AP应用接口文档

日期	更新内容	备注
2024.09.29	v3.2版本更新如下： 1.设置类接口增加参数校验和返回值，0表示成功，非0表示失败，该改动兼容老版本 2.新增接口 2.1 uc_wiota_set_boost_level_0_5 2.2 uc_wiota_get_sm_resend_times 2.3 uc_wiota_set_sm_resend_times 2.4 uc_wiota_register_fn_refresh_callback 2.5 uc_wiota_recv_send_by_fn 2.6 uc_wiota_set_aagc_idx 2.7 uc_wiota_send_data_order 2.8 uc_wiota_register_recv_data_detail_callback 2.9 uc_wiota_register_time_service_info_callback 2.10 uc_wiota_query_corrdinate_lla 3.兼容性说明 3.1 uc_wiota_gnss_query_fix_pos替换为uc_wiota_query_coordinate_xyz 3.2 uc_wiota_paging_tx_start返回值改变，统一为设置类接口的返回值 4.修复部分接口说明错误	

前言说明:

1）AP_8288只提供镜像文件。

库接口说明

以下接口全声明在“uc_wiota_api.h”文件中。

1. 初始化WIoTa

- 目的
  WIoTa协议栈初始化。
- 语法

```
void uc_wiota_init(void);
```

- 描述
  初始化WIoTa协议栈资源，比如：线程，内存等。
- 返回值
  无。
- 参数
  无。

2. 启动WIoTa

- 目的
  启动WIoTa协议栈。
- 语法

```
void uc_wiota_run(void);
```

- 描述
  启动WIoTa协议栈，并启动基带。
- 返回值
  无。
- 参数
  无。

3. 关闭WIoTa

- 目的
  关闭WIoTa协议栈，基带也将停止。
- 语法

```
void uc_wiota_exit(void);
```

- 描述
  关闭WIoTa协议栈，回收所有WIoTa协议栈资源。
- 返回值
  无。
- 参数
  无。

4. 获取WIoTa库版本信息

- 目的
  获取当前WIoTa版本信息和构建时间。
- 语法

```
void uc_wiota_get_version(unsigned char *wiota_version_8088,
                          unsigned char *git_info_8088,
                          unsigned char *make_time_8088
                          unsigned char *wiota_version_8288,
                          unsigned char *git_info_8288,
                          unsigned char *make_time_8288
                          unsigned int *cce_version);
```

- 描述
  获取WIoTa版本信息和构建时间，包括AP和基带的版本信息，需自行开辟空间或使用数组接收出参，wiota_version大于等于15个字节，git_info大于等于36个字节，make_time大于等于36个字节， cce_version 4个字节。
  注意：如果未启动WIoTa协议栈（2. 启动WIoTa）调用该接口只能得到UC8088的版本信息，要得到UC8288和CCE的版本信息必须先启动WIoTa协议栈。
- 返回值
  无。
- 参数

```
wiota_version_8088          //当前UC8088 WIoTa库版本号
git_info_8088               //当前UC8088 WIoTa库版本git信息
make_time_8088              //当前UC8088 WIoTa库版本构建时间
wiota_version_8288          //当前UC8288 WIoTa库版本号
git_info_8288               //当前UC8288 WIoTa库版本git信息
make_time_8288              //当前UC8288 WIoTa库版本构建时间
cce_version                 //当前CCE版本号
```

5. 配置系统参数

5.1 获取系统配置

- 目的
  获取系统配置 。
- 语法

```
void uc_wiota_get_system_config(sub_system_config_t *config);
```

- 描述
  获取系统配置。
- 返回值
  无。
- 参数

```
config                      //结构体指针
```

- 参数类型

```
typedef struct
{
    unsigned char  ap_max_pow;
    unsigned char  id_len;
    unsigned char  pp;
    unsigned char  symbol_length;
    unsigned char  dlul_ratio;
    unsigned char  bt_value;
    unsigned char  group_number;
    unsigned char  spectrum_idx;
    unsigned char  old_subsys_v;
    unsigned char  bitscb;
    unsigned char  freq_idx;        // v2.7版本将频点加入系统配置中
    unsigned char  reserved;
    unsigned int system_id;         // v2.5版本之后无改参数
    unsigned int subsystem_id;      // v2.9版本之后，子系统id的高12bit将固定为mask，
用户设置之后也将被强行替换
}sub_system_config_t;
```

参数类型描述：

- ap_max_pow：AP最大发射功率，默认22dbm. 范围 0 - 29 dbm。
- id_len：user_id长度，取值0,1,2,3代表2,4,6,8字节，默认四字节，IOTE该变量需要与AP保持一致，现在只支持设置为1，即四字节。
- pp：固定为1，此值涉及同步灵敏度、传输效率等系统性能，暂时不提供修改。
- symbol_length：帧配置，取值0,1,2,3代表128,256,512,1024。

- dlul_ratio：帧配置，该值代表一帧里面上下行的比例，取值0,1代表1:1和1:2。
- bt_value：该值和调制信号的滤波器带宽对应，BT越大，信号带宽越大，取值0,1代表BT配置为1.2和BT配置为0.3，bt_value为0时，代表使用的是低阶mcs组，即低码率传输组。bt_value为1时，代表使用的是高mcs组，即高码率传输组。
- group_number：帧配置，取值0,1,2,3代表一帧里包含1,2,4,8个上行group数量。
- spectrum_idx：频谱序列号，默认为3，即470-510M(具体见下图)。
- old_subsys_v：匹配老版本iote（v2.3及之前的版本）标志位，默认值为0，表示不匹配老版本，如果需要匹配老版本，将该值设为1。
- freq_idx：频点配置，默认值160，v2.7版本将频点加入系统配置。
- bitscb：比特加扰标志位，默认值为1，表示开启比特加扰，为0表示关闭比特加扰。
- system_id：系统id，预留值，必须设置，但是不起作用。（注意：v2.5版本之后无该参数）
- subsystem_id：子系统id（子系统的识别码，终端IOTE如果要连接该子系统（AP），需要将config配置里的子系统ID参数配置成该ID）。
- na：48个字节预留位。

频谱idx	低频MHz	高频MHz	中心频率MHz	带宽MHz	频点stepMHz	频点idx	频点个数
0(other1)	223	235	229	12	0.2	0-60	61
1(other2)	430	432	431	2	0.2	0-10	11
2(EU433)	433.05	434.79	433.92	1.74	0.2	0-8	9
3(CN470-510)	470	510	490	40	0.2	0-200	201
4(CN779-787)	779	787	783	8	0.2	0-40	41
5(other3)	840	845	842.5	5	0.2	0-25	26
6(EU863-870)	863	870	866.5	7	0.2	0-35	36
7(US902-928)	902	928	915	26	0.2	0-130	131

5.2 设置系统配置

- 目的
  设置系统配置。
- 语法

```
int uc_wiota_set_system_config(sub_system_config_t *config);
```

- 描述
  设置系统配置时，注意参数个数，强烈建议先获取系统配置，再更改相关参数，最后设置系统配置。v2.3版本后子系统id支持热配置，在wiota run之后也可以设置
- 返回值
  0表示成功，非0表示失败。
- 参数
  同5.1 获取系统配置。
- 注意
  终端的系统配置需要跟AP的系统配置保持一致才能与AP同步。

6. AP端上下行状态信息（状态信息相关接口在v2.4及之后的版本将不再支持）

6.1 查询单个终端的单个状态信息

- 目的
  查询单个终端的单个状态信息。
- 语法

```
unsigned int uc_wiota_get_single_state_info_of_iote(unsigned int user_id,

                                                    uc_state_type_e state_type);
```

- 描述
  查询单个终端的单个状态信息。
- 返回值
  查询到的单个状态值。
- 参数

```
  user_id                          //要查询的终端id。
  state_type                       //状态类型
```

- 参数类型

```
  typedef enum
  {
      TYPE_UL_RECV_LEN = 1,
      TYPE_UL_RECV_SUC = 2,
      TYPE_DL_SEND_LEN = 3,
      TYPE_DL_SEND_SUC = 4,
      TYPE_DL_SEND_FAIL = 5,
      UC_STATE_TYPE_MAX
  } uc_state_type_e;
```

- 参数类型描述
  - TYPE_UL_RECV_LEN：上行接受成功的数据长度状态。
  - TYPE_UL_RECV_SUC：上行接受成功的次数状态。
  - TYPE_DL_SEND_LEN：下行发送成功的数据长度状态。
  - TYPE_DL_SEND_SUC：下行发送成功次数的状态。
  - TYPE_DL_SEND_FAIL：下行发送失败次数的状态。
  - UC_STATE_TYPE_MAX：无效状态。

6.2 查询单个终端的所有状态信息

- 目的
  查询单个终端的所有状态信息。

  注意：返回的结构体指针禁止释放。
- 语法

```
uc_state_info_t *uc_wiota_get_all_state_info_of_iote(unsigned int user_id);
```

- 描述
  查询单个终端的所有状态信息。
- 返回值

```
uc_state_info_t                    //结构体指针
```

- 返回值类型

```
typedef struct uc_state_info
{
    slist_t node;
    unsigned int user_id;
    unsigned int ul_recv_len;
    unsigned int ul_recv_suc;
    unsigned int dl_send_len;
    unsigned int dl_send_suc;
    unsigned int dl_send_fail;
}uc_state_info_t;

//单链表(下同)
typedef struct slist
{
    struct slist *next;
}slist_t
```

- 返回值类型描述
  - user_id：终端的user id。
  - ul_recv_len：单个终端上行成功接受数据的总长度，单位：byte。
  - ul_recv_suc：单个终端上行成功接受数据的次数，接受完一次完整数据后加1。
  - dl_send_len：单个终端下行成功发送数据的总长度，单位：byte。
  - dl_send_suc：单个终端下行成功发送数据的次数，发送完一次完整数据后加1。
  - dl_send_fail：单个终端下行发送数据失败的次数，一次下行数据发送失败后加1。
  - node：单链表节点。
- 参数

```
  user_id                          //要查询的终端id。
```

6.3 查询所有终端的所有状态信息

- 目的
  查询所有终端的所有状态信息。
  注意：返回的结构体指针禁止释放。
- 语法

```
uc_state_info_t *uc_wiota_get_all_state_info(void);
```

- 描述
  查询所有终端的全部状态信息。此函数返回该信息链表结构的首节点，如果节点的next不为空，则代表有相应终端的状态信息。
- 返回值

```
uc_state_info_t                    //结构体指针
```

- 返回值类型
  同6.2 查询单个终端的所有状态信息。
- 参数
  无。

6.4 重置单个终端的单个状态信息

- 目的
  重置单个终端的单个状态信息。
- 语法

```
void uc_wiota_reset_single_state_info_of_iote(unsigned int user_id,
                                              uc_state_type_e state_type);
```

- 描述
  重置单个终端的单个状态信息，即单个状态变量归零。
- 返回值
  无。
- 参数

```
  user_id                       //要重置的终端id。
  state_type                    //状态类型，同6.1
```

6.5 重置单个终端的所有状态信息

- 目的
  重置单个终端的所有状态信息。
- 语法

```
void uc_wiota_reset_all_state_info_of_iote(unsigned int user_id);
```

- 描述
  重置单个终端的所有状态信息，即所有状态变量都归零。
- 返回值
  无。
- 参数

```
  user_id                       //要重置的终端id。
```

6.6 重置所有终端的所有状态信息

- 目的
  重置所有终端的所有状态信息。
- 语法

```
void uc_wiota_reset_all_state_info(void);
```

- 描述
  重置所有终端的所有状态信息，即所有终端的所有状态变量都归零。
- 返回值
  无。

- 参数
  无。

7. 频点相关

7.1 扫描频点集合

- 目的
  扫描频点集合。
- 语法

```
uc_result_e uc_wiota_scan_freq(unsigned char *freq,
                               unsigned char freq_num,
                               unsigned char scan_type,
                               signed int timeout,
                               uc_scan_callback callback,
                               uc_scan_recv_t *scan_result);
```

- 描述
  扫描频点集合，返回各频点的详细结果，包括snr、rssi、is_synced（详细解释看本小节末尾）。
  *freq以及freq_num的意思如下：

```
unsigned char freq[freq_num] = {100,101,102……}; //freq 代表数组名，freq_num代表有效的数据个数，不一定是数组的大小。
```

函数返回执行结果超时，即UC_OP_TIMEOUT，我们期待的扫频结果scan_result将为空。
如果freq为NULL， freq_num 为 0， timeout 为 -1时，为全扫（0-200共201个频点），全扫大约需要4分钟，注意把控超时时间 。

- 返回值

```
uc_result_e
```

- 返回值类型

```
typedef enum
{
    UC_OP_SUC = 0,
    UC_OP_TIMEOUT = 1,
    UC_OP_FAIL = 2,
    UC_OP_MP_POOL = 3,
    UC_OP_DROP_CLEAR = 4,
}uc_result_e;
```

- 返回值类型描述
  - UC_OP_SUC：函数执行结果成功。
  - UC_OP_TIMEOUT：函数执行超时。
  - UC_OP_FAIL：函数执行失败。
  - UC_OP_MP_POOL：AP内存池耗光，发送数据将被丢弃，并返回该值。
- UC_OP_DROP_CLEAR：终端离开连接态时还有数据未发送，将全部清除，并返回该值。此种情况一般为一直发送失败导致终端离开连接态。
  注：下面用到uc_result_e的地方都表示相同含义。

- 参数

```
freq                    //频点集合
freq_num                //频点数量
scan_type               //扫频类型，0表示正常扫频，1表示快速扫频（只扫rssi）
timeout                 //超时时间
callback                //执行结果回调函数指针
scan_result             //扫频结果
```

- 参数类型

```
typedef void (*uc_scan_callback)(uc_scan_recv_t *result)
typedef struct
{
    unsigned short data_len;
    unsigned char  *data;
    unsigned char  result;
} uc_scan_recv_t;

//频点的信息
typedef struct
{
    unsigned char freq_idx;
    signed char snr;
    signed char rssi;
    unsigned char is_synced;
} uc_scan_freq_info_t;
```

- 参数类型描述
  - uc_scan_recv_t：扫频结果信息结构体。
    - data_len：扫频结果数据的总长度。
    - data：扫频结果数据，将类型转换为uc_scan_freq_info_t即可得到各频点的详细信息，在使用完成后，该指针需要调用者手动释放。
    - result：uc_result_e
  - uc_scan_freq_info_t：频点信息结构体。
    - freq_idx：频点。
    - snr：该频点的信噪比。
    - rssi：该频点的接收信号强度指示。
    - is_synced：该频点是否能同步上，能同步上该值为1，不能同步上该值为0。

7.2 设置默认频点

- 目的
  设置默认频点，支持热配置。
- 语法

```
int uc_wiota_set_freq_info(unsigned char freq_idx);
```

- 描述
  设置默认频点，频点范围470M-510M，每200K一个频点，v2.3版本后频点支持热配置，在wiota run之后也可以设置。
- 返回值

0表示成功，非0表示失败。

- 参数

```
freq_idx                    //范围0 ~ 200，代表频点（470 + 0.2 * freq_idx）
```

7.3 查询默认频点

- 目的
  获取当前设置的默认频点。
- 语法

```
unsigned char uc_wiota_get_freq_info(void);
```

- 描述
  获取设置的默认频点。
- 返回值

```
freq_idx                    // 频点，范围0 ~ 200
```

- 参数
  无。

7.4 设置跳频频点

- 目的
  设置跳频频点，支持的配置。
- 语法

```
int uc_wiota_set_hopping_freq(unsigned char hopping_freq);
```

- 描述
  设置跳频频点，频点范围470M-510M，每200K一个频点。 v2.3版本后频点支持热配置，在wiota run之后也可以设置。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
hopping_freq                //范围0 ~ 200，代表频点（470 + 0.2 * hopping_freq）
```

7.5 设置跳频模式

- 目的
  设置跳频模式。
- 语法

```
int uc_wiota_set_hopping_mode(unsigned char ori_freq_frame, unsigned char
hopping_freq_frame);
```

- 描述
  设置跳频模式，默认不跳频。 例如：ori_freq_frame为10， hopping_freq_frame为20则表示，在原频点工作10帧后在跳频频点工作20帧，如此循环。

- 返回值
  0表示成功，非0表示失败。
- 参数

```
ori_freq_frame                  //在原频点工作的帧数
hopping_freq_frame              //在跳频频点工作的帧数
```

8. 连接态相关

8.1 设置连接态保持时间

- 目的
  设置连接态的保持时间。
- 语法

```
int uc_wiota_set_active_time(unsigned int active_s);
```

- 描述
  设置连接态的保持时间（需要与终端保持一致）。
  终端在接入后，即进入连接态，当无数据发送或者接收时，会保持一段时间的连接态状态，在此期间AP和终端双方如果有数据需要发送则不需要再进行接入操 作，一旦传输数据就会重置连接时间，而在时间到期后，终端自动退出连接态，AP同时删除该终端连接态信息。正常流程是终端接入后发完上行数据，AP再开始发送下行数据，显然，这段时间不能太短，否则底层会自动丢掉终端的信息，导致下行无法发送成功。默认连接时间是3秒，也就是说AP侧应用层在收到终端接入后，需要在3秒内下发下行数据，超过3秒AP端将走寻呼流程，当然，重走寻呼过程再下发数据，这全是协议栈完成，应用层不可见。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
active_s                        //单位：秒，根据symbol length的不同默认值稍有不同：对应关系为symbol length为128，256，512，1024分别对应的连接态时间为2，3，4，8
```

8.2 查询连接态保持时间

- 目的
  查询连接态的连接态保持时间。
- 语法

```
unsigned int uc_wiota_get_active_time(void);
```

- 描述
  查询连接态的保持时间，单位：秒。
- 返回值

```
unsigned int                    //连接态保持的时间
```

- 参数
  无。

8.3 设置和查询连接态终端数量

- 目的
  设置和查询最大的连接态终端的数量。
- 语法

```
int uc_wiota_set_max_num_of_active_iote(unsigned short max_iote_num);
unsigned short uc_wiota_get_max_num_of_active_iote(void);
```

- 描述
  用于设置和插叙最大的连接态终端数量，默认1:1配置为72个，1:2配置为144个，一般不建议用户设置，如果有特殊需求才设置。
- 返回值
  0表示成功，非0表示失败或连接态终端数量。
- 参数

```
max_iote_num                    //默认1:1配置为72个，1:2配置为144个
```

8.4 获取终端信息（获取终端信息接口在v2.4及之后的版本将不再支持）

- 目的
  查询当前在线或离线的终端信息。
- 语法

```
iote_info_t *uc_wiota_get_iote_info(unsigned short *connected_iote_num,
                                    unsigned short *disconnected_iote_num);
```

- 描述
  查询当前终端的信息，返回信息链表头和在线总个数、离线总个数（这两个数据以参数方式返回）。
- 返回值

```
iote_info_t                     //结构体指针，使用完成后不需要手动释放
```

- 返回值类型

```
//终端信息
typedef struct iote_info
{
    slist_t node;
    unsigned int user_id;
    unsigned char iote_status;
    unsigned char group_idx;
    unsigned char subframe_idx;
}iote_info_t;

//终端状态
typedef enum
{
    STATUS_DISCONNECTED = 0,
    STATUS_CONNECTED = 1,
    STATUS_MAX
}iote_status_e;
```

- 返回值描述

- iote_info_t：终端信息。
  - user_id：终端id。
  - iote_status：终端状态，iote_status_e。
  - group_idx：终端所在的group位置信息。
  - subframe_idx：终端所在的子帧位置信息。
  - node：单链表节点。
- iote_status_e：终端状态。
  - STATUS_DISCONNECTED：表示终端处于离线状态。
  - STATUS_DISCONNECTED：表示终端处于在线状态。
  - STATUS_MAX：无效状态。
- 参数

```
connected_iote_num              //传出当前在线终端的总个数
disconnected_iote_num           //传出当前离线终端的总个数
```

8.5 打印获取的终端信息（打印获取的终端信息接口在v2.4及之后的版本将不再支持）

- 目的
  串口打印连接态的终端信息或离线的终端信息。
- 语法

```
void uc_wiota_print_iote_info(iote_info_t *head_node,
                              unsigned short connected_iote_num,
                              unsigned short disconnected_iote_num);
```

- 描述
  根据查询到的结果，串口打印终端信息。
- 返回值
  无。
- 参数

```
head_node                       //获取到的信息链表头，类型同8.4
connected_iote_num              //传出当前在线终端的总个数
disconnected_iote_num           //传出当前离线终端的总个数
```

9. 黑名单

9.1 添加终端到黑名单

- 目的
  添加一个或多个终端到黑名单（可用于删除指定id的终端，将该终端的id添加到黑名单即可）。
- 语法

```
int uc_wiota_add_iote_to_blacklist(unsigned int *user_id, unsigned short
user_id_num);
```

- 描述
  根据传入的user_id和数量，将该组user_id添加到黑名单，黑名单中的user_id将不再处理。
- 返回值
  0表示成功，非0表示失败。

- 参数

```
user_id                          //user id数组首地址
user_id_num                      //数组有效id数量
```

9.2 从黑名单中移除终端

- 目的
  将一个或多个终端从黑名单中移除。
- 语法

```
int uc_wiota_remove_iote_from_blacklist(unsigned int *user_id,
                                        unsigned short user_id_num);
```

- 描述
  根据传入的user_id和数量，将该组user_id从黑名单中移除，如果某个user_id本来就不在黑名单里，就跳过这个user_id，不做任何处理，执行下一个user_id。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
user_id                          //user id数组首地址
user_id_num                      //数组有效id数量
```

9.3 获取黑名单

- 目的
  获取已设置的黑名单信息。
- 语法

```
blacklist_t *uc_wiota_get_blacklist(unsigned short *blacklist_num);
```

- 描述
  获取已设置的黑名单链表头。
- 返回值

```
blacklist_t                      //黑名单链表头，使用完后不需要手动释放
```

- 返回值类型

```
typedef struct blacklist
{
    slist_t node;
    unsigned int user_id;
}blacklist_t
```

- 返回值描述
  - user_id：已添加的终端id。
  - node：单链表节点。
- 参数

```
blacklist_num                    //返回已添加的黑名单数量
```

9.4 打印黑名单

- 目的
  打印已获取到的黑名单内容。
- 语法

```
void uc_wiota_print_blacklist(blacklist_t *head_node, unsigned short
blacklist_num);
```

- 描述
  根据获取到的黑名单链表头,通过串口输出打印所有节点信息。
- 返回值
  无。
- 参数

```
head_node                        //获取到的黑名单链表头，类型见9.3
blacklist_num                    //获取到的黑名单总个数
```

10. 回调注册 （禁止在回调函数中加延迟或者大量操作）

10.1 终端掉线提示

- 目的
  终端掉线提示回调注册。
- 语法

```
void uc_wiota_register_iote_dropped_callback(uc_drop_callback callback);
```

- 描述
  当有终端掉线时主动上报哪一个user_id的终端掉线，可在1. 初始化WIoTa 之后或者2. 启动WIoTa 之后注册。
- 返回值
  无。
- 参数

```
typedef void (*uc_drop_callback)(unsigned int user_id);
callback                         //回调函数函数指针(参数可增加，目前只有user_id)
```

10.2 接收数据主动上报

- 目的
  数据被动上报回调注册， v2.3版本后将接入提示回调注册接口取消，合并到数据接收回调中，通过上报的数据类型区分是接入短消息还是连接态短消息（便于上层业务做终端位置管理），并在数据上报的同时上报该终端在帧结构的位置信息 。
- 语法

```
void uc_wiota_register_recv_data_callback(uc_recv_callback callback);
```

- 描述
  当上行数据接受完成后上报给应用层，可在1. 初始化WIoTa 之后或者2. 启动WIoTa之后注册。
- 返回值
  无。
- 参数

```
callback                              //回调函数函数指针
```

- 参数类型

```
typedef void (*uc_recv_callback)(unsigned int user_id, uc_dev_pos_t dev_pos,
unsigned char *data, unsigned int data_len, uc_recv_data_type_e type);

typedef struct
{
    unsigned char group_idx;
    unsigned char burst_idx;
    unsigned char slot_idx;
    unsigned char reserved;
} uc_dev_pos_t;

typedef enum
{
    DATA_TYPE_ACCESS = 0,
    DATA_TYPE_ACTIVE = 1,
    DATA_TYPE_SUBF_DATA = 2,  // v3.0新增，上行为子帧数据
} uc_recv_data_type_e;
```

- 参数类型描述
  uc_recv_callback：回调函数指针。
  - user_id：终端id。
  - dev_pos：终端在帧结构上的位置信息，包括终端所在的group_idx, burst_idx, slot_idx。
  - data：接收到的数据指针，不需要调用者释放。
  - data_len：接收到的数据长度。
  - type：接收到的数据类型，DATA_TYPE_ACCESS表示接入短消息，DATA_TYPE_ACTIVE表示连接态短消息，DATA_TYPE_SUBF_DATA为上行子帧数据，开启上行子帧接收模式后才能收到。

10.3 接收数据主动上报 (更多信息)

- 目的
  将收到的终端数据上报给用户，相比10.2上报的信息更多，两个接收回调只能生效一个。
- 语法

```
void uc_wiota_register_recv_data_detail_callback(uc_recv_detail_callback
callback);
```

- 描述
  当上行数据接受完成后上报给应用层，可在1. 初始化WIoTa 之后或者2. 启动WIoTa之后注册。
- 返回值
  无。
- 参数

```
callback                                    //回调函数函数指针
```

- 参数类型

```
typedef void (*uc_recv_detail_callback)(uc_recv_detail_t *recv_detail);

typedef struct
{
    unsigned int user_id;
    uc_recv_data_type_t data_type;
    unsigned char *data;
    unsigned short data_len;
    signed char rssi;
    unsigned char delay;
    unsigned char fn_cnt;
    unsigned char group_idx;
    unsigned char burst_idx;
    unsigned char slot_idx;
    unsigned int frame_num;
} uc_recv_detail_t;
```

- 参数类型描述
  uc_recv_detail_t：回调函数指针。
  - user_id：终端id。
  - data_type：接收到的数据类型，DATA_TYPE_ACCESS表示接入短消息，DATA_TYPE_ACTIVE表示连接态短消息，DATA_TYPE_SUBF_DATA为上行子帧数据，开启上行子帧接收模式后才能收到。
  - data：接收到的数据指针，不需要调用者释放。
  - data_len：接收到的数据长度。
  - rssi：接收到该数据时的信号强度，这个值不一定准确，只能作为参考。
  - delay：接入消息的延迟，只在data_type为DATA_TYPE_ACCESS时有效。
  - fn_cnt：子帧数据的计数，只在data_type为DATA_TYPE_SUBF_DATA时有效。
  - group_idx、burst_idx、slot_idx：终端在帧结构上的位置信息，包括终端所在的group_idx, burst_idx, slot_idx。
  - frame_num：收到该数据时的帧号，也是终端发送完成时的帧号。

10.4 AP帧号刷新

- 目的
  实时监控AP帧号变化。
- 语法

```
void uc_wiota_register_fn_refresh_callback(uc_fn_refresh_callback callback);
```

- 描述
  每当帧号更新时，调用通知用户，帧号刷新，主要用于需要帧号的业务，比如靠指定帧收发实现的分时收发策略。
- 返回值
  无。
- 参数

```
callback                                    //回调函数函数指针
```

- 参数类型

```
typedef void (*uc_fn_refresh_callback)(unsigned int frame_num);
```

- 参数类型描述
  uc_fn_refresh_callback：回调函数指针。
  - frame_num：AP帧号，在一帧的开头更新

11. 数据发送

11.1 设置和查询广播的传输速率

- 目的
  设置和查询广播的mcs（包括普通广播和OTA）。
- 语法

```
int uc_wiota_set_broadcast_mcs(uc_mcs_level_e mcs);
uc_mcs_level_e uc_wiota_get_broadcast_mcs(void);
```

- 描述
  设置广播的传输速率，分为7个等级，OTA默认等级2，等级越高，速率越高。
- 返回值
  0表示成功，非0表示失败或广播MCS。
- 参数

```
mcs                                 //mcs等级
```

- 参数类型

```
typedef enum
{
    UC_MCS_LEVEL_0 = 0,
    UC_MCS_LEVEL_1,
    UC_MCS_LEVEL_2,
    UC_MCS_LEVEL_3,
    UC_MCS_LEVEL_4,
    UC_MCS_LEVEL_5,
    UC_MCS_LEVEL_6,
    UC_MCS_LEVEL_7,
    UC_MCS_LEVEL_AUTO = 8,
    UC_MCS_LEVEL_INVALID = 9,
}uc_mcs_level_e;
```

- 参数描述
  BT=0.3（即bt_value = 1时，5.1 获取系统配置） 时在不同symbol length和不同MCS下，对应每帧传输的应用数据量（byte）会有差别，NA表示不支持，见下表：

  (备注：下表中为单播数据包的数据量，如果是普通广播包，下表每项减2，如果是OTA包，下表每项减1)

symbol length	mcs0	mcs1	mcs2	mcs3	mcs4	mcs5	mcs6	mcs7
128	6	8	51	65	79	NA	NA	NA
256	6	14	21	51	107	156	191	NA
512	6	14	30	41	72	135	254	296
1024	6	14	30	62	107	219	450	618

Note1：由于协议限制，广播和单波在不同symbol_length下支持的最大MCS不同，但设置超过最大MCS时，默认设置为最大MCS，见下表：

symbol length	广播最大MCS	单波最大MCS
128	4	4
256	6	6
512	6	7
1024	5	7

Note2：当OTA的MCS为高阶MCS且一直发送时，此时发送上行会失败，在此种场景下要发上行，请采用低阶MCS发送OTA。128配置MCS大于等于MCS2为高阶小于MCS2为低阶；256配置MCS大于等于MCS3为高阶小于MCS3为低阶；512和1024配置MCS大于等于MCS4为高阶小于MCS4为低阶。

11.2 设置和查询广播发送轮数

- 目的
  设置普通广播、OTA或下行子帧数据的发送轮数。
- 语法

```
int uc_wiota_set_broadcast_send_round(unsigned char round);
unsigned char uc_wiota_get_broadcast_send_round(void);
```

- 描述
  由于广播没有ACK，一次广播数据会默认会发3轮，保证IOTE接收成功率，当信号好时用户可设置发送轮数，缩短发送时间 。
- 返回值
  0表示成功，非0表示失败或发送轮数。
- 参数

```
round                           //广播发送轮数
```

11.3 广播数据发送

- 目的
  发送广播数据给所有終端，现在发送广播（OTA或普通广播）时可同时进行上下行业务。
- 语法

```
uc_result_e uc_wiota_send_broadcast_data(unsigned char *send_data,
                                         unsigned short send_data_len,
                                         broadcast_mode_e mode,
                                         signed int timeout,
                                         uc_send_callback callback,
                                         void *para);
```

- 描述
  发送广播数据给所有终端，有两种模式，设置mode的值决定为哪种模式。
  如果callback为NULL，为阻塞调用，需要等到函数返回值为UC_SUCCESS才能发送下一个包。
  如果callback不NULL，为非阻塞调用。

  备注：（禁止在回调函数中加延迟或者大量操作）
- 返回值

```
uc_result_e                         //函数执行结果
//当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
send_data                           //要发送的数据，该指针如果是调用者malloc的空间，需要调用者自己释放，且调用完该接口后即可释放
send_data_len                       //要发送的数据长度，最大为1024byte
mode                                //发送的模式广播或OTA，见下说明
timeout                             //超时时间，发送1k数据的时间大约为4s，若要发送大量数据请将数据分段并控制发送频率
callback                            //执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用，结构见下
para                                //用于非阻塞发送，使应用成感知每段数据的发送情况，一般传入发送数据的地址，在数据发送成功后返回该地址，表示某段数据发送结束，不需要该功能时填NULL
```

- 参数类型

```
typedef enum
{
    NORMAL_BROADCAST = 0,
    OTA_BROADCAST    = 1,
    INVALID_BROADCAST,
}broadcast_mode_e;

typedef void (*uc_send_callback)(uc_send_recv_t *result)
typedef struct
{
    unsigned int user_id; // 发送广播时，该id无意义
    unsigned int data_id;
    unsigned char result;
}uc_send_recv_t;
```

- 参数类型描述
  - broadcast_mode_e： 广播类型。
    - NORMAL_BROADCAST：普通广播模式，数据量小，速率相对较低。
    - OTA_BROADCAST：OTA模式，数据量大，速率相对较高。

11.4 设置组播ID

- 目的
  设置用于发送组播的组播ID，需要跟终端约定，最多设置8个组播ID。
- 语法

```
int uc_wiota_set_multicast_id(unsigned int *multicast_id, unsigned int id_num);
```

- 描述
  设置一组组播ID用于发送组播。最多设置8个，可多次设置，只有设置了组播ID才能向该ID发送组播。 故发送组播前必须先设置组播ID，否则会提示发送失败。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
multicast_id                    //组播id数组首地址
id_num                          //组播id个数
```

11.5 删除组播ID

- 目的
  用于删除组播ID，对于设置错误或不再需要的组播ID，可进行删除。
- 语法

```
int uc_wiota_del_multicast_id(unsigned int *multicast_id, unsigned int id_num);
```

- 描述
  删除一组组播ID，删除后不可再用该组ID发送组播。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
multicast_id                    //组播id数组首地址
id_num                          //组播id个数
```

11.6 发送组播数据

- 目的
  设置好组播id后可发送组播消息。设置了相同组播id的终端可接收到消息
- 语法

```
uc_result_e uc_wiota_send_multicast_data(unsigned char *send_data,
                                         unsigned short send_data_len,
                                         unsigned int multicast_id,
                                         signed int timeout,
                                         uc_send_callback callback
                                         void *para);
```

- 描述
  可向一组终端发送数据。
  如果回调函数不为NULL，则非阻塞模式，成功发送数据或者超时后会调用callback返回结果。
  如果回调函数为NULL，则为阻塞模式，成功发送数据或者超时该函数才会返回结果。

  备注：（禁止在回调函数中加延迟或者大量操作）

- 返回值

```
uc_result_e                          //函数执行结果
//当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
send_data                            //要发送的数据，该指针如果是调用者malloc的空间，需要调用者自己释放，且调用完该接口后即可释放
send_data_len                        //要发送的数据长度，最大为300byte
user_id                              //要发送数据的终端的组播ID
timeout                              //超时时间
callback                             //执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用
para                                 //用于非阻塞发送，使应用成感知每段数据的发送情况，一般传入发送数据的地址，在数据发送成功后返回该地址，表示某段数据发送结束，不需要该功能时填NULL
```

- 参数类型

```
typedef void (*uc_send_callback)(uc_send_recv_t *result)
typedef struct
{
    unsigned int user_id; // 发送组播时，该id表示组播id
    unsigned int data_id;
    unsigned char result;
}uc_send_recv_t;
```

11.7 设置和查询单播重发次数

- 目的
  设置单播重发次数，指单包的重发次数。
- 语法

```
int uc_wiota_set_sm_resend_times(unsigned char resend_times);
unsigned char uc_wiota_get_sm_resend_times(void);
```

- 描述
  设置重发次数，比如发送300字节，采用mcs3，要分为6包发送，设置的是每包的重发次数。
- 返回值
  0表示成功，非0表示失败或重发次数。
- 参数

```
resend_times                         //单播重发次数
```

11.8 指定终端发送数据

- 目的
  指定终端发送数据，只要终端同步上该AP，在任何情况下都可发送。
- 语法

```
uc_result_e uc_wiota_send_data(unsigned char *send_data,
                               unsigned short send_data_len,
                               unsigned int user_id,
                               signed int timeout,
                               uc_send_callback callback
                               void *para);
```

- 描述
  可向一个终端发送数据。
  如果回调函数不为NULL，则非阻塞模式，成功发送数据或者超时后会调用callback返回结果。
  如果回调函数为NULL，则为阻塞模式，成功发送数据或者超时该函数才会返回结果。

  备注：（禁止在回调函数中加延迟或者大量操作）
- 返回值

```
uc_result_e                         //函数执行结果
//当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
send_data                           //要发送的数据，该指针如果是调用者malloc的空间，需要调用
者自己释放，且调用完该接口后即可释放
send_data_len                       //要发送的数据长度，最大为300byte
user_id                             //要发送数据的终端的user_id
timeout                             //超时时间
callback                            //执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用
para                                //用于非阻塞发送，使应用成感知每段数据的发送情况，一般传
入发送数据的地址，在数据发送成功后返回该地址，表示某段数据发送结束，不需要该功能时填NULL
```

- 参数类型

```
typedef void (*uc_send_callback)(uc_send_recv_t *result)
typedef struct
{
    unsigned int user_id;               //发送单播时该id表示user_id
    unsigned int data_id;
    unsigned char result;
}uc_send_recv_t;
```

11.9 发送顺序业务数据

- 目的
  指定终端发送顺序业务数据，顺序业务是指：同一个子帧的多个终端顺序执行，必须等前一个终端执行完后且离开连接态后，下一个终端才能执行，是一种解决冲突的方式。
- 语法

```
uc_result_e uc_wiota_send_data_order(unsigned char *send_data,
                                     unsigned short send_data_len,
                                     unsigned int user_id,
                                     signed int timeout,
                                     unsigned int order_business,
                                     uc_send_callback callback
                                     void *para);
```

- 描述
  可向一个终端发送数据。
  如果回调函数不为NULL，则非阻塞模式，成功发送数据或者超时后会调用callback返回结果。
  如果回调函数为NULL，则为阻塞模式，成功发送数据或者超时该函数才会返回结果。

  备注：（禁止在回调函数中加延迟或者大量操作）
- 返回值

```
uc_result_e                          //函数执行结果
//当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
send_data                            //要发送的数据，该指针如果是调用者malloc的空间，需要调用
者自己释放，且调用完该接口后即可释放
send_data_len                        //要发送的数据长度，最大为300byte
user_id                              //要发送数据的终端的user_id
timeout                              //超时时间
order_business                       //顺序业务标识，为0时和uc_wiota_send_data一样，为1时
表示顺序业务开启
callback                             //执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用
para                                 //用于非阻塞发送，使应用成感知每段数据的发送情况，一般传
入发送数据的地址，在数据发送成功后返回该地址，表示某段数据发送结束，不需要该功能时填NULL
```

11.10 指定帧收发业务

- 目的
  在某一帧开始安排接收短消息和发送短消息任务，用于分时策略的精准控制。
- 语法

```
uc_result_e uc_wiota_recv_send_sm_by_fn(recv_send_by_fn_t *rs_fn);
```

- 描述
  安排某个终端在指定帧完成接收和发送，用于分时策略的实现，要完成指定帧收发需要知道AP的实时帧号，可使用10.4的接口完成帧号刷新回调的注册。
- 返回值

```
uc_result_e                          //函数执行结果
```

- 参数

```
typedef struct
{
    unsigned int user_id;        //要安排定帧任务的终端的user_id
    unsigned int start_recv_fn; //接收任务开始的帧号，该帧号必须大于等于当前帧号
    unsigned char recv_fns;      //连续安排接收任务的帧数
    unsigned char send_fns;      //发送数据的帧数，发送任务必须在接受任务结束后才能执行，
该参数可为0
    unsigned short data_len;     //发送数据的长度，可为0
    unsigned char *data;         //数据指针，可为NULL
    uc_send_callback callback;  //发送结果回调，如果有下行数据时，一定不为NULL，只支持非
阻塞模式
    void *para;                  //用户数据，一般为发送data的地址
} recv_send_by_fn_t;
// 当data_len为0或data为NULL时，只有接收任务
```

12. 授时相关接口

12.1 开启帧边界对齐功能

- 目的
  利用授时校准同步帧边界的功能是否开启。
- 语法

```
int uc_wiota_set_frame_boundary_align_func(unsigned char is_open);
```

- 描述
  开启GPS、1588或同步助手授时功能后开启帧边界对齐功能可周期性校准帧边界，如果只开授时不开启该功能，只能完成授时和定位信息，无法实现多AP的帧结构同步。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
is_open                          //是否开启帧边界对齐功能，0：关闭，1：打开
```

12.2 设置和查询开关某种授时

- 目的
  支持GPS功能的版本固件或有带1588协议的网关版本可开启此功能，开启后会周期性进行授时。
- 语法

```
int uc_wiota_set_time_service_func(time_service_type_e type, unsigned char
is_open);
unsigned char uc_wiota_get_time_service_func(time_service_type_e type);
```

- 描述
  开启授时功能后开启帧边界对齐功能可周期性校准帧边界。只能同时设置一种授时类型。
- 返回值
  0表示成功，非0表示失败或授时状态。
- 参数

```
type                             //授时类型，GPS、1588协议或同步助手
is_open                          //是否开启帧边界对齐功能，0：关闭，1：打开
```

- 参数类型

```
typedef enum
{
    TIME_SERVICE_GNSS = 0,                  //设置授时类型为GPS
    TIME_SERVICE_1588_PROTOCOL = 1,         //设置授时类型为1588协议
    TIME_SERVICE_SYNC_ASSISTANT = 2,        //设置授时类型为同步助手
} time_service_type_e;
```

12.3 查询授时过程状态

- 目的
  查询授时过程状态。
- 语法

```
time_service_state_e uc_wiota_get_time_service_state(void);
```

- 描述
  查询授时过程的状态。
- 返回值
  time_service_state_e。
- 返回值类型

```
typedef enum
{
    TIME_SERVICE_NULL = 0,                  //授时线程创建，未开启gps或1588授时的状态
    TIME_SERVICE_START = 1,                 //授时开始的状态
    TIME_SERVICE_SUC = 2,                   //一次授时成功的状态
    TIME_SERVICE_FAIL = 3,                  //授时结果偏差过大无法完成对齐校验的状态
    TIME_SERVICE_INIT_END = 4,              //初次开机经过授时完成帧头计算成功后的状态，通过
次状态可判断授时是否成功
    TIME_SERVICE_ALIGN_END = 5,             //非初次开机，每隔固定时间进行帧头对齐校准成功的
状态
    TIME_SERVICE_STOP = 6,                  //一次授时停止的状态
} time_service_state_e;
```

12.4 设置1588时间到协议栈

- 目的
  将1588授时时间传入协议栈。
- 语法

```
int uc_wiota_set_1588_protocol_rtc(unsigned int timestamp, unsigned int usec);
```

- 描述
  开启1588授时后，将外部1588获取的世界时钟源传入协议栈。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
timestamp                               //1588授时时钟源的整秒部分
usec                                    //1588授时时钟源的微秒部分
```

12.5 查询GNSS授时时的位置信息

- 目的
  当GPS授时成功后可通过该接口查询授时时的三维坐标位置和经纬度（v3.2版本新增）。
- 语法

```
void uc_wiota_gnss_query_coordinate_xyz(int *pos_x, int *pos_y, int *pos_z); //
v3.2接口名字改变
void uc_wiota_gnss_query_coordinate_lla(float *longitude,
                                        float *latotude,
                                        float *altitude);   // v3.2新增
```

- 描述
  开启GPS授时功能并成功授时后会记录位置信息，此接口可查询该位置信息。
- 返回值
  无。
- 参数

```
pos_x                           //位置信息x
pos_y                           //位置信息y
pos_z                           //位置信息z
longitude                       // 经度
latotude                        // 纬度
altitude                        // 海拔
```

12.6 设置GNSS重新定位

- 目的

一般来说AP只会在第一次启动后定位一次，后续只会授时，但如果AP移动了位置或者连续发生GNSS授时失败时，需要上层决策是否重新定位。

- 语法

```
int uc_wiota_gnss_relocation(unsigned char is_relocation);
```

- 描述
  开启功能可让GPS在下次授时时重新定位后再授时。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
is_relocation                   //是否重新定位，0：关闭重新定位，1：打开重新定位
```

12.7 注册授时状态回调函数

- 目的
  在wiota init之前注册，可监测授时过程状态。
- 语法

```
void uc_wiota_register_time_service_state_callback(
uc_time_service_callback callback);
void uc_wiota_register_time_service_info_callback(
uc_time_service_info_callback callback);
// 只生效一个回调，后者为3.2新增，上报信息更多
```

- 描述
  开启授时功能后，上报授时过程状态，应用可根据状态进行一些处理。
- 返回值
  无。
- 参数

```
uc_time_service_callback                //授时状态回调函数
uc_time_service_info_callback           //授时信息回调函数
```

- 参数类型

```
typedef void (*uc_time_service_callback)(time_service_state_e state);
typedef void (*uc_time_service_info_callback)(uc_ts_info_t *ts_info);

typedef struct
{
    unsigned int ts_state;              //授时状态，time_service_state_e
    unsigned int frame_head;            //当前帧头
    int frame_head_offset;              //当前帧头相对于当前世界时间的偏差
    unsigned int cur_time_s;            //当前世界时间的整秒部分
    unsigned int cur_time_us;           //当前世界时间的微妙部分
    int pos_x;                          //授时时的位置x
    int pos_y;                          //授时时的位置y
    int pos_z;                          //授时时的位置z
    float longitude;                    // 经度
    float latitude;                     // 纬度
    float altitude;                     // 海拔
} uc_ts_info_t;
```

12.8 授时开始

- 目的
  开始授时。
- 语法

```
void uc_wiota_time_service_start(void);
```

- 描述
  当设置授时类型为GPS、1588或同步助手后，调用该接口授时开始运行，周期校准定时器开始工作，校准周期为15分钟。
- 返回值
  无。

12.9 授时停止

- 目的
  停止授时
- 语法

```
void uc_wiota_time_service_stop(void);
```

- 描述
  停止授时，周期校准定时器也将停止。
- 返回值
  无。

12.10 授时周期设置

- 目的
  v3.1版本新增接口，可设置授时周期
- 语法

```
int uc_wiota_set_time_service_cycle(unsigned char cycle_min);
```

- 描述
  设置授时周期，默认5分钟，范围5分钟~60分钟。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
cycle_min                          //授时周期，单位：分钟
```

12.11 广播世界时间功能

- 目的
  v3.1版本新增接口，授时完成后可通过AP广播世界时间给终端
- 语法

```
int uc_wiota_set_broadcast_utc_func(unsigned char is_bc_utc);
```

- 描述
  该功能依赖于授时功能开启，授时完成后，AP根据广播帧周期广播世界时间给终端。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
is_bc_utc                          //0：关闭广播utc，1：开启广播utc
```

13. 其他接口说明

13.1 查询ap8288芯片温度

- 目的
  可实时获取到ap8288芯片的温度。
- 语法

```
uc_result_e uc_wiota_read_temperature(uc_temp_callback callback,
                                      uc_temp_recv_t *read_temp,
                                      signed int timeout)
```

- 描述
  调用该接口可读取基带芯片内部的实时温度，读取温度需要两帧左右，需要在没有任务的时候读取，有任务时会直接返回读取失败。
  如果回调函数不为NULL，则为非阻塞模式，成功执行或者超时后会调用callback返回结果。
  如果回调函数为NULL，则为阻塞模式，成功执行或者超时该函数才会返回结果。
- 返回值

```
uc_result_e                         //函数执行结果
//当callack!=NULL时直接返回成功,真正的结果由callback返回
```

- 参数

```
callback                            //函数执行结果回调，为NULL时为阻塞调用，非NULL时为非阻塞调用
read_temp                           //出参，返回读取的温度和执行结果
timeout                             //函数执行超时时间
```

- 参数类型

```
typedef void (*uc_temp_callback)(uc_temp_recv_t *result)
typedef struct
{
    signed char temp;
    unsigned char result;
}uc_temp_recv_t;
```

- 参数类型描述
  - uc_temp_callback：函数指针。
  - uc_temp_recv_t：查询结果结构体。
    - temp：查询到的温度值。
    - result：查询到的结果，uc_result_e。

13.2 设置WIoTa log开关

- 目的
  设置协议层的log开关。
- 语法

```
int uc_wiota_log_switch(uc_log_type_e log_type, unsigned char is_open);
```

- 描述
  开关协议层的log，包括uart和spi两种，可开启其中一种log，也可以同时开启（该函数不同参数设置两次）。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
log_type                            //uart和spi两种
is_open                             //是否开启该log
```

- 参数类型

```
typedef enum
{
    UC_LOG_UART = 0,
    UC_LOG_SPI = 1
}uc_log_type_e;
```

- 参数类型说明
  - UC_LOG_UART：串口log。
  - UC_LOG_SPI：spi log。

13.3 设置和查询AP CRC开关

- 目的
  设置和查询AP CRC开关。
- 语法

```
int uc_wiota_set_crc(unsigned short crc_limit);
unsigned short uc_wiota_get_crc(void);
```

- 描述
  开关协议层的CRC校验和设置检验长度。大于等于设定值则自动添加CRC，否则不添加，默认为100，即当发送的数据大于等于100字节时，协议层自动加CRC，小于100时不加CRC。
- 返回值
  0表示成功，非0表示失败或crc_limit。
- 参数

```
crc_limit                              //开启CRC的检验长度
//0：关闭CRC校验，不管数据长度多长都不加CRC
//大于0：表示加CRC的数据长度，如：crc_limit为50，则表示大于等于50个字节的数据开启CRC校验
```

13.4 设置和查询AP数据传输模式和速率

- 目的
  根据应用需求设置、查询数据传输模式和速率 。
- 语法

```
int uc_wiota_set_data_rate(uc_data_rate_mode_e rate_mode, unsigned int
rate_value);
unsigned int uc_wiota_get_data_rate(uc_data_rate_mode_e rate_mode);
```

- 描述
  四种模式：
  第一种基本模式，是基本速率设置。
  在第一种模式的基础上，在系统配置中dlul_ratio为1:2时，才能打开第二种模式，打开该模式能够提高该帧结构情况下两倍速率，默认第二种模式开启状态。
  在第一种模式的基础上，打开第三种模式，能够提升（8*(1 << group_number)）倍单终端的速率，但是会影响网络中其他终端的上行，建议在大数据量快速传输需求时使用。
  备注：group_number为系统配置中的参数。

  第四种为减少基带CRC为1字节，提高上层单帧数据量3字节。
- 返回值

  0表示成功，非0表示失败或速率值。

- 参数

```
rate_mode                           //传输模式
rate_value                          //当rate_mode为UC_RATE_NORMAL时，rate_value为UC_MCS_LEVEL
                                    //当rate_mode为UC_RATE_MID时，rate_value为0或1，表示关闭或打开，必须和终端的状态保持一致
                                    //当rate_mode为UC_RATE_HIGH时，rate_value为0，表示关闭rate_value为其他值，表示当实际发送数据量（byte）大于等于该值时才会真正开启该模式，常用建议设置rate_value为100，可单独开启，建议最好和终端状态保持一致
```

- 参数类型

```
typedef enum
{
    UC_RATE_MORMAL = 0,             //普通模式
    UC_RATE_MID = 1,                //dlul_ratio为1:2时可开启
    UC_RATE_HIGH = 2,               //连续数据包模式
    UC_RATE_CRC_TYPE = 3,           //基带CRC类型，默认为0，表示四字节CRC校验，为1表示1字节CRC校验，也就是说开启该模式后，上层单帧可多携带3字节数据
}uc_data_rate_mode_e;
```

- 参数类型描述
  - UC_RATE_MORMAL：普通模式。
  - UC_RATE_MID：dlul_ratio为1:2时可开启。
  - UC_RATE_HIGH：连续数据包模式。
  - UC_RATE_CRC_TYPE：基带CRC类型

13.5 查询id在帧结构上的位置

- 目的
  根据user_id查询对应的位置信息。
- 语法

```
uc_dev_pos_t *uc_wiota_query_dev_pos_by_userid(unsigned int *user_id, unsigned int user_id_num);
```

- 描述
  根据user id返回该id在帧结构上的位置，将运行协议栈后才能查询。
- 返回值

```
uc_dev_pos_t                        //函数执行结果

typedef struct
{
    unsigned char group_idx;
    unsigned char burst_idx;
    unsigned char slot_idx;
    unsigned char reserved;
} uc_dev_pos_t;
```

- 参数

```
user_id                         //要查询的id数组首地址
user_id_num                     //要查询的id个数
```

13.6 查询AP8288运行状态（AP主线程运行状态）

- 目的
  查询AP8288运行是否正常，该接口可大致判断AP整体的运行是否异常。
- 语法

```
ap8288_state_e uc_wiota_get_ap8288_state(void);
```

- 描述
  AP主线程依赖AP8288产生的中断驱动，如果AP8288发生异常，则AP主线程会停止运行，整个AP也就发生异常，该状态只能大概判断AP8288是否异常，如发生异常并不能准确反馈造成异常的原因。
- 返回值

```
ap8288_state_e                  //AP8288运行状态，正常或异常
```

- 返回值类型

```
typedef enum
{
    STATE_ABNORMAL = 0,             //AP8288运行状态异常
    STATE_NORMAL = 1,               //AP8288运行状态正常
} ap8288_state_e;
```

13.7 获取当前配置帧长

- 目的
  获取当期配置下的帧长。
- 语法

```
unsigned int uc_wiota_get_frame_len(void);
```

- 描述
  调用接口，返回当前配置的帧长，单位：微妙。
- 返回值
  帧长。

13.8 设置广播帧发送周期

- 目的
  设置广播帧发送周期，广播帧固定在子帧6周期发送，用于AP同步帧号给终端。
- 语法

```
int uc_wiota_set_broadcast_fn_cycle(unsigned char bc_fn_cycle);
```

- 描述
  广播帧主要用于发送帧号给IOTE，当使用同步DTU的低功耗时设为1，其他情况不建议设置，默认值11，范围0~11，0表示关闭，1~11表示间隔1~11帧发送一次广播帧。
- 参数

```
bc_fn_cycle                    //发送周期，最小0，最大11
```

- 返回值

  0表示成功，非0表示失败。

13.9 获取广播帧发送周期

- 目的
  获取广播帧发送周期。
- 语法

```
unsigned char uc_wiota_get_broadcast_fn_cycle(void);
```

- 描述
  用于获取当前的广播帧发送周期。
- 返回值
  广播帧发送周期。

13.10 开启或关闭单音发送

- 目的
  AP控制发送单音。
- 语法

```
int uc_wiota_set_single_tone(unsigned char is_open);
```

- 描述
  通过AP控制AP8288发送单音，或结束发送单音。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
is_open                    //1：打开，0：关闭
```

13.11 获取当前帧号

- 目的
  获取AP当前的帧号。
- 语法

```
unsigned int uc_wiota_get_frame_num(void);
```

- 描述
  获取AP当前的帧号。
- 返回值
  帧号。
- 参数

  无。

13.12 IOTE主动离开连接态

- 目的
  控制IOTE主动离开连接态。
- 语法

```
int uc_wiota_iote_leaving_active_state(unsigned int *user_id, unsigned int
id_num);
```

- 描述
  调用此接口终端如果在连接态将在下一帧离开连接态，不管连接态时间是否超时，如果此时该终端有上下行业务，都将失败，慎用。
- 返回值
  0表示成功，非。
- 参数

```
user_id                    //要主动离开连接态终端的id数组首地址
id_num                     //要主动离开连接态终端的id个数
```

13.13 获取模组ID

- 目的
  获取模组ID。
- 语法

```
void uc_wiota_get_module_id(unsigned char *module_id);
```

- 描述
  查询AP模组的模组ID。
- 返回值
  无。
- 参数

```
module_id                  //AP模组ID，预留20个字节
```

13.14 设置aagc idx

- 目的
  设置AP aagc idx。
- 语法

```
int uc_wiota_set_aagc_idx(unsigned char aagc_idx);
```

- 描述
  设置aagc值。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
aagc_idx                   //aagc idx
```

13.15 设置开关bl05的接收

- 目的
  设置开启或关闭boost level0.5的接收。
- 语法

```
int uc_wiota_set_boost_level_0_5(unsigned char is_open);
```

- 描述
  boost level0.5的消息接收，默认关闭。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
is_open                  //0表示关闭，1表示打开
```

14. 空中唤醒相关接口

14.1 设置连续信号唤醒配置

- 目的
  设置空中唤醒终端的寻呼配置。
- 语法

```
int uc_wiota_set_paging_tx_cfg(uc_lpm_tx_cfg_t *config);
```

- 描述
  设置空中唤醒终端的配置，开始寻呼前必须设置配置。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
config                        //寻呼配置信息
```

- 参数类型

```
typedef struct
{
    unsigned char freq;
    unsigned char spectrum_idx;
    unsigned char bandwidth;
    unsigned char symbol_length;
    unsigned short awaken_id;
    unsigned char mode;          // v3.1版本新增
    unsigned char reserved;
    unsigned int send_time;
}uc_lpm_tx_cfg_t;
```

- 参数类型说明
  - freq：频点。
  - spectrum_idx：频谱。
  - bandwidth：带宽。
  - symbol_length：symbol length。
  - awaken_id：指示需要唤醒的ID。

- mode：为0时，为默认模式，mode为1时，为扩展ID模式，ID范围更大
- reserved：对齐预留位
- send_time： 最小值为接收端检测周期

14.2 获取连续信号唤醒配置

- 目的
  获取连续信号唤醒配置。
- 语法

```
void uc_wiota_get_paging_tx_cfg(uc_lpm_tx_cfg_t *config);
```

- 描述
  获取连续信号唤醒配置。
- 返回值
  无。
- 参数

```
config                          //连续信号唤醒配置信息
```

14.3 开始发送连续信号唤醒终端

- 目的
  开始发送连续唤醒信号。
- 语法

```
int uc_wiota_paging_tx_start(void);
```

- 描述
  设置好连续信号唤醒配置后，可调用该接口进行空中唤醒对应配置的终端设备。
- 返回值
  0表示成功，非0表示失败。
- 参数
  无。

14.4 发送周期信号唤醒

- 目的
  发送周期性唤醒信号，在接收到终端发送的paging ctrl消息后，调用该接口唤醒进入同步paging低功耗的终端。
- 语法

```
void uc_wiota_register_sync_paging_callback(uc_paging_ctrl_callback callback);
uc_result_e uc_wiota_sync_paging(uc_paging_info_t *paging_info,
uc_paging_callback callback);
```

- 描述
  该接口为发送周期唤醒信号，需要提前知道终端睡眠时的帧号，并在需要唤醒时将该帧号传入。

上层如何知道终端睡眠时的帧号，通过uc_wiota_register_sync_paging_callback该接口注册回调，当终端进入sync paging低功耗前会发送一个ctrl消息通知AP，AP收到该消息后会将当前帧号通过该回调函数告知上层，当需要唤醒时，上层再将该帧号传入paging_info中通过uc_wiota_sync_paging下发周期唤醒信号，从而唤醒该终端。

- 返回值
  无或uc_result_e。
- 参数

```
paging_info                    // 同步paging信息结构体指针
callback                       //用于返回多帧寻呼是否完成，当该参数为NULL时，表示阻塞等待寻呼完成结果；当该参数不为NULL时，表示非阻塞调用，寻呼完成后会调用callback返回结果
```

- 参数类型

```
typedef void (*uc_paging_callback)(uc_paging_recv_t *result);
typedef void (*uc_paging_ctrl_callback)(unsigned int user_id, //发送paging ctrl消息的终端id
                                        unsigned char burst_idx, // 该id在帧结构的burst位置
                                        unsigned int fn_index); // 该终端实际睡眠的帧号，上层需要记录该值

typedef struct
{
    unsigned int user_id;            //同步寻呼的设备id
    unsigned int result;             //多帧寻呼是否完成
} uc_paging_recv_t;

typedef struct
{
    unsigned int user_id;            //同步寻呼的设备id
    unsigned int fn_index;           //终端睡眠时的帧号，由callback传出，paging时传入
    unsigned int detection_period;   //检测周期，与终端检测paging信号周期一致
    unsigned int send_round;         //发送唤醒信号的轮数，一般为1，表示发送一轮
    unsigned int continue_fn;        //单轮发送唤醒信号的帧数，一般为1，表示只在周期点发送，如果为2，则表示在周期点再前后各发送一帧唤醒信号，为3则表示前后各发两帧，以此类推，当超过detection_period的一半时，将变为每帧都发送唤醒信号
} uc_paging_info_t;
```

14.5 查询当前进行周期信号唤醒的任务个数

- 目的
  查询帧结构某位置上当前有多少个同步paging的任务，便于上层安排寻呼任务
- 语法

```
unsigned char uc_wiota_get_syncpaging_num(unsigned char group_idx, unsigned char subframe_idx);
```

- 描述
  传入位置信息group_idx和subframe_idx，返回该位置上正在进行同步paging的终端个数。
- 返回值
  同步paging的个数。
- 参数

```
group_idx                      //帧结构group位置，范围0到7
subframe_idx                   //帧结构子帧位置，范围0~
```

15. AP低功耗接口（需新的AP模组支持）

15.1 设置AP低功耗配置

- 目的
  AP进入paging接收检测模式，降低AP功耗。
- 语法

```
int uc_wiota_set_paging_rx_cfg(uc_lpm_rx_cfg_t *config);
```

- 描述
  设置paging rx配置。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
config                          //paging rx配置
```

- 参数类型

```
typedef struct
{
    unsigned char freq;           // 频点
    unsigned char spectrum_idx;   // 频带
    unsigned char bandwidth;      // 带宽
    unsigned char symbol_length;
    unsigned char lpm_nlen;       // 检测头配置，1,2,3,4，默认值4
    unsigned char lpm_utimes;     // 检测头配置，1,2,3，默认值2
    unsigned char threshold;      // 3~15，默认值10
    unsigned char extra_flag;     // defalut 0, if set 1, last period will use
extra_period, then wake up
    unsigned short awaken_id;     // 指示需要唤醒的ID
    unsigned short reserved;      // 4字节对齐预留位
    unsigned int detect_period;   // 接收端检测周期
    unsigned int extra_period;    // ms, extra new period before wake up
    unsigned char mode;           // old id range(narrow), 1: extend id
range(wide)
    unsigned char period_multiple;    //the multiples of detect_period using
awaken_id_ano, if 0, no need
    unsigned short awaken_id_another; //another awaken_id
  }uc_lpm_rx_cfg_t;
```

- 参数类型说明
  config：配置结构体指针
  is_need_32k_div：0，不需要降低32K时钟频率；1，需要降低32K时钟频率。
  降低32K时钟频率会导致32K定时不准。
  timeout_max：如果在检测最大次数后仍未检测到信号，也强制醒来，防止时偏过大醒不过来的情况，如果配置为0，则不会强制醒来。
  symbol_length：取值0,1,2,3代表128,256,512,1024
  threshold：检测门限，3~15，默认值10。增大该值，漏检率增大，虚警率减小。（虚警率即对噪声的敏感程度，漏检率即对唤醒信号的敏感程度）
  awaken_id：唤醒ID，根据symbol length不同，最大值不同，当symbol length为[0,1,2,3]时，当mode为0时，唤醒ID最大值限制分别为[41,82,168,339]（可等于，最小值为0，实际可能变化，以代码接口为准），可根据接口uc_wiota_get_awaken_id_limit 获取。当mode为1时，最大值限制为[1023, 4095,16383, 65535]（可等于，最小值为0，不用接口获取！）

detect_period：接收端检测周期（单位ms，最大值44000），每隔该时间，基带会自动单独起来检测一次信号，如果检测到信号，则唤醒整个系统，如果没有则继续sleep，该时间越长，整体功耗越低，相应的发送端想要唤醒接收端时则需要发送更长的时间。扩展ID模式时，detect_time必须与paging tx配置中的send time相同。
extra_flag：物理层检测到唤醒信号后，自动继续休眠的功能flag配置，设为1则开启该功能，该功能开启时，进休眠不建议32K时钟降频。
extra_period：物理层检测到唤醒信号后，自动继续休眠的时长配置，单位ms，如果 extra_period 小于等于 （detect_period + 10）ms，则继续休眠 detect_period 时长，否则继续休眠 extra_period 时长。
awaken_id_another：第二个唤醒id，范围与第一个一样，不建议两个awaken id相同，当period_multiple不为0时才有效。
period_multiple：第二个唤醒id的检测周期只能是第一个唤醒id的检测周期的倍数，该参数即为倍数，当倍数为0时，表示不检测第二个唤醒id，当倍数为1时，周期与第一个唤醒id相同，以此类推，注意，换算之后的周期，仍然有44秒的限制。（v2.9版本新增），扩展ID模式，仅支持相同周期！
mode：为0时，为默认模式，mode为1时，为扩展ID模式，ID范围更大。

15.2 获取AP低功耗配置

- 目的
  获取paging rx配置。
- 语法

```
void uc_wiota_get_paging_rx_cfg(uc_lpm_rx_cfg_t *config);
```

- 描述
  获取paging rx配置。
- 返回值
  无。
- 参数

```
config                          //paging rx配置
```

- 参数类型
  同上。

15.3 进入低功耗

- 目的
  AP进入低功耗。
- 语法

```
int uc_wiota_paging_rx_enter(unsigned char is_need_32k_div, unsigned int
timeout_max);
```

- 描述
  AP进入低功耗模式。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
is_need_32k_div                         //0，不需要降低32K时钟频率；1，需要降低32K时钟频率
timeout_max                             //进入低功耗的最大超时时间，超过该时间没被唤醒，将自动醒来
```

15.4 获取唤醒原因

- 目的
  获取AP被唤醒的原因。
- 语法

```
void uc_wiota_get_awakend_cause(uc_awaken_cause_t *awaken_cause);
```

- 描述
  获取paging的唤醒原因。
  在系统启动后，当使用接口uc_wiota_get_awakened_cause获取原因为AWAKENED_CAUSE_PAGING时，可使用本接口进一步获取paging的唤醒原因，
  如果为PAGING_WAKEN_CAUSE_NULL：则表示系统之前在paging下，可能是rtc或者spi cs唤醒；
  如果为PAGING_WAKEN_CAUSE_PAGING_TIMEOUT，则表示系统之前在paging下，并且没有检测到信号，在达到最大次数后被基带强制唤醒；
  如果为PAGING_WAKEN_CAUSE_PAGING_SIGNAL，则表示系统之前在paging下，并且基带检测到唤醒信号后唤醒系统。
  其他唤醒类型暂未使用。
- 参数

```
awaken_cause                            //唤醒原因
```

- 参数类型

```
typedef struct
{
    unsigned char is_cs_awakend;         // 是否为CS脚唤醒
    unsigned char awaken_cause;          // uc_awakened_cause_e
    unsigned char lpm_last_wakeup_cause;// uc_lpm_paging_waken_cause_e
    unsigned char lpm_last_wakeup_idx;  // 用于获取当前被唤醒时的唤醒id的idx，0或者1，表示第一个或者第二个唤醒id
    unsigned int lpm_detected_times;     // 被唤醒时的检测次数
}uc_awaken_cause_t;

typedef enum
{
    AWAKENED_CAUSE_HARD_RESET = 0,       // also watchdog reset, spi cs reset
    AWAKENED_CAUSE_SLEEP = 1,
    AWAKENED_CAUSE_PAGING = 2,
    AWAKENED_CAUSE_GATING = 3,           // no need care
    AWAKENED_CAUSE_FORCED_INTERNAL = 4, // not use
    AWAKENED_CAUSE_OTHERS,
} uc_awakened_cause_e;

typedef enum
{
    PAGING_WAKEN_CAUSE_NULL = 0,             // not from paging
    PAGING_WAKEN_CAUSE_PAGING_TIMEOUT = 1,   // from lpm timeout
    PAGING_WAKEN_CAUSE_PAGING_SIGNAL = 2,    // from lpm signal
```

```
    PAGING_WAKEN_CAUSE_SYNC_PG_TIMEOUT = 3, // from sync paging timeout
    PAGING_WAKEN_CAUSE_SYNC_PG_SIGNAL = 4,  // from sync paging signal
    PAGING_WAKEN_CAUSE_SYNC_PG_TIMING = 5,  // from sync paging timing set
    PAGING_WAKEN_CAUSE_MAX,
} uc_lpm_paging_waken_cause_e;
```

15.5 获取是否为新硬件

- 目的
  获取当前AP模组是否支持paging rx功能。
- 语法

```
unsigned char uc_wiota_get_is_new_hardware(void);
```

- 描述
  获取当前AP模组是否为新硬件，只有新模组才支持paging rx功能。
- 返回值
  1为新硬件，0为旧硬件。

16. 子帧模式相关接口(目前只用于传输语音数据)

16.1 设置子帧模式配置

- 目的
  设置子帧模式配置，协议栈根据该配置接受上行子帧数据或发送下行子帧数据，一般情况下无需设置，使用默认值即可。
- 语法

```
int uc_wiota_set_subframe_mode_cfg(uc_subf_cfg_t *subf_cfg);
```

- 描述
  设置子帧模式配置，目前只设置语音数据单帧大小和子帧数据发送轮数，协议栈根据大小组包上报或组包发送。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
subf_cfg                            //子帧模式配置
```

- 参数类型

```
typedef struct
{
    unsigned char block_size;
    unsigned char send_round;
    unsigned char na[2];
}uc_subf_cfg_t;
```

- 参数类型说明
  - block_size：语音数据单帧数据大小，协议栈编码使用。
  - send_round：下行子帧数据发送轮数，默认为1，表示发送一轮，最多三轮。

16.2 获取子帧模式配置

- 目的
  获取子帧模式配置。
- 语法

```
void uc_wiota_get_subframe_mode_cfg(uc_subf_cfg_t *subf_cfg);
```

- 描述
  获取子帧模式配置。
- 返回值
  无。
- 参数

```
subf_cfg                    //子帧模式配
```

16.3 设置上行子帧模式

- 目的
  设置上行子帧接收模式。
- 语法

```
int uc_wiota_set_ul_subframe_mode(unsigned char subf_mode, unsigned int user_id,
unsigned char rach_delay);
```

- 描述
  对单个终端开启或关闭上行子帧接收模式，对应的终端也需要相同配置，设置该模式后该终端将一直处于连接态，目前主要用于语音数据的接收。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
subf_mode                   //上行子帧模式，0表示关闭，1表示一帧只接收一包数据，2表示一帧
接收两包数据，为2时只有在dlul_radio为1时有效
user_id                     //要设置上行子帧模式的终端ID
rach_delay                  //接入上报，由用户保存并设置
```

16.4 添加下行子帧数据

- 目的
  添加下行子帧数据，即发送下行子帧数据。
- 语法

```
int uc_wiota_add_dl_subframe_data(unsigned char *data, unsigned char data_len,
unsigned char fn);
```

- 描述
  非阻塞添加下行子帧数据，协议栈根据子帧模式配置发送数据，目前主要用于语音数据的发送。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
data                    //下行子帧数据
data_len                //下行子帧数据长度
fn                      //上行子帧数据接收时的帧号，发送下行时回填
```

16.5 子帧模式测试(测试用)

- 目的
  开启或关闭子帧模式测试，需要和终端配合使用。
- 语法

```
int uc_wiota_set_subframe_test(unsigned char mode);
```

- 描述
  开启或关闭子帧模式测试。
- 返回值
  0表示成功，非0表示失败。
- 参数

```
mode                    //子帧数据测试模式，取值0~2，分别表示关闭测试，开启测试，清除测试结果
```

16.6 获取子帧模式测试结果(测试用)

- 目的
  查询子帧模式测试结果。
- 语法

```
uc_subf_test_t *uc_wiota_get_subframe_test(void);
```

- 描述
  获取子帧模式测试结果，包括有几个终端收到几次上行子帧数据和下行子帧数据发送次数。
- 返回值
  测试结果。
- 返回值类型

```
typedef struct
{
    unsigned int user_id;
    unsigned int recv_cnt;
} uc_subf_node_t

typedef struct
{
    uc_subf_node_t *subf_node;      // ul
    unsigned int subf_node_num;     // ul
    unsigned int send_cnt;          // dl
} uc_subf_test_t
```

- 返回值描述
  - user_id：接收上行子帧数据的ID。
  - recv_cnt：接收次数。
- subf_node_num：ID个数

    - send_cnt：下行发送次数
- 参数

  无。