AP应用接口文档

前言说明:

1）AP_8288只提供镜像文件。
2）本文档基本数据类型说明：

- u64_t表示数据类型unsigned long long
- s64_t表示数据类型signed long long
- ul32_t表示数据类型unsigned long
- sl32_t表示数据类型signed long
- u32_t表示数据类型unsigned int
- s32_t表示数据类型signed int
- u16_t表示数据类型unsigned short
- s16_t表示数据类型signed short
- u8_t表示数据类型unsigned char
- s8_t表示数据类型signed char
- n8_t表示数据类型char
- boolean表示数据类型unsigned char

3）本文档中参数为其他数据类型的，将在接口说明中的参数类型和参数类型描述项中说明。

4）本文档中返回值为其他类型的，将在接口说明中的返回值类型和返回值类型描述项中说明。

库接口说明

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，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。
- 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
  获取当前版本信息和构建时间。
- 语法

```
void uc_wiota_get_version(u8_t *wiota_version_8088,
                          u8_t *git_info_8088,
                          u8_t *make_time_8088
                          u8_t *wiota_version_8288,
                          u8_t *git_info_8288,
                          u8_t *make_time_8288
                          u32_t *cce_version);
```

- 描述
  获取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。
- 返回值
  无。
- 参数

```
wiota_version_8088          //当前ap8088 wiota库版本号
git_info_8088               //当前ap8088 wiota库版本git信息
make_time_8088              //当前ap8088 wiota库版本构建时间
wiota_version_8288          //当前ap8288 wiota库版本号
git_info_8288               //当前ap8288 wiota库版本git信息
make_time_8288              //当前ap8288 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
{
    u8_t  ap_max_pow;
    u8_t  id_len;
    u8_t  pn_num;
    u8_t  symbol_length;
    u8_t  dlul_ratio;
    u8_t  bt_value;
    u8_t  group_number;
    u8_t  spectrum_idx;
    u32_t system_id;
    u32_t subsystem_id;
    u8_t  na[48];
}sub_system_config_t;
```

- 参数类型描述
  - ap_max_pow：AP最大发射功率，默认27db. 范围 -1 - 29 db
  - id_len：id长度，取值0,1,2,3代表2,4,6,8字节
  - pn_num：固定为1，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，下上行比例，取值0,1代表1:1和1:2
  - bt_value：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
  - group_number：帧配置，取值0,1,2,3代表1,2,4,8个上行group数量，在symbol_length为0/1/2/3时，group_number最高限制为3/2/1/0
  - spectrum_idx：频谱序列号，默认为3，即470-510M(具体见频谱idx表)
  - system_id：系统id
  - subsystem_id：子系统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step MHz	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

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

- 描述
  设置系统配置时需要先获取系统配置。
- 返回值
  无 。
- 参数
  同5.1 获取系统配置。

注意：系统配置表需要与终端一样才能与终端同步

6. AP端上下行状态信息

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

```
u32_t uc_wiota_get_single_state_info_of_iote(u32_t 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_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。
- 语法

```
uc_state_info_t *uc_wiota_get_all_state_info_of_iote(u32_t 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 uc_state_info
{
    u32_t user_id;
    u32_t ul_recv_len;
    u32_t ul_recv_suc;
    u32_t dl_send_len;
    u32_t dl_send_suc;
    u32_t dl_send_fail;
    struct uc_state_info *next;
}uc_state_info_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
  - next：指向下一个节点的指针
- 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。
- 语法

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

- 描述
  查询所有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
  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(u32_t user_id,
                                              uc_state_type_e state_type);
```

- 描述
  重置单个终端的单个状态信息。
- 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(u32_t user_id);
```

- 描述
  重置单个终端的所有状态信息。
- 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。
- 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（见例子test_handle_scan_freq()）。
- 语法

```
uc_result_e uc_wiota_scan_freq(u8_t *freq,
                               u8_t freq_num,
                               s32_t 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。
  如果freq==NULL && freq_num == 0 && timeout == -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_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
    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
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
{
    u16_t data_len;
    u8_t  *data;
    u8_t  result;
} uc_scan_recv_t;

//频点的信息
typedef struct
{
    u8_t freq_idx;
    n8_t snr;
    s8_t rssi;
    u8_t 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。
- 语法

```
void uc_wiota_set_freq_info(u8_t freq_idx);
```

- 描述
  设置默认频点，频点范围470M-510M，每200K一个频点。
- 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

```
u8_t 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。
- 语法

```
void uc_wiota_set_hopping_freq(u8_t hopping_freq);
```

- 描述
  设置跳频频点，频点范围470M-510M，每200K一个频点。
- 返回值
  无。
- 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

```
void uc_wiota_set_hopping_mode(hopping_mode_e hopping_mode);
```

- 描述
  设置跳频模式，默认模式为HOPPING_MODE_0，不跳频。
- 返回值
  无。
- 参数

```
hopping_mode                //跳频模式
```

- 参数类型

```
typedef enum
{
    HOPPING_MODE_0 = 0,
    HOPPING_MODE_1 = 1,
    HOPPING_MODE_2 = 2,
    HOPPING_MODE_3 = 3,
    HOPPING_MODE_INVALID,
}hopping_mode_e;
```

- 参数描述
  注：0代表这一帧不跳频工作在设置的默认频点上，1代表这一帧工作在设置的跳频频点。
  - HOPPING_MODE_0：默认类型，不跳频
  - HOPPING_MODE_1：跳频模式为010101...
  - HOPPING_MODE_2：跳频模式为00110011...
  - HOPPING_MODE_3：跳频模式为000111000111...
  - HOPPING_MODE_INVALID：无效模式

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

```
void uc_wiota_set_active_time(u32_t 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。
- 返回值
  无。
- 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

```
u32_t uc_wiota_get_active_time(void);
```

- 描述
  查询连接态的保持时间，单位：秒。
- 返回值

```
active_s                        //连接态保持的时间
```

- 参数
  无。

8.3 设置连接态终端数量

- 目的
  设置同一个子帧上的最大的连接态终端的数量。
- 语法

```
void uc_wiota_set_max_active_iote_num_in_the_same_subframe(u8_t max_iote_num);
```

- 描述
  用于设置同一个子帧位置上最大的连接态终端数量。
- 返回值
  无。
- 参数

```
max_iote_num                    //默认为4，最大为8
```

8.4 获取终端信息

- 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(u16_t *connected_iote_num,
                                    u16_t *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。
- 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

```
typedef struct iote_info
{
    u32_t user_id;
    u8_t iote_status;
    u8_t group_idx;
    u8_t subframe_idx;
    struct iote_info *next;
}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
  - next：结构体指针，指向下一个节点
- 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

- 目的
  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,
                              u16_t connected_iote_num,
                              u16_t disconnected_iote_num);
```

- 描述
  根据查询到的结果，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

```
void uc_wiota_add_iote_to_blacklist(u32_t *user_id, u16_t 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
  无。
- 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

```
void uc_wiota_remove_iote_from_blacklist(u32_t *user_id,
                                         u16_t 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。
- 返回值
  无。
- 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(u16_t *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
{
    u32_t user_id;
    struct blacklist *next;
}blacklist_t
```

- 返回值描述
  - user_id：已添加的终端id
  - next：结构体指针，指向下一个节点
- 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,
                              u16_t blacklist_num);
```

- 描述
  根据获取到的黑名单链表头打印所有节点信息。
- 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

10.1 终端接入提示

- 目的
  终端接入提示回调注册。
- 语法

```
void uc_wiota_register_iote_access_callback(uc_iote_access callback);
```

- 描述
  当有终端接入时主动上报哪一个user_id的终端接入，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_iote_access)(u32_t user_id);
callback                  //回调函数函数指针(参数可增加，目前只有user_id)
```

10.2 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_iote_drop 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_iota_drop)(u32_t user_id);
callback                  //回调函数函数指针(参数可增加，目前只有user_id)
```

10.3 接收数据主动上报

- 目的
  数据被动上报回调注册。
- 语法

```
void uc_wiota_register_recv_data_callback(uc_recv callback);
```

- 描述
  当有数据时上报完成数据，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_recv)(u32_t user_id, u8_t *data, u32_t data_len, u8_t type);
callback                    //回调函数函数指针
```

- 参数类型

```
typedef void (*uc_recv)(u32_t user_id, u8_t *data, u32_t data_len, u8_t type);
```

- 参数类型描述
  uc_recv：回调函数指针
  - user_id：终端id
  - data：接收到的数据指针，不需要手动释放
  - data_len：接收到的数据长度
  - type：接收到的数据类型，为0表示普通的上行数据

11. 数据发送

11.1 设置广播的传输速率

- 目的
  设置广播的mcs（包括普通广播和OTA）。
- 语法

```
void uc_wiota_set_broadcast_mcs(uc_mcs_level_e mcs)
```

- 描述
  设置广播的传输速率，分为7个等级，OTA默认等级2，等级越高每个包可携带的数据量越大。
- 返回值
  无。
- 参数

```
mcs                         //mcs等级
```

- 参数类型

```
typedef enum
{
    UC_MCS_LEVEL_0 = 0,
    UC_MCS_LEVEL_1 = 1,
    UC_MCS_LEVEL_2 = 2,
    UC_MCS_LEVEL_3 = 3,
    UC_MCS_LEVEL_4 = 4,
    UC_MCS_LEVEL_5 = 5,
    UC_MCS_LEVEL_6 = 6,
    UC_MCS_LEVEL_7 = 7
}broadcast_mode_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：

symbol length	mcs0	mcs1	mcs2	mcs3	mcs4	mcs5	mcs6	mcs7
128	5	7	50	64	78	NA	NA	NA
256	5	13	20	50	106	155	190	NA
512	5	13	29	40	71	134	253	295
1024	5	13	29	61	106	218	449	617

11.2 广播数据发送

- 目的
  发送广播数据给所有终端，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(u8_t *send_data,
                                         u16_t send_data_len,
                                         broadcast_mode_e mode,
                                         s32_t timeout,
                                         uc_send_callback callback);
```

- 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，发送的数据大于1k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，发送的数据大于1k需要等到注册的回调返回UC_SUCCESS才能发送下一个包。
  详见：uc_wiota_interface_test.c中test_send_broadcast_data();的例子。
- 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
```

- 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
{
    u32_t user_id;
    u8_t result;
}uc_send_recv_t;
```

注：下面用到uc_send_recv_t的地方都表示相同含义

- 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
  - uc_send_recv_t： 发行结构信息结构体
    - user_id：该id在发送广播或OTA时无效，发送普通下行数据时生效
    - result：发送结果，uc_result_e

11.3 指定终端发送数据

- 目的
  指定终端发送数据，只要终端连接过，就可以调用该接口发送数据，不管该终端是不是连接态。在连接态发送普通数据也用该接口。
- 语法

```
uc_result_e uc_wiota_send_data(u8_t *send_data,
                               u16_t send_data_len,
                               u32_t *user_id,
                               u32_t user_id_num,
                               s32_t timeout,
                               uc_send_callback callback);
```

- 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。
  目前只支持单个寻呼不支持组播。
- 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数据首地址，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
user_id_num              //终端的个数
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，具体结构见11.2
```

12. 其他接口说明

12.1 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,
                                      s32_t timeout)
```

- 描述
  调用该接口可读取基带芯片的实时温度，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
{
    s8_t temp;
    u8_t 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

12.2 设置WIoTa log开关

- 目的
  设置协议层的log开关。
- 语法

```
void uc_wiota_log_switch(uc_log_type_e log_type, u8_t 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。
- 返回值
  无。
- 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

12.3 设置AP CRC开关

- 目的
  设置AP CRC开关。
- 语法

```
void uc_wiota_set_crc(u16_t crc_limit);
```

- 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
  无。
- 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：为50，则表示大于等于50个字节的数据开启CRC校验
```

12.4 设置AP数据传输模式和速率

- 目的
  根据应用需求设置数据传输模式和速率 。

- 语法

```
void uc_wiota_set_data_rate(uc_data_rate_mode_e rate_mode, u32_t rate_value);
```

- 描述
  三种模式：
  第一种基本模式，是基本速率设置，AP侧暂不支持。
  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。
- 返回值
  无。
- 参数

```
rate_mode                       //传输模式
rate_value                      //当rate_mode为UC_RATE_NORMAL时，rate_value为UC_MCS_LEVEL，AP侧暂不支持该模式
                                //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，暂不支持
    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_data_rate_mode_e;
```

- 参数类型描述
  - UC_RATE_MORMAL：普通模式，暂不支持
  - UC_RATE_MID：dlul_ratio为1:2时可开启
  - UC_RATE_HIGH：连续数据包模式

12.5 通过终端id查询对应的scramble id

- 目的
  根据user_id查询对应的scramble id 。
- 语法

```
uc_result_e uc_wiota_query_scrambleid_by_userid(u32_t *user_id,
                                                u32_t user_id_num,
                                                uc_query_callback callback,
                                                uc_query_recv_t *query_result);
```

- 描述
  根据user id返回对应的scramble id结果。
- 返回值

```
uc_result_e          //函数执行结果
```

- 参数

```
user_id              //要查询的id数组首地址
user_id_num          //要查询的id个数，一次最多100个
callback             //为NULL时为非阻塞查询，不为空时为阻塞查询
query_result         //查询到的结果
```

- 参数类型

```
typedef void (*uc_qurey_callback)(uc_query_recv_t *result)
typedef struct
{
    u32_t result;
    u32_t scramble_id_num;
    u32_t *scramble_id;
}uc_query_recv_t;
```

- 参数类型说明
  - uc_qurey_callback：回调函数指针
  - uc_query_recv_t：
    - result：查询的结果，uc_result_e
    - scramble_id_num：查询到的scramble id个数
    - scramble_id：查询到的scramble id指针，使用完成后需要调用者手动释放

12.6 查询某地址内容

- 目的
  查询内存地址的值，程序异常时使用 。
- 语法

```
u32_t uc_wiota_query_addr_content(u32_t type, u32_t addr);
```

- 描述
  查询内存地址的值，用于判断某些寄存器的工作状态。
- 返回值
  内存地址的值，默认4个字节。
- 参数

```
type                 //0:查询8088内存地址；1：查询ap8288内存地址
addr                 //内存地址，如0x3b1014
```