终端应用接口文档

库接口

1. 初始化

- 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
- 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. 连接同步ap

- 目的
  iote同步ap
- 语法
  void uc_wiota_connect(void);

- 描述
  同步ap，同步帧结构，进入SYNC状态，wiota系统处于待命状态，随时可发起随机接入。
- 返回值
  无
- 参数
  无
- 注意
  在wiota启动之后调用

5. 断开与ap的同步

- 目的
  断开同步状态
- 语法
  void uc_wiota_disconnect(void);
- 描述
  断开与AP的同步连接，回到NULL状态
- 返回值
  无
- 参数
  无

6. 查询wiota当前状态

- 目的
  查询wiota状态，为下一步操作做准备
- 语法
  UC_WIOTA_STATUS uc_wiota_get_state(void);
- 描述
  查询wiota当前状态
- 返回值
  状态枚举值
  typedef enum {
  UC_STATUS_NULL = 0,
  UC_STATUS_SYNC,
  UC_STATUS_SLEEP,
  UC_STATUS_ERROR,
  }UC_WITOA_STATUS;
- 参数
  无

7. 设置频点

- 目的
  设置频点，iote和ap需要设置相同频点才能同步
- 语法
  void uc_wiota_set_freq_info(u8_t freq_idx);
- 描述
  设置频点，频点范围470M-510M，每200K一个频点
- 返回值
  无
- 参数
  频点idx，范围0~200，代表频点（470+0.2*idx）

- 注意
  在初始化系统之后，在系统启动之前调用，否则无法生效

8. 查询频点

- 目的
  获取频点idx
- 语法
  u8_t uc_wiota_get_freq_info();
- 描述
  查询频点，频点范围470M-510M，每200K一个频点
- 返回值
  频点idx，范围0~200，代表频点（470+0.2*idx）
- 参数
  无

9. 设置用户id

- 目的
  设置用户id
- 语法
  int uc_wiota_set_userid(u32_t* id, u8_t id_len);
- 描述
  设置用户id，此id为终端唯一标识
- 返回值
  0：正常
  1：参数异常
- 参数
  id：user_id
  id建议使用2个32bit的16进制数值列表，方便传输，例：uid_list = [0x12345678,0x0]
  id_len：id长度，取值范围1~8字节
- 注意
  目前只支持4字节长度的user id

10. 获取用户id

- 目的
  获取用户id
- 语法
  void uc_wiota_get_userid(u32_t* id, u8_t* id_len);
- 描述
  获取用户id，此id为终端唯一标识
- 返回值
  id：user_id
  id_len：id长度，取值2,4,6,8字节
- 参数
  无
- 注意
  目前只支持4字节长度的user id

11. 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
  子系统配置结构表
- 参数
  无
- 结构体
  typedef struct {
  u8_t ap_max_pow; // ap最大发射功率，默认21db. 范围 0 - 31 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
  u8_t spectrum_idx; // 频谱序列号，默认为3，即470-510M
  u32_t systemid; // 系统id
  u32_t subsystemid; // 子系统id
  u8_t na[48];
  }sub_systrm_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

- 注意
  子系统配置表需要与ap一样才能同步

12. 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
- 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

13. 获取无线信道状态

- 目的
  获取信道参数
- 语法
  void uc_wiota_get_radio_info(radio_info_t *radio);
- 描述
  设置系统配置
- 返回值
  无线信道参数表，目前支持
  snr，范围 -25dB ~ 30dB
  power，范围 -18~21dBm
- 参数
  无
- 结构体
  typedef struct {
  u32_t rssi;
  u32_t ber;
  s8_t snr;
  s8_t power;
- }radio_info_t;

14. 发送数据

- 目的
  发送数据给ap
- 语法
  UC_OP_RESULT uc_wiota_send_data(u8_t* data, u16_t len, u16_t timeout, uc_send callback);
- 描述
  发送数据给ap，等待返回结果，提供两种模式
  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
- 返回值
  阻塞模式时该返回值有效
  typedef enum {
  UC_OP_SUCC = 0,

UC_OP_TIMEOUT,
UC_OP_FAIL,
}UC_OP_RESULT;

- 参数
  data：需要传输的数据的头指针，在收到返回结果之前不能释放
  len：数据长度
  callback：回调函数，非阻塞时处理返回结果
  timeout：超时时间，单位ms
- 结构体
  typedef struct {
  u16_t result;
  }uc_send_back_t,*uc_send_back_p;*
  *typedef void (*uc_send)(uc_send_back_p send_result);

15. 被动接收数据接口注册

- 目的
  被动接收数据
- 语法
  void uc_wiota_gegister_recv_data(uc_recv callback);
- 描述
  注册一个接收数据的被动回调函数，只需要系统启动后注册一次即可，每当iote收到普通数据或者广播数据时，会调用改回调函数上报数据。
- 返回值
  无
- 参数
  回调函数用于接收数据结果
- 结构体
  typedef struct {
  u8_t result;
  u8_t type; // UC_RECV_DATA_TYPE
  u16_t data_len;
  u8_t* data;
  }uc_recv_back_t,*uc_recv_back_p;*
  typedef enum {
  UC_RECV_MSG = 0,
  UC_RECV_BC,
  UC_RECV_OTA,
  UC_RECV_SCAN_RESULT,
  UC_RECV_SYNC_LOST,
  }UC_RECV_DATA_TYPE;
  *typedef void (*uc_recv)(uc_recv_back_p recv_data);

16. 主动接收数据

- 目的
  iote主动向ap申请下行数据
- 语法
  void uc_wiota_recv_data(uc_recv_back_p recv_result, u16_t timeout, uc_recv callback);
- 描述
  发送申请给ap，等待返回数据结果，提供两种模式
  如果回调函数不为NULL，则非阻塞模式，成功收到数据或者超时后会调用callback返回数据和结果
  如果回调函数为NULL，则为阻塞模式，成功收到数据或者超时该函数才会返回数据结果

- 返回值
  recv_result:阻塞模式时，返回的结果
- 参数
  timeout：超时时间，单位ms
  callback：回调函数，非阻塞时处理返回结果
- 结构体
  参见上述接口

17. 设置DCXO

- 目的
  设置频偏
- 语法
  void uc_wiota_set_dcxo(u32_t dcxo);
- 描述
  每块芯片的频偏不同，在系统启动之前需要单独配置，测试模式使用，之后量产时会测好后固定写在系统静态变量中，不需要应用管理。
- 返回值
  无
- 参数
  dcxo：频偏
- 注意
  在初始化系统之后，在系统启动之前调用，否则无法生效

18. 设置终端连接时间

- 目的
  设置终端接入后保持的时间
- 语法
  void uc_wiota_set_activetime(u32_t active_s);
- 描述
  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：生存周期，单位秒
- 注意
  需要跟AP侧同步设置，否则终端状态会不同步。

19. 获取终端连接时间

- 目的
  获取终端接入后保持的时间
- 语法
  u32_t uc_wiota_get_activetime(void);
- 描述
  同上

- 返回值
  active_s，单位秒
- 参数
  无
- 注意
  无

20. 扫频

- 目的
  扫频，获取可接入频点的RSSI和SNR，用于判断接入哪个频点
- 语法
  void uc_wiota_scan_freq(u8_t* data, u16_t len, u32_t timeout, uc_recv callback, uc_recv_back_p recv_result);
- 描述
  发送扫频频点数据，等待返回结果，提供两种模式
  如果回调函数不为NULL，则非阻塞模式，扫频结束或者超时后会调用callback返回结果
  如果回调函数为NULL，则为阻塞模式，扫频结束或者超时该函数才会返回结果
- 返回值
  recv_result
  结构体见上
  其中的data，内容为uc_freq_scan_result_t的结构体数组，其频点个数需要根据len计算得到
- 参数
  data：需要传输的数据的头指针，在收到返回结果之前不能释放，数据内容为uc_freq_scan_req_t的结构体数组
  len：数据长度，由于频点idx为8bit，所以该len也代表频点个数，如果len为0并且data为空，则代表需要全频带扫频
  callback：回调函数，非阻塞时处理返回结果
  timeout：超时时间，单位ms
  - 结构体
    typedef struct {
      unsigned char freq_idx;
    }uc_freq_scan_req_t,*uc_freq_scan_req_p;*
    typedef struct {
      unsigned char freq_idx;
      signed char   snr;
      signed char   rssi;
      unsigned char is_synced;
    *}uc_freq_scan_result_t,*uc_freq_scan_result_p;
- 注意
  需要先初始化协议栈，并且配置系统参数，特别是其中的频带信息，再启动协议栈后才能扫频操作，每次扫频只能扫一个频带的频点
  上报结果目前固定为4个，如果能同步的频点不满4个，则会从RSSI最大的频点依次上报，填满4个频点为止。