



Midea Open Smart™ Development Guide

MSmartSDK and HTTP API

V1.0.9

REVISION HISTORY.....	- 2 -
1. Overview.....	- 3 -
2. MSmartSDK & HTTP API	- 4 -
2.1. MSmartSDK	- 4 -
2.2. HTTP API for User Account Management	- 4 -
2.3. HTTP API for Appliance Management.....	- 4 -
3. Developing with MSmartSDK.....	- 5 -
3.1. Importing MSmartSDK into APP project.....	- 5 -
3.1.1. Importing in Android.....	- 5 -
3.1.2. Importing in IOS	- 7 -
3.2. Initializing MSmartSDK.....	- 9 -
3.2.1. Key parameters for initialization.....	- 9 -
3.2.2. Initializing in Android.....	- 10 -
3.2.3. Initializing in IOS.....	- 11 -
3.3. Calling MSmartSDK.....	- 12 -
3.3.1. Managing Device Token.....	- 12 -
3.3.2. configure appliance network by Wi-Fi AP	- 13 -
3.3.3. configure appliance network by Bluetooth.....	- 18 -
3.3.4. Error code definition.....	- 23 -
3.3.5. Miscellanea.....	- 23 -
3.4. Connecting appliance to Midea IoT Core.....	- 24 -
4. HTTP API Reference.....	- 27 -
4.1. HTTP API for User Account Management	- 27 -
4.1.1. HTTP Common header	- 27 -
4.1.2. Security	- 28 -
4.1.3. API	- 31 -
4.1.4. Response Code definition.....	- 56 -
4.2. HTTP API for Appliance Management.....	- 57 -
4.2.1. HTTP Common header	- 57 -
4.2.2. Security	- 57 -
4.2.3. API	- 61 -
4.2.4. Response Code definition.....	- 83 -
5. Receiving appliance upload data.....	- 84 -
6. Appendixes.....	- 87 -
6.1. Error code of MSmartSDK.....	- 87 -
6.2. Thing model definition for Air Conditioner.....	- 101 -

REVISION HISTORY

Version	Author	Date	Comment
V1.0.0	Lauman Liu	2022-08-02	This document content describes content about MSmartSDK & HTTP API.
V1.0.1	Lauman Liu	2022-08-08	1. Add control API based on thing model. 2. Add thing model of air conditioner to appendix.
v1.0.2	Lauman Liu	2022-8-16	1. Adjust page layout of AC thing model. 2. Add error codes for “Send hexadecimal command” HTTP API.
V1.0.3	Lauman Liu	2022-9-06	1. Revised thing model definition for air conditioner product. 2. Added signature calculation description for appliance management API.
V1.0.4	Lauman Liu	2022-12-01	Modified the architecture chart in “Overview” section.
v1.0.5	Lauman Liu	2022-12-23	1. Updated signature calculating paragraph. 2. Updated response parameters of login API.
v1.0.6	Lauman Liu	2023-1-11	Updated the AC thing model.
v1.0.7	Lauman Liu	2023-1-16	Added a new chapter about receiving appliance upload data.
v1.0.8	Lauman Liu	2023-3-6	Updated the description about calculating signature in “http api appliance management” section.
V1.0.9	Lauman Liu	2023-3-23	Supplemented description to configure appliance network by Wi-FI AP.

1. Overview

Midea Open Smart is a solution that allow Midea partners to integrate Midea IoT feature into their business.

As a leading player in appliance design and manufacture industry, Midea has been dedicating to make consumer life more wonderful by continuous product and technical innovation. With widely use of IoT technology in recent years, Midea also constructed own IoT platform and keeps it evolving.

Midea IoT Open Cloud, which is based on Midea IoT Core, is a cloud service providing IoT-related business support with SaaS model.

Midea Open Smart, a suite of client tools for Midea IoT Open Cloud, can be used to integrate Midea IoT capabilities into mobile APP or cloud service. It enables third-party application easily to access Midea IoT feature through Midea IoT Open Cloud, such as to sign up for a user account, to setup network connection for appliance, to control appliance through mobile APP instead of a hand-held remote controller etc.

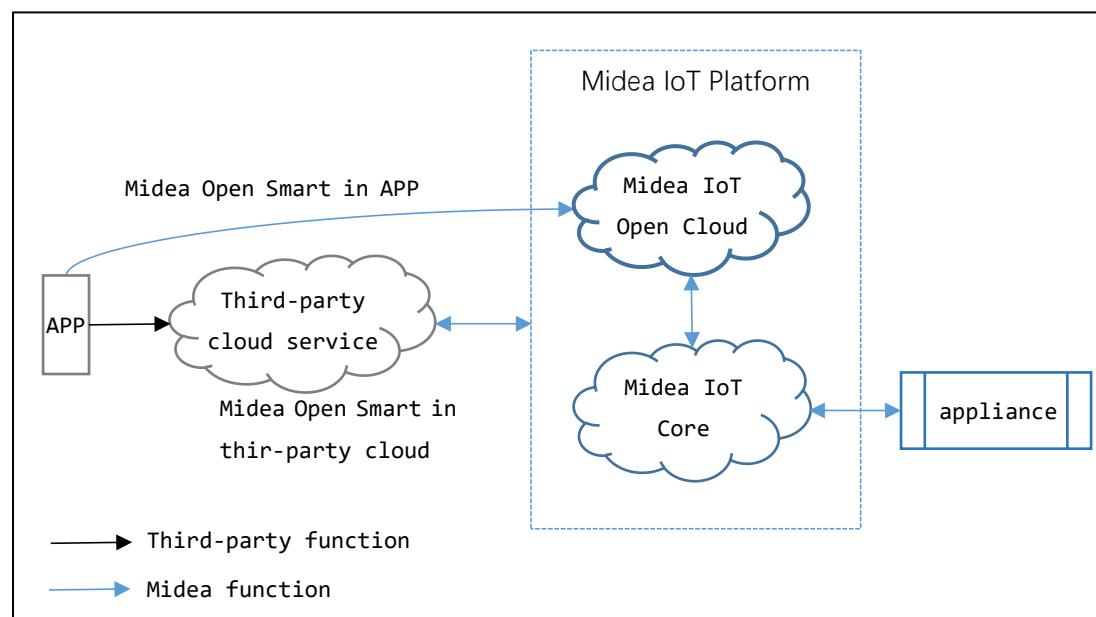


Chart: The overall architecture

MSmartSDK and HTTP API, a subset of Midea Open Smart, can be used to do network configuration on appliance, create user account for the end user, and control the appliance via mobile APP.

2. MSmartSDK & HTTP API

Midea Open Smart contains a series of SDK modules for mobile APP development, and a set of http API, which can use in APP or cloud server, in actuality, the SDK module used in APP also call the http APIs.

2.1. MSmartSDK

This SDK is mainly used to set the credentials for appliance so that appliance can access to internet, we call this process as “appliance network configuration process”, also known as “IoT commissioning process”.

MSmartSDK can complete the network configuration via Bluetooth or Wi-Fi AP connection.

2.2. HTTP API for User Account Management

User Account Management API can be used to create and manage user account for your mobile APP. You can create, modify, and delete user account, login a user account to Midea IoT Open Cloud, even use a third-party account to login. These API is multilingual, you can specify the language code according to where the end user locates.

Note: if you have an existing user account system, you don't have to use these APIs, just feel free to keep using your account system.

2.3. HTTP API for Appliance Management

Appliance Management API provides the capability to retrieve related information when do the network configuration via MSmartSDK, to manage user appliance list and control them by your APP, it has a different definition specification to User Account Management API.

3. Developing with MSmartSDK

3.1. Importing MSmartSDK into APP project

3.1.1. Importing in Android

MSmartSDK is a standalone .aar file, and you need to copy it into the “app\libs” folder of your android project.

- Add following lines in build.gradle file located in “\app” folder of android project.

```
1. repositories {  
2.     flatDir {  
3.         dirs 'libs' //specify the path of MSmartSDK module here  
4.     }  
5. }
```

- Add dependencies for SDK modules in **build.gradle** file

```
1. dependencies {  
2.     //dependency for third-party libs  
3.     implementation "com.google.code.gson:gson:2.8.0"  
4.     implementation 'androidx.annotation:annotation:1.0.0'  
5.     implementation 'com.madgag.spongycastle:core:1.54.0.0'  
6.     implementation 'com.madgag.spongycastle:prov:1.54.0.0'  
7.     implementation 'com.madgag.spongycastle:pkix:1.54.0.0'  
8.     implementation 'com.madgag.spongycastle:pg:1.54.0.0'  
9.  
10.    //dependency for MSmartSDK  
11.    implementation(name: 'xxx.aar', ext: 'aar')  
12. }
```

- Apply for following permissions in **AndroidManifest.xml** file.

```
1.<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>  
2.<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>  
3.<uses-permission android:name="android.permission.BROADCAST_STICKY"/>  
4.<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>  
5.<uses-permission android:name="android.permission.INTERNET"/>  
6.<uses-permission android:name="android.permission.BLUETOOTH" />  
7.<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />  
8.<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
9.<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

```
|10. <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
|               />
```

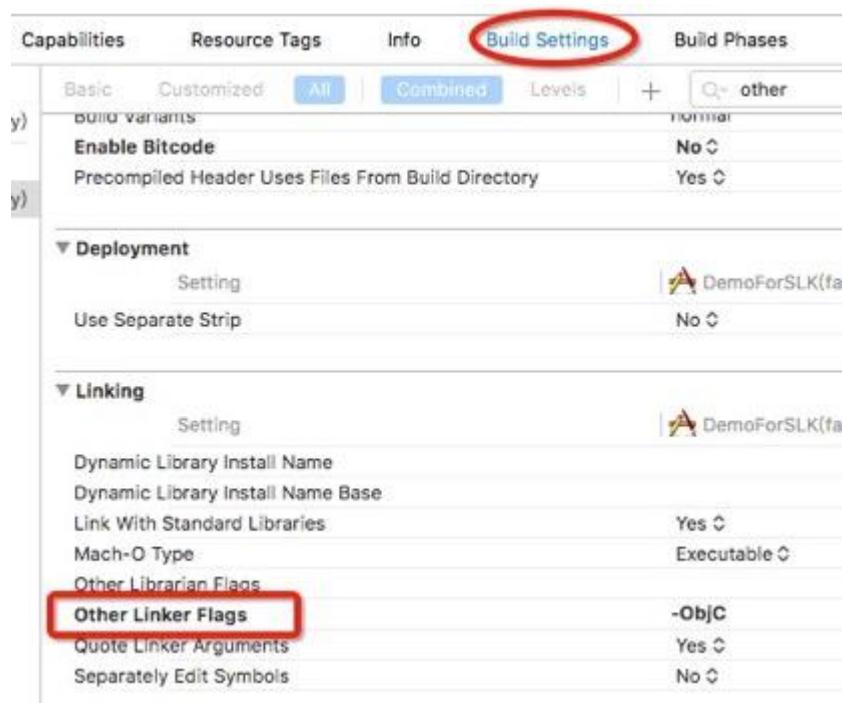
3.1.2. Importing in IOS

For IOS development, followings are required:

- ✧ Xcode version: v12.5 or later.
- ✧ CocoaPods version: v1.10.1 (recommended)
- ✧ IOS version supported: v11.0 or later

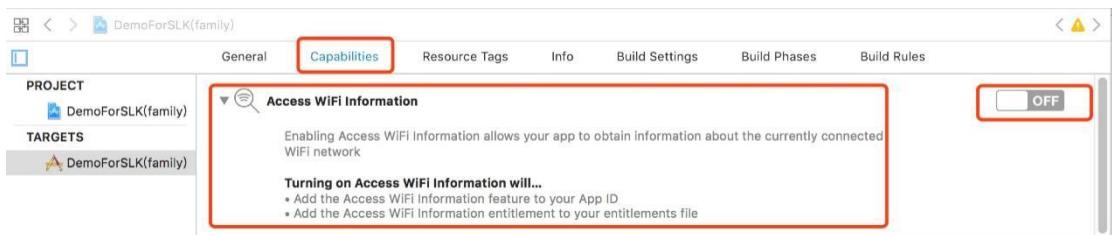
- Put MSmartSDK.framework and MSSecretSDK.framework folder into your IOS project.
- Set linker options

In your IOS project, go to “TARGETS->Build Settings”, set “Other Linker Flags” with “-all_load” or “-ObjC” or “-fore_load <SDK path>”.



- Allow Wi-Fi access permission.

Turn the “Access WiFi Information” switch on manually in the UI of development tool.



- Note for IOS v13

When setting the network configuration for appliance, MSSmartSDK uses `CNCopyCurrentNetworkInfo` API to get the Wi-Fi information currently connected by mobile phone, there is more limitation for this IOS API in IOS v13, in order to make `CNCopyCurrentNetworkInfo` work normally as expected, the APP must satisfy at least one of following conditions.

- ✧ Have the permission to access user location through Location service.
- ✧ Provide VPN service.
- ✧ Use `NEHotspotConfiguration` API to config current Wi-Fi network.

3.2. Initializing MSmartSDK

After importing MSmart SDK into project successfully, the APP also need to initialize SDK resource before calling other methods.

3.2.1. Key parameters for initialization

The following table lists key parameters required to initialize MSmartSDK.

Parameter	Description
clientId	Be used to identify a client session from a client in Midea IoT Open Cloud, used by Appliance management API and MSmartSDK.
clientSecret	The are allocated by Midea.
serverHost	The whole URL locating the server host, the domain name is same as “domain” parameter.

Table: key parameters used when Initializing SDKs

3.2.2. Initializing in Android

You can initiate SDKs at the entry of Application in your android code project like following code.

```
1. //init SDKs for Android
2. import com.midea.iot.msmart.MSConfig
3. import com.midea.iot.msmart.MSInterface
4.
5. //--init MSmartSDK
6. MSConfig config = new MSConfig();
7. config.serverHost ="https://sit-us.dollin.net";
8. config.enableLog = true;      //enable Log in MSmartSDK, suggest to disable if
     make a release.
9. config.clientId="e2ca9a96-fa55-6c70-690c-e1e4b06e8c15";
10. config.clientSecret="c9ea24be-2318-330c-32f6-045665a71110";
11. MSInterface.getInstance().initSDK(context, MSInterface.WorkMode.OVERSEAS_OEM,
    config); //use OVERSEAS_OEM for non-Midea APP.
```

3.2.3. Initializing in IOS

Similarly, following code show how to initialize MSmartSDK as an example.

```
1. MSConfig *config = [[MSConfig alloc] init];
2. config.serverHost = server_host_of_midea_IoT_open_cloud;
3. config.clientId = client_id;
4. config.clientSecret = client_secret;
5. config.enableLog = YES;
6.
7. MSInterface * interface = [MSInterface sharedInstance];
8.
9. //set work mode for OEM partner
10. BOOL success = [interface initSDK:config MSmartWorkModeOverSeaOEM extra:nil]
    ;
11.
12. // you can set a delegate method that whould be
    called when token become expired.
13. [interface setTokenRefreshDelegate:self];
14. if (success) {
15.     NSLog(@"init SDK successfully");
16. }
```

3.3. Calling MSmartSDK

3.3.1. Managing Device Token

MSmartSDK must keep a valid device token when APP call a method. You can get a valid device token by Appliance Management API. After getting a device token, you can transfer it into MSmartSDK by calling its setAccessToken() method.

On the other hand, you should also refresh the device token in time, the device token maybe becomes expired after a period.

You can set a delegate method calling setTokenRefreshDelegate() method for MSmartSDK, and refresh device token when MSmartSDK call the delegate method. Following is an example code for android.

```
1. MSInterface.getInstance().setTokenRefreshDelegate(new MSTokenRefresh() {  
2.     @Override  
3.     public boolean refreshToken() {  
4.         //get a new device token by calling Appliance Management API  
5.         //call HTTP API and get a new access token.  
6.  
7.         //set the new device token for MSmartSDK  
8.         MSInterface.getInstance().setAccessToken(accessToken);  
9.  
10.        //if succeed in getting a new device token, return true, otherwise return  
11.        false.  
12.    }  
13.});
```

3.3.2. configure appliance network by Wi-Fi AP

In Android, MSmartSDK provides MSDeviceConfigManager to support network configuration, following code shows how to use it for network configuration by Wi-Fi AP.

```
1. MSDeviceApConfigParams params = new MSDeviceApConfigParams(context); //context is of Context type
2. params.setDeviceSSID(deviceSSID); //"midea_fd_0002"
3. params.setRouterBSSID(routerBSSID); //"00:9a:cd:8d:fe:0c"
4. params.setRouterSSID(routerSSID); //"APP_TEST2"
5. params.setRouterPassword(routerPassword); //"1234567890"
6. params.setRouterSecurityParams(routerSecurity); //"["WPA2-PSK-CCMP"] [RSN-PSK-CMP] [ESS] [WPS]"
7. params.setFrequency(frequency);
8. params.setDeviceSecurityParams("WPA/PSK");
9. params.setDevicePassword("12345678");
10. params.setNeedCheckPassword(true);
11. params.setAdsId(adsid);
12. params.setRegionId(regionId);
13. params.setCountryChannelList(channels);
14. params.setTimeZone(timeZone);
15. params.setCountryCode(countryCode);
16. params.setFunctionType(functionType);
17. params.setModuleServerDomain(moduleDomain);
18. params.setModuleServerPort(modulePort);
19.
20. MSDeviceConfigManager.getInstance().startConfigureDevice(params, MSDeviceConfigType.MSDeviceConfigAP, new MSProgressCallback<MSDevice, MSDeviceConfigStep>() {
21.     @Override
22.     public void onProgressUpdate(MSDeviceConfigStep deviceConfigStep) {
23.         LogUtils.i("progress:" + String.valueOf(deviceConfigStep));
24.     }
25.     @Override
26.     public void onComplete(MSDevice data) {
27.         LogUtils.i("network config is success:" + data);
28.         LogUtils.d(data.getDeviceID() + "-" + data.getVerificationCode());
29.         //you can call HTTP API to bind the appliance in cloud server.
30.     }
31.
32.     @Override
33.     public void onError(MSErrorMessage errMsg) {
34.         LogUtils.d("network config is failed:" + errMsg.getErrorMessage());
35.     }
}
```

```
36. });
```

Parameters assigned to the “params” variable in above demonstrated code explains as below.

- deviceSSID: the SSID name of the appliance that is activated as a Wi-Fi AP, you can scan relevant Wi-Fi AP with a name pattern through system call.
- routerBSSID: the BSSID name of the Wi-Fi router. You can get BSSID from android system when mobile phone is connecting to Wi-Fi router.
- routerSSID: the SSID name of the Wi-Fi router.
- routerPassword: the password of the Wi-Fi router, generally let user input it explicitly.
- routerSecurity: the encryption type used by Wi-Fi router.
- frequency: the frequency of the Wi-Fi router.
- adsid: the ADS ID representing the gateway of Midea, you can get ADS resource by calling the corresponding “[/token](#)” API of Appliance Management API.
- regionId: the region ID associated with time zone. See “[Get RegionID](#)” API for details.
- channels: an array of MSCountryChannel entity, a Wi-Fi channel setting list, you can get channel information with a country code¹ by calling corresponding Appliance Management API, and convert the channel information to this “channels” array like following example code.

```
1. public static MSCountryChannel[] getChannels(){  
2.     MSCountryChannel[] channels = new MSCountryChannel[4];  
3.     channels[0] = new MSCountryChannel((byte)1,(byte)13,(byte)27,false);  
4.     channels[1] = new MSCountryChannel((byte)36,(byte)4,(byte)23,false);  
5.     channels[2] = new MSCountryChannel((byte)52,(byte)4,(byte)23,true);  
6.     channels[3] = new MSCountryChannel((byte)149,(byte)5,(byte)27,false);  
7.     return channels;  
8. }
```

Each channel includes 4 elements described as follows:

```
channels[0] = new MSCountryChannel(  
    (byte)1, //the first channel number.  
    (byte)13, //the amount of channel.  
    (byte)27, //the maximum power of transmission.  
    false); // whether to enable DFS feature for channel.
```

¹ Country Code: Compliance with ISO 3166-1 and it uses alpha-2 format, example: CN,US

- timeZone: the current time zone where APP user locates. Such as MSTimeZone.EAST_8.
- countryCode: the country code where APP user locates, such as: CN,US. see [country code](#) for more information.
- functionType: set it as MSFunctionType.OVERSEAS.
- moduleDomain: the server host name in ADS information, such as:iotlab.midea.com.cn.
- modulePort: the port number in ADS information, such as 28443.

Note:

adsId, moduleDomain, and modulePort parameters are ADS resources, you can get ADS resource information by calling setToken() method of OEMDeviceManager in OEMDeviceSDK for Android; calling getAuthData() method of BaseService instead after setToken() was called in IOS.

On the other hand, you can also call getAdsInfo() method of OEMDeviceManager in any time to retrieve ADS resource directly after you got a valid device token. OEMDeviceManager. gatAdsInfo() actually calls a http interface, see also: [Get ADS resource](#)

Once the network configuration process completes successfully, onComplete callback would be called with a MSDevice data. In the callback, you still need to bind the configured device in cloud server by calling “[/bind](#)” API of Appliance Management API.

In IOS, the network configuration process for appliance is similar to the one in android, below is example code for reference.

```

1. //set core parameters for network configuration.
2. + (NSDictionary *)createFunctionSetDic{
3.     //channel List example
4.     NSArray *channelList = @[@{kMSSDKConfigFirstChannelNumber:@"1", //
5.                             kMSSDKConfigChannelCount:@"13", //
6.                             kMSSDKConfigChannelMaxPower:@"27", //
7.                             kMSSDKConfigChannelDFSEnable:@"0"}, //
8.                             @{@"kMSSDKConfigFirstChannelNumber:@"36",
9.                             kMSSDKConfigChannelCount:@"4",
10.                            kMSSDKConfigChannelMaxPower:@"23",
11.                            kMSSDKConfigChannelDFSEnable:@"0"}, //
12.                            @{@"kMSSDKConfigFirstChannelNumber:@"52",

```

```

13.                                     kMSSDKConfigChannelCount:@"4",
14.                                     kMSSDKConfigChannelMaxPower:@"23",
15.                                     kMSSDKConfigChannelDFSEnable:@"1",
16.                                     @{@"kMSSDKConfigFirstChannelNumber:@"149",
17.                                     kMSSDKConfigChannelCount:@"5",
18.                                     kMSSDKConfigChannelMaxPower:@"27",
19.                                     kMSSDKConfigChannelDFSEnable:@"0"},
20.                                     ];
21.     NSMutableDictionary *dic = [NSMutableDictionary dictionary];
22.     dic[kMSSDKConfigFunctionType] = @"1";//"functionType";
23.     dic[kMSSDKConfigRegionID] = @"1";//"regionID";
24.     dic[kMSSDKConfigModuleServerDomain] = @"iot1.midea.com.cn";
25.     dic[kMSSDKConfigModuleServerPort] = @"28443"; //moduleServerPort";
26.     dic[kMSSDKConfigCountryCode] = @"CN"; //countryCode";
27.     dic[kMSSDKConfigTimeZone] = @"8"; //timeZone";
28.     dic[kMSSDKConfigChannelList] = channelList; //channel list";
29.     return [NSDictionary dictionaryWithDictionary:dic];
30. }
31.
32. //configure network example code.
33. - (void)startAddDevice:(id)sender {
34.     MSDeviceApConfigParams *configParams = [[MSDeviceApConfigParams alloc] i
nit];
35.     configParams.deviceSSID = self.codeResultParamDic[@"deviceSSID"]; //
36.     configParams.routerSSID = self.codeResultParamDic[@"routerSSID"]; //
37.     configParams.routerBSSID = self.codeResultParamDic[@"routerBSSID"]; //
38.     configParams.routerPwd = self.codeResultParamDic[@"routerPwd"]; //
39.
40.     NSDictionary *extraDic = [MideaConfigDeviceTool createFunctionSetDic];//
41.     configParams.countryCode = extraDic[kMSSDKConfigCountryCode];//
42.     configParams.channelList = extraDic[kMSSDKConfigChannelList];//
43.     configParams.timeZone = extraDic[kMSSDKConfigTimeZone];//
44.     configParams.functionType = extraDic[kMSSDKConfigFunctionType];//
45.     configParams.moduleServerPort = extraDic[kMSSDKConfigModuleServerPort];
46.     //
47.     configParams.moduleServerDomain = extraDic[kMSSDKConfigModuleServerDomai
n];//
48.     configParams.regionId = extraDic[kMSSDKConfigRegionID];//
49.     configParams.adsClusterId = self.codeResultParamDic[@"clusterId"];//
50.     [self.configManager startConfigureDevice:configParams configType:MSDevic
eConfigAP progressCallback:^(MSmartDeviceConfigStepType apStep, MSBleConnects
tateType bleStep) {
51.         //handle

```

```

52.         if(apStep == MSmartDeviceConfigStepTypeConnectSuc) {
53.             self.flowLabel.text = @"connected to appliance successfully...";
54.         } else if(apStep == MSmartDeviceConfigStepTypeConnectFailed){
55.             self.flowLabel.text = @"failed to connect to appliance...";
56.         } else if(apStep == MSmartDeviceConfigStepTypeStartCheckCloud) {
57.             self.flowLabel.text = @"start check in cloud...";
58.         }
59.
60.     } completioncallback:^(NSError *_Nullable error, MSDevice * _Nullable de
vice) {
61.         if (error.code == MSsmartErrorCode_Suc) {
62.             self.flowLabel.text = @"appliance succeeded to connect to intern
et...";
63.         } else if (error.code == MSsmartErrorCode_WriteWifiInfoBy70_WrongPass
word
64.                     || error.code == (MSsmartErrorCode_WriteWifiInfoBy70_Wrong
Password + 110000)) {
65.             self.flowLabel.text = @"the wifi password is wrong...";
66.
67.         } else if (error.code == 110000 + MSsmartErrorCode_CurrentWifi_Unable
AccessCloud){
68.             self.flowLabel.text = @"the current wifi can not access to inter
net, please change another wifi...";
69.
70.     }else{
71.         self.flowLabel.text = [NSString stringWithFormat:@"%@",error.use
rInfo[NSLocalizedStringKey]];
72.     }
73.     [self.startAddDeviceBtn setTitle:@"start configuring network..." fo
rState:UIControlStateNormal];
74. ];
75. }

```

There is a timeout setting of 90 seconds for the whole network configuration process.

The parameters used in above IOS code have the same semantic meaning as the ones used in android.

NOTE: the Wi-Fi module of Midea appliance can not support Wi-Fi AP running with 5G frequency band.

3.3.3. configure appliance network by Bluetooth

You can use MSBLEDDeviceManager and MSDeviceConfigManager objects in MSsmartSDK to configure appliance network by Bluetooth, use MSBLEDDeviceManager to scan, find and filter appliance what you want to configure, then use MSDeviceConfigManager to complete the whole configuration process. Following code is an example for android.

```
1. //scan for appliance
2. MSBLEDDeviceManager.getInstance().startScan(type, timeOut, new MSBleScanCallback() {
3.     @Override
4.     public void onComplete(MSBleScanInfo data) {
5.         //save the found appliance information in this callback.
6.         if(!isContain(data)){
7.             Map<String,MSBleScanInfo> map = new HashMap<>();
8.             map.put(data.getMac(),data);
9.             datas.add(map);
10.        }
11.        myAdapter.notifyDataSetChanged();
12.    }
13.
14.     @Override
15.     public void onError(MSErrorMessage errMsg) {
16.    }
17. }
18.
19. //Launch network configuration by Bluetooth
20. MSDeviceBleConfigParams params = new MSDeviceBleConfigParams();
21. params.setContext(context);
22. params.setMsBleScanInfo(msBleScanInfo);//the object returned when scan
23. params.setRouterSSID(routerSSID);
24. params.setRouterPassword(routerPassword);
25. params.setRouterSecurityParams(routerSecurity);
26. params.setBleSecondConfigType(secondConfigType);
27. params.setBindActionType(bindActionType);
28.
29. MSDeviceConfigManager.getInstance().startConfigureDevice(params, MSDeviceConfigType.MSDeviceBindBluetooth, new MSProgressCallback<MSDevice, MSDeviceConfigStep>() {
30.     @Override
31.     public void onProgressUpdate(MSDeviceConfigStep msDeviceConfigStep) {
32.         LogUtils.d("BleConfigActivity", msDeviceConfigStep.getStepName().name())
33.     }
34. }
```

```

34.
35.     @Override
36.     public void onComplete(MSDevice data) {
37.         LogUtils.d("succeeded to config network for appliance. ", data.toString()
38.             ());
39.         LogUtils.d(data.getDeviceID()+"-"+data.getVerificationCode());
40.         //you can call HTTP API to bind the appliance in cloud server.
41.     }
42.     @Override
43.     public void onError(MSErrorMessage errMsg) {
44.         LogUtils.d("BleConfigActivity failed to config network for appliance",
45.             errMsg.getErrorCode() + "//" + errMsg.getMessage());
46.         Toast.makeText(BleConfigActivity.this,"network configuration is failed:
47.             "+errMsg.getErrorCode(),Toast.LENGTH_LONG).show();
48.     }
49. });

```

The `startScan()` method of `MSBLEDeviceManager` has 3 parameters listed below:

- type: scan type
 - `MSBLEModuleType.ALL`: scan all device supporting bluetooth.
 - `MSBLEModuleType.MS`: only scan device supporting MSmart protocol.
 - `MSBLEModuleType.MS_SingleModule`: scan MSmart device with single mode bluetooth.
 - `MSBLEModuleType.MS_ComboModule`: scan MSmart devcie supporting both bluetooth and Wi-Fi.
- timeout: set timeout value in second.
- callback: a callback to be called when scan successfully or failure.

The parameters assinged to “params” variable of `MSDeviceBleConfigParams` type explains as following.

- `msBleScanInfo`: one object returned when call `startScan()` of `MSBLEDeviceManager` to scan bluetooth devcie.
- `secondConfigType`: the network configuration type for the 2nd generation Bluetooth module of Midea, for general case, you can set it as `MSBleSecondConfigType.CONNECT`.
 - `MSBleSecondConfigType.PRE_CONNECT`: pre-configure appliance network.
 - `MSBleSecondConfigType.CONNECT`: this option is for common case

- use.
- MSBleSecondConfigType.AUTH_CONNECT: configure appliance network and authenticate appliance.
 - bindActionType: the bingding type for Bluetooth appliance.
 - MSBLEBindActionType.BIND: use this option for most of cases.
 - MSBLEBindActionType.BIND_WITHOUT_AUTH: this option is to bind appliance without authentication.

In IOS, the network configuration process for appliance is similar to the one in android, following is example code.

```

1.  /* scan method declaration
2.  start scaning device process
3.  @param filters : an array to filter found device when scanning
4.  @param timeout: timeout of scan in second, -1 means keep scanning continu
   usly
5.  @param scanback : a callback that would be called once a device is found.
6.  - peripheral : this parameter contains the information of a Bluetooth dev
   ice. see CBPeripheral+MSExtensions.h
7.  */
8.  - (void)startScanWithFilter:(NSArray * __nonnull)filters
9.          timeout:(NSTimeInterval)timeout
10.         scanback:(MSBleScanCallback)scanback;
11.
12.     NSDictionary *options = @{
13.         CBCentralManagerRestoredStateScanOptionsKey:@(YES),
14.         CBCentralManagerScanOptionAllowDuplicatesKey:@(YES),
15.         CBCentralManagerOptionShowPowerAlertKey:@(NO)
16.     };
17.
18.     [[MSBLEDeviceManager shareInstance] setMSBluetoothOptionsWithScanForPeri
   pheralsWithOptions:options connectPeripheralWithOptions:nil scanForPeripheral
   sWithServices:nil discoverWithServices:nil discoverWithCharacteristics:nil];
19.
20.     @weakify(self);
21.     [[MSBLEDeviceManager shareInstance] startScanWithFilter:@[@"midea"] time
   out:TimeInterval scanback:^(CBPeripheral * _Nullable peripheral) {
22.         @strongify(self);
23.         [self dealFindBlePeripheral:peripheral error:nil];
24.     }];
25.
26. //process the data of found device.
27. NSData *data = peripheral.msAdvertiDictionary[@"kCBAAdvDataManufacturerData"];

```

```

28. NSString *ssid = peripheral.msAdvertiDictionary[@"kCBAAdvDataLocalName"];
29. NSDictionary *bleInfo = [MSmartBleTool bleAdvDicWithPeripheral:data bleName:
   ssid];
30.
31.
32. //startConfigureDevice method declaration
33. @note there is a 90 seconds timeout. Tip: the Wi-Fi module of appliance can
   not support 5G frequency band.
34. @brief launch network configuration for an appliance
35. @param params : contains necessary parameter values used when configuring
   appliance network.
36. @param configType: the type of network configuration
37. @param progresscallback : callback function that can receive progress infor
   mation during network configuration, it
38. - MSmartDeviceConfigStepType apStep : progress information for configurat
   ing network with Wi-Fi AP.
39. - MSBleConnectStateType bleStep : progress information for configuring
   network with Bluetooth.
40. @param completioncallback : callback function when the whole network config
   uration process is completed.
41. */
42. - (void)startConfigureDevice:(MSDeviceConfigParams *__nonnull)params
43.                      configType:(MSDeviceConfigType)configType
44.                      progressCallback:(MSProgressCallback)progresscallback
45.                      completioncallback:(MSCompletionCallback)completioncallback;
46.
47. //Launch network configuration
48. - (void)startBleConnect {
49.     DDLogDeviceInfo(@"params for 2nd generatoin Bluetooth module : deviceMac
   = %@", routerBSSID = %@", routerSSID = %@", routerPassword = %@", familyId =
   %@", connectMode = %lu, \n Peripheral = %@", self.deviceMac, self.wifiBssid,
   self.wifiSsid, self.wifiPassword, self.familyId, (unsigned long)self.mode, s
   elf.peripheral );
50.
51.     MSDeviceBleConfigParams *params = [[MSDeviceBleConfigParams alloc] init]
   ;
52.     params.peripheral = self.peripheral; //
53.     params.routerSSID = self.wifiSsid; // wifissid "APP_TEST2"
54.     params.routerPwd =self.wifiPassword; //wifi password
55.     params.routerBSSID = self.wifiBssid;//wifiBssid "5E:6E:0A:0F:E2:41"
56.     params.bleSecondConfigType = self.mode; // MSBLEConfigNetMode_Connection
57.
58.     NSDictionary *tmpDic = [MideaConfigDeviceTool createFunctionSetDic];//
59.

```

```

60.     params.countryCode = tmpDic[kMSSDKConfigCountryCode];//  

61.     params.timeZone = tmpDic[kMSSDKConfigTimeZone];//  

62.     params.channelList = tmpDic[kMSSDKConfigChannelList];//  

63.     params.regionId = tmpDic[kMSSDKConfigRegionID];//  

64.     params.moduleServerDomain = tmpDic[kMSSDKConfigModuleServerDomain];//  

65.     params.moduleServerPort = tmpDic[kMSSDKConfigModuleServerPort];//  

66.     params.functionType = tmpDic[kMSSDKConfigFunctionType];//  

67.     params.adsClusterId = [NSString stringWithFormat:@"%@,[MideaUserManager  

shareInstance].currentUser.clusterId];//  

68.  

69.     @weakify(self);  

70.     [[MSDeviceConfigManager shareInstance] startConfigureDevice:params confi  

gType:MSDeviceConfigBluetooth progressCallback:^(MSmartDeviceConfigStepType a  

pStep, MSBleConnectStateType bleStep) {  

71.         @strongify(self);  

72.         switch (self.bleStep) {  

73.             case MSBleConnectStepTypeStartConfig: // 6(prepare to Launch  

connection)  

74.  

75.                 break;  

76.             case MSBleConnectStepTypeFindingDeviceInCloud: //11(check appli  

ance in cloud)  

77.  

78.                 break;  

79.             case MSBleConnectStepTypeFindDeviceInCloudSuccess: //12 (succ  

ed to check appliance in cloud)  

80.  

81.                 break;  

82.             default:  

83.                 break;  

84.         }  

85.     } completioncallback:^(NSError * _Nullable error, MSDevice * _Nullable d  

evice) {  

86.         DDLogDeviceInfo(@"Network configuration result---%---%@", error, [de  

vice mj_JSONObject]);  

87.         if(error.userInfo){  

88.  

89.             }  

90.  

91.             //the whole network configuration is completed without error.  

92.             if (error.code == 0) {  

93.                 //make appliance available to operate in APP UI.  

94.                 [self activeDevice:device];  

95.             }

```

```
96.         else{//error occurred
97.             if (error.code == MSmartErrorCode_Ble_WriteWifiInfo_WrongPasswor
d
98.                 || error.code == (MSmartErrorCode_Ble_WriteWifiInfo_WrongPas
sword + 110000)) {//password is wrong
99.
100.            }
101.        }
102.    }];
103.}
```

3.3.4. Error code definition

You can find error code definition in the “[Error code of MSmartSDK](#)” of appendixes section.

3.3.5. Miscellanea

If you intend to obfuscate source code for your android APP, remember to add “`-keep class com.midea.iot.*{*;}`” option in the setting file.

3.4. Connecting appliance to Midea IoT Core

An appliance device must setup a network connection to Midea IoT Core before the APP can manipulate it, this means that appliance need access to internet.

MSmartSDK have network setup methods that can transfer Wi-Fi credentials (the Wi-Fi SSID and password) of the router to the appliance, once appliance successfully connects to Midea IoT Core, APP can call Appliance Management API to manipulate it.

Midea appliance product usually supports Bluetooth or/and Wi-Fi AP link, APP need use either of them to get connected with appliance device. Following chart shows the general network structure between APP, Midea IoT and appliance.

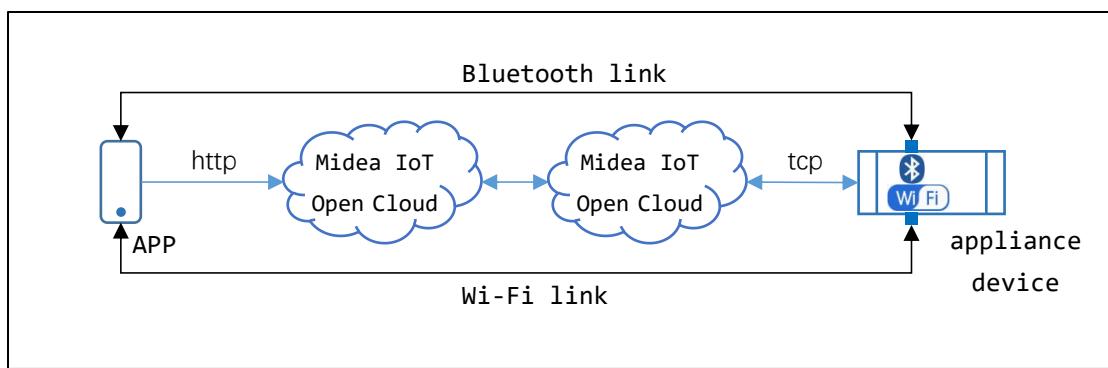


Chart: network connections between APP, Midea IoT and appliance

In order to complete the whole network configuration for appliance, the mobile APP need use Appliance Management API and MSmartSDK to work together, a common guideline describes as below:

- Call Appliance Management API to get a valid device token and the target ADS² resource; also need to get Wi-Fi channel setting for a specified country.
- Set a valid device token to MSmartSDK by calling MSmartSDK's `setAccessToken()` method. Don't remember refresh device token if expired, see [managing device token](#).
- Use MSmartSDK to scan device via Bluetooth and call `startConfigDevice()` method to launch network configuration; or directly call `startConfigDevice()` method after manually connect mobile phone to the Wi-Fi AP of appliance.

² ADS: appliance domain-name system, a network connection distribution system of Midea IoT. It is responsible to allocate a particular target server for each appliance that need to connect to Midea IoT Core.

Tips: your APP should show some guideline information to prompt user on how to operate appliance and make it to work in network configuration mode.

Following charts shows relevant activities about the network configuration process for an appliance. The overall process consists of two stages: preparation stage and network configuration stage.

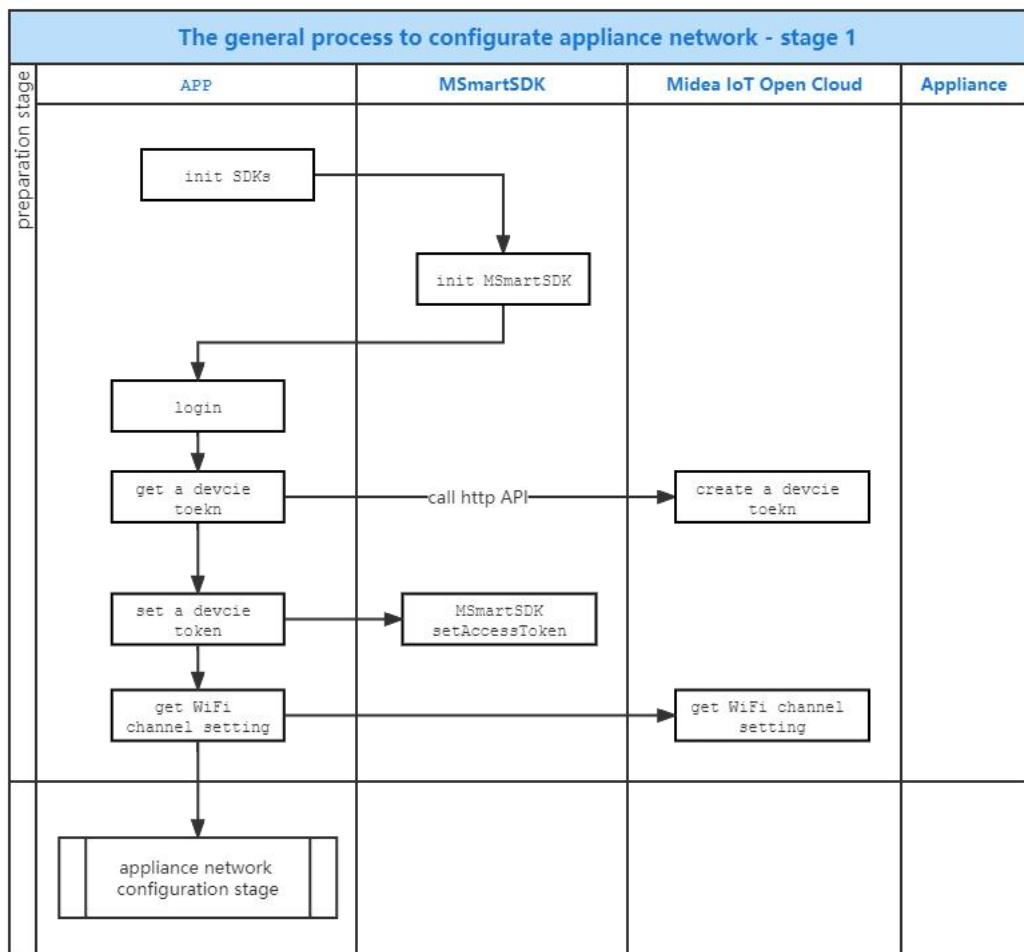


Chart: appliance network configuration stage 1

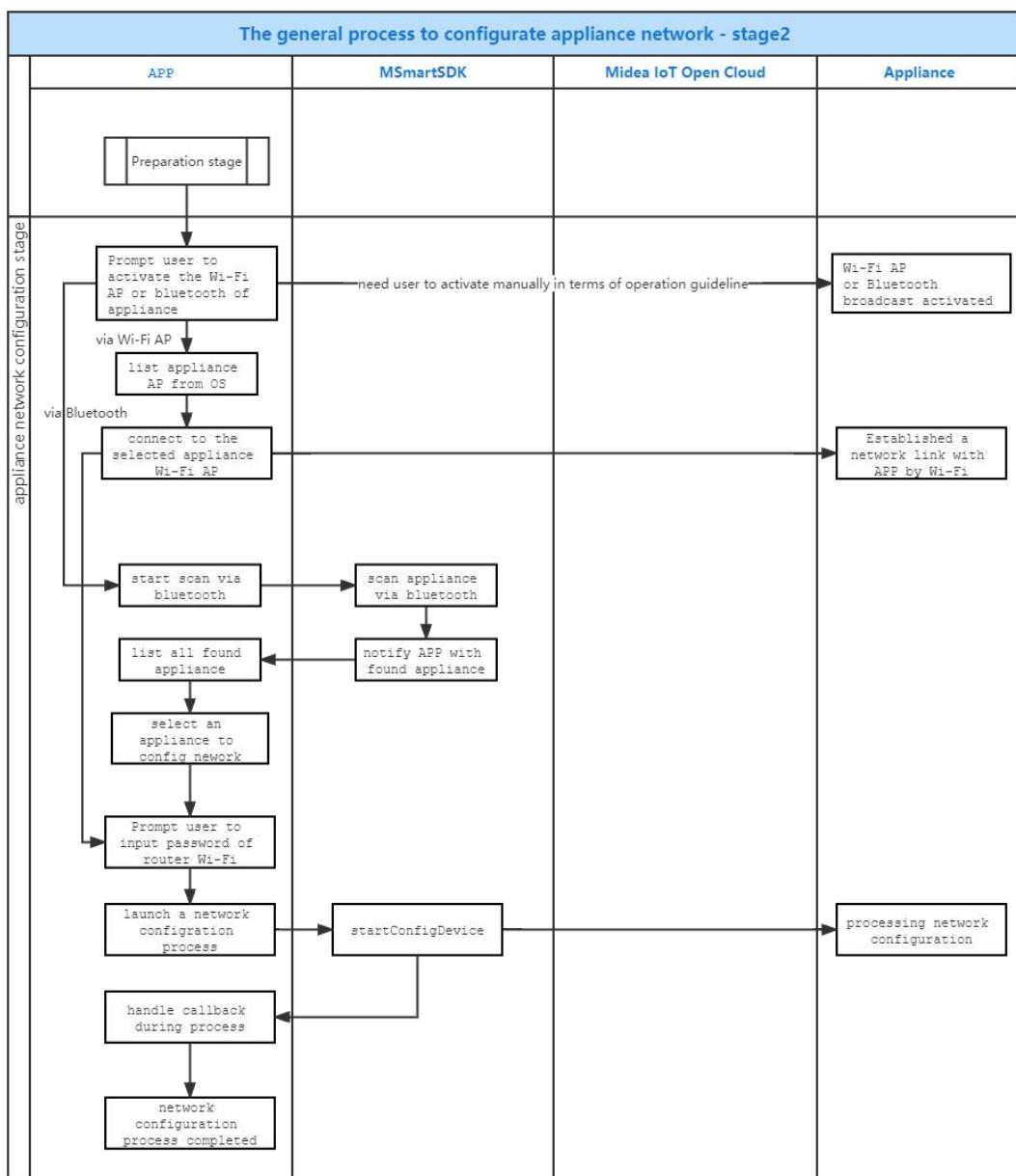


Chart: appliance network configuration stage 2

4. HTTP API Reference

4.1. HTTP API for User Account Management

Midea IoT Open Cloud provides relevant APIs to allow you to create and manage user account in terms of your business requirement, you can develop your own APP with these APIs by calling in your cloud server or APP.

4.1.1. HTTP Common header

HTTP header	required	Description
Accept-Language	true	Such as:accept-language:zh-CN,zh;q=0.9
Content-Type	true	application/json
User-Agent	true	set by referencing system.
appId	true	App id, allocated by Midea, uniquely identify an APP client in Midea IoT Open Cloud.
src	true	Request source, usually set it with 2 last digits of appId.
clientType	true	1:Android, 2:IOS
stamp	true	Timestamp, format:yyyyMMddHHmmss
reqId	true	Request id, a UUID string without '-' char. Example:cf72452d76bf1da40c47f806a43c95f4
language	false	The language code used to specify the expected language in which server spells textual message in http response. It's compliance with RFC1766, but follow "<primary tag>_<sub tag>" format instead of "<primary tag>-<sub tag>". Currently support: zh_CN,zh_HK,en_US,ru_RU ar_SA,ko
deviceId	true	A unique ID of mobile phone.
appVNum	true	The version number of APP
random	false	A random integer.
sign	false	the signature of request parameters, this signature is a hash value calculated with a certain method, see following security section for more information.
secretVersion	false	
accessToken	false	The token created after a user login.

4.1.2. Security

4.1.2.1. Encrypting key data object

The HTTP API require to encrypt key data object when transporting them between client(APP) and server. Midea IoT Open cloud provides utility unit to help call encryption method.

There are following data objects required to encrypt.

- The password of user login
- Verification code
- Third-pard account OpenID: facebook, AppleID, twitter etc.

4.1.2.2. Calculating signature for request parameters

Most of HTTP API requires the “sign” parameter, Midea IoT Open Cloud use “sign” to verify request parameters.

Following is an example code to calculate the signature.

```
1. // for java
2. String random = "1632894808272";
3. String body = "{\"stamp\": \"20210929135328\", \"reqId\": \"7f58b69eb1ed413785f5eff0ed6b1907\"}";
4. try {
5.     String unSign = "meicloud" + body + random;
6.     String checkSign = EncryptUtils.encryptHmacSHA256ToString(unSign, "SIT_4VjZdg19laDoIrut").toLowerCase(Locale.getDefault());
7. } catch (Exception e) {
8.     e.printStackTrace();
9. }
```

Note: Midea technical staff can contribute the code of EncryptUtils package.

```
1. // for iOS
2. - (NSString *)signpostData:(NSData *)postData
3.                     forUrl:(NSString *)url
4.                     withRandom:(NSString *)randomStr {
5.     if (!url) {
6.         return @"";
7.     }
8.
9.     NSString *paramsJson = [BusinessNetworkTools dataToJSONString:postData] ?
10.        @"";
11.    NSString *oriStr = [NSString stringWithFormat:@"%@%@", [MSAppInfo appKey], paramsJson, randomStr];
```

```

11.     NSString *result = [[self class] hmacSHA256WithSecret:[MSAppInfo appSecret] content:oriStr];
12.
13.     return result;
14. }
15.
16. + (NSString *)hmacSHA256WithSecret:(NSString *)secret
17.                               content:(NSString *)content {
18.     const char *cKey = [secret cStringUsingEncoding:NSUTF8StringEncoding];
19.     const char *cData = [content cStringUsingEncoding:NSUTF8StringEncoding];
20.     unsigned char cHMAC[CC_SHA256_DIGEST_LENGTH];
21.     CCHmac(kCCHmacAlgSHA256, cKey, strlen(cKey), cData, strlen(cData), cHMAC)
22. ;
23.     NSData *HMACData = [NSData dataWithBytes:cHMAC length:sizeof(cHMAC)];
24.     const unsigned char *buffer = (const unsigned char *)[HMACData bytes];
25.     NSMutableString *HMAC = [NSMutableString stringWithCapacity:HMACData.length * 2];
26.     for (int i = 0; i < HMACData.length; ++i){
27.         [HMAC appendFormat:@"%02x", buffer[i]];
28.     }
29.     return HMAC;
30.
31.
32. + (NSMutableDictionary *)getCommonData {
33.     NSMutableDictionary *commonData = [NSMutableDictionary new];
34.     [commonData setValue:[self getAppId] forKey:@"appId"];
35.     [commonData setValue:[self getSrc] forKey:@"src"];
36.     [commonData setValue:MSLogin_Client_Type_IOS forKey:@"clientType"];
37.     NSString *timestamp = [NSDate dof_getNSStringFromNSDate:[NSDate date] withFormat:@"yyyyMMddHHmmss"];
38.     [commonData setValue:timestamp forKey:@"stamp"];
39.     [commonData setValue:[MideaTool getReqid] forKey:@"reqId"];
40.     NSString *languageCode = [[OEMInternationalization sharedInstance] convertSystemLanguageCodeToUnifyLanguageCode:OEMCurrentLanguage.code];
41.     [commonData setValue:languageCode forKey:@"language"];
42.     [commonData setValue:[MideaTool getUUID] forKey:@"deviceId"];
43.     [commonData setValue:[MSAppInfo appShortVersion] forKey:@"appVNum"];
44.     return commonData;
45. }
46.
47. // 配置请求头信息
48. - (NSDictionary *)configRequestHeaderForUrl:(NSString *)url
49.                                         postData:(NSData *)postData

```

```

50.                     header:(NSDictionary *)addHeader {
51.             NSMutableDictionary *mutHeaders = [NSMutableDictionary new];
52.             NSArray * originHeaders = [[OFRequestManager manager] getGlobalConfigure
53.                                         Header];
53.             for (NSDictionary * head in originHeaders) {
54.                 if ([head count] > 0) {
55.                     [mutHeaders addEntriesFromDictionary:head];
56.                 }
57.             }
58.             //HTTPHeader(name: "Content-Type", value: "application/json")
59.             [mutHeaders setObject:@"application/json" forKey:@"Content-Type"];
60.
61.             NSDictionary *dict = [MSAppInfo getCommonData]; //公共参数添加到头部
62.             [mutHeaders addEntriesFromDictionary:dict];
63.             NSString *randomStr = [self createRandomString];
64.             mutHeaders[@"random"] = randomStr;
65.             mutHeaders[@"sign"] = [self signpostData:postData
66.                                     forUrl:url
67.                                     withRandom:randomStr];
68.
69.             if ([[MSUserInfoManager shareManager].loginInfoModel.accessToken] {
70.                 mutHeaders[@"accessToken"] = [MSUserInfoManager shareManager].loginI
71.                 nfoModel.accessToken;
71.                 // mutHeaders[@"uid"] = [MSUserInfoManager shareManager].LoginInfoMod
72.                 el.uid; //去掉uid
72.             }
73.
74.             if ([addHeader isKindOfClass:[NSDictionary class]]) {
75.                 [mutHeaders addEntriesFromDictionary:addHeader];
76.             }
77.             return mutHeaders;
78.         }

```

Note: appKey and appSecret are distributed by Midea technical staff.

4.1.3. API

4.1.3.1. Get country code list

Get country code list supported by Midea IoT Open Cloud sever.

URL: `http://{domain}/v1/user/area/get`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Comment
none		

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
list	Json array, each element is a json object.
id	Element id
area	Country name.
countryCode	Compliance with RFC3177 and it uses alpha-2 format. Example: CN,US
phoneCode	The international telephone code, compliance with E.164 standard, such as: 1, 86
url	API server host of Midea IoT Open Cloud.
mqttBroker	MQTT server host of Midea IoT Open Cloud. NOTE: If your APP want to interact asynchronously with Midea IoT Open Cloud, the APP need to integrate another SDK named OEMMessageSDK.

```
# Response example
[
  {
    "id":1,
    "area":"China",
    "countryCode":"CN",
    "phoneCode":"86",
```

```
        "url":"sit-cn.dollin.net",
        "mqttBroker":"mqtxkfi.iot.cn-west-2.amazonaws.com"
    },
    {
        "id":2,
        "area":"United State",
        "countryCode":"US",
        "phoneCode":"1",
        "url":"sit-us.dollin.net",
        "mqttBroker":"mqtxkfi.iot.us-west-2.amazonaws.com"
    },
    {
        "id":3,
        "area":"France",
        "countryCode":"FR",
        "phoneCode":"33",
        "url":"sit-eu.dollin.net",
        "mqttBroker":"mqtxkfi.iot.eu-west-2.amazonaws.com"
    }
]
```

Remark:

If your APP need to service multiple countries, it should call this API to get available countries supported by Midea IoT Open Cloud, the APP also connect to Midea IoT Open Cloud with the server information corresponding to the country selected by user or where is located by APP.

On the other hand, if you APP don't service other country, it don't have to call this API, and can connect to Midea IoT Open Cloud with a consolidated server host, Midea technical support team would provide relevant information.

4.1.3.2. Send a verification code

Request server to send a verification code to the end user. Occasionally you need to call this API again if user would not receive the verification code.

URL: `http://{domain}/v1/user/verify/code/send`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
verifyIdReceiver	true	string	the email or mobile phone number that will receive verification code. NOTE: mobile phone number must prefix with international area code, such as: <u>86138123456786</u>
type	false	int	What the verification code is for? 1:user sign-up 2:forgot password 3:change password 4:login with third party account 5:delete user account use 1 by default if this parameter is not specified in the request.
channel	false	int	When type = 4, this parameter is required. 32:facebook 79:apple 89:Twitter
userFlag	false	int	Specify the type of verifyIdReceiver parameter. 0:email address 1:mobile phone 0 by default.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

4.1.3.3. Verify a verification code

Request server to verification code.

URL: http://{domain}/v1/user/auth/verifyId

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
verifyId	true	string	<p>The verification code which is encrypted.</p> <p>please contact Midea technical support team to get the utility unit for encryption.</p> <p>Once the code is verified successfully, the verification would be invalid in server.</p> <p>You can call this API to verify code by a configurable amount of times in Midea IoT Open Cloud server.</p>
verifyIdReceiver	true	string	Must be the same value as the one when sending verification code.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
verifyId	<p>The “second verification code” used in the specific process.</p> <p>This code is also encrypted like as “verifyId” request parameter.</p> <p>Must use this encrypted code when process get to final step. For example: the APP need to submit user sign-up information to server after user input verification code and verified successfully, then the submit request must contain this code.</p>

4.1.3.4. Check existence for a user

Check if a user has already signed up in Midea IoT Open Cloud server.

URL: http://{domain}/v1/user/exist/verify

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
account	true	string	Email address or a mobile phone number. The mobile phone number must include the international area code with “+” char. Example: 86138123456786
reverse	false	boolean	true or false, false by default. Whether return with a reversed result. true: if account does not exist in server, the response is like:("14001", "User is not existed") false: if account exists in server, the response is like:("14003", "User is already existed")

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

4.1.3.5. Create a new user account

Submit user sign-up information to server.

URL: `http://{domain}/v1/user/register`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
email	false		An email address, it is mandatory when using an email to sign up.
phoneAreaCode	false		The international area code of mobile phone number, it is mandatory when using an mobile phone number to sign up. Such as: 86 for China, 1 for USA.
mobile	false		The mobile phone number, it is mandatory when using an mobile.
countryCode	true		Such as: CN,US See country code for details.
userFlag	false		0: sign up with an email. 1: sign up with an mobile phone number. 0 by default.
verifyIdReceiver	true		Must be same as the “verifyIdReceiver” parameter when calling “user/verify/code/send” API
privateVersion	false		The version number of privacy terms.
password	true		The encrypted password. please contact Midea technical support team to get the utility unit for encryption.
verifyId	true		The verification code returned by “/user/auth/verifyId” API. see “ second verification code ” for more information.
type	false		The user type 0: the end user of APP 1: developer user

			2: manager user.
nickName	false		If not assign this parameter, use email or the mobile phone number by default.
gender	false		0: female 1: male
birthday	false		Format: YYYY-MM-DD
face	false		A picture as the avatar. Format: base64 string.
tenantCode	false		Same as appId that is distributed by Midea.
memo	false		Some extra information for the user.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
accessToken	The token used to call other API.
uid	The user id created by server.
nickName	User nick name
email	Email address
versionCode	Deprecated field.
headPhoto	User avatar picture URL.
expiredDate	The unix timestamp when the accessToken will become expired.

4.1.3.6. User login

A user login to Midea IoT Open Cloud.

URL: http://{domain}/v1/user/login

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
loginAccount	True	string	User account name. Email address or a mobile phone number with international area code. such as: <u>8613812345678</u>
password	True	string	The encrypted password. Refer to “ <u>password</u> ” in “/user/register” API for more information.
phoneModel	false	string	The mobile phone model name Example: IPHONE 6
phoneSysVNum	false	string	The OS version number of mobile phone. Example:IOS 9.3.1
encryptVersion	false	string	Deprecated parameter.
pushToken	false		A push Token for android or IOS phone. Server will compare the server-saved push token(old push token) to this one, if they are different, server would generate a message and push to the phone from which the old push token was.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
accessToken	The token used to call other API.
uid	The user id created by server.
userName	The user name assigned by server.
nickName	User nick name
userFlag	Indicate what the user name is: 0: email 1: mobile phone number 32:a string generated from Facebook user OpenID. 79:a string generated from Apple user OpenID. 89:a string generated from Twitter user OpenID.
migrateStatus	An internal field used by server.
email	Email address
mobile	Mobile phone number.
phoneAreaCode	International area code of mobile phone number. Example:1 for USA, 86 for China.
headPhoto	User avatar picture URL.
versionCode	Deprecated.
expiredDate	The unix timestamp when the accessToken will become expired.
gender	0: female 1: male
birthday	Format:YYYY-MM-DD
memo	Some extra information for the user.

4.1.3.7. Login with a third party account

A user login to Midea IoT Open Cloud with a third party account, such as: Facebook, Twitter or apple ID.

URL: `http://{domain}/v1/user/third/login`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
thirdUID	true		The third-party uid(also called user OpenID), must be encrypted. Refer to “ password ” in “/user/register” API for more information.
token	true		a token generated by third-party application used to login to your APP.
channel	true		Identifier to distinguish third party application. 32: facebook 79: apple 89: twitter
email	false		You can specify an email address to bind with third party UID.
mobile	false		You can specify a mobile phone number to bind with third party UID.
phoneAreacode	false		Specify the international area code of mobile phone number.
countryCode	false		You can specify a country code to bind with third party UID.
userFlag	false		Deprecated parameter.
verifyIdReceiver	false		Deprecated parameter.
extraInfo	false		Some extra information binding to the account created in server when third-party login is success.
password	false		Deprecated parameter. Originally this parameter is required when the email parameter is set with a Facebook

		account.
verifyId	false	Deprecated parameter.
phoneModel	false	The mobile phone model name Example: IPHONE 6
phoneSysVNum	false	The OS version number of mobile phone. Example:IOS 9.3.1
encryptVersion	false	Deprecated parameter.
pushToken	false	A push Token for android or IOS phone. Server will compare the server-saved push token(old push token) to this one, if they are different, server would generate a message and push to the phone from which the old push token was.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	JSON object
accessToken	The access token is required when calling other API.
uid	user id
userName	The user name assigned by server.
userFlag	Indicate what the user name is: 0: email 1: mobile phone number 32:a string generated from Facebook user OpenID. 79:a string generated from Apple user OpenID. 89:a string generated from Twitter user OpenID.
nickName	User nick name.
email	Email address bound to the user.
mobile	Mobile phone number.
phoneAreaCode	International area code of mobile phone number. Example:1 for USA, 86 for China.
headPhoto	User avatar picture URL.
versionCode	Deprecated.

	<code>expiredDate</code>	The unix timestamp when the accessToken will become expired.
	<code>gender</code>	0: female 1: male
	<code>birthday</code>	Format:YYYY-MM-DD
	<code>memo</code>	Some extra information for the user.

4.1.3.8. Bind a third-party account

Binding a third-party account to a login user.

URL: http://{domain}/v1/user/third/bind

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Comment
thirdUID	True	The third-party uid(also called user OpenID), must be encrypted. Refer to “ password ” in “/user/register” API for more information.
token	True	a token generated by third-party application used to login to your APP.
channel	True	Identifier to distinguish third party. application. 32: facebook 79: apple 89: twitter
extraInfo	false	Some extra information binding to the account created in server when third-party login is success.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

4.1.3.9. Unbind a third-party account

Bind a third-party account from a login user.

URL: `http://{domain}/v1/user/third/unbind`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Comment
channel	True	Identifier to distinguish third party application. 32: facebook 79: apple 89: twitter

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

Remark:

User can only bind one for For a specified third party account(Facebook, Twitter or AppleID), this means that you can only bind one Facebook account to a user at all time.

4.1.3.10. Renew the access token

Renew the access token if API return an error code meaning “token is expired.”, see error code [10011](#).

URL: `http://{domain}/v1/user/token/extend`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Comment
nond		

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
accessToken	The new access token.
expiredDate	The expired date of accessToken.

4.1.3.11. Set push token for google Firebase

Your must renew the access token if API calling returns an error code meaning “token is expired.”, see error code [10011](#).

URL: `http://{domain}/v1/common/bind/pushtoken`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
pushToken	True	String	The push token.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

Remark:

Midea IoT Open Cloud support using google Firebase to push message to your APP. With google Firebase SDK, you can use your own Firebase developer account to get a push token from firebase and tell Midea IoT Open Cloud via this API, moreover, you must provide necessary certificate information and we will make a configuration in server, so that Midea IoT Open Cloud can push message to your APP with your own Firebase messaging channel.

On the other hand, if you don't have a google Firebase developer account, Midea IoT Open Cloud could use a default messaging channel to push related message to your APP, and your APP need to integrate google Firebase SDK and import the required certificate provided by Midea IoT Open Cloud.

Midea IoT Open Cloud supports both google Firebase and MQTT messaging, Firebase messaging is usually used for notification triggered by user operation, and the MQTT messaging is for notification when appliance report an event to Midea IoT Open Cloud.

4.1.3.12. User logout

User logout.

URL: http://{domain}/v1/user/logout

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
none			

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

4.1.3.13. Delete an user account

Delete the account of login user.

URL: `http://{domain}/v1/user/account/unregister`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
verifyId	true	String	The verification code returned by “/user/auth/verifyId” API. see “ the second verification code ” for more information.
verifyIdReceiver	true	string	Must be same as the “verifyIdReceiver” parameter when calling “ user/verify/code/send ” API.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

Remark:

It is a cautious and considered behavior to delete the account of a login user, so this API requires a verification code as parameter to help validate deletion operation.

4.1.3.14. Query user information

Query detail information of the login user.

URL: http://{domain}/v1/user/query

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
none			

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
uid	The user id.
status	“0”: normal “1”: blocked
createTime	The create time of user account.
updateTime	The recent update time of user account.
type	“0”: APP user
userName	User name, may be an email address, mobile phone number or a short string representing a user account created by a third-party account.
nickName	Nick name
gender	“0”: female “1”: male
birthday	Format: YYYY-MM-DD
mobile	Mobile phone number without international area code.
phoneAreacode	The international area code for mobile phone number.
email	Email address
headPhoto	The picture URL of user avatar
tenantCode	App Id
language	
memo	

	facebook	A JSON object containing binding information, format: <table border="1"> <thead> <tr> <th>Field</th><th>Comment</th></tr> </thead> <tbody> <tr> <td>bound</td><td>True: a facebook account has bound to user. False: not bound.</td></tr> <tr> <td>identifier</td><td>The open user ID of Facebook.</td></tr> <tr> <td>extra info</td><td>The extra info bound to "identifier".</td></tr> </tbody> </table>	Field	Comment	bound	True: a facebook account has bound to user. False: not bound.	identifier	The open user ID of Facebook.	extra info	The extra info bound to "identifier".
Field	Comment									
bound	True: a facebook account has bound to user. False: not bound.									
identifier	The open user ID of Facebook.									
extra info	The extra info bound to "identifier".									
	apple	A JSON object containing binding information, format: <table border="1"> <thead> <tr> <th>Field</th><th>Comment</th></tr> </thead> <tbody> <tr> <td>bound</td><td>True: a AppleID account has bound to user. False: not bound.</td></tr> <tr> <td>identifier</td><td>The open user ID of AppleID.</td></tr> <tr> <td>extra info</td><td>The extra info bound to "identifier".</td></tr> </tbody> </table>	Field	Comment	bound	True: a AppleID account has bound to user. False: not bound.	identifier	The open user ID of AppleID.	extra info	The extra info bound to "identifier".
Field	Comment									
bound	True: a AppleID account has bound to user. False: not bound.									
identifier	The open user ID of AppleID.									
extra info	The extra info bound to "identifier".									
	twitter	A JSON object containing binding information, format: <table border="1"> <thead> <tr> <th>Field</th><th>Comment</th></tr> </thead> <tbody> <tr> <td>bound</td><td>True: a twitter account has bound to user. False: not bound.</td></tr> <tr> <td>identifier</td><td>The open user ID of Twitter.</td></tr> <tr> <td>extra info</td><td>The extra info bound to "identifier".</td></tr> </tbody> </table>	Field	Comment	bound	True: a twitter account has bound to user. False: not bound.	identifier	The open user ID of Twitter.	extra info	The extra info bound to "identifier".
Field	Comment									
bound	True: a twitter account has bound to user. False: not bound.									
identifier	The open user ID of Twitter.									
extra info	The extra info bound to "identifier".									
	pushTokens	push token list								
	fireBaseToken	fireBase Token								

Example:

```
## response
{
  "code": 0,
  "msg": "success!",
  "data": {
    "uid": "106",
    "status": "0"
    "birthday": null,
    "userFlag": "0",
    "fireBaseToken": "",
    "gender": "0",
    "nickName": "huangxy174@midea.com",
    "facebook": {
      "identifier": "347732473921497",
      "bound": true,
      "extraInfo": ""
    }
  }
}
```

```
},
"mobile": "",
"pushTokens": [
  ""
],
"memo": "",
"updateTime": "2022-07-11 08:21:42",
"language": "",
"tenantCode": 1020,
"type": "0",
"userName": "huangxy174@midea.com",
"phoneAreacode": "",
"headPhoto": "https://x.y.z.com/672ef985-0951-4483-a117-c3284ab1f310.jpg",
"apple": {
  "bound": false
},
"twitter": {
  "bound": false
},
"createTime": "2022-04-22 01:52:46",
"email": "huangxy174@midea.com",
}
}
```

4.1.3.15. Update user information

Update account information of the login user.

URL: http://{domain}/v1/user/update

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
nickName	false	String	Nick name
gender	false	String	0: female 1: male
birthday	false	string	Format: YYYY-MM-DD
mobile	false	String	Mobile phone number without a international area code.
face	false		A user avatar picture data encoded as BASE64 string.
memo	false		

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

4.1.3.16. Modify the password of login user

Modify password for the login user.

URL: `http://{domain}/v1/user/passwd/reset`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
password	true	string	See “ password ” parameter in “Create a new user account” section for details.
verifyId	true	string	See the second verification code for more information.
verifyIdReceiver	true	string	Must be same as the “verifyIdReceiver” parameter when calling “user/verify/code/send” API .

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

4.1.3.17. Change the language code for login user

When APP call API, Midea IoT Open Cloud server could return text message to APP written in a specific language, this API can set the language that server should use when server generated return text message.

URL: `http://{domain}/v1/user/lan/area/setting`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
language	true	string	See “ language code ” for details.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object

Remark:

The language code set by this API will be saved in user account, and be used by some objects running in backend server, such as schedule handler. For the text message returned by an API, server usually writes the message in a response in terms of the “language” HTTP header.

4.1.3.18. Upload an image

This API can upload an image and return a URL locating the image.

URL: `http://{domain}/v1/common/picture/upload`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Required	Data Type	Comment
format	string	false	The image format. Such as: jpg,png
base64	string	true	The image data encoded as BASE64.

Response:

Field	Comment
code	The result code, see response code definition for details.
msg	A response message.
data	Json object
fileName	The file name saved in server.
url	The URL locating the image. This URL will keep valid within a period of time, currently it will become expired after 24 hours.
filePath	null
fileType	IMEI type, such as: image/jpg

4.1.4. Response Code definition

Code	Description
0	success
10XXX	General error
11XXX	Error code related to user sign-up.
12XXX	Error code related to user login
10000	System exception
10001	Some parameter is missed or the format of parameter is incorrect.
10002	The secret key is not existed in server.
10003	Signature is expired.
10004	Signature is wrong.
10010	Token is invalid.
10011	token is expired.
11001	The verification is expired.
11002	Failed to send verification code.
12001	The user account is not existed.
12002	User name or password is wrong.
11003	The verification code is invalid.
13001	Failed to create user account in server.
14003	The user is already existed.
9503	The appliance has already bound to user.
1130	Some Parameter is wrong.
99999	System error.
14001	Account does not exist.

4.2. HTTP API for Appliance Management

Midea IoT Open Cloud provides a suit of APIs to manage appliance device , some of APIs need to use with MSmartSDK to complete appliance network configuration.

4.2.1. HTTP Common header

HTTP header	Data type	Required	Description
ClientId	String	Y	Distributed by Midea, see client id for more information.
Authorization	String	N	Format: Bearer <token> Except for the “/token” API, all of other API require to set this header.
SignatureVersion	String	Y	The version number of signature. Currently use “2.0”
Signature	String	Y	A hash value calculated on request parameters.

4.2.2. Security

Most of APIs need to contain a “signature” value as the fingerprint of the parameters in http request, the signature is calculated based on a composed way by using hmacSHA256, base64 encoding.

4.2.2.1. Generate the source string

The source string is used to calculate signature, and consists of multiple parts. It is a string concatenated by following order and format:

```
<http method><API URI><request query string><request body(utf-8 format)>
```

For example:

```
#request
POST
/v1/open/device/list/get?uid=123&uname=tom
HTTP/1.1
Host: xx.com
Content-Type: */*
```

```
{"reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01", "applianceList": [{"applianceCode": "17592186044420", "type": "0xAC", "name": "airc", "modelNumber": "1", "onlineStatus": "1"}]}
```

```
#source_string =
POST/v1/open/device/list/getuid=123&uname=tom{"reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01", "applianceList": [{"applianceCode": "17592186044420", "type": "0xAC", "name": "airc", "modelNumber": "1", "onlineStatus": "1"}]}
```

4.2.2.2. Calculate signature

```
Signature = Base64(HMAC_SHA_256(client_secret,source_string))
```

NOTE: client_secret is created and distributed to developer by Midea.

4.2.2.3. Implementation reference

```
1. import org.apache.commons.codec.binary.Base64;
2. import org.apache.commons.codec.digest.HmacAlgorithms;
3. import org.apache.commons.codec.digest.HmacUtils;
4. import org.apache.commons.lang3.StringUtils;
5. import org.slf4j.Logger;
6. import org.slf4j.LoggerFactory;
7.
8. import java.net.URLDecoder;
9. import java.nio.charset.StandardCharsets;
10.
11. /**
12. * 签名工具类
13. *
14. * created on 2021/10/18
15. * @author hejian34
16. */
17. public class SignatureUtil {
18.
19.     private static final Logger log = LoggerFactory.getLogger(SignatureUtil.class);
20.
21.     public static String getSignature(String clientSecret,
22.                                     String requestMethod, String request
23.                                     URI,
```

```
23.                                     String queryString, String body) {
24.             return getSignature(clientSecret.getBytes(StandardCharsets.UTF_8),re
25.                 questMethod,requestURI,
26.                     queryString,body);
27.
28.         public static String getSignature(byte[] clientSecret,
29.                                         String requestMethod, String request
30.                                         URI,
31.                                         String queryString, String body) {
32.             try {
33.                 String queryStringAfterDeal = StringUtils.isEmpty(queryString) ?
34.                     ""
35.                         : URLDecoder.decode(queryString,
36.                             "utf-8");
37.                 String valueToDigest = requestMethod + requestURI +
38.                     queryStringAfterDeal + body;
39.                 log.debug("requestURI:{} ,queryString:{} ,body:{}" ,requestURI,quer
40.                     yStringAfterDeal, body);
41.                 String signatureResult = createSignature(clientSecret, valueToDi
42.                     gest);
43.                 log.debug("signatureResult:{}" , signatureResult);
44.                 return signatureResult;
45.             } catch (Exception e) {
46.                 log.error("createSignature error " , e);
47.             }
48.             return null;
49.         }
50.         /**
51.          * 生成消息认证码
52.          *
53.          * @param clientSecret 密钥
54.          * @param valueToDigest 待签名字段
55.          * @return
56.          */
57.         public static String createSignature(byte[] clientSecret,
58.                                         final String valueToDigest) {
59.
60.             final byte[] digest = new HmacUtils(HmacAlgorithms.HMAC_SHA_256, cli
entSecret)
61.                 .hmac(valueToDigest);
62.
63.             return Base64.encodeBase64String(digest);
```

```
61.     }
62.
63.     public static void main(String[] args) {
64.         System.out.println( );
65.     }
66. }
```

4.2.3. API

4.2.3.1. Get a device token

Get a device token from Midea IoT Open cloud server.

URL: `http://{domain}/midea/open/business/v1/token`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
openUserId	String	Y	An open user ID that is mapping to a real user id. If you has a legacy user account system and continue to use it when integrating Midea Open Smart, you could generate an openUserId for each user like: <code>openUserId = md5(uid)</code> or it's also ok to directly use uid as openUserId, if you believe enough in Midea IoT Open Cloud server. Once you use an open user id, must keep it same in future.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	Json object
accessToken	string	The devcie token.
adsId	string	The ADS ID, see ADS for more information. NOTE: MSmartSDK require ADS resource when configuring appliance network.
adsDomain	string	ADS server host.

	adsPort	int	The port of ADS server
--	---------	-----	------------------------

Example:

```
## request
POST {host}/midea/open/v1/token
Content-Type: application/json
{
    "reqId": "98200f78-98d3-467f-a82e-123e7910deal",
    "openUserId": "100"
}

## response
{
    "code": 0,
    "msg": "",
    "data": {
        "accessToken": "a5e49368a8eb46648a23194229a34522",
        "adsId": "65794",
        "adsDomain": "iotlab.midea.com.cn",
        "adsPort": 28443
    }
}
```

4.2.3.2. Get ADS resource

Get ADS resource information.

URL: http://{domain}/midea/open/business/v1/ads

Body format: JSON

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	Json object
	adsId	string The ADS ID, see ADS for more information. NOTE: MSmartSDK require ADS resource when configuring appliance network.
	adsDomain	string ADS server host.
	adsPort	int The port of ADS server

Example:

```
## request
POST {host}/midea/open/v1/token
Content-Type: application/json
{
    "reqId": "98200f78-98d3-467f-a82e-123e7910deal",
    "stamp": "1593740640770"
}

## response
{
    "code": 0,
    "msg": "",
```

```
"data": {  
    "adsId": "65794",  
    "adsDomain": "moduletest.appsmb.com",  
    "adsPort": 28443  
}  
}
```

4.2.3.3. Get RegionID

Query the region id in terms of a specified time zone.

URL: http://{domain}//midea/open/business/v1/area/city

Body format: JSON

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
cityCode	String	Y	A city code represented as time zone ID, refer to https://nodatime.org/TimeZones for detail.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	
cityId	int	The region id

Example:

```
## request
POST {host}/midea/open/business/v1/appliance/auth
Content-Type: application/json
{
  "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
  "stamp": "1593740640770",
  "cityCode": "America/Anchorage"
}
## response
{
  "code": 0,
  "msg": "",
  "data": {
    "cityId": "10246"
  }
}
```

4.2.3.4. Get channel list

Get Wi-Fi channel setting information for a specific country.

URL: `http://{domain}/midea/open/business/v1/country/channel/retrieve`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
countryCode	String	Y	Example: CN,US See country code for details.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	json array
channelNum	String	The first channel number.
channelCounts	String	Amount of channel.
maxTransferPower	String	The maximum power of transmission.
dfs	String	The enable flag of DFS feature. 1:enable 0:disable
countryCode	String	The country code.
country	String	The country name.

Example:

```
## request
POST {host}/midea/open/business/v1/country/channel/retrieve
Content-Type: application/json
{
    "reqId": "98200f78-98d3-467f-a82e-123e7910dea1",
    "stamp": "1593740640770",
```

```
        "countryCode": "US"
    }
## response
{
    "code": 0,
    "msg": "",
    "data": [
        {
            "channelNum": "36",
            "channelCounts": "4",
            "maxTransferPower": "17",
            "dfs": "0",
            "countryCode": "US",
            "country": "USA"
        },
        {
            "channelNum": "52",
            "channelCounts": "4",
            "maxTransferPower": "24",
            "dfs": "1",
            "countryCode": "US",
            "country": "USA"
        },
        {
            "channelNum": "100",
            "channelCounts": "11",
            "maxTransferPower": "24",
            "dfs": "1",
            "countryCode": "US",
            "country": "USA"
        },
    ],
}
```

4.2.3.5. Bind an appliance to user

Bind appliance to the user represented by [openUserID](#) in cloud server.

URL: http://{domain}/midea/open/business/v1/bind

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
verificationCode	String	Y	MSmartSDK can get the verification code when it completes network configuration process, see how to get verification code in the example code.
applianceCode	String	Y	The unique device ID, you can also get the device id referring to example code

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	

Example:

```
## request
POST {host}/midea/open/business/v1/bind

Content-Type: application/json
{
    "reqId": "98200f78-98d3-467f-a82e-123e7910dea1",
    "stamp": "123456789",
    "applianceCode": "1234567890",
    "verificationCode": "123456789"
}
## response
```

```
{  
    "code": 0,  
    "msg": "",  
    "data": null  
}
```

4.2.3.6. Unbind an appliance from user

Bind appliance from the user represented by [operUserID](#) in cloud server.

URL: http://{domain}/midea/open/business/v1/unbind

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance to unbind.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	

Example:

```
## request
POST {host}/midea/open/business/v1/unbind
Content-Type: application/json
{
    "reqId": "98200f78-98d3-467f-a82e-123e7910dea1",
    "stamp": "123456789",
    "applianceCode": "1234567890"
}
## response
{
    "code": 0,
    "msg": "",
    "data": null
}
```

4.2.3.7. Get authentication state of appliance

Query the authentication status for a specified appliance.

URL: http://{domain}/midea/open/business/v1/appliance/auth/status

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance id.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	
	status	<p>Status code.</p> <p>0: the appliance has authenticated.</p> <p>1: the appliance has entered authentication state and is waiting to finish.</p> <p>2: the appliance has not authenticated yet.</p> <p>3: the appliance does not support authentication process.</p>

Example:

```
## request
POST {host}/midea/open/v1/appliance/auth/status
Content-Type: application/json
{
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
    "stamp": "1593740640770",
    "applianceCode": "17592186044420"
}
## response
{
```

```
"code":0,  
"msg":"",
"data":{  
    "status":1  
}  
}
```

4.2.3.8. Authenticate an appliance

Authenticate appliance in Midea IoT Core.

URL: http://{domain}/midea/open/business/v1/appliance/auth

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance id.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	

Example:

```
## request
POST {host}/midea/open/business/v1/appliance/auth
Content-Type: application/json
{
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
    "stamp": "1593740640770",
    "applianceCode": "17592186044420"
}
## response
{
    "code": 0,
    "msg": "",
    "data": null
}
```

4.2.3.9. Get appliance state

Get the status of a specified appliance.

URL: `http://{domain}/midea/open/business/v1/appliance/fullQuery`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance id.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	
query	Json	JSON object. Include full state of appliance, the content is different for variant of Midea products. the state definition for appliance is same as the command definition.
fault	Json	Include fault code if appliance is in malfunction state.

Example:

```
## request
POST {host}/midea/open/business/v1/appliance/status
Content-Type: application/json
{
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
    "stamp": "1593740640770",
    "applianceCode": "17592186044420"
}
```

```
## response
{
  "code": 0,
  "msg": "",
  "data": {
    "query": {
      "power": "1"
    },
    "fault": {
      "faultCode": "0"
    }
  }
}
```

4.2.3.10. Send JSON command

Send a command to appliance by a JSON string.

URL: http://{domain}/midea/open/business/v1/appliance/control/json

Body format: application/json

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance id.
query	Boolean	N	<p>Indicate whether cloud server need to query full state of appliance before send control instruction.</p> <p>Default value is false.</p> <p>Note: the full state contains all mandatory properties of appliance, some Midea product requires a full state to control due to not supporting control with incomplete property.</p>
instruction	json	Y	<p>Command formatted by json, format:</p> <pre>{ <command key1>:<command value1>, <command key2>:<command value2> }</pre> <p><command value> is of string type.</p> <p>Example:</p> <pre>{ "power": "1", "temp": "26", "mode": "2", "windSpeed": "40" }</pre> <p>NOTE: Midea defines relevant commands with a thing model based on JSON for each product respectively, please contact Midea technical support team to get the</p>

			corresponding thing model for your products. The appendix includes the thing model definition for air conditioner product.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	json	null

Example:

```
## request
POST {host}/midea/open/v1/appliance/control
Content-Type: application/json
{
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
    "stamp": "1593740640770",
    "applianceCode": "17592186044420",
    "query": true,
    "instruction": {
        "power": "on",
        "temp": 23
    }
}

## response
{
    "code": 0,
    "msg": "",
    "data": null
}
```

4.2.3.11. Send hexadecimal command

Send a command to appliance by a hexadecimal command string.

URL: `http://{domain}/midea/open/business/v1/appliance/control/hexadecimal`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance id.
command	String	Y	a hexadecimal command string.

Response:

Field	Data Type	Comment
code	int	0:success. 1016:time out. 1202:the appliance does not belong user. 1201:the appliance does not exist. 9003:parameter can not be empty. others:please consult to Midea technical support team for details.
msg	string	A response message.
data	string	A string represented command execution result.

Remark:

A hexadecimal command is constructed based on a binary command structure, it is required to learn about msmaart protocol that is the binary communication specification between appliance and Midea IoT Core.

Example:

```
## request
POST {host}/midea/open/business/v1/appliance/control/hexadecimal
Content-Type: application/json
```

```
{  
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",  
    "stamp": "1593740640770",  
    "applianceCode": "17592186044420",  
    "command": "000102030405060708090a0b0c0d0e"  
}  
  
## response  
{  
    "code": 0,  
    "msg": "",  
    "data": "5a,5a,01,00,5a,00,20,80,ad,29,00,00,38,0c,06,00,1c,06,15,14,ad,1d,00,0  
0,00,09,00,00,00,00,00,00,00,01,80,00,00,00,00,aa,21,fb,00,00,00,00,00,02,03,01,  
00,04,03,0b,00,00,00,02,00,00,00,00,00,00,00,00,00,00,10,0a,00,01,af,00,00,00,00,  
00,00,00,00,00,00,00,00,00,00,00"  
}
```

4.2.3.12. Get appliance list

Retrieve all appliances binding to the login user represented by device token.

URL: `http://{domain}/midea/open/business/v1/appliance/list`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	string	A string represented command execution result.
applianceCode	string	Appliance id
name	string	Appliance name
applianceType	string	Category code
activeStatus	int	Activated status 0: appliance not activated. 1: activated.
online	int	0:offline 1:online, appliance is connected to server.
modelNumber	string	Model number
sn	string	The serial number of appliance
sn8	string	A serial number with 8 characters.
productImage	string	Image URL for appliance.
authStatus	int	Authentication status code. 0: the appliance has authenticated. 1: the appliance has entered authentication state and is waiting to finish. 2: the appliance has not authenticated yet.

		3: the appliance does not support authentication process.
--	--	---

Example:

```
## request
POST {host}/midea/open/business/v1/appliance/list

Content-Type: application/json
{
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
    "stamp": "20191106142639"
}

## response
{
    "code": 0,
    "msg": "success",
    "data": [
        {
            "applianceCode": "9895604656561",
            "name": "smart door lock",
            "applianceType": "0x09",
            "modelNumber": "0",
            "online": null,
            "activeStatus": 0,
            "sn": "0000093112001095015140000000UP70",
            "sn8": "20010950",
            "productImage": "http://x.y.com/img/prodimg/pm632A2082.png",
            "authStatus":0
        }
    ]
}
```

4.2.3.13. Update appliance information

Update information of appliance, currently it only support to modify appliance name.

URL: `http://{domain}/midea/open/business/v1/appliance/updateDeviceInfo`

Body format: `application/json`

HTTP method: POST

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	The request ID, usually use a UUID.
stamp	String	N	A unix timestamp by millisecond precise.
applianceCode	String	Y	The appliance id.
name	String	Y	A new name of appliance.

Response:

Field	Data Type	Comment
code	int	The result code, see response code definition for details.
msg	string	A response message.
data	Json	null

Example:

```
## request
Content-Type: application/json
{
    "reqId": "fe8234bf-e94c-4cdf-8ea9-c3112962ab01",
    "stamp": "20191106142639",
    "applianceCode": "9895604656561",
    "name": "哈哈"
}
## response
{
    "code": 0,
    "msg": "success",
    "data": null
}
```

4.2.4. Response Code definition

Code	Description
0	success
1204	The appliance is offline.
5000	Client id is invalid. See also: <u>client id</u>
5001	Failed to subscribe appliance in server.
5300	Appliance is not existed in server.
6001	Unsupported authentication type for appliance.
9001	Parameter is wrong.
1100001	Failed to get appliance full status.
1100002	The format of command sent to appliance is wrong.
1100003	Failed to control appliance.
9998	Failed to call system service.
9999	System error, usually exception occurred.

5. Receiving appliance upload data

An appliance may actively upload its status or data when running, you can receive them in cloud side by providing a web hook endpoint, Midea IoT Open Cloud will call your web hook endpoint once it receive an uploaded data from appliance, your cloud must implement the web hook by compliance.

URL: `http://{domain of your cloud}/v1/oem2third/appliance/data/report`

Body format: `application/json`

HTTP method: POST

HTTP Header:

HTTP Header	Data Type	Required	Comment
ClientId	String	y	Distributed by Midea, see client id for more information.
Signature	String	y	A hash value calculated on request parameters, please refer to security section on how to calculate the signature for a request.
SignatureVersion	String	y	The version number of signature. Currently use "2.0"

Request Parameter:

Parameter	Data Type	Required	Comment
reqId	String	N	A request id used to identify a single call.
stamp	String	N	the unix time stamp(UTC+0) when issuing a call from Midea IoT Open Cloud.
applianceCode	String	Y	Appliance id
msgId	String	Y	The message id to identify a upload from appliance.
msgTime	Long	Y	The unix time stamp(UTC+0) when appliance upload data, with ms precision.
hexData	String	Y	The uploaded data of appliance, in hex

			<p>string format, you can decode it in terms of Midea msmart protocol.</p> <p>example:</p> <p>"AA0034EB87C23723DE873EB87C2372"</p>
jsonData	Json	N	<p>The uploaded data of appliance, representing in thing model, this field is only available when the appliance has a thing model implementation in Midea IoT Open Cloud, otherwise it will set as Null.</p> <p>Example:</p> <p>{"power":"0", "mode":"2", "eco":"1"}</p>

Response:

After your cloud handles a web hook call from Midea IoT Cloud, your cloud should return a JSON response defined as below.

Field	Data Type	Comment
code	int	<p>0 means success. Other value means failure.</p> <p>Note: Midea IoT Open Cloud won't make a retry call although code is not 0.</p>
msg	string	A response message.

Example:

```
## request

POST {host} /v1/oem2third/appliance/data/report

Content-Type: application/json

{
    "reqId": "hxxbhlg5vueth0p05qnpssc52h78c5b7i",
    "stamp": "20191106142639",
    "applianceId": "30786325578468",
    "msgId": 1,
    "msgTime": 1672185600467,
```

```
"dataFormat":1,  
"reportData":"{\"power\":\"0\"}  
}  
  
## response  
{  
"code": 0,  
"msg": ""  
}
```

Remark:

Your cloud can receive web hook calls only after the appliance has already bound to a user.

For the sake of security consideration, your cloud should verify the “signature” value of the request from Midea IoT Open Cloud, and it’s better accept HTTPS request for your cloud.

6. Appendixes

6.1. Error code of MSmartSDK

Error code	Description	Cause and action
0	The method call is success.	
4003	MSmartSDK failed to connect Wi-Fi router because of a wrong password.	Ensure to provide a right Wi-Fi password when setting network for appliance.
4004	It is timeout to connect Wi-Fi router.	App may pop up a UI to guide user manually connect Wi-Fi router again.
4005	MSmartSDK failed to connect Wi-Fi router due to other errors.	
4008	MSmartSDK failed to connect to Wi-Fi AP of appliance because the password is wrong.	In the Wi-Fi setting page of mobile, select the Wi-Fi AP entry naming like "mdiea_xx_xxxx", and do "forget password" or "don't save" operation.
4009	It is timeout to connect to Wi-Fi AP of appliance.	App may pop up a UI to guide user manually connect the Wi-Fi AP of appliance again.
4010	Failed to connect to Wi-Fi AP of appliance due to other errors.	
4011	MSmartSDK cannot find the device before timeout when setting network for appliance by Wi-Fi AP.	<p>1) The Wi-Fi signal of the appliance is not stable.</p> <p>2) The mobile phone switch to another Wi-Fi connection instead of the appliance AP.</p> <p>3) After connected to the Wi-Fi AP of the appliance, user make the APP foreground before the Wi-Fi connection icon appears in the status bar of mobile phone.</p> <p>This could happen probably when connecting to a Wi-Fi AP and password is not required. Because the mobile system has some limitation against this kind of Wi-Fi and needs a longer period to connect it.</p> <p>Action: get the mobile phone closer to the appliance if cause 1).</p>
4013	It is timeout to find the appliance when the Wi-Fi module of appliance works	<p>1) Set a wrong password for the Wi-Fi module of appliance.</p> <p>2) The Wi-Fi router is overloaded and the</p>

Error code	Description	Cause and action
	under STA mode.	appliance can not connect to it. 3) Something is wrong in the Wi-Fi module of appliance. You can get help from Midea technical support team and get the log of Wi-Fi module to analyze.
4014	MSmartSDK received acknowledge timeout when setting a temporary ID for appliance.	1) the Wi-Fi AP is not stable. 2) The TCP connection between mobile phone and the appliance terminated abnormally.
4015	MSmartSDK got an error return when setting a temporary ID for appliance.	Possible cause: 1) The Wi-Fi module of appliance get something wrong. 2) The Wi-Fi module was set a different configuration mode than the MeiJu-APP configuration mode in multi-mode supported module.
4016	Some network parameter worked failed.	The network is not stable.
4017	Set a temporary ID with exception.	The Wi-Fi module probably close the socket.
4018	Other error occurred when setting a temporary ID.	Please contact Midea technical support.
4023	It is timeout without acknowledge when setting password for SSID for new Wi-Fi module of appliance.	When setting network configuration for appliance by Wi-Fi AP, command 0x70 got no acknowledge: 1) The TCP connection between mobile phone and the appliance terminated abnormally. 2) After connected to the Wi-Fi AP of the appliance, user make the APP foreground before the Wi-Fi connection icon appears in the status bar of mobile phone. This could happen probably when connecting to a Wi-Fi AP and password is not required. Because the mobile system has some limitation against this kind of Wi-Fi and needs a longer period to connect it. 3) the Wi-Fi AP is not stable. 4) Once the Wi-Fi module of appliance received the command 0x70, it immediately switched to STA mode from AP mode, this would disconnect the TCP connection between mobile phone and the appliance and no acknowledge would return to MSsmartSDK.

Error code	Description	Cause and action
4024	when setting password for SSID for old appliance Wi-Fi module, it returned an error.	When setting network configuration for appliance by Wi-Fi AP, command 0x68 sent by MSmartSDK got no acknowledge from appliance.
4025	It is timeout without acknowledge when setting password for SSID for old Wi-Fi module of appliance.	When setting network configuration for appliance by Wi-Fi AP, command 0x68 got no acknowledge: 1) The TCP connection between mobile phone and the appliance terminated abnormally. 2) the Wi-Fi AP is not stable.
4026	(Network Config by AP) MSmartSDK can not parse the returned data when writing config data to appliance.	It is probable that the appliance can not send the completed data due to an unsteady network connection.
4027	(Network Config by AP) I/O error occurred when writing config data to appliance.	When setting network configuration for appliance by Wi-Fi AP, if there is I/O error occurred, the usual reason is that the socket connection is closed actively; on the other hand, it is also possible the Wi-Fi module of appliance maybe something wrong. User can get mobile phone closer to the appliance and try again, if get failed after multiple retries, user can contact Midea customer service for help.
4028	(Network Config by AP) Unknown error occurred when writing config data to appliance.	Need to contact Midea technical support.
4029	(Network Config by AP) The Wi-Fi module was failed to enter STA mode.	1) the Wi-Fi AP is not stable. 2) The TCP connection between mobile phone and the appliance terminated abnormally. 3) Once the Wi-Fi module of appliance received command and immediately switched to STA mode from AP mode, this would disconnect the TCP connection between mobile phone and the appliance and no acknowledge would return to MSmartSDK.
4031	(Quick Network Config) Under the Quick Network config mode, MSmartSDK	1) Set a wrong password for the Wi-Fi module of appliance. 2) The Wi-Fi router is overloaded and the

Error code	Description	Cause and action
	find appliance timeout in LAN side or cloud server side.	appliance can not connect to it. 3) Something is wrong in the Wi-Fi module of appliance. 4) If The Wi-Fi router is running in either 11bgn fixed or 11n mode, please make it running in other mode. 5) The appliance module does not support Quick Network config.
4032	MSmartSDK does not support configuration on this appliance module, need to upgrade SDK.	The version of MSmartSDK is not matched with the firmware version of appliance.
4033	(Network Config by AP) Failed to open the WiFi of mobile phone.	
4034	(Network Config by AP) The encryption method of router is not supported by appliance WiFi module.	
4035	(Network Config by AP) The password was wrong when appliance tried to connect to router.	
4036	(Network Config by AP) It was time out when appliance tried to connect to router.	
4037	(Network Config by AP) Appliance failed to connect to router due to other reason.	
4038	(Network Config by AP) I/O error occurred when connecting to appliance.	Appliance maybe closed the socket object.
4039	(Network Config by AP) Failed to connect to appliance due to other reason.	Possible reason: 1. Appliance does not listen to a port. 2. Appliance is listening to a port that is different to the port in broadcast datagram.
4040	(Network Config by AP) The command A0 can not get appliance information.	The network is not stable probably.
4041	(Network Config by AP)	The reason may be that appliance closed the

Error code	Description	Cause and action
	The command A0 occurred exception.	socket proactively.
4042	(Network Config by AP) It is time out for MSmartSDK to receive response of command A0.	Appliance didn't respond to command A0, or did not send response before timeout.
4043	(Network Config by AP) The format is incorrect for response data against command A0.	
4044	(Network Config by AP) Other error occurred for command A0.	You can contact Midea technical support team for help.
4045	(Quick Network Config) The encryption method of router is not supported by appliance WiFi module.	
4046	(Quick Network Config) The password was wrong when appliance tried to connect to router.	
4047	(Quick Network Config) It was time out when appliance tried to connect to router.	
4048	(Quick Network Config) Appliance failed to connect to router due to other reason.	
4049	(Quick Network Config) Failed to send multicast datagram.	
4050	(Network Config by Bluetooth) The blueooth connection is disconnected.	Recommended actions: 1. Reset Bluetooth and retry. 2. Make mobile phone closer to the appliance.
4051	(Network Config by Bluetooth) Failed to get the public key.	
4052	(Network Config by Bluetooth) Failed to verify the	

Error code	Description	Cause and action
	secret key.	
4053	(Network Config by Bluetooth) Failed to parse the response data sent by appliance.	Unknown reply for command 0x69.
4054	(Network Config by Bluetooth) Failed to verify the existence of appliance in cloud server.	
4055	(Network Config by Bluetooth) Token is invalid when configuring appliance network or control appliance.	
4056	(Network Config by Bluetooth) Can not find SSID.	Command 0x63, reply from appliance.
4057	(Network Config by Bluetooth) Failed to connect to router.(the first generation bluetooth module)	Command 0x63, reply from appliance: 1: password is wrong. 2: the phone is too far away from appliance. 3: 5G band is not supported by appliance. 4: MAC address whitelist. 5: DHCP function is not enabled. (configuration in router)
4058	(Network Config by Bluetooth) Appliance failed to resolve server DNS domain.	Command 0x63, reply from appliance: 1. WiFi connection disconnects intermittently. 2. Router does not allow to access Midea Cloud IoT Core. 3. Router was set with a specified DNS server.
4059	(Network Config by Bluetooth) Failed to establish TCP connection to server.	Command 0x63, reply from appliance.
4060	(Network Config by Bluetooth) Heartbeat communication is timeout.	Command 0x63, reply from appliance.
4061	(Network Config by	Command 0x63, reply from appliance.

Error code	Description	Cause and action
	Bluetooth) SST error occurred during login.	
4062	(Network Config by Bluetooth) Appliance restarted proactively.	Command 0x63, reply from appliance.
4063	(Network Config by Bluetooth) Appliance restarted passively.	Command 0x63, reply from appliance.
4064	(Network Config by Bluetooth) Failed to authenticate SDK.	Command 0x63, reply from appliance.
4065	(Network Config by Bluetooth) Server closed connection proactively during login.	Command 0x63, reply from appliance.
4066	(Network Config by Bluetooth) Failed to send data during login.	Command 0x63, reply from appliance.
4079	(Network Config by Bluetooth) Bluetooth is not opened.	The Bluetooth is not enabled in mobile phone.
4082	(Network Config by Bluetooth) Other Bluetooth error occurred.	
4083	(Network Config by Bluetooth) Failed to resume network configuration.	
4084	(Network Config by Bluetooth) It's time out for Bluetooth communication.	
4090	(Network Config by AP) Appliance returned response indicating password is incorrect.	
4091	(Network Config by 2 nd	It's based on the protocol for 2 nd

Error code	Description	Cause and action
	Generation Bluetooth) Failed to get license.	Bluetooth. it's possible to fail to negotiate secret key, or cloud server returned empty license(cloud server does not configure the corresponding license for appliance MAC address.) If the endpoint has error internally, it will return an error code.
4092	(Network Config by 2 nd Generation Bluetooth) Failed to negotiate secret key in terms of license.	
4093	(Network Config by 2 nd Generation Bluetooth) Command 0x69, failed to establish connection.(including error code returned by appliance.)	
4094	(Network Config by 2 nd Generation Bluetooth) Appliance returned an error indicating that the router password is incorrect.	
4095	Appliance returned error indicating the Wi-Fi channel is not supported when writing network setting parameters to the WiFi module.	
4096	(Network Config by Bluetooth) Appliance returned error indicating	
4097	(Network Config by Bluetooth) Appliance returned that Wi-Fi parameter is incorrect.	
4098	(Network Config by Bluetooth) Appliance returned that	

Error code	Description	Cause and action
	the router signal is too weak.	
4099	(Network Config by Bluetooth) Appliance returned that DHCP function failed.	You can enable DHCP feature in router configuration UI.
4100	(Network Config by Bluetooth) Appliance returned that router authenticated with failure.	
4101	(Network Config by Bluetooth) Appliance returned error related to router.	
4115	(Network Config by 2nd MSmartSDK received or retrieved an error code (0x10) indicating login to sever timed out.	
4116	(Network Config by 2nd Generation Bluetooth) MSmartSDK received or retrieved an error code (0x11) indicating the Wi-Fi module failed to scan router AP.	Context: The Wi-Fi channel is in 1-13, and SSID contains “5G/-5G/_5G” letters.
4117	(Network Config by 2nd Generation Bluetooth) MSmartSDK received or retrieved an error code (0x12) indicating the Wi-Fi module failed to scan router AP.	Context: The Wi-Fi channel is in 1-13, and SSID does not contain “5G/-5G/_5G” letters.
4118	(Network Config by 2nd Generation Bluetooth) MSmartSDK received or retrieved an error code (0x13) indicating the Wi-Fi module failed to scan router AP.	Context: The Wi-Fi signal channel is 0, and SSID does not contain “5G/-5G/_5G” letters.
4119	(Network Config by 2nd Generation Bluetooth)	Context: The Wi-Fi signal channel is greater than

Error code	Description	Cause and action
	MSmartSDK received or retrieved an error code (0x14) indicating the Wi-Fi module failed to scan router AP.	13, and SSID does not contain “5G/-5G/_5G” letters.
4120	(Network Config by 2nd Generation Bluetooth) MSmartSDK received or retrieved an error code (0x15) indicating the Wi-Fi module failed to scan router AP.	Context: The Wi-Fi signal channel is greater than 13, and SSID contains “5G/-5G/_5G” letters.
4121	(Network Config by 2nd Generation Bluetooth) MSmartSDK received or retrieved an error code (0x16) indicating the Wi-Fi module failed to scan router AP.	Context: The Wi-Fi signal channel is 0, and SSID contains “5G/-5G/_5G” letters.
4123	(Network Config by Bluetooth) Appliance returned country code is wrong.	Context: command 0x67
4124	(Network Config by Bluetooth) Appliance returned time zone is wrong.	Context: command 0x67
4125	(Network Config by Bluetooth) Appliance returned Wi-Fi channel list is wrong.	Context: command 0x67
4127	(Network Config by AP) Appliance returned country code is wrong.	Possible reason: 1. APP does not provide a value for country code parameter when calling method of MSmartSDK. 2. The value of country code contains lower-case letter, it must be all capital letters(A-Z).
4128	(Network Config by AP) Appliance returned time zone is wrong.	Possible reason: 1. APP does not provide a value for time zone parameter when calling method of MSmartSDK. 2. The value of time zone is greater than

Error code	Description	Cause and action
		24.
4129	(Network Config by AP) Appliance returned Wi-Fi channel list is wrong.	Possible reason: 1. APP does not provide a channel list when calling method of MSmartSDK. 2. The number of entry in channel list is less than 1 or greater than 10.
4130	(Network Config by AP) Failed to send country code or channel list.	The probable reason is that country code or channel list is missing or in incorrect format.
4131	Bluetooth mesh device, failed to get deviceId from cloud server.	Possible reason: the interface request is failed.
4132	Bluetooth mesh device, Failed to send netkey information to appliance.	Possible reason: 1. Bluetooth connection is broken. 2. Frequency deviated.
4133	Bluetooth mesh device, The time to send netkey information is not right.	Possible reason: The appliance has already connected to network, but still try to send netkey information to appliance.
4134	Failed to send the response returned by mesh device.	Possible reason: the interface request is failed.
4135	(Network Config by AP) Appliance entered AP working mode again.	Possible reason: 1. password is wrong. 2. Appliance Wi-Fi module does not support the routing format. Only exists in android 1307(seemingly password is incorrect.)
4136	(Network Config by AP) The domain name is incorrect.	Possible reason: 1. In the extra function datagram broadcasted by 2in1 module of appliance, when the bit0 of high byte equals to 1, the APP does not provide the server domain to MSmartSDK. 2. The length of server domain is greater than 50.
4137	(Network Config by AP) The server port is incorrect.	The APP does not provide the port number to MSmartSDK.(the port number flag is 1 in the extra function of 2in1 module.)
4138	(Network Config by AP) The region ID is incorrect.	The APP does not provide the region ID to MSmartSDK.(the region ID flag is 1 in the extra function of 2in1 module.)

Error code	Description	Cause and action
4139	(Network Config by AP) The function type is incorrect.	The APP does not provide the function type to MSmartSDK.(the function type flag is 1 in the extra function of 2in1 module.)
4140	(Network Config by Bluetooth) The domain name is incorrect.	The APP does not provide the domain name to MSmartSDK.(2in1 module, the flag in the appliance Bluetooth broadcast indicates 2in1 firmware)
4141	(Network Config by Bluetooth) The server port is incorrect.	The APP does not provide the port number to MSmartSDK.(2in1 module, the flag in the appliance Bluetooth broadcast indicates 2in1 firmware)
4142	(Network Config by Bluetooth) The region ID is incorrect.	The APP does not provide the region ID to MSmartSDK.(2in1 module, the flag in the appliance Bluetooth broadcast indicates 2in1 firmware)
4143	(Network Config by Bluetooth) The function type is incorrect.	The APP does not provide the function type to MSmartSDK.(2in1 module, the flag in the appliance Bluetooth broadcast indicates 2in1 firmware)
4144	(Network Config by Bluetooth) The appliance does not support 5G WiFi.	
4145	(Network Config by Bluetooth) The appliance has been bound.	If appliance has been bound, it should not enter network configuration mode, need to analyse the broadcast packet.
4146	(Network Config by Bluetooth) Appliance failed to receive data	The first generation Bluetooth, appliance responds with 0xff against command 0x67.
4147	Appliance does not support Bluetooth biding.	Possible reason: The firmware in appliance is not correct, it requires application to verify.
4148	Failed to query post-authentication state via Bluetooth.	
4149	Failed to set post-authentication state via Bluetooth.	
4150	Bluetooth device failed to report authentication state.	

Error code	Description	Cause and action
4151	Failed to execute additional action during network configuration process.	Possible reason: The current network configuration process does not support such additional action.
4153	Network parameter is incorrect.	
4154	(Network Config by Bluetooth) Failed to send network information.	
4160	The ADS ID is incorrect.	
4161	Failed to resolve ADS domain.	
4162	Failed to make a TCP connection to ADS server.	
4163	Failed to request access layer domain.	
4164	(Network Config by Bluetooth) The ADS ID is incorrect.	
4165	(Network Config by Bluetooth) Failed to resolve ADS domain.	
4166	(Network Config by Bluetooth) Failed to make a TCP connection to ADS server.	
4167	(Network Config by Bluetooth) Failed to request access layer domain.	
4168	(Network Config by Bluetooth) The cloud server can not find appliance sn and random digit generated during network configuration.	
4169	(Network Config by AP) The cloud server could not find appliance sn and random number generated	

Error code	Description	Cause and action
	during network configuration.	
4177	The appliance can not support the docking mode to which network bridge has switched.	
4178	Network bridge has switched docking mode, but appliance does not verify.	
1321	The cloud server could find the appliance sn, but failed to verify the random number.	
1307	The cloud server could not find appliance sn and random number, furthermore also could not find appliance in the LAN of router.	Possible reason: The appliance does not connect to the router.
4170	(Network config via Network bridge) The parameter of family ID is missing.	
4171	(Network config via Network bridge) Failed to get deviceId in cloud server.	
4172	(Network config via Network bridge) Failed to send deviceId and familyId parameters to appliance.	
4173	The ADS ID is incorrect.	
4174	Appliance failed to request the ADS domain.	
4175	Appliance returned that the router signal is too weak.	

6.2. Thing model definition for Air Conditioner

JSON key	Value	Description	Usage ³
power	0 :power off 1 :power on		command/property
mode	1	1:automatic mode	command/property
	2	2:Refrigeration mode	
	3	3:Dehumidifying mode	
	4	4:HEAT	
	5	5:FAN	
	6	6:smart Dehumidifying mode	
windSpeed	102	102:Automatic WIND	command/property
	101	101:Fixed WIND	property
	80	80:HIGH WIND	command/property
	60	60:MIDDLE WIND	command/property
	40	40:LOW WIND	command/property
btnSound	0:OFF 1:ON	Indicate whether appliance need to make a sound when executing control command.	command
temp	Value range: 12-30	When it is used as a command key, you can only set its value in centigrade unit. For fahrenheit value , you need to convert to centigrade value in terms of following table:	command/property

³ Usage describes that a JSON key can be used as a **command** key to send command to appliance; or as a **property** key in response returned from HTTP API, or **event** notification from appliance.

JSON key	Value	Description		Usage ³
		Fahrenheit	Centigrade	
	16	60		
	16.5	61		
	17	62		
	17.5	63		
	18	64		
	18.5	65		
	19	66		
	19.5	67		
	20	68		
	20.5	69		
	21	70		
	21.5	71		
	22	72		
	23	73		
	23.5	74		
	24	75		
	24.5	76		
	25	77		
	25.5	78		
	26	79		
	26.5	80		
	27	81		
	28	82		

JSON key	Value	Description		Usage ³
		28.5 29 29.5 30 30.5 31 31.5	83 84 85 86 87 88 89	
faultCode	0:no error; 1:EH0b 2:EH00 3:EH01 4:EH02 5:EH03 6:EC52 7:EC53 8:EC54 9:EC51 10:EH60 11:EH61 12:EC07 13:PC00 14:PC01 15:PC02 16:PC0L	Error code of air conditioner.		property

JSON key	Value	Description	Usage ³
	33:EL0C		
roomTemp	Value range: From -15 to 50 in centigrade.	Indoor temperature.	property
sleep	0:OFF 1:ON		command/property
horzWind	0:OFF 3:ON	Wind from side to side.	command/property
vertWind	0:OFF 3:ON	Wind up and down	command/property
filterFlag	0:normal 1:need to clean	flag indicating filter state.	property
outTemp	Value range: From -15 to 50 in centigrade.	Out door temperature.	property
tempModeSwitch	0:centigrade 1:fahrenheit	The temperature unit.	command/property
Following definitions are only for extended functionalities of Air Conditioner. Before using them, It's required to confirm whether your product support these function or not.			
strong	0:OFF 1:ON	Strong wind	command/property
eco	0:OFF 1:ON	eco mode	command/property
8DegreeHot	0:OFF 1:ON	Heating in 8 centigrade.	command/property
unDirectBlow	1:OFF	Prevent from blowing human straightly.	command/property

JSON key	Value	Description	Usage ³
	2:ON		
buzzer	0:OFF 1:ON	open or close buzzer.	command/property
selfClean	0:OFF 1:ON	Self cleaning	command/property
windBreezed	0:OFF 1:ON	breezed wind feature.	command/property
windNone	0:OFF 1:ON	no wind feeling feature.	command/property
windSoft	0:OFF 1:ON	soft wind feature.	command/property
notWindStatus	0	Prevent from blowing human straightly in:	command/property
	1	0: left direction	command/property
	2	1: right direction	command/property
		2: upward direction	command/property
	3	3: downward direction	
		Note: it's required to set unDirectBlow=2 when using this key.	
ledDisplay	Control appliance: 0: turn on LED display. 7: turn off LED display. Event notification from appliance: 0: LED display is on.		command/property

JSON key	Value	Description	Usage ³
	<code>!= 0(2 or 7):LED display is off.</code>		
smartEye	0:OFF 1:ON	Open or close smart eye feature.	command/property
windNotBreezed	0:OFF 1:ON	open or close "not breezing feature"	command/property
lightClass	0:OFF 1:ON	turn on or turn off light.	command/property
roundWind	0,1,2	surrounding wind feature. 0: turn off this feature. 1: turn on this feature in upward direction. 2: turn on this feature in downward direction.	command/property
totalPowerConsume	value of total electricity consumption.	Value is in 0.01KW unit.	property
totalRunPower	value of accumulated running electricity consumption.	Value is in 0.01KW unit.	property
curRunPower	Value of current running electricity consumption.	Value is in 0.01KW unit.	property
curRealTimePower	Value of real time power.	Value is in 0.01KW unit.	property
masterLouverMsg	The louver deflector state of the primary air conditioner unit	<p>deflector1:masterLouverMsg_1 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>deflector2:masterLouverMsg_2 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p>	property

JSON key	Value	Description	Usage ³
		<p>deflector3:masterLouverMsg_3 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>deflector4:masterLouverMsg_4 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>Value construction: <code>masterLouverMsg = masterLouverMsg_1 << 24 +masterLouverMsg_2 << 16 + masterLouverMsg_3 << 8 +masterLouverMsg_4</code></p>	
slaveLouverMsg	The louver deflector state of the secondary air conditioner unit.	<p>deflector1:slaveLouverMsg_1 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>deflector2:slaveLouverMsg_2 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>deflector3:slaveLouverMsg_3 0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>deflector4:slaveLouverMsg_4</p>	property

JSON key	Value	Description	Usage ³
		<p>0: swing is off; 1-100: swinging with angle; 102: swinging is on.</p> <p>Value construction:</p> <pre>slaveLouverMsg = slaveLouverMsg_1 << 24 +slaveLouverMsg_2 << 16 + slaveLouverMsg_3 << 8 +slaveLouverMsg_4</pre>	