AP应用接口文档

前言说明:

1）AP_8288只提供镜像文件。

库接口说明

1. 初始化wiota

- 目的
  wiota系统初始化。
- 语法

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

- 描述
  初始化wiota资源，初始化线程，内存等。
- 返回值
  无。
- 参数
  无。

2. 启动wiota

- 目的
  启动wiota系统。
- 语法

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

- 描述
  启动wiota系统，进入NULL状态。
- 返回值
  无。
- 参数
  无。

3. 关闭wiota

- 目的
  关闭wiota系统，ap8288也将停止。
- 语法

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

- 描述
  关闭wiota系统，回收所有wiota系统资源。
- 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, u8_t git_info, u8_t *time);
```

- 描述
  获取版本信息和构建时间，需自行开辟空间或使用数组接收出参，wiota_version>=5个字节，git_info>=24个字节，time>=20个字节。
- 返回值
  无。
- 参数

```
wiota_version       //当前wiota库版本号
git_info            //当前wiota库版本git信息
time                //当前wiota库版本构建时间
```

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

```
  typedef struct
  {
    u8_t  ap_max_pow;     // ap最大发射功率，默认21db. 范围 -1 - 29 db.
    u8_t  id_len;         // id长度，取值0,1,2,3代表2,4,6,8字节
    u8_t  pn_num;         // 固定为1，暂时不提供修改
    u8_t  symbol_length; // 帧配置，取值0,1,2,3代表128,256,512,1024
    u8_t  dlul_ratio;     // 帧配置，下上行比例，取值0,1代表1:1和1:2
    u8_t  btvalue;        // 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
1.2和0.3，BT=1.2的数据速率比BT=0.3的高
    u8_t  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
```

```
    u8_t  spectrum_idx;  // 频谱序列号，默认为3，即470-510M(具体见频谱idx表)
    u32_t systemid;      // 系统id
    u32_t subsystemid;   // 子系统id
    u8_t  na[48];
}sub_system_config_t;
```

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
  子系统配置结构表
- 结构体
  同前一个接口
- 注意
  子系统配置表需要与ap一样才能同步

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
  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;
```

- 结构体
  无。

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
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;
```

- 参数

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

- 结构体
  无。

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，同上
```

- 参数
  无。
- 结构体
  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                 //要查询的终端id。
  state_type              //状态类型，同上
```

- 结构体
  无。

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                //要查询的终端id。
```

- 结构体
  无。

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。
- 结构体
  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。
- 返回值

```
uc_result_e
```

- 结构体

```
typedef struct
{
```

```
    u16_t data_len;      //扫频后的结果数据总长度
    u8_t  *data;         //扫频后的结果数据，，即可得到各频点的详细信息
    u8_t  result;        //扫频操作执行的结果是否成功
} uc_scan_recv_t;

typedef void (*uc_scan_callback)(uc_scan_recv_t *result)

//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

```
freq                    //频点集合
freq_num                //频点数量
timeout                 //超时时间
callback                //执行结果回调
scan_result             //扫频结果
//注：如果freq==NULL && freq_num == 0 && timeout == -1时，为全扫，全扫大约需要4分钟
```

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
```

- 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(u8_t hopping_mode);
```

- 描述
  设置跳频模式，默认为模式0，不跳频。
- 返回值
  无。
- 参数

```
hopping_mode            //0代表这一帧不跳，1代表这一帧工作在设置的跳频频点上
typedef enum
{
    HOPPING_MODE_0 = 0,     //default mode, not hopping
    HOPPING_MODE_1 = 1,     //010101...
    HOPPING_MODE_2 = 2,     //00110011...
    HOPPING_MODE_3 = 3,     //000111000111...
    HOPPING_MODE_INVALID,
}hopping_mode_e;
```

8. active态相关

8.1 设置active态保持时间

- 目的
  设置连接态的连接态保持时间。
- 语法

```
void uc_wiota_set_active_time(u32_t active_s);
```

- 描述
  设置连接态的连接态保持时间（需要与iote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，否则会底层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，系统默认值为3秒
```

8.2 查询active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active态终端数量

- 目的
  设置同一个子帧上的最大的active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active态终端数量。
- 返回值
  无。
- 参数

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

8.4 获取连接态终端信息

- 目的
  查询当前连接态的iote信息。
- 语法

```
IoteInfo_t *uc_wiota_get_connected_iotes(u16_t *iote_num);
```

- 描述
  查询当前已连接iote的信息，返回信息链表头和总个数。
- 返回值

```
IoteInfo_t              //信息链表头
```

- 参数

```
iote_num                //传出当前iote的总个数
```

8.5 获取离线终端信息

- 目的
  查询当前离线的iote信息。
- 语法

```
IoteInfo_t *uc_wiota_get_disconnected_iotes(u16_t *iote_num);
```

- 描述
  查询当前已连接iote的信息，返回信息链表头和总个数。
- 返回值

```
IoteInfo_t              //信息链表头
```

- 参数

```
iote_num                //传出当前iote的总个数
```

8.6 打印获取的终端信息

- 目的
  打印连接态的iote信息或离线的iote信息。
- 语法

```
void uc_wiota_print_iote_info(IoteInfo_T *head_node, u16_t iote_num);
```

- 描述
  根据查询到的结果，打印iote信息。
- 返回值
  无。
- 参数

```
head_node               //获取到的信息链表头
iote_num                //获取到的iote数量
```

9. 黑名单

9.1 添加iote到黑名单

- 目的
  添加一个或多个iote到黑名单（可用于删除指定id的iote，将该iote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userid添加到黑名单，黑名单中的userid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iote

- 目的
  将一个或多个iote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userid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
```

- 参数

```
blacklist_num           //返回总共的黑名单数量
```

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
blacklist_num           //获取到的黑名单总个数
```

10. 回调注册

10.1 iote接入提示

- 目的
  iote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iote接入时主动上报哪一个userid的iote接入，可在初始化之后或者wiota start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 iote掉线提示

- 目的
  iote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iote掉线时主动上报哪一个userid的iote掉线，可在初始化之后或者wiota start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，可在初始化之后或者wiota start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);
callback                    //回调函数函数指针
```

11. 数据发送

11.1 广播数据发送

- 目的
  发送广播数据给所有iote，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iote，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
send_data_len            //要发送的数据长度，最大为1024byte
//mode：
typedef enum
{
    NORMAL_BROADCAST = 0,       //模式0：数据量小，速率相对较低
    OTA_BROADCAST    = 1,       //模式1：数据量大，速率相对较高
    INVALID_BROADCAST,
}broadcast_mode_e;
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
```

11.2 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
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;
```

11.3 指定iote数据发送

- 目的
  指定iote发送数据，只要iote连接过，就可以调用该接口发送数据，不管该iote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iote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

```
typedef void (*uc_send_callback)(uc_send_recv_t *result)

send_data               //要发送的数据
send_data_len           //要发送的数据长度，最大为300byte
user_id                 //要发送数据的iote的userid数据首地址
user_id_num             //iote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
```

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ap8288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

```
typedef void (*uc_temp_callback)(uc_temp_recv_t *result)

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
```

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。
- 返回值
  无。
- 参数

```
log_type                //uart和spi两种
typedef enum
{
    UC_LOG_UART = 0,
    UC_LOG_SPI = 1
}uc_log_type_e;
is_open                 //是否开启该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。默认值为100，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
```

12.3 设置AP Grant开关

- 目的
  设置AP 连续数据包模式开关。
- 语法

```
void uc_wiota_set_crc(u16_t grant_limit);
```

- 描述
  开关协议层的是否为连续数据包模式。默认值为0关闭。
- 返回值
  无。
- 参数

```
grant_limit                 //开启grant模式长度
//0：关闭连续数据包模式，不管数据长度多长都发连续数据包
//大于0：表示发起连续数据包模式的数据长度，如：为50，则表示大于50个字节的数据下行会开启连续数据包模式
```

注：必须和终端配置一致