MI WLAN API

Version 2.03

REVISION HISTORY

Revision No.	Description	Date
2.03	• Initial release	08/14/2019

TABLE OF CONTENTS

1. API 参考

1.1. 概述

本模块（无线局域网），提供了简单的 wifi 信号的扫描，连接功能。提供了对 AP 以及 STA 模式的支持。

1.2. 功能模块 API

API 名	功能
MI_WLAN_Init	WLAN 设备初始化
MI_WLAN_DeInit	WLAN 设备注销
MI_WLAN_Open	使能 WLAN 设备，设置工作参数
MI_WLAN_Close	关闭 WLAN 设备。
MI_WLAN_Connect	连接 WIFI 热点或者启动 AP 工作模式
MI_WLAN_Disconnect	断开 WIFI 热点连接诶或者关闭 AP 模式
MI_WLAN_Scan	扫描可用热点
MI_WLAN_GetStatus	获取连接状态
MI_WLAN_GetWlanChipVersion	获取 wlan 设备参数

1.2.1 MI_WLAN_Init

- 功能

  WLAN 设备初始化

- 语法

  MI_RESULT MI_WLAN_Init(MI_WLAN_InitParams_t *pstInitParams);

- 形参

参数名称	描述	输入/输出
pstInitParams	Wlan 设备初始化参数指针	输入

- 返回值

  返回值
  - 0 成功。
  - 非 0 失败，参照**错误码**。

- 依赖
  - 头文件: mi_wlan.h mi_wlan_datatype.h
  - 库: libmi_wlan libcjson libcrypto libssl libnl
  - 可执行档: iwlist wpa_cli wpa_supplicant

※ 注意

Wlan 设备的初始化，依赖一个 json 配置文件，公版名为 wlan.json，位于/config/wifi/wlan.json,
该参数目前不支持传入 NULL。

- 因为 WLAN 模块的可配置项目多，项目类型复杂，而且有不确定性，所以引入一个 json 配置文件该文件的内容，以及配置项介绍，请参考 APPENDIX:A。

- 举例

  下面的代码实现设置 WLAN 设备初始化。

```
MI_WLAN_InitParams_t stParm = {"/config/wifi/wlan.json"};
MI_WLAN_Init(&stParm);
```

1.2.2 MI_WLAN_DeInit

- 功能

  注销 WLAN 设备。

- 语法

  MI_RESULT MI_WLAN_DeInit(void);

- 形参

  无

➢ 返回值

返回值 { 0 成功。 / 非 0 失败，参照**错误码**。

➢ 依赖

- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意

无

➢ 举例

```
MI_RESULT ret;
ret = MI_WLAN_DeInit();;
if(MI_SUCCESS != ret)
{
    printf(“ wlan deinit fail\n”);
    return ret;
}
```

1.2.3 MI_WLAN_Open

➢ 功能

使能 WLAN 设备，设置工作参数。

➢ 语法

```
MI_RESULT MI_WLAN_Open(MI_WLAN_OpenParams_t *pstParam);
```

➢ 形参

参数名称	描述	输入/输出
pstParam	Wlan设备的工作参数指针	输入

➢ 返回值

返回值 { 0 成功。 / 非 0 失败，参照**错误码**。

➢ 依赖

- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意

- 需要已经 WLAN 初始化

➢ 举例

请参见 MI_WLAN_OpenParams_t 的举例。

1.2.4 MI_WLAN_Close

➢ 功能

关闭 WLAN 设备。

➢ 语法

MI_RESULT MI_WLAN_Close(void);

➢ 形参

无

➢ 返回值

返回值		
	0	成功。
	非 0	失败，参照错误码。

➢ 依赖

- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意

无

➢ 举例

```
MI_RESULT ret;
ret = MI_WLAN_Close ();;
if(MI_SUCCESS != ret)
{
        printf(“ wlan close fail\n”);
        return ret;
}
```

1.2.5 MI_WLAN_Connect

➢ 功能

建立 wifi 连接服务

➢ 语法

MI_RESULT MI_WLAN_Connect(WLAN_HANDLE *hWLan, MI_WLAN_ConnectParam_t *pstConnectParam);

➢ 形参

参数名称	描述	输入/输出
hWLan	Wlan 句柄指针 (<0)表示要建立一个新连接，函数执行成功，会被赋予新连接的句柄 (>0)表示是一个已经存在的连接。	输入/输出
pstConnectParam	Wifi连接的参数设定指针 详见MI_WLAN_ConnectParam_t	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照错误码。

➢ 依赖

- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意

- 如果 wifi 热点没有设置密码，需要传入一个空字符串 ””
- 目前超时设定没有正确生效，连接的超时由 wlan.json 中，connect:infra:dhcp –T –t 参数给出详见 **APPENDIX:A**

➢ 举例

```
WLAN_HANDLE wlanHdl = -1;
MI_WLAN_ConnectParam_t      stConnectParam =    \
            {E_MI_WLAN_SECURITY_WPA2, "123456", "", 5000};
MI_WLAN_Connect(&wlanHdl, &stConnectParam);
```

1.2.6 MI_WLAN_Disconnect

➢ 功能

断开 wifi 连接服务

➢ 语法

```
MI_RESULT MI_WLAN_Disconnect(WLAN_HANDLE hWLan);
```

➢ 形参

参数名称	描述	输入/输出
hWLan	Wlan 句柄	输入

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照**错误码**。

➢ 依赖
- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意
确保 hWLan>0

➢ 举例
```
WLAN_HANDLE wlanHdl = -1;
MI_WLAN_ConnectParam_t stConnectParam =  \
        {E_MI_WLAN_SECURITY_WPA2, "123456", "", 5000};
MI_WLAN_Connect(&wlanHdl, &stConnectParam);
MI_WLAN_Disconnect(wlanHdl);
```

1.2.7 MI_WLAN_Scan

➢ 功能
扫描当前可用 wifi 连接

➢ 语法
MI_RESULT MI_WLAN_Scan(MI_WLAN_ScanParam_t *pstParam, MI_WLAN_ScanResult_t *pstResult);

➢ 形参

参数名称	描述	输入/输出
pstParam	扫描参数指针 目前没有实际使用	输入
pstResult	扫描到的当前可用wifi热点的信息指针	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照**错误码**。

➢ 依赖
- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意
- 输入参数 pstResult 中需要设定 u8APNumber，表示你需要获得的热点信息个数，最大为：MI_WLAN_MAX_APINFO_NUM。扫描完成后会被设定为实际热点个数

➢ 举例

```
MI_WLAN_ScanResult_t stRst;
 while(1)
 {
       stRst.u8APNumber = 64;
       MI_WLAN_Scan(NULL, &stRst);
       for(i = 0 ; i < stRst.u8APNumber; i ++)
       {
            printf("%s %d %s\n", __FUNCTION__, __LINE__, stRst.stAPInfo[i].au8SSId);
       }
        memset(&stRst,0,sizeof(stRst));
        stRst.u8APNumber = 64;
       sleep(3);
 }
```

1.2.8 MI_WLAN_GetStatus

➢ 功能

获取当前所有 wifi 连接信息

➢ 语法

MI_RESULT MI_WLAN_GetStatus(MI_WLAN_Status_t *pstStatus);

➢ 形参

参数名称	描述	输入/输出
pstStatus	当前所有wifi连接信息	输出

➢ 返回值

返回值
- 0 成功。
- 非 0 失败，参照**错误码**。

➢ 依赖

- 头文件: mi_wlan.h mi_wlan_datatype.h
- 库: libmi_wlan libcjson libcrypto libssl libnl
- 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意

- MI_WLAN_Status_t 为一个 union，根据当前 open 时设置的网络类型，返回已经连接的热点信息或者连接到本设备的主机的信息。

➢ 举例

无

1.2.9 MI_WLAN_GetWlanChipVersion

➢ 功能

获取 wlan 设备 FW 的版本号

- 语法

  MI_RESULT MI_WLAN_GetWlanChipVersion(MI_U8 *pstChipVersion);

- 形参

参数名称	描述	输入/输出
pstChipVersion	wlan设备FW的版本号	输出

- 返回值

  返回值
  - 0 成功。
  - 非 0 失败，参照**错误码**。

- 依赖
  - 头文件: mi_wlan.h mi_wlan_datatype.h
  - 库: libmi_wlan libcjson libcrypto libssl libnl
  - 可执行档:iwlist wpa_cli wpa_supplicant

※ 注意

无

- 举例

无

2. 数据类型

2.1. 模块相关数据类型

WLAN_HANDLE	定义 WLAN 句柄
MI_WLAN_Security_e	定义 WLAN 连接的安全协议类型
MI_WLAN_Encrypt_e	定义 WLAN 连接的加密协议类型
MI_WLAN_NetworkType_e	定义 WLAN 设备的工作模式类型
MI_WLAN_Authentication_Suite_e	定义 WLAN 连接认证的协议类型
MI_WLAN_WPAStatus_e	定义 WLAN 连接的状态类型
MI_WLAN_InitParams_t	定义 WLAN 设备的初始化参数
MI_WLAN_OpenParams_t	定义 WLAN 设备使能的参数
MI_WLAN_ConnectParam_t	定义 WLAN 连接的参数
MI_WLAN_Quality_t	定义 wifi 热点的信号质量
MI_WLAN_Cipher_t	定义 wifi 热点工作的加密信息
MI_WLAN_APInfo_t	定义 wifi 热点的信息
MI_WLAN_ScanParam_t	定义 wifi 热点扫描的参数信息
MI_WLAN_ScanResult_t	定义 wifi 热点扫描的结果信息
MI_WLAN_Status_sta_t	定义正在连接的 wifi 热点的信息
MI_WLAN_Status_host_t	定义某个接入的主机的信息
MI_WLAN_Status_ap_t	定义所有接入的主机的信息
MI_WLAN_Status_t	Union of MI_WLAN_Status_host_t and MI_WLAN_Status_sta_t

2.1.1 WLAN_HANDLE

➢ 说明

Wlan 设备句柄，标记并区分 wifi 连接。

➢ 定义

typedef MI_S32 WLAN_HANDLE

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.2 MI_WLAN_Security_e

➢ 说明

Wifi 通讯的安全协议标准类型。

➢ 定义

```
typedef enum
{
    /// WLAN module security key off
    E_MI_WLAN_SECURITY_NONE          = 1 << 0,
    /// WLAN module security key unknow
    E_MI_WLAN_SECURITY_UNKNOWTYPE = 1 << 1,
    /// WLAN module security key WEP
    E_MI_WLAN_SECURITY_WEP           = 1 << 2,
    /// WLAN module security key WPA
    E_MI_WLAN_SECURITY_WPA           = 1 << 3,
    /// WLAN module security key WPA2
    E_MI_WLAN_SECURITY_WPA2          = 1 << 4,
    /// WLAN module max
    E_MI_WLAN_SECURITY_MAX           = 0xff,
} MI_WLAN_Security_e;
```

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.3 MI_WLAN_Encrypt_e

➢ 说明

Wifi 密钥加密类型。

➢ 定义

```
typedef enum
{
    /// WLAN module encrypt type none
    E_MI_WLAN_ENCRYPT_NONE          = 1 << 0,
    /// WLAN module encrypt type unknown
    E_MI_WLAN_ENCRYPT_UNKNOWN       = 1 << 1,
    /// WLAN module encrypt type WEP
    E_MI_WLAN_ENCRYPT_WEP           = 1 << 2,
    /// WLAN module encrypt type TKIP
    E_MI_WLAN_ENCRYPT_TKIP          = 1 << 3,
    /// WLAN module encrypt type AES
    E_MI_WLAN_ENCRYPT_AES           = 1 << 4,
    /// WLAN module max
    E_MI_WLAN_ENCRYPT_MAX           = 0xff,
} MI_WLAN_Encrypt_e;
```

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.4 MI_WLAN_NetworkType_e

➢ 说明

Wlan 设备的工作模式。

➢ 定义

```
typedef enum
{
    /// WLAN network infrastructure type
    E_MI_WLAN_NETWORKTYPE_INFRA,
    /// WLAN network AP type
    E_MI_WLAN_NETWORKTYPE_AP,
    /// WLAN network AdHoc type
    E_MI_WLAN_NETWORKTYPE_ADHOC,
    /// WLAN network Monitor type
    E_MI_WLAN_NETWORKTYPE_MONITOR,
    /// WLAN network mode master
    E_MI_WLAN_NETWORKTYPE_MASTER,
    /// WLAN network mode slave
    E_MI_WLAN_NETWORKTYPE_SLAVE,
    /// WLAN param max
    E_MI_WLAN_NETWORKTYPE_MAX
} MI_WLAN_NetworkType_e;
```

※ 注意事项

目前支持 E_MI_WLAN_NETWORKTYPE_INFRA 和 E_MI_WLAN_NETWORKTYPE_AP

➢ 相关数据类型及接口

无。

2.1.5 MI_WLAN_Authentication_Suite_e

➢ 说明

WIFI 认证方式。

➢ 定义

```
typedef enum
{
    /// Authentication Suite PSK
    E_MI_WLAN_AUTH_SUITE_PSK,
    /// Authentication Suite unknown
    E_MI_WLAN_AUTH_SUITE_UNKNOWN,
    E_MI_WLAN_AUTH_SUITE_MAX

} MI_WLAN_Authentication_Suite_e;
```

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.6 MI_WLAN_WPAStatus_e

➢ 说明

Wifi 连接过程中的状态。

➢ 定义

```
typedef enum
{
    WPA_DISCONNECTED,
    WPA_INTERFACE_DISABLED,
    WPA_INACTIVE,
    WPA_SCANNING,
    WPA_AUTHENTICATING,
    WPA_ASSOCIATING,
    WPA_ASSOCIATED,
    WPA_4WAY_HANDSHAKE,
    WPA_GROUP_HANDSHAKE,
    WPA_COMPLETED
} MI_WLAN_WPAStatus_e;
```

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.7 MI_WLAN_InitParams_t

➢ 说明

Wlan 设备初始化参数。

➢ 定义

```
typedef struct MI_WLAN_InitParams_s
{
    /// json description file of wifi dongle
    MI_U8 au8JsonConfFilePath[MI_WLAN_MAX_FOLDERPATH_LEN];
    /// reserved
    MI_U64 u64Reserved;
} MI_WLAN_InitParams_t;
```

➢ 成员

成员名称	描述
au8JsonConfFilePath	Wlan 设备初始化依赖的 json 配置文件绝对路径。
u64Reserved	预留设置，未启用。

※ 注意事项

Wlan 设备的初始化，依赖一个 json 配置文件，公版名为 wlan.json，位于/config/wifi/wlan.json，该参数目前不支持传入 NULL。

➢ 相关数据类型及接口

无。

2.1.8 MI_WLAN_OpenParams_t

➢ 说明

Wlan 设备的工作参数。

➢ 定义

```
typedef struct MI_WLAN_OpenParam_s
{
    // WLAN network type
    MI_WLAN_NetworkType_e eNetworkType;
    // reserved
    MI_BOOL bReserved;
} MI_WLAN_OpenParams_t;
```

➢ 成员

成员名称	描述
MI_WLAN_NetworkType_e	Wlan 设备的工作服务类型。
bReserved	预留参数，未启用。

※ 注意事项

无

➢ 相关数据类型及接口

MI_WLAN_NetworkType_e

2.1.9 MI_WLAN_ConnectParam_t

➢ 说明

Wlan 设备连接的具体参数设定。

➢ 定义

```
typedef struct
{
    // WLan security mode
    MI_WLAN_Security_e eSecurity;
    // WLan SSID
    MI_U8 au8SSId[MI_WLAN_MAX_SSID_LEN];
    // WLan password
    MI_U8 au8Password[MI_WLAN_MAX_PASSWD_LEN];
    // WLAN connect overtime
    MI_U32 OverTimeMs;
} MI_WLAN_ConnectParam_t;
```

➢ 成员

成员名称	描述
MI_WLAN_Security_e	infra 模式下为 wifi 热点的 security type。 ap 模式下为接入 wifi 连接时采用的 security type
au8SSId	wifi 热点的名字。
au8Password	wifi 热点的密码
OverTimeMs	建立 wifi 连接的超时时间

※ 注意事项

意见使用 WLAN module security key WPA/ WLAN module security key WPA2

➢ 相关数据类型及接口

MI_WLAN_NetworkType_e MI_WLAN_Security_e

2.1.10 MI_WLAN_Quality_t

➢ 说明

Wifi 热点的信号质量。

➢ 定义

```
typedef struct
{
    MI_U8 curLVL;
    MI_U8 maxLVL;
    MI_S8 signalSTR;
} MI_WLAN_Quality_t;
```

➢ 成员

成员名称	描述
curLVL	当前 wifi 热点的信号水平
maxLVL	Wifi 热点的总信号水平
signalSTR	当前 wifi 热点的信号强度(mdb)

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.11 MI_WLAN_Cipher_t

➢ 说明

Wifi 连接的加密设定

➢ 定义

```
typedef struct
{
    // WLAN Security mode
    MI_WLAN_Security_e eSecurity;
    // WLAN Encryption type
    MI_WLAN_Encrypt_e eGroupCipher;
    // WLAN Encryption type
    MI_WLAN_Encrypt_e ePairCipher;
    // WLAN authenticationb suite
    MI_WLAN_Authentication_Suite_e eAuthSuite;
} MI_WLAN_Cipher_t;
```

➢ 成员

成员名称	描述
eSecurity	Wifi 热点的安全协议类型
eGroupCipher	Wifi 热点的组加密类型
ePairCipher	Wifi 热点的配对加密类型
eAuthSuite	Wifi 连接的认证协议组

※ 注意事项

无

➢ 相关数据类型及接口

MI_WLAN_Security_e MI_WLAN_Encrypt_e MI_WLAN_Authentication_Suite_e

2.1.12 MI_WLAN_APInfo_t

➢ 说明

Wifi 热点信息。

➢ 定义

```
typedef struct MI_WLAN_APInfo_s
{
    // WLAN CELL ID
    MI_U16 u16CellId;
    // WLAN Frequency GHz
    MI_FLOAT fFrequency;
    // WLAN Bitrate Mb/s
    MI_FLOAT fBitRate;
    // WLAN Quality
    MI_WLAN_Quality_t stQuality;
    // WLAN Encryption key on/off
    MI_BOOL bEncryptKey;
    // WLAN SSID
    MI_U8 au8SSId[MI_WLAN_MAX_SSID_LEN];
    // WLAN Channel
    MI_U8 u8Channel;
    // WLAN MAC
    MI_U8 au8Mac[MI_WLAN_MAX_MAC_LEN];
    // WLAN Encryption type
    MI_WLAN_Encrypt_e eEncrypt;
    // WLAN AP type (Infrastructure / Ad-Hoc)
    MI_WLAN_NetworkType_e eMode;
    // WLAN cipher kit
    MI_WLAN_Cipher_t stCipher[2];
} MI_WLAN_APInfo_t;
```

➢ 成员

成员名称	描述
u16CellId	Wifi 热点编号
fFrequency	Wifi 热点的工作频率
fBitRate	Wifi 热点的工作比特率。
stQuality	Wifi 热点信息
bEncryptKey	Wifi 热点是否需要密码
au8SSId	Wifi 热点名字
u8Channel	Wifi 热点工作信道
au8Mac	Wifi 热点的 mac 地址
eEncrypt	Wifi 热点加密协议
eMode	Wifi 热点的工作模式
stCipher	Wifi 热点的加密协议组

※ 注意事项

无。

➢ 相关数据类型及接口

MI_WLAN_Quality_t MI_WLAN_Encrypt_e MI_WLAN_NetworkType_e MI_WLAN_Cipher_t

2.1.13 MI_WLAN_ScanParam_t

➢ 说明

wifi 热点扫描设定。

➢ 定义

```
typedef struct MI_WLAN_ScanParam_s
{
    // WLan set block mode
    MI_BOOL bBlock; //reserved
} MI_WLAN_ScanParam_t;
```

➢ 成员

成员名称	描述
bBlock	是否阻塞 未启用，是否阻塞在 json 配置文件当中配置

※ 注意事项

无。

➢ 相关数据类型及接口

2.1.14 MI_WLAN_ScanResult_t

➢ 说明

Wifi 热点扫描结果。

➢ 定义

```
typedef struct MI_WLAN_ScanResult_s
{
    // WLan AP number
    MI_WLAN_APInfo_t stAPInfo[MI_WLAN_MAX_APINFO_NUM];
    MI_U8 u8APNumber;
} MI_WLAN_ScanResult_t;
```

➢ 成员

成员名称	描述
stAPInfo	保留，未使用。
u8APNumber	

※ 注意事项

无。

➢ 相关数据类型及接口

MI_WLAN_APInfo_t

2.1.15 MI_WLAN_Status_sta_t

➢ 说明

Wifi 连接状态。

➢ 定义

```
typedef struct MI_WLAN_Status_sta_s
{
    MI_U8 bssid[MI_WLAN_BSSID_LEN];
    MI_U32 freq;
    MI_U8 ssid[MI_WLAN_MAX_SSID_LEN];
    MI_U16 id;
    MI_WLAN_NetworkType_e mode;
    MI_WLAN_Cipher_t stCipher;
    MI_U8 key_mgmt[12];
    MI_WLAN_WPAStatus_e state;
    MI_U8 address[MI_WLAN_BSSID_LEN];
    MI_U8 ip_address[16];
    MI_U32 channel;
    MI_U32 RSSI;
    MI_U8 Bandwidth[8];
} MI_WLAN_Status_sta_t;
```

➢ 成员

成员名称	描述
bssid	Wifi 热点名字 hexdecimal 值
freq	正在连接的 wifi 热点频率。
ssid	Wifi 热点名字。
id	Wifi 热点 id
mode	Wifi 热点工作模式。
stCipher	Wifi 热点加密协议组。
key_mgmt	Wifi 热点密钥管理类型。
state	Wifi 热点当前的连接状态
address	Wifi 热点给主机分配的 ip 地址
channel	当前 wifi 连接工作信道
RSSI	当前 wifi 连接的 RSSI 水平
Bandwidth	当前 wifi 连接工作带宽

※ 注意事项

无

➢ 相关数据类型及接口

MI_WLAN_NetworkType_e MI_WLAN_Cipher_t MI_WLAN_WPAStatus_e

2.1.16 MI_WLAN_Status_host_t

➢ 说明

Ap 模式下接入 wifi 连接的状态信息。

➢ 定义

```
typedef struct MI_WLAN_Status_host_s
{
    MI_U8 hostname[MI_WLAN_MAX_HOST_NAME_LEN];
    MI_U8 ipaddr[16];
    MI_U8 macaddr[18];
    MI_U64 connectedtime;

} MI_WLAN_Status_host_t;
```

➢ 成员

成员名称	描述
hostname	接入主机的名字。
ipaddr	接入主机的 ip 地址。
macaddr	接入主机的 mac 地址
connectedtime	接入主机的已连接时间

※ 注意事项

无。

➢ 相关数据类型及接口

无。

2.1.17 MI_WLAN_Status_ap_t

➢ 说明

所有接入 wifi 接连的状态信息。

➢ 定义

```
typedef union MI_WLAN_Status_s
{
    MI_WLAN_Status_sta_t stStaStatus;
    MI_WLAN_Status_ap_t stApStatus;
} MI_WLAN_Status_t;
```

➢ 成员

成员名称	描述
stStaStatus	当前连接的 wifi 热点信息
stApStatus	当前接入的所有主机的信息

※ 注意事项

无

➢ 相关数据类型及接口

MI_WLAN_Status_sta_t MI_WLAN_Status_ap_t

3. 错误码

AI API 错误码如表 3-1 所示：

表 3-1 AI API 错误码

宏定义	描述
MI_WLAN_ERR_FAIL	未识别错误
MI_WLAN_ERR_INVALID_DEVID	音频输入信道号无效
MI_WLAN_ERR_ILLEGAL_PARAM	输入参数不合法
MI_WLAN_ERR_NOT_SUPPORT	输入参数为不支持类型
MI_WLAN_ERR_MOD_INITED	设备已经初始化
MI_WLAN_ERR_MOD_NOT_INIT	设备未初始化
MI_WLAN_ERR_NOT_CONFIG	设备未使能
MI_WLAN_ERR_INVAL_HANDLE	句柄非法
MI_WLAN_ERR_NULL_PTR	空输入参数
MI_WLAN_ERR_INITED	模块已经初始化

4. APPENDIX A

4.1. wlan.json

```
{
    /*当前选中的 wifi 设备，本配置文件支持多个 wifi 设配的配置
    **本例选择 ssw01b,那么 MI_WLAN 就会去解析 ssw01b 标签下的信息
    */
    "selfie": {
        "wifi": "ssw01b"
    },
    /*wifi 设备标签*/
    "ssw01b": {
        /*wifi 设备的自我描述*/
        "selfie": {
            /*wifi 设备的固件版本*/
            "version": "0.0.0",
            /*wifi 设备的名字*/
            "name": "ssw01b40m",
            /*wifi 设备初始化脚本的路径，会在 MI_WLAN_Init 中被调用*/
            "scriptPath": "/config/wifi",
            /*本 wifi 设备所有 script 标签下的脚本解析缺省解析 bin
            **脚本不一定需要是 shell，也可以是 python perl 等
            */
            "parser": "sh"
        },
        /*wifi 的初始化标签
        **在 MI_WLAN_Init 函数中被解析执行
        */
        "init": {
            /*本 wifi 设备运行需要配置的 linux 环境变量
            **环境变量的 name 与 value 需要一一对应
            **个数不限
            */
            "env": {
                "name": ["LD_LIBRARY_PATH","PATH"],
                "value": ["/config/wifi","/config/wifi"]
            },
            /*本 wifi 设备初始化需要解析的脚本
            **@parser 解析这个脚本的 bin 档，可选标签
            **@name 脚本文件
            **@option 脚本参数，个数不限，可选标签
```

```
        */
        "script": {
            "parser": "source",
            "name": "ssw01bInit.sh",
            "option": ["dummy1","dummy2"]
        }
    },
    /*wifi 设备的注销标签*/
    "deinit": {
        /*本 wifi 设备注销需要解析的脚本
        **@parser 可选标签，未选
        **@name 脚本文件
        **@option 可选标签，未选
        */
        "script": {
            "name": "ssw01bDeInit.sh"
        }
    },
    /*wifi 设备的全局信息配置标签
    **本标签给出了一些 wifi 服务需要的基本设定
    **用户可以根据自身对 wifi 工作的定制需求，
    **配置该标签下的内容
    */
    "universal": {
        /*主机模式下配置标签*/
        "infra": {
            /*wifi 设备提供的 wifi 热点的接口名字
            **不同 wifi 设备提供的接口名字可能会有差异
            */
            "iface": "wlan0",
            /*系统提供给 wifi 设备的控制接口目录
            **需要在 init 之前创建好，默认是在 init 标签中指示的脚本中创建
            **需要用户同步路径
            */
            "ctrl_interface": "/config/wifi/run/wpa_supplicant"
        },
        /*热点模式下的配置标签*/
        "ap": {
            /*wifi 设备提供的 wifi 热点的接口名字
            **不同 wifi 设备提供的接口名字可能会有差异
            */
            "iface": "p2p0",
            /*热点自己的 static ip 地址*/
            "ipaddr": "192.168.1.100",
            /*热点自己的子网掩码*/
            "netmask": "255.255.255.0",
```

```
        /*系统提供给 wifi 设备的控制接口目录
        **需要在 init 之前创建好，默认是在 init 标签中指示的脚本中创建
        **需要用户同步路径
        */
        "ctrl_interface": "/var/run/hostapd"

    },
    /*udhcpc 需要的配置脚本路径，该脚本用于配置获取的网络连接信息,系统提供*/
    "dhcp_script": "/etc/init.d/udhcpc.script",
    /*wpa_supplicant 需要的配置文件路径，需要用户构建*/
    "wpa_supplicant_conf": "/config/wifi/wpa_supplicant.conf",
    /*hostapd 需要的配置文件路径，需要用户构建*/
    "hostapd_conf": "/config/wifi/hostapd.conf",
    /*dnsmasq 需要的配置文件路径，需要用户构建*/
    "dnsmasq_conf": "/config/wifi/dnsmasq.conf",
    /*dhcp 需要的租约记录文件，需要用户构建目录*/
    "dhcp-leasefile":"/var/lib/misc/dnsmasq.leases"
},
/*
**individial 标签定义了 MI_WLAN 支持的所有 action
**action 子标签 目前总共定义了
**@scan 扫描操作
**@open 打开 WLAN 设备操作
**@close 关闭 WLAN 设备操作
**@connect wifi 连接服务
**@disconnect wifi 连接服务断开
**@status 获取当前 wifi 连接服务状态
**每一个 action 在不同的工作模式下会有不同的行为
**action 定义模式为
**{
    "binary":["bin1",...."binx"],
    "option0":["opt1",...."optX"],
        ................,
        ................,
        ................,
    "option#N":["opt1",...."optX"]
**}
**binary 数组的 bin 和 option 数组是一一对应的，option 标签有明确的编号尾缀，从 0 开始递增
**option 内容，支持对 universal 标签的内容引用，语法为'$TAGNAME'支持逐级引用
**'$TAGNAME:$SUB_TAGNAME'
}
*/
"individual": {
    "action": {
        "scan": {
            "binary": ["iwlist"],
```

```
            "option0": ["$infra:$iface","scanning"]
        },
        "open": {
            "deviceup": {
                "ap": {
                    "binary": ["ifconfig"],
                    "option0": ["$ap:$iface","up"]
                },
                "infra": {
                    "binary": ["ifconfig"],
                    "option0": ["$infra:$iface","up"]
                }
            },
            "serviceup": {
                "ap": {
                    "binary": ["ifconfig","hostapd"],
                    "option0": ["$ap:$iface","$ap:$ipaddr","netmask","$ap:$netmask"],
                    "option1": ["-B","$hostapd_conf"]
                },
                "infra": {
                    "binary": ["wpa_supplicant"],
                    "option0": ["-i","$infra:$iface","-Dnl80211","-c","$wpa_supplicant_conf","-d","&"]
                }
            }
        },
        "close": {
            "devicedown": {
                "ap": {
                    "binary": ["ifconfig"],
                    "option0": ["$ap:$iface","down"]
                },
                "infra": {
                    "binary": ["ifconfig"],
                    "option0": ["$infra:$iface","down"]
                }
            },
        "servicedown": {
            "ap": {
                "script":{
                    "parser":"sh",
                    "name":"atbmClose.sh",
                    "option": ["ap"]
                }
            },
            "infra": {
                "script":{
```

```
                        "name":"atbmClose.sh",
                        "option": ["infra"]
                    }
                }
            }
        },
        "connect": {
            "ap": {
                "binary": ["ifconfig","dnsmasq"],
                "option0": ["$ap:$iface","up"],
                "option1": ["-i","$ap:$iface","--no-daemon","-C","$dnsmasq_conf","&"]
            },
            "infra": {
                "add": {
                    "binary": ["wpa_cli"],
                    "option0": ["-i","$infra:$iface","-p","$infra:$ctrl_interface","add_network"]
                },
                "setup": {
                    "ssid":{
                        "binary":["wpa_cli"],
                        "option0":
["-i","$infra:$iface","-p","$infra:$ctrl_interface","set_network","%d","ssid","\\\"%s\\\""]
                    },
                    "password":
                    {
                        "keyon":{
                            "wpa":{
                                "binary":["wpa_cli"],
                                "option0":
["-i","$infra:$iface","-p","$infra:$ctrl_interface","set_network","%d","psk","\\\"%s\\\""]
                            },
                            "wep":{
                                "binary":["wpa_cli"],
                                "option0":
["-i","$infra:$iface","-p","$infra:$ctrl_interface","set_network","%d","wep_key0","\\\"%s\\\""]
                            }
                        },
                        "keyoff":{
                            "binary":["wpa_cli"],
                            "option0":
["-i","$infra:$iface","-p","$infra:$ctrl_interface","set_network","%d","key_mgmt","NONE"]
                        }
                    }

                },
                "enable": {
```

```
                    "binary": ["wpa_cli","wpa_cli"],
                    "option0": ["-i","$infra:$iface","-p","$infra:$ctrl_interface","select_network","%d"],
                    "option1": ["-i","$infra:$iface","-p","$infra:$ctrl_interface","enable_network","%d"]
                },
                "dhcpc": {
                    "binary": ["udhcpc"],
                    "option0": ["-i","$infra:$iface","-s","$dhcp_script","-a","-q"," -b","-t","20","-T"," 1"]
                },
                "save": {
                    "binary": ["wpa_cli"],
                    "option0": ["-i","$infra:$iface","-p","$infra:$ctrl_interface","save_config"]
                }
            }
        },
        "disconnect": {
            "ap": {
                "binary": ["ifconfig"],
                "option0": ["$ap:$iface","down"]
            },
            "infra": {
                "binary": ["wpa_cli"],
                "option0": ["-i ","$infra:$iface","-p ","$infra:$ctrl_interface","disable_network","%d"]
            }
        },
        "status": {
            "ap": {
                "all_sta":{
                    "binary": ["hostapd_cli"],
                    "option0": ["-i","$ap:$iface","-p","$ap:$ctrl_interface","all_sta"]
                }
            },
            "infra": {
                "binary": ["wpa_cli"],
                "option0": ["-i","$infra:$iface","-p","$infra:$ctrl_interface","status"]
            }
        }
    }
}
}
```