError *)result isSubscribed:(BOOL)isSubscribed {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 订阅或解除订阅成功<br>  } else {<br>  // 操作失败<br>  }<br>} | |

## 【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：<br>{<br>"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<br>"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<br>"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<br>"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<br>} |
| | 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<br>mDevice.delegate = self;<br>[mDevice getDeviceStatus:nil];<br><br>// 实现回调<br>- (void)device:(GizWifiDevice *)device didReceiveData:(NSError *)result data:(NSDictionary *)data withSN:(NSNumber *)sn {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 查询成功<br>} else {<br>// 查询失败<br>}<br>} | |

## 【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：<br>（1）如果数据点为布尔类型，则 value 为 NSNumber 类型；<br>（2）如果数据点为数值类型，则 value 为 NSNumber 类型；<br>（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）；<br>如果数据点为扩展类型，则 value 为 NSData 类型；<br>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 | 控制指令序号，用于对应控制指令应答数据。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：<br>{<br>"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<br>"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<br>"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<br>"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<br>} |
| | 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<br>mDevice.delegate = self;<br>// 开灯<br>int sn = 5;<br>[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 | |
| 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：<br>{<br>"wifiHardVersion": [value], // value 为 NSString 类型，设备的 Wifi 模组硬件版本号<br>"wifiSoftVersion": [value], // value 为 NSString 类型，设备的 Wifi 模组软件版本号<br>"wifiFirmwareId": [value], // value 为 NSString 类型，设备的 Wifi 固件 ID<br>"wifiFirmwareVer": [value], // value 为 NSString 类型，设备的 Wifi 固件版本<br>"mcuHardVersion": [value], // value 为 NSString 类型，设备的硬件版本号<br>"mcuSoftVersion": [value], // value 为 NSString 类型，设备的软件版本号<br>"productKey": [value], // value 为 NSString 类型，设备的产品唯一标识码<br>} |
| 代码示例 | // mDevice为从设备列表中取到的设备对象<br>mDevice.delegate = self;<br>[mDevice getHardwareInfo];<br><br>// 实现回调<br>- (void)device:(GizWifiDevice *)device didGetHardwareInfo:(NSError *)result hardwareInfo:(NSDictionary *)hardwareInfo {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 获取成功<br>} else {<br>// 获取失败<br>}<br>} | |

### 【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<br>mDevice.delegate = self;<br>[mDevice exitProductionTesting];<br><br>// 实现回调<br>- (void)device:(GizWifiDevice *)device didExitProductionTesting:(NSError *)result {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 成功<br>  } else {<br>  // 失败<br>  }<br>} | |

# 3. GizWifiCentralControlDevice 类

## 3.1. 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。

## 3.2. 属性

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

## 3.3. 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

## 3.4. API

### 【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。<br>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<br>mDevice.delegate = self;<br><br>// 实现回调<br>- (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 接收变更的子设备列表<br>  } else {<br>  // 失败处理<br>  }<br>} | |

### 【addSubDevice】

| | | |
|---|---|---|
| 定义 | - (void)addSubDevice:(NSArray*)deviceMacs; | |
| 功能描述 | 添加子设备，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<br>mDevice.delegate = self;<br>[mDevice addSubDevice:nil];<br><br>// 实现回调<br>- (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 添加成功<br>} else {<br>// 添加失败<br>}<br>} | |

## 【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 | deviceID | 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 |
| 代码示例 | // mDevice为从设备列表中取到的中控设备对象<br>mDevice.delegate = self;<br>// mSubDevice是从子设备列表中获取到的要删除的设备实体对象<br>[mDevice deleteSubDevice: mSubDevice];<br><br>// 实现回调<br>- (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 删除成功<br>} else {<br>// 删除失败<br>}<br>} | |

### 【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调用该接口。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<br>mDevice.delegate = self;<br>[mDevice updateSubDevices];<br><br>// 实现回调<br>- (void)didUpdateSubDevices:(GizWifiCentralControlDevice *)device result:(NSError*)result subDeviceList:(NSArray *)subDeviceList {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 获取成功<br>  } else {<br>  // 获取失败<br>  }<br>} |

# 4. GizUserInfo 类

## 4.1. 简介

GizUserInfo 类为开发者提供用户信息存取属性。

## 4.2. 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 |

| 属性 | 描述 |
| --- | --- |
| 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 |

# 5. GizWifiSSID 类

## 5.1. 简介

路由的 SSID 信息类，包括 SSID 名和信号强度。

## 5.2. 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 |

# 6. GizDeviceSchedulerCenter 类

## 6.1. 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。

## 6.2. 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：

- didUpdateSchedulers：定时任务列表回调

## 6.3. API

### 【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。创建定时任务、修改定时任务信息、删除定时任务、同步更新定时任务列表都使用该回调接口。 |

| | | |
|---|---|---|
| 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。GizWifiDevice 对象数组 |
| 代码示例 | // mDevice为从设备列表中取到的设备对象<br>mDevice.delegate = self;<br><br>// 实现回调<br>- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner result:(NSError*)result schedulerList:(NSArray*)schedulerList {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 接收变更的定时任务列表<br>} else {<br>// 失败处理<br>}<br>} | |

## 【setDelegate】

| | | |
|---|---|---|
| 定义 | + (void)setDelegate:(id <GizDeviceSchedulerCenterDelegate>)delegate; | |
| 功能描述 | 设置定时任务委托 | |
| 参数 | delegate | 定时任务委托 |
| 代码示例 | [GizDeviceSchedulerCenter setDelegate:self]; | |

## 【createScheduler】

| | | |
|---|---|---|
| 定义 | + (void)createScheduler:(NSString*)uid token(NSString*)token schedulerOwner:(GizWifiDevice*)schedulerOwner scheduler:(GizDeviceScheduler*)scheduler; | |
| 功能描述 | 创建设备定时任务。每一个定时任务创建成功后都会被分配一个定时任务 ID，SDK 通过回调接口 didUpdateSchedulers 给 App 返回创建结果。创建普通设备的定时任务时，只返回成功或失败的错误信息，不返回定时任务列表 | |
| 参数 | uid | 用户 uid |
| | token | 用户 token |
| | schedulerOwner | 执行定时任务的设备对象 |

| | | |
|---|---|---|
| | scheduler | 定时任务内容。需要创建一个 GizDeviceScheduler 类对象，填写好定时任务内容，在接口调用时传这个 GizDeviceScheduler 对象 |

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 alloc] init];
scheduler.date = @"2017-01-16";
scheduler.time = @"06:30";
scheduler.remark = @"开灯任务";
scheduler.attrs = @{@"LED_OnOff": @YES};

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

// 创建设备的定时任务，mDevice 为从设备列表中取到的要创建定时任务的设备对象
[GizDeviceSchedulerCenter createScheduler:@"your_uid" token:@"your_token" schedulerOwner:mDevice scheduler:scheduler];

// 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
    } else {
        // 创建失败
    }
}
```

## 【deleteScheduler】

| | | |
|---|---|---|
| 定义 | + (void)deleteScheduler:(NSString *)uid token:(NSString *)token schedulerOwner:(GizWifiDevice *)schedulerOwner schedulerID:(NSString *)schedulerID; | |
| 功能描述 | 删除设备定时任务。在定时任务列表中找到要删除的定时任务 ID，就可以删除了，SDK 通过回调接口 didUpdateSchedulers 给 App 返回删除结果。删除普通设备的定时任务时，只返回成功或失败的错误信息，不返回定时任务列表 | |
| 参数 | uid | 用户 uid |
| | token | 用户 token |

| | | |
|---|---|---|
| | schedulerOwner | 删除定时任务的设备对象 |
| | schedulerID | 要删除的定时任务 ID |
| 代码示例 | // 设置定时任务委托<br>[GizDeviceSchedulerCenter setDelegate:self];<br><br>// 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<br>[GizDeviceSchedulerCenter deleteScheduler:@"your_uid"<br>token:@"your_token" schedulerOwner:mDevice<br>schedulerID:@"schedulerID_to_delete"];<br><br>// 实现回调<br>- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner<br>result:(NSError*)result schedulerList:(NSArray*)schedulerList {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 定时任务删除成功<br>} else {<br>// 删除失败<br>}<br>} | |

## 【editScheduler】

| | | |
|---|---|---|
| 定义 | + (void)editScheduler:(NSString *)uid token:(NSString *)token<br>schedulerOwner:(GizWifiDevice *)schedulerOwner<br>scheduler:(GizDeviceScheduler*)scheduler; | |
| 功能描述 | 修改设备定时任务。要修改的定时任务必须是已经创建过的，App 编辑好修改内容后，SDK 通过回调接口 didUpdateSchedulers 给 App 返回修改结果。修改普通设备的定时任务时，只返回成功或失败的错误信息，不返回定时任务列表 | |
| 参数 | uid | 用户 uid |
| | token | 用户 token |
| | schedulerOwner | 修改定时任务的设备对象 |
| | schedulerID | 要修改的定时任务对象。App 在已经得到的定时任务列表中找到要修改的定时任务对象，修改好内容后，在接口调用时传这个对象 |
| 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<br>scheduler.date = @"2017-01-16";<br>scheduler.time = @"06:30";<br>scheduler.remark = @"开灯任务"; | |

```
scheduler.attrs = @{@"LED_OnOff": @YES};
scheduler.monthDays = @[@1, @15];

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

// 修改设备的定时任务，mDevice 是设备列表中要创建定时任务的设备对象
[GizDeviceSchedulerCenter editScheduler:@"your_uid"
token:@"your_token" schedulerOwner:mDevice scheduler:scheduler];

// 实现回调
- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner
result:(NSError*)result schedulerList:(NSArray*)schedulerList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 定时任务修改成功
    } else {
        // 修改失败
    }
}
```

## 【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。App 调用此接口可以与当前设备上的定时任务列表保持同步，SDK 通过回调接口 didUpdateSchedulers 返回同步更新结果。成功时返回最新的定时任务列表，失败时返回错误信息 | |
| 参数 | uid | 用户 uid |
| | token | 用户 token |
| | schedulerOwner | 同步更新定时任务列表的设备对象 |
| 代码示例 | // 设置定时任务委托<br>[GizDeviceSchedulerCenter setDelegate:self];<br><br>// 与设备的同步定时任务，mDevice为从设备列表中取到的要同步更新定时任务的设备对象<br>[GizDeviceSchedulerCenter updateSchedulers:@"your_uid"<br>token:@"your_token" schedulerOwner:mDevice];<br><br>// 实现回调<br>- (void)didUpdateSchedulers:(GizWifiDevice*)schedulerOwner | |

```
result:(NSError*)result schedulerList:(NSArray*)schedulerList {
    if(result.code == GIZ_SDK_SUCCESS) {
        // 定时任务同步成功
    } else {
        // 同步失败
    }
}
```

# 7. GizDeviceScheduler 类

## 7.1. 简介

GizDeviceScheduler 类是基于设备的定时任务类，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。

## 7.2. 属性

| 属性 | 描述 |
| --- | --- |
| schedulerID | NSString 类型，只读不可写。定时任务 ID，定时任务创建成功时会被分配一个 ID |
| createdDateTime | NSString 类型，只读不可写。定时任务的创建时间 |
| date | NSString 类型，可读写。定时任务的执行日期，年月日以“－”符号分割，例如：2017-01-30。此变量默认值为 nil |
| time | NSString 类型，可读写。定时任务的执行时间，24 小时制，例如：06:30。无论是一次性定时任务还是按周、按月重复的定时任务，必须指定执行时间，此变量必须有值 |
| weekdays | GizScheduleWeekday 枚举类型数组，可读写。此变量为 nil，表示定时任务不需要按周重复。需要按周重复时，此变量可填写为一周中的某一天或者某几天，例如：想在周一和周三执行定时任务，就把 GizScheduleMonday、GizScheduleWednesday 这两个值放到数组里。此变量默认值为 nil。<br>说明：定时任务是按周重复还是按月重复，只能二选一。如果定时任务是按周重复，则忽略按月重复。例如：weekdays 和 monthDays 都被赋值时，会优先取 weekdays 的值 |
| monthDays | NSInteger 类型数组，可读写。此变量为 nil，表示定时任务不需要按月重复。需要按月重复时，此变量可填写为一个月中的某一天或某几天。例如，想在一个月的 1 号和 15 号执行定时任务，可把 1、15 这两个值放到数组里。此变量默认值为 nil。<br>说明：定时任务是按周重复还是按月重复，只能二选一。如果定时任务是按月重复，需要把按周重复变量 weekdays 清空，按月重复才能生效 |

| 属性 | 描述 |
| --- | --- |
| enabled | BOOL 类型，可读写。值为 YES 表示启动定时任务，默认值为 NO |
| remark | NSString 类型，可读写。定时任务备注信息，默认值为 nil |
| startDate | NSString 类型，可读写。定时任务启动日期，年月日以“－”符号分割。格式为：2017-01-29。默认值为 nil |
| endDate | NSString 类型，可读写。定时任务结束日期，年月日以“－”符号分割。格式为：2017-02-01。默认值为 nil |
| attrs | NSDictionary 类型，可读写。定时任务要执行的动作，为数据点名称和值的键值对。定时任务必须指定要执行的动作，否则无法创建定时任务。此变量必须有值 |

# 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<br>[GizDeviceSharing setDelegate:self];<br><br>// 解绑其他用户<br>[GizDeviceSharing unbindUser:@"your_token" deviceID: @"your_device_id" guestUID:@"guest_uid_to_unbind"];<br><br>// 实现回调<br>- (void)didUnbindUser:(NSError*)result deviceID:(NSString *)deviceID guestUID:(NSString*)guestUID {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 解绑成功<br>  } else {<br>  // 解绑失败<br>  }<br>} | |

## 【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<br>deviceID:(NSString*)deviceID<br>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<br>sharingWay:(GizDeviceSharingWay)sharingWay<br>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
```

| | |
|---|---|
| | }<br>} |

## 【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<br>[GizDeviceSharing setDelegate:self];<br><br>// 撤回分享邀请<br>[GizDeviceSharing revokeDeviceSharing:@"your_token" sharingID: your_sharing_id];<br><br>// 实现回调<br>- (void)didRevokeDeviceSharing:(NSError*)result sharingID:(NSInteger)sharingID {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 撤回成功<br>} else {<br>// 撤回失败<br>}<br>} | |

## 【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<br>[GizDeviceSharing setDelegate:self];<br><br>// 接受邀请<br>[GizDeviceSharing acceptDeviceSharing:@"your_token" sharingID: your_sharing_id accept:YES];<br><br>// 实现回调<br>- (void)didAcceptDeviceSharing:(NSError*)result sharingID:(NSInteger)sharingID {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 接受成功<br>  } else {<br>  // 接受失败<br>  }<br>} | |

## 【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<br>[GizDeviceSharing setDelegate:self];<br><br>// 查看扫码邀请信息<br>[GizDeviceSharing checkDeviceSharingInfoByQRCode:@"your_token" QRCode:@"your_sharing_code"];<br><br>// 实现回调<br>- (void)didCheckDeviceSharingInfoByQRCode:(NSError*)result userName:(NSString*)userName productName:(NSString*)productName deviceAlias:(NSString*)deviceAlias expiredAt:(NSString*)expiredAt {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 成功<br>  } else {<br>  // 失败<br>  }<br>} | |

## 【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) {<br>// 成功<br>} else {<br>// 失败<br>}<br>} |

## 【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<br>[GizDeviceSharing setDelegate:self];<br><br>// 查询消息列表<br>[GizDeviceSharing queryMessageList:@"your_token" messageType: GizMessageSharing];<br><br>// 实现回调<br>- (void)didQueryMessageList:(NSError*)result messageList:(NSArray*)messageList {<br>if(result.code == GIZ_SDK_SUCCESS) {<br>// 成功<br>} else {<br>// 失败<br>}<br>} | |

### 【markMessageStatus】

| | | |
|---|---|---|
| 定义 | + (void)markMessageStatus:(NSString*)token<br>messageID:(NSString*)messageID<br>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<br>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<br>[GizDeviceSharing setDelegate:self];<br><br>// 标记已读<br>[GizDeviceSharing markMessageStatus:@"your_token" messageID :<br>@"your_message_id"  messageType: GizMessageRead];<br><br>// 实现回调<br>- (void)didMarkMessageStatus:(NSError*)result<br>messageID:(NSString*)messageID {<br>  if(result.code == GIZ_SDK_SUCCESS) {<br>  // 成功<br>  } else {<br>  // 失败<br>  }<br>} | |

# 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 |
| type | GizDeviceSharingType 枚举类型，只读不可写。分享邀请是分享给自己的还是自己分享给别人的 |
| 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. 枚举定义

## 11.1. 简介

本节说明 GizWifiSDK 中使用的所有枚举定义。

## 11.2. 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 |

### 【GizWifiDeviceType】

功能描述：设备类型。

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 0 | GizDeviceNormal | 普通设备 |
| 1 | GizDeviceCenterControl | 中控设备 |
| 2 | GizDeviceSub | 子设备 |

## 【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 |

## 【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 |
| 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） |

## 【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 | 庆科模组 |

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 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 |
| 8 | GizGAgentMXCHIP3 | 庆科 mico 模组 |
| 9 | GizGAgentBL | 古北模组 |
| 10 | GizGAgentAtmelEE | Atmel 模组 |
| 11 | GizGAgentOther | 其他模组 |

## 【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） |
| 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 |

## 【GizWifErrorCode】

功能描述：错误码定义。

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 0 | GIZ_SDK_SUCCESS | Client 发出的请求执行成功 |
| 8001 | GIZ_SDK_PARAM_FORM_INVALID | Client 发给 Daemon 的 json 格式错误 |
| 8002 | GIZ_SDK_CLIENT_NOT_AUTHEN | Client 与 Daemon 之间如果没有通过握手认证，任何数据交互都无效 |
| 8003 | GIZ_SDK_CLIENT_VERSION_INVALID | Client 版本号无效 |
| 8004 | GIZ_SDK_UDP_PORT_BIND_FAILED | udp 端口绑定失败 |
| 8005 | GIZ_SDK_DAEMON_EXCEPTION | Daemon 系统错误 |
| 8006 | GIZ_SDK_PARAM_INVALID | Client 发出的数据请求，Json 格式正确，但参数无效；APP 传入参数无效 |
| 8007 | GIZ_SDK_APPID_LENGTH_ERROR | appid 长度错误 |
| 8008 | GIZ_SDK_LOG_PATH_INVALID | 日志路径无效 |
| 8009 | GIZ_SDK_LOG_LEVEL_INVALID | 日志级别无效 |
| 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 | 设备正在配置 |
| 8023 | GIZ_SDK_DEVICE_CONFIG_TIMEOUT | 设备配置超时 |
| 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 did 无效 |
| 8027 | GIZ_SDK_DEVICE_PASSCODE_INVALID | 设备 passcode 无效 |
| 8028 | GIZ_SDK_DEVICE_NOT_CENTERCONTROL | 不是中控设备 |
| 8029 | GIZ_SDK_DEVICE_NOT_SUBSCRIBED | 设备未订阅 |
| 8030 | GIZ_SDK_DEVICE_NO_RESPONSE | 设备未响应 |
| 8031 | GIZ_SDK_DEVICE_NOT_READY | 设备未就绪 |
| 8032 | GIZ_SDK_DEVICE_NOT_BINDED | 设备未绑定 |
| 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 | 设备控制指令执行失败 |
| 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 |

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 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 连接成功 |
| 8043 | GIZ_SDK_SET_SOCKET_NON_BLOCK_FAILED | socket 设置非阻塞失败 |
| 8044 | GIZ_SDK_CONNECTION_TIMEOUT | 连接超时 |
| 8045 | GIZ_SDK_CONNECTION_REFUSED | 连接被拒绝 |
| 8046 | GIZ_SDK_CONNECTION_ERROR | 连接错误 |
| 8047 | GIZ_SDK_CONNECTION_CLOSED | 连接被关闭 |
| 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 | 当前外网不可达 |
| 8095 | GIZ_SDK_HTTP_SERVER_NOT_SUPPORT_API | HTTP 服务不支持此 API |
| 8096 | GIZ_SDK_HTTP_ANSWER_FORMAT_ERROR | openapi 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，比如返回 404 等 |
| 8100 | GIZ_SDK_OTHERWISE | 其他错误 |
| 8101 | GIZ_SDK_MEMORY_MALLOC_FAILED | Daemon 内存分配失败 |
| 8102 | GIZ_SDK_THREAD_CREATE_FAILED | Daemon 内部线程创建失败 |
| 8150 | GIZ_SDK_USER_ID_INVALID | 用户 ID 无效 |
| 8151 | GIZ_SDK_TOKEN_INVALID | 用户 token 无效 |
| 8152 | GIZ_SDK_GROUP_ID_INVALID | 组 ID 无效 |
| 8153 | GIZ_SDK_GROUPNAME_INVALID | 组名称无效 |
| 8154 | GIZ_SDK_GROUP_PRODUCTKEY_INVALID | 组类型无效 |
| 8155 | GIZ_SDK_GROUP_FAILED_DELETE_DEVICE | 组设备删除失败 |
| 8156 | GIZ_SDK_GROUP_FAILED_ADD_DEVICE | 组设备添加失败 |
| 8157 | GIZ_SDK_GROUP_GET_DEVICE_FAILED | 组设备获取失败 |
| 8201 | GIZ_SDK_DATAPOINT_NOT_DOWNLOAD | 配置文件还未下载 |

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 8202 | GIZ_SDK_DATAPOINT_SERVICE_UNAVAILABLE | 配置文件服务不可用 |
| 8203 | GIZ_SDK_DATAPOINT_PARSE_FAILED | 配置文件解析失败 |
| 8300 | GIZ_SDK_SDK_NOT_INITIALIZED | SDK 未初始化 |
| 8301 | GIZ_SDK_APK_CONTEXT_IS_NULL | Context 无效，无法启动 |
| 8302 | GIZ_SDK_APK_PERMISSION_NOT_SET | app 权限不足 |
| 8303 | GIZ_SDK_CHMOD_DAEMON_REFUSED | 无法修改 daemon 的执行权限 |
| 8304 | GIZ_SDK_EXEC_DAEMON_FAILED | daemon 程序执行失败 |
| 8305 | GIZ_SDK_EXEC_CATCH_EXCEPTION | 尝试运行 daemon 时发生异常 |
| 8306 | GIZ_SDK_APPID_IS_EMPTY | APPID 为空 |
| 8307 | GIZ_SDK_UNSUPPORTED_API | 不支持的 API |
| 8308 | GIZ_SDK_REQUEST_TIMEOUT | Client 如果等不到 Daemon 的回复，就向 APP 返回操作超时 |
| 8309 | GIZ_SDK_DAEMON_VERSION_INVALID | Daemon 版本号无效 |
| 8310 | GIZ_SDK_PHONE_NOT_CONNECT_TO_SOFTAP_SSID | 手机没有连接软 AP 热点 |
| 8311 | GIZ_SDK_DEVICE_CONFIG_SSID_NOT_MATCHED | 手机热点和要配置的路由 ssid 不匹配 |
| 8312 | GIZ_SDK_NOT_IN_SOFTAPMODE | 设备不在 softap 模式 |
| 8313 | GIZ_SDK_CONFIG_NO_AVAILABLE_WIFI | 设备配置时无可用 wifi |
| 8314 | GIZ_SDK_RAW_DATA_TRANSMIT | 设备上报透传数据的标识 |
| 8315 | GIZ_SDK_PRODUCT_IS_DOWNLOADING | 正在下载设备的产品定义 |
| 8316 | GIZ_SDK_START_SUCCESS | SDK 启动成功 |
| 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 |
| 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! |

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 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 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! |

| 枚举 ID | 枚举定义 | 描述 |
| --- | --- | --- |
| 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! |
| 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! |
| 9999 | GIZ_OPENAPI_RESERVED | reserved |