

APPLICATION NOTE

ATBM IOCTL 控制接口使用说明



ATBM603X

1x1 802.11b/g/n
Wi-Fi 芯片

Table of contents

一、 DEVICE IOCTL 接口.....	4
概述:	4
1 获取当前 STATION 的连接状态.....	5
2 获取当前 STATION 连接 AP 的信号强度 RSSI	5
3 获取 AP 模式下, 所有已经连接的 STATION 信息.....	6
4 获取 STA 模式下, 已经连接 AP 的信息	6
5 获取 STA 模式下, 启动私有扫描	7
6 设置特殊频点	8
7 设置特殊字段的数据内容	9
8 STATION 模式下主动断掉与 AP 之间的连接	10
9 MONITOR 模式的 ENABLE/DISABLE	10
10 WI-FI 自适应 (CE 认证)	11
11 MAP 导入与 MAP 内容读写.....	11
12 功率设置.....	12
13 获取当前工作的信道.....	13
14 最优信道.....	13
15 获取 AP 列表.....	14
16 获取吞吐量	15
17 速率上限设置	16

18	固定速率设置	16
19	速率控制取消	17
20	获取 EFUSE	17
21	更改 EFUSE MAC 地址	17
22	设置 EFUSE DCXO	18
23	设置 EFUSE DELTA GAIN	18
24	设置 SET MIN RATE.....	19
25	设置 SET RATE POWER.....	19
26	设置过滤功能	20
27	设置国家码	21
28	获取驱动版本	21
29	定频 START_TX	22
30	定频 STOP_TX.....	23
31	定频 START_RX.....	23
32	定频 STOP_RX	23
33	定频 START_RX_RESULT	24
34	设置单信道扫描	24
35	启动频偏自校准功能.....	25
36	获取频偏自校准结果.....	25
37	设置功率补偿值	26

Version	Date	Author	Contents
001	20190918	Altobeam	Draft version.
	20200610	Yuzhihuang	修改了获取最优信道的参数
	20200703	Yuzhihuang	增加过滤特定帧的功能
	20210423	Yuzhihuang	增加 ETF 相关功能，增加获取版本号、efsue 功能， 国家码信道扫描限制功能
	20220126	Yuzhihuang	修正设置功率命令
	20220428	Yuzhihuang	增加 34~37 cmd 说明
	20220805	Yuzhihuang	Efuse dcxo deltagain 设置是否写到 rom 里面去，增 加功率补偿说明
	20221101	Yuzhihuang	增加获取私有协议的数据，以及设置监听信道

数据类型说明:

struct altm_wext_msg 结构体用于应用层与 wifi 驱动的数据交互。

```
struct altm_wext_msg{
    int type;
    int value;
    char externData[256];
};
```

type----具体定义如下:

```
enum atbm_msg_type{
    ATBM_DEV_IO_GET_STA_STATUS    = 0,
    ATBM_DEV_IO_GET_STA_RSSI      = 1, //STA connected AP's RSSI
    ATBM_DEV_IO_GET_AP_INFO       = 2,  //STA or AP
    ATBM_DEV_IO_GET_STA_INFO      = 3,
    ATBM_DEV_IO_SET_STA_SCAN      = 4,
    ATBM_DEV_IO_SET_FREQ          = 5,
    ATBM_DEV_IO_SET_SPECIAL_OUI   = 6, //use for Beacon and Probe package
    ATBM_DEV_IO_SET_STA_DIS       = 7,
    ATBM_DEV_IO_SET_IFTYPE        = 8,
    ATBM_DEV_IO_SET_ADAPTIVE      = 9,
    ATBM_DEV_IO_SET_TXPWR_DCXO    = 10,
    ATBM_DEV_IO_SET_TXPWR         = 11,
    ATBM_DEV_IO_GET_WORK_CHANNEL  = 12,
    ATBM_DEV_IO_SET_BEST_CHANNEL_SCAN = 13,
    ATBM_DEV_IO_GET_AP_LIST      = 14,
    ATBM_DEV_IO_GET_TP_RATE      = 15,
    ATBM_DEV_IO_FIX_TX_RATE      = 22,
    ATBM_DEV_IO_MAX_TX_RATE      = 23,
    ATBM_DEV_IO_TX_RATE_FREE     = 24,
    ATBM_DEV_IO_SET_EFUSE_MAC     = 25,
    ATBM_DEV_IO_SET_EFUSE_DCXO    = 26,
    ATBM_DEV_IO_SET_EFUSE_DELTAGAIN = 27,
    ATBM_DEV_IO_MIN_TX_RATE      = 28,
    ATBM_DEV_IO_SET_RATE_POWER    = 29,
    ATBM_DEV_IO_SET_SPECIAL_FILTER = 30,
    ATBM_DEV_IO_SET_COUNTRY_CODE = 31,

    ATBM_DEV_IO_GET_DRIVER_VERSION = 32,
    ATBM_DEV_IO_GET_EFUSE          = 33,
    ATBM_DEV_IO_GET ETF_START_RX_RESULTS = 34,
    ATBM_DEV_IO_SET_FIX_SCAN_CHANNEL = 36,
    ATBM_DEV_IO_SET_AUTO_CALI_PPM  = 37,
    ATBM_DEV_IO_GET_CALI_REAULTS   = 38,
```

```

ATBM_DEV_IO_SET_EFUSE_GAIN_COMPENSATION_VALUE = 39,
ATBM_DEV_IO_GET_VENDOR_SPECIAL_IE = 40,
#ifdef CONFIG_ATBM_STA_LISTEN
    ATBM_DEV_IO_SET_STA_LISTEN_CHANNEL = 41,
#endif
};

```

value----预留参数；

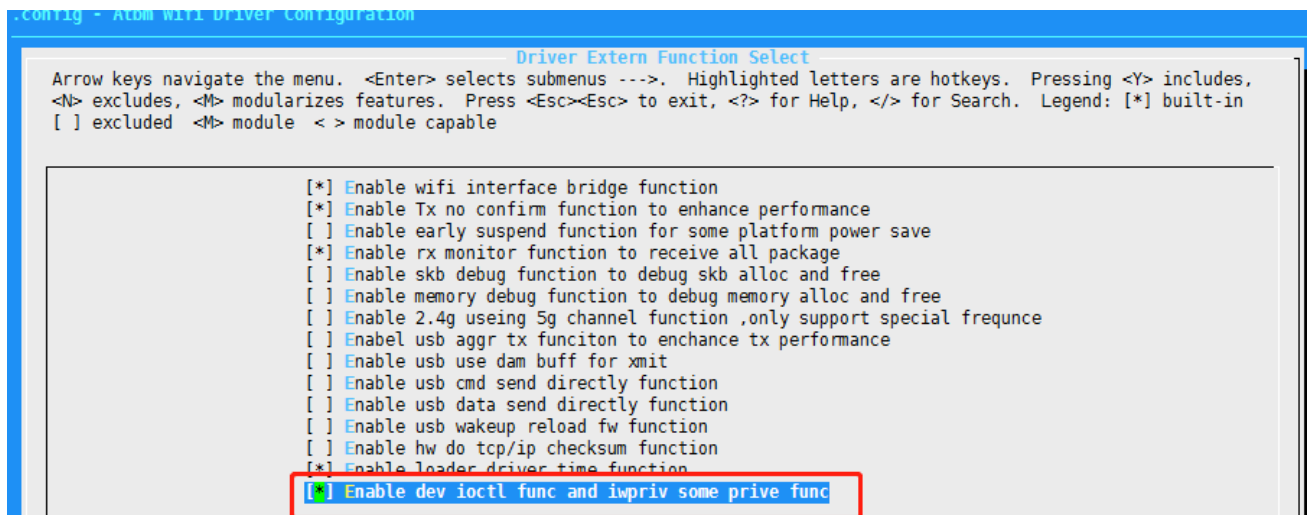
externData----传递数据。

一、Device IOCTL 接口

概述：

通过 SIOCDEVPRIVATE，用于应用层与 Wi-Fi 驱动的交互。

改功能需要打开驱动的 PRIVATE_IE 宏 15323 以上的版本驱动需要选上：



驱动版本低于 1532，需要在打开 hal_apollo/Makefile 里面打开宏：

```
ccflags-y += -DCONFIG_ATBM_AF0LEO_SUPPORT_S01
ifeq ($(USB_BUS),y)
ccflags-y += -DUSB_BUS
ccflags-y += -DUSB_USE_TASTLET_TXRX
ccflags-y += -DUSB_BUS_BUG
ccflags-y += -DCONFIG_USB_URB_RX_SUBMIT_DIRECTLY
#ccflags-y += -DCONFIG_WSM_CMD_XMIT_DIRECTLY
#ccflags-y += -DCONFIG_USB_DATA_XMIT_DIRECTLY
#ccflags-y += -DCONFIG_USE_DMA_ADDR_BUFFER
#ccflags-y += -DCONFIG_USB_AGGR_URB_TX
#ccflags-y += -DCONFIG_TX_NO_CONFIRM
#ccflags-y += -DATBM_IPC_ASSOC
#ccflags-y += -DATBM_PRIVATE_IE
#ccflags-y += -DSIGMASTAR_FILTER_MACADDR_ONLY
#ccflags-y += -DCONFIG_ATBM_5G_PRETEND_2G
endif
```

1 获取当前 station 的连接状态

Type ID	ATBM_DEV_IO_GET_STA_STATUS
输入参数	无
返回值	1 类型: bool 2 数据: externData[0:3] 3 说明: 0 (disconnect) 1 (associated)
例子	{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_STA_STATUS; ioctl(..., &msg, ...); }

2 获取当前 station 连接 AP 的信号强度 RSSI

Type ID	ATBM_DEV_IO_GET_STA_RSSI
输入参数	无
返回值	1 类型: int 2 数据: externData[0:3] 3 说明: 收到最后一包数据的 RSSI 值
例子	{ struct altm_wext_msg msg; int rssi = 0; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_STA_RSSI; ioctl(..., &msg, ...); }

	<pre>memcpy(&rssj, &msg.externData[0], sizeof(int)); }</pre>
--	--

3 获取 AP 模式下，所有已经连接的 Station 信息

Type ID	ATBM_DEV_IO_GET_AP_INFO
输入参数	<p>1 类型: char *</p> <p>2 数据: externData[0:3]</p> <p>3 说明: 四个字节的内存地址，传给 wifi 驱动，用于保存所有 STA 信息。</p>
返回值	<p>1 类型:</p> <pre>typedef struct _atbm_wifi_ap_info_{ int wext_rssi; //信号强度 unsigned long rx_packets; //收包数量 unsigned long tx_packets; //发包数量 unsigned long tx_retry_count; //重传次数 int last_rx_rate_idx; //收包速率 unsigned char wext_mac[6]; //Station 的 MAC 地址 unsigned char sta_cnt; //已经连接的 AP 个数 }atbm_wifi_ap_info;</pre> <p>2 数据: 数据 copy 到输入参数的内存地址中</p> <p>3 说明:</p> <p>因为，AP 模式下，最多支持 14 个 STA 的连接。</p> <p>所以，STA 信息的 total size 大于 256 bytes，不用 externData 传送数据。</p>
例子	<pre>{ struct altm_wext_msg msg; atbm_wifi_ap_info atbm_ap_info[14]; int addr_val = (int>(&atbm_ap_info[0])); memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_AP_INFO; memcpy(&msg.externData[0], &addr_val, sizeof(int)); ioctl(..., &msg, ...); }</pre>

4 获取 STA 模式下，已经连接 AP 的信息

Type ID	ATBM_DEV_IO_GET_STA_INFO
输入参数	无
返回值	1 类型:

	<pre> typedef struct _atbm_wifi_sta_info_{ int rssi; //信号强度 unsigned long rx_packets; //收包数量 unsigned long tx_packets; //发包数量 unsigned long tx_retry_count; //重传次数 unsigned char bssid[6]; //AP 的 MAC 地址 unsigned char ssid[32]; //AP 的 SSID unsigned long ssid_len; //AP 的 SSID 长度 }atbm_wifi_sta_info; 2 数据: externData[0:N] --N 是数据类型的长度 3 说明: </pre>
例子	<pre> { struct atbm_wext_msg msg; atbm_wifi_sta_info atbm_sta_info; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_STA_INFO; ioctl(..., &msg, ...); memcpy((u8*)&atbm_sta_info, (u8*)msg.externData, sizeof(atbm_sta_info)); } </pre>

5 获取 STA 模式下，启动私有扫描

Type ID	ATBM_DEV_IO_SET_STA_SCAN
输入参数	<p>参数一</p> <p>1 类型: unsigned short</p> <p>2 数据: externData[0:1]</p> <p>3 说明: 信道号 (1~13, 0: 全信道扫描)</p> <p>参数二</p> <p>1 类型: unsigned char filter_mac[8][6];</p> <p>2 数据: externData[2:49]</p> <p>3 说明: 现在支持 8 组 MAC 设定, 其它 MAC 地址的 AP 将会被过滤掉; 如果不设置 MAC 地址, 驱动将不会进行过滤。</p> <p>注: 如果参数二为空 (即, 不设置 MAC 地址), wifi 驱动将不会根据 MAC 地址对扫描到的 AP 进行过滤。</p> <p>参数三</p> <p>1 类型: char *</p> <p>2 数据: externData[50:53]</p> <p>3 说明: 四个字节的内存地址, 传给 wifi 驱动, 用于保存扫描结果。</p>
返回值	1 类型:

	<pre>typedef struct _Wifi_Recv_Info { unsigned char Ssid[32]; unsigned char channel; unsigned int Rssi; unsigned int Quality; unsigned char phy_noise; unsigned char Bssid[6]; unsigned char User_data[USER_DATE_LEN+1]; }Wifi_Recv_Info_t;</pre> <p>2 数据：数据 copy 到输入参数的内存地址中。</p> <p>3 说明： 扫描到的 AP 会首先判断是否包含特殊数据字段，如果不包含特殊字段将被过滤掉，其次再根据扫描的 channel 进行过滤，最后根据设定的 MAC 进行过滤，扫描结果最大保存 8 个 AP 的信息。</p>	
例子	<pre>{ struct altm_wext_msg msg; unsigned short channel = 11; unsigned char filter_mac[8][6]; Wifi_Recv_Info_t private_ap_info[8]; int addr_val = (int>(&private_ap_info[0])); memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_STA_SCAN; memcpy(&msg.externData[0], &channel, (sizeof(channel))); memcpy(&msg.externData[2], &filter_mac[0][0], 8*6); memcpy(&msg.externData[50], &addr_val, sizeof(int)); ioctl(..., &msg, ...); }</pre>	

6 设置特殊频点

Type ID	ATBM_DEV_IO_SET_FREQ
输入参数	<p>参数一</p> <p>1 类型：unsigned short</p> <p>2 数据：externData[0:1]</p> <p>3 说明：信道号（1~13）</p> <p>参数二</p> <p>1 类型：int</p> <p>2 数据：externData[2:5]</p> <p>3 说明：特殊频率（2300 ~ 2400（5M 一个频点），2407 - 2489 ， 2500 ~ 2600（5M 一个频点））</p>

	<p>目前我们只验证了 2380 和 2504 的频点，其它的特殊频点性能不确定。</p> <p>注：信道和频点会保存到文件系统中， sys/module/atbm_wifi/parameters/wifi_freq;</p> <p>如果没有设置特殊频点值，会显示 NULL：</p> <pre>~ # cat sys/module/atbm_wifi/parameters/wifi_freq NULL ~ #</pre> <p>如果有设置成功特殊频点，会显示如下信息：</p> <pre>~ # cat sys/module/atbm_wifi/parameters/wifi_freq ch:11, freq:2455 ~ #</pre>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; unsigned short channel = 11; int freq = 2407; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_FREQ; memcpy(&msg.externData[0], &channel, (sizeof(channel))); memcpy(&msg.externData[2], &freq, sizeof(int)); ioctl(..., &msg, ...); }</pre>

7 设置特殊字段的数据内容

Type ID	ATBM_DEV_IO_SET_SPECIAL_OUI
输入参数	<p>1 类型：char</p> <p>2 数据：externData[0:254]</p> <p>3 说明：设置私有字段的数据内容，用于 Probe Req，Probe Resp 和 beacon 封包，数据长度 0~255 bytes；</p> <p>私有字段的 ElementId=233，OUI Type = {0x41,0x54,0x42,0x4D};//ATBM</p> <p>注：ElementId 和 OUI Type 是暂定的，可以根据用户的实际需求进行修改。</p> <p>设置的数据会被保存到文件系统 sys/module/atbm_wifi/parameters/spec_oui</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg;</pre>

	<pre> memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_SPECIAL_OUI; memcpy(&msg.externData[0], "11223344aabbcc", 14); ioctl(..., &msg, ...); } </pre>
--	--

8 Station 模式下主动断掉与 AP 之间的连接

Type ID	ATBM_DEV_IO_SET_STA_DIS
输入参数	无
返回值	无
例子	<pre> { struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_STA_DIS; ioctl(..., &msg, ...); } </pre>

9 Monitor 模式的 Enable/Disable

Type ID	ATBM_DEV_IO_SET_IFTYPE
输入参数	<p>参数一</p> <p>1 类型: char</p> <p>2 数据: externData[0]</p> <p>3 说明: iftype</p> <p>内核的枚举定义如下:</p> <pre>enum ieee80211_internal_iftype{ IEEE80211_INTERNAL_IFTYPE_REQ__MANAGED, IEEE80211_INTERNAL_IFTYPE_REQ__MONITOR, };</pre> <p>参数二</p> <p>1 类型: char</p> <p>2 数据: externData[1]</p> <p>3 说明: 信道号</p> <p>注: 使用的时候需要先把应用关掉。把 hostapd kill 掉。</p>
返回值	无
例子	{

	<pre> struct altm_wext_msg msg; char ch_type = 0; char channel= 1; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_IFTYPE; memcpy(&msg.externData[0], &ch_type, sizeof(char)); memcpy(&msg.externData[1], & channel, sizeof(char)); ioctl(..., &msg, ...); </pre>
--	--

10 Wi-Fi 自适应（CE 认证）

Type ID	ATBM_DEV_IO_SET_ADAPTIVE
输入参数	1 类型: int 2 数据: externData[0:3] 3 说明: 0:disable 1:enable
返回值	无
例子	<pre> { struct altm_wext_msg msg; int adaptive= 1;//enable memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_ADAPTIVE; memcpy(&msg.externData[0], &adaptive, sizeof(int)); ioctl(..., &msg, ...); } </pre>

11 Map 导入与 Map 内容读写

Type ID	ATBM_DEV_IO_SET_TXPWR_DCXO
输入参数	参数一 1 类型: int 2 数据: externData[0:3] 3 说明: 低频, tx power (-32~32) 参数二 1 类型: int 2 数据: externData[4:7] 3 说明: 中频, tx power (-32~32)

	<p>参数三</p> <p>1 类型: int</p> <p>2 数据: externData[8:11]</p> <p>3 说明: 高频, tx power (-32~32)</p> <p>参数四</p> <p>1 类型: int</p> <p>2 数据: externData[12:15]</p> <p>3 说明: dcxo (0~127)</p> <p>该功能设置到寄存器, 不写到 efuse</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; int tx_pwr_L= 30; int tx_pwr_M= 30; int tx_pwr_H= 30; Int dcxo = 20 memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_TXPWR_DCXO; memcpy(&msg.externData[0], &tx_pwr_L, sizeof(int)); memcpy(&msg.externData[4], &tx_pwr_M, sizeof(int)); memcpy(&msg.externData[8], &tx_pwr_H, sizeof(int)); memcpy(&msg.externData[12], &dcxo, sizeof(int)); ioctl(..., &msg, ...); }</pre>

12 功率设置

Type ID	ATBM_DEV_IO_SET_TXPWR
输入参数	<p>1 类型: char</p> <p>2 数据: externData[0]</p> <p>3 说明: tx power (-16,16)</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; int tx_pwr =3; memset(&msg, 0, sizeof(msg));</pre>

	<pre> msg.type = ATBM_DEV_IO_SET_TXPWR; msg.externData[0] = tx_pwr; ioctl(..., &msg, ...); </pre>
--	---

13 获取当前工作的信道

Type ID	ATBM_DEV_IO_GET_WORK_CHANNEL
输入参数	无
返回值	<p>1 类型: unsigned short</p> <p>2 数据: externData[0:1]</p> <p>3 说明: 当前的信道号 (1~13)</p> <p>注: 不支持 5G 信道。</p>
例子	<pre> { struct altm_wext_msg msg; unsigned short channel = 0; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_WORK_CHANNEL; ioctl(..., &msg, ...); memcpy(&channel, &msg.externData[0], sizeof(unsigned short)); } </pre>

14 最优信道

Type ID	ATBM_DEV_IO_SET_BEST_CHANNEL_SCAN
输入参数	无
返回值	<p>1 类型:</p> <pre> typedef struct _best_ch_scan_result_ { unsigned int channel_ap_num[14]; // 每个信道的 AP 个数 unsigned int busy_ratio[14]; // 每个信道的繁忙比例 unsigned char suggest_ch; // 建议的最优信道号 } Best_Channel_Scan_Result; </pre> <p>2 数据: externData[0:N]</p> <p>3 说明: 增加信道限制功能</p>

	<p>参数:</p> <p>msg.externData[0]起始信道</p> <p>msg.externData[1]截止信道</p> <p>最佳信道选择会在该范围内选择。</p>
例子	<pre> { struct altm_wext_msg msg; Best_Channel_Scan_Result rls_buf; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_BEST_CHANNEL_SCAN; if(argv[0]){ msg.externData[0] = atoi(argv[0]); if(argv[1]) msg.externData[1] = atoi(argv[1]); } ioctl(..., &msg, ...); memcpy(&rls_buf, &msg.externData[0], sizeof(rls_buf)); } </pre>

15 获取 AP 列表

Type ID	ATBM_DEV_IO_GET_AP_LIST
输入参数	<p>1 类型: char *</p> <p>2 数据: externData[0:3]</p> <p>3 说明: 四个字节的内存地址, 传给 wifi 驱动, 用于保存所有扫描到的 AP 信息。</p>
返回值	<p>1 类型:</p> <pre> typedef struct _scan_ap_info_ { u8 ssid[32]; u8 mac_addr[6]; u8 rssi; u8 flag; u8 enc_type; }SCAN_AP_INFO; </pre> <p>2 数据: AP list 列表存放在输入参数的内存地址中。</p> <p>3 说明:</p> <p>需要的内存 buffer 比较大, 18Kbytes;</p> <p>现在每个信道保存 32 个 AP 信息, 每个 AP 信息的结构如上所示, 所以需要的内存是 $14 \times 32 \times 41 = 18368$ bytes.</p>

	4enc_type 参数说明 0 open 1 WPA 2 WPA2 3 WPA/WPA2 4 WEP
例子	<pre> { struct altm_wext_msg msg; SCAN_AP_INFO scan_ap_info_buff[14][32]; int addr_val = (int)(&scan_ap_info_buff[0][0]); memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_AP_LIST; memcpy(&msg.externData[0], &addr_val, 4); ioctl(..., &msg, ...); } </pre>

16 获取吞吐量

Type ID	ATBM_DEV_IO_GET_TP_RATE
输入参数	1 类型: char 2 数据: externData[0:5] 3 说明: AP 模式需要获取连接的 sta 的当前环境的吞吐量,该参数对应的是 sta 的 mac 地址; STA 模式该参数可以为空。
返回值	1 类型: int 2 数据: externData[0:3] 3 说明: rate value
例子	<pre> { struct altm_wext_msg msg; char mac[6] = {0x11, 0x12, 0x13, 0x14, 0x15, 0x16}; int rate_val = 0; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_TP_RATE; memcpy(&msg.externData[0], mac, 6); ioctl(..., &msg, ...); memcpy(&rate_val, &msg.externData[0], sizeof(int)); } </pre>

	}
--	---

17 速率上限设置

Type ID	ATBM_DEV_IO_MAX_TX_RATE
输入参数	<p>1 类型: int</p> <p>2 数据: value, 设置为 0 恢复默认</p> <p>3 说明: 设置发送最大速率</p> <p>11b: 10,20,55,110</p> <p>11g: 60,90,120,180,240,360,480,540</p> <p>11n: 65,130,195,260,390,520,585,650</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; msg.type = ATBM_DEV_IO_MAX_TX_RATE; msg.value = 650; ioctl(..., &msg, ...); }</pre>

18 固定速率设置

Type ID	ATBM_DEV_IO_FIX_TX_RATE
输入参数	<p>1 类型: int</p> <p>2 数据: value, 设置为 0 恢复默认</p> <p>3 说明: 固定发送速率</p> <p>11b: 10,20,55,110</p> <p>11g: 60,90,120,180,240,360,480,540</p> <p>11n: 65,130,195,260,390,520,585,650</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_FIX_TX_RATE; msg.value = 650; ioctl(..., &msg, ...); }</pre>

19 速率控制取消

Type ID	ATBM_DEV_IO_TX_RATE_FREE
输入参数	无
返回值	1 类型: int 2 数据: externData[0:3] 3 说明: rate value
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_TX_RATE_FREE; ioctl(..., &msg, ...); }</pre>

20 获取 efuse

Type ID	ATBM_DEV_IO_GET_EFUSE
输入参数	无
返回值	无
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_EFUSE; ioctl(..., &msg, ...); }</pre>

21 更改 efuse mac 地址

Type ID	ATBM_DEV_IO_SET_EFUSE_MAC
输入参数	U8 mac[6]
返回值	无
例子	<pre>{ struct altm_wext_msg msg; u8 mac[6] = {0x00,0x01,0x22,0x33,0x44,0x55}; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_EFUSE_MAC; memcpy(&msg.externData[0], mac, 6); ioctl(..., &msg, ...); }</pre>

22 设置 efuse dcxo

Type ID	ATBM_DEV_IO_SET_EFUSE_DCXO
输入参数	U8 dcxo U8 write_rom
返回值	无
例子	<pre> { struct altm_wext_msg msg; u8 dcxo = 18; u8 wrie_rom = 0; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_EFUSE_DCXO; msg.externData[0] = dcxo; msg.value = wrie_rom; ioctl(..., &msg, ...); } </pre>

23 设置 efuse delta gain

Type ID	ATBM_DEV_IO_SET_EFUSE_DELTAGAIN
输入参数	U8 delta_gain1 U8 delta_gain2 U8 delta_gain3 U8 write_rom
返回值	无
例子	<pre> { struct altm_wext_msg msg; u8 delta_gain1= 18; u8 delta_gain2= 19; u8 delta_gain3= 20; u8 wrie_rom = 0; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_EFUSE_DCXO; msg.externData[0] = delta_gain1; msg.externData[1] = delta_gain2; msg.externData[2] = delta_gain3; msg.value = wrie_rom; ioctl(..., &msg, ...); } </pre>

24 设置 set min rate

Type ID	ATBM_DEV_IO_MIN_TX_RATE
输入参数	<p>1 类型: int</p> <p>2 数据: value , 设置为 0 恢复默认</p> <p>3 说明: 固定发送速率</p> <p>11b: 10,20,55,110</p> <p>11g: 60,90,120,180,240,360,480,540</p> <p>11n: 65,130,195,260,390,520,585,650</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_MIN_TX_RATE; msg.value = 650; ioctl(..., &msg, ...); }</pre>

25 设置 set rate power

Type ID	ATBM_DEV_IO_SET_RATE_POWER
输入参数	<p>Char rate_index [0:10]</p> <p>Char power [-16:16]</p> <p>rate_idx: 0~10, 对应的速率如下,</p> <p>0→ 1/2M</p> <p>1→ 5.5/11M</p> <p>2→ 6/6.5M</p> <p>3→ 9M</p> <p>4→ 12/13M</p> <p>5→ 18/19.5M</p> <p>6→ 24/26M</p> <p>7→ 36/39M</p> <p>8→ 48/52M</p> <p>9→ 54/58.5M</p> <p>10→ 65M</p> <p>rate_txpower: -16~16</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; char rate_index = 1;</pre>

	<pre> char power=1; memset(&msg, 0, sizeof(msg)); msg.type =ATBM_DEV_IO_SET_RATE_POWER; msg.externData[0] = rate_index; msg.externData[1] = power; ioctl(..., &msg, ...); } </pre>
--	--

26 设置过滤功能

Type ID	ATBM_DEV_IO_SET_SPECIAL_FILTER
输入参数	<p>msg.value:</p> <ul style="list-style-type: none"> 1 filter_frame 过滤的帧类型，只能过滤 beacon/probe req 2 filter_ie 过滤携带 IE 的帧 3 filter clean 4 filter show <p>msg.externData[0]:</p> <ul style="list-style-type: none"> filter_frame : 80 or 40 , 80 是 beacon,40 是 probe req filter_ie : ie , 携带该 IE 的帧，十进制值 <p>msg.externData[1~3]:</p> <ul style="list-style-type: none"> filter_ie : oui1 oui2 oui3 , IE 里面的 OUI，十进制值 <pre> enum SPECIAL_FILTER_TYPE{ FILTER_FRAME = 1, FILTER_IE = 2, FILTER_CLEAR = 3, FILTER_SHOW = 4, }; </pre>
返回值	无
例子	<pre> { int ret = -1; int status; struct altm_wext_msg msg; memset(&msg,0,sizeof(msg)); msg.type = ATBM_DEV_IO_SET_SPECIAL_FILTER; if(argc > 0) msg.value = atoi(argv[0]); if(argc > 1) msg.externData[0] = atoi(argv[1]); if(argc > 2) msg.externData[1] = atoi(argv[2]); if(argc > 3) msg.externData[2] = atoi(argv[3]); } </pre>

	<pre> if(argc > 4) msg.externData[3] = atoi(argv[4]); ret = ioctl_wext_send_ack(nl_connect, &msg, &status); if(ret != 0){ printf("error, ioctl send\n"); } if(msg.value == FILTER_SHOW) printf("[show] %s \n ",msg.externData); return ret; } </pre>
具体使用	<p>过滤 beacon 帧</p> <p>Atbm_Wext wlan0 filter_special 1 80</p> <p>过滤 IE 为 243 的帧</p> <p>Atbm_wext wlan0 filter_special 2 243</p> <p>过滤 IE 为 243 ， OUI 为 04-16-21 的帧</p> <p>Atbm_Wext wlan0 filter_special 2 243 4 22 33</p> <p>查看设置了哪些过滤条件</p> <p>Atbm_Wext wlan0 filter_special 4</p> <p>清除所有过滤条件</p> <p>Atbm_wext wlan0 filter_special 3</p>

27 设置国家码

Type ID	ATBM_DEV_IO_SET_COUNTRY_CODE = 31
输入参数	<p>目前只支持 2.4G 的部分设置：</p> <p>中国：CN 信道限制：1~13</p> <p>美国：US 信道限制：1~11</p> <p>日本：JP 信道限制：1~14</p>
返回值	无
例子	<pre> { struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_SET_COUNTRY_CODE; memcpy(msg.externData,"CN",2); ioctl(..., &msg, ...); } </pre>

28 获取驱动版本

Type ID	ATBM_DEV_IO_GET_DRIVER_VERSION
输入参数	无

返回值	无
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_DRIVER_VERSION; ioctl(..., &msg, ...); }</pre>

29 定频 start_tx

Type ID	ATBM_DEV_IO ETF_START_TX
输入参数	<p>msg.externData[0]: 信道: 1~14</p> <p>msg.externData[1]~ msg.externData[2]: 发包速率: 11b:10 20 55 110 11g:60 90 120 180 240 360 480 540 11n:65 130 195 260 390 520 585 650</p> <p>msg.externData[3]~ msg.externData[4]: 发包长度</p> <p>msg.externData[5]: 发包带宽: 1 →HT40 0 →HT20</p> <p>msg.externData[6]: 发包模式: 1 →greedfiled mode 0 →legacy mode</p>
返回值	无
例子	<pre>{ struct altm_wext_msg msg; short *p; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO ETF_START_TX; msg.externData[0] =1; p = (short *)&msg.externData[1]; *p = 650; p = (short *)&msg.externData[3]; *p = 1000; msg.externData[5] = 0; msg.externData[6] = 0; ioctl(..., &msg, ...); }</pre>

	}
--	---

30 定频 stop_tx

Type ID	ATBM_DEV_IO ETF_STOP_TX
输入参数	无
返回值	无
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO ETF_STOP_TX; ioctl(..., &msg, ...); }</pre>

31 定频 start_rx

Type ID	ATBM_DEV_IO ETF_START_RX
输入参数	msg.externData[0]: 信道: 1~14 msg.externData[1]: 收包带宽: 1 -->HT40 0 -->HT20
返回值	无
例子	<pre>{ struct altm_wext_msg msg; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO ETF_START_RX; msg.externData[0] = 1; msg.externData[1] = 0; ioctl(..., &msg, ...); }</pre>

32 定频 stop_rx

Type ID	ATBM_DEV_IO ETF_START_TX
输入参数	无
返回值	无
例子	<pre>{ struct altm_wext_msg msg;</pre>

	<pre> short *p; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO ETF_STOP_RX; ioctl(..., &msg, ...); } </pre>
--	--

33 定频 start_rx_result

Type ID	ATBM_DEV_IO_GET ETF_START_RX_RESULTS
输入参数	无 需要注意改命令需要在 start_rx 执行的前提下执行
返回值	无
例子	<pre> { struct altm_wext_msg msg; short *p; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET ETF_START_RX_RESULTS; ioctl(..., &msg, ...); } </pre>

34 设置单信道扫描

Type ID	ATBM_DEV_IO_SET_FIX_SCAN_CHANNEL
输入参数	无 需要注意驱动需要打开 CONFIG_ATBM_SUPPORT_AP_CONFIG 宏
返回值	无
例子	<pre> struct ieee80211_internal_ap_conf{ u8 bssid[6]; u8 ssid[32]; u8 ssid_len; u8 channel; /*others password or enc type*/ }; { struct altm_wext_msg msg; int status = 0; struct ieee80211_internal_ap_conf conf_req; memset(&msg, 0, sizeof(msg)); memset(&conf_req, 0, sizeof(conf_req)); } </pre>

	<pre> conf_req.channel = atoi(argv[0]); msg.type = ATBM_DEV_IO_SET_FIX_SCAN_CHANNEL; memcpy(msg.externData,&conf_req,sizeof(conf_req)); ret = ioctl_wext_send_ack(nl_connect, &msg, &status); } </pre>

35 启动频偏自校准功能

Type ID	ATBM_DEV_IO_SET_AUTO_CALI_PPM
输入参数	无 0 关闭 1 打开
返回值	无
例子	<pre> { struct altm_wext_msg msg; msg.type = ATBM_DEV_IO_SET_AUTO_CALI_PPM; msg.value = atoi(argv[0]); ioctl(..., &msg, ...); } </pre>

36 获取频偏自校准结果

Type ID	ATBM_DEV_IO_GET_CALI_RESULTS
输入参数	无
返回值	<p>msg.externData[0] == 2 ， 代表为共享晶体此时 dcxo 无法控制频偏，firmware 通过调节射频参数进行自动校准</p> <p>msg.externData[1] 保存 wifi 为独立晶体情况下 校准后的 dcxo 值，如果自动校准功能没有使能，该值默认为 efsue->dcxo</p> <p>msg.externData[2] ， 保存校准后的实际频偏值</p>
例子	<pre> { struct altm_wext_msg msg; msg.type = ATBM_DEV_IO_GET_CALI_RESULTS; ioctl(..., &msg, ...); if(ret != 0){ printf("error, ioctl send\n"); } } </pre>

	<pre> } if(msg.externData[0] == 2) dcxo = -1; else dcxo = msg.externData[1]; cfo_val = msg.externData[2]; printf("%s , dcxo:%d, cfo:%dppm \n",msg.externData[0] == 2?"share crystal, dcxo Setting is invalid ":"independent crystal",dcxo,cfo_val); } </pre>
--	--

37 设置功率补偿值

Type ID	ATBM_DEV_IO_SET_EFUSE_GAIN_COMPENSATION_VALUE
输入参数	<p>无</p> <p>Param1 : 补偿值, 精确到小数</p> <p>Param2 : 1 写入 efuse , 0 不写到 efuse</p> <p>功能说明:</p> <p>通过修改 efuse deltagain 寄存器进行功率补偿, 参数 1 是每次增加的功率值。</p> <p>公式为:</p> $\text{Effuse_register} = \text{effuse_register} + \text{param1} * 4$ <p>effuse_register 为临时 efuse deltagain 值</p> <p>当 efuse 值达到极限, 功率不在增加或减少!</p>
返回值	成功返回 0 失败返回-1
例子	<pre> { struct altm_wext_msg msg; msg.type = ATBM_DEV_IO_SET_EFUSE_GAIN_COMPENSATION_VALUE; compensation_data = atof(argv[0]); compensation_data *= 4; printf("compensation_data = %f \n",compensation_data); val = (int)compensation_data; msg.externData[0] = val; printf("msg.externData[0] = %d , val = %d\n",msg.externData[0],val); msg.value = atoi(argv[1]); ioctl(..., &msg, ...); } </pre>

38 获取私有自定义的 IE 数据

Type ID	ATBM_DEV_IO_GET_VENDOR_SPECIAL_IE
输入参数	无
返回值	成功返回 0 失败返回-1
例子	<pre> { struct altm_wext_msg msg; struct atbm_vendor_cfg_ie *private_ie; memset(&msg, 0, sizeof(msg)); msg.type = ATBM_DEV_IO_GET_VENDOR_SPECIAL_IE; ret = ioctl_wext_send_ack(nl_connect, &msg, &status); if(ret != 0){ printf("error, ioctl send\n"); } private_ie = (struct atbm_vendor_cfg_ie *)msg.externData; if(private_ie->ie_id == 221){ printf("ssid[%s] pwd[%s] \n",private_ie->ssid,private_ie->password); ret = 0; }else{ printf("not recive special vendor data \n"); ret = -1; } } </pre>

39 获取私有自定义的 IE 数据

Type ID	ATBM_DEV_IO_SET_STA_LISTEN_CHANNEL
输入参数	Param1: 需要监听的信道
返回值	成功返回 0 失败返回-1
例子	<pre> { struct altm_wext_msg msg; int status = 0; memset(&msg, 0, sizeof(msg)); </pre>

	<pre>msg.type = ATBM_DEV_IO_SET_STA_LISTEN_CHANNEL; msg.value = atoi(argv[0]); printf("set listen channel = %d \n",msg.value); ret = ioctl_wext_send_ack(nl_connect, &msg, &status); if(ret != 0){ printf("error, ioctl send\n"); } return 0; }</pre>	
--	--	--



CONTACT INFORMATION

AltoBeam (China) Inc.

Address: B808, Tsinghua Tongfang Hi-Tech Plaza, Haidian, Beijing, China 100083

Tel: (8610) 6270 1811

Fax: (8610) 6270 1830

Website: www.altobeam.com

Email: support@altobeam.com

DISCLAIMER

Information in this document is provided in connection with AltoBeam products. No license, express or implied, by estoppels or otherwise, to any intellectual property rights is granted by this document. Except as provided in AltoBeam's terms and conditions of sale for such products, AltoBeam assumes no liability whatsoever, and AltoBeam disclaims any express or implied warranty, relating to sale and/or use of AltoBeam products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right.

AltoBeam may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." AltoBeam reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Unauthorized use of information contained herein, disclosure or distribution to any third party without written permission of AltoBeam is prohibited.

AltoBeam™ is the trademark of AltoBeam. All other trademarks and product names are properties of their respective owners.

Copyright © 2007~2020 AltoBeam, all rights reserved