终端应用接口文档

库接口说明

1. 初始化WIoTa

- 目的
  WIoTa协议栈的初始化。
- 语法

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

- 描述
  初始化WIoTa协议栈的资源，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，进入空闲状态，即UC_STATUS_NULL。
- 返回值
  无。
- 参数
  无。

3. 关闭WIoTa

- 目的
  关闭WIoTa协议栈。
- 语法

```
void uc_wiota_exit(void);
void uc_wiota_set_exit_save_static(unsigned char is_save); // v3.1新增接口
```

- 描述
  关闭WIoTa协议栈，回收所有WIoTa协议栈资源。默认会保存静态数据（如果发现本地数据与静态区不同才会写入），可通过接口配置是否保存。
- 返回值
  无。

- 参数
  无。

4. 版本信息和设备地址

4.1 获取WIoTa库版本信息

- 目的
  WIoTa库的版本号、git信息以及编译时间。
- 语法

```
void uc_wiota_get_version(unsigned char *wiota_version, unsigned char *git_info,
unsigned char *time, unsigned int *cce_version);
```

- 描述
  WIoTa库的版本号、git信息以及编译时间。
- 返回值
  无。
- 参数
  wiota_version: wiota 库的版本信息。字符串，最大字符长度16字节。
  git_info：wiota库对应git信息。字符串，最大字符长度36字节。
  time： wiota库编译时间。字符串，最大字符长度36字节。
  cce_version: CCE版本号。无符号int型，长度4字节。
- 示例

```
  unsigned char version[16] = {0};
  unsigned char git_info[36] = {0};
  unsigned char time[36] = {0};
  unsigned int cce_version = 0;
  uc_wiota_get_version(version, git_info, time, &cce_version);
```

4.2 设备地址

- 目的
  查询和修改设置地址。
- 语法

```
void uc_wiota_set_dev_addr(unsigned int dev_addr);
void uc_wiota_get_dev_serial(unsigned char* serial);
```

- 描述
  查询和修改设备地址。注意，获取的是设备串号，其中的4字节是设备地址。 目前只支持修改设备地址，不支持修改设备串号。
- 返回值
  无。
- 参数
  dev_addr: set入参，4字节无符号数。
  serial：字符串，16字节，获取的是设备串口，其中第5个字节（1开始计数时）开始的连续4字节才是设备地址，可直接强转成4字节无符号数，即转换成设备地址，具体请看示例。
- 示例

```
  unsigned int dev_addr = 0xabcd1234;
  unsigned char serial[16] = {0};
  unsigned int* dev_addr_out = (unsigned int*)(&serial[4]);
  uc_wiota_set_dev_addr(dev_addr);
  uc_wiota_get_dev_serial(serial);
  kt_kprintf("dev addr 0x%x\n", *dev_addr_out);
```

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
  无。
- 结构体

```
WIOTA IOTE 2.5以前的版本sub_system_config_t结构如下：
typedef struct {
    unsigned char  ap_max_pow;    // ap最大发射功率，默认21db. 范围 0 - 31 db.
    unsigned char  id_len;        // id长度，取值0,1,2,3代表2,4,6,8字节
    unsigned char  pp;            // 固定为1，暂时不提供修改
    unsigned char  symbol_length; // 帧配置，取值0,1,2,3代表128,256,512,1024
    unsigned char  dlul_ratio;    // 帧配置，下上行比例，取值0,1代表1:1和1:2
    unsigned char  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BT_1.2和BT_0.3，BT_1.2的数据速率比BT_0.3的高
    unsigned char  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
    unsigned char  spectrum_idx;  // 频谱序列号，默认为3，即470-510M(具体见频谱idx表)

    unsigned char old_subsys_v;    // default 0, if set 1, match old
version(v2.3_ap8088) subsystem id
    unsigned char bitscb;        // bit scb func, defautl open after v2.3(not
include), set 1
    unsigned char reserved[2];   // for 4bytes alain
    unsigned int   systemid;      // 系统id
    unsigned int   subsystemid;   // 子系统id
    unsigned char  freq_list[16]; // 用于自动管理功能
    unsigned int  subsystemid_list[8];     // total 64 byte
  } sub_system_config_t;

WIOTA IOTE 2.5以及以后的版本主要增加了子系统ID列表，删除没有作用的系统ID值。
sub_system_config_t结构体如下：
  typedef struct {
    unsigned char  ap_tx_power;    // ap发射功率，默认21db. 范围 0 - 29 db. v2.8修改
名字
    unsigned char  id_len;        // id长度，取值0,1,2,3代表2,4,6,8字节
```

```
    unsigned char  pp;             // 固定为1，暂时不提供修改
    unsigned char  symbol_length; // 帧配置，取值0,1,2,3代表128,256,512,1024
    unsigned char  dlul_ratio;     // 帧配置，下上行比例，取值0,1代表1:1和1:2
    unsigned char  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BT_1.2和BT_0.3，BT_1.2的数据速率比BT_0.3的高
    unsigned char  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
    unsigned char  spectrum_idx;  // 频谱序列号，默认为3，即470-510M(具体见频谱idx表)

    unsigned char old_subsys_v;    // default 0, if set 1, match old
version(v2.3_ap8088) subsystem id
    unsigned char bitscb;         // bit scb func, defautl open after v2.3(not
include), set 1
    unsigned char freq_idx;   // 频点，v2.9增加此参数，同时reserved减小1字节
    unsigned char reserved;   // for 4bytes alain
    unsigned int   subsystemid;   // 子系统id
    unsigned char  freq_list[16]; // 用于自动管理功能
    unsigned int  subsystemid_list[8];     //子系统id列表，在自动运行功能打开后生效。
  } sub_system_config_t;
```

参数类型描述：

- ap_tx_power：AP发射功率，默认27dbm. 范围 -1 - 29 dbm，v2.8修改名字。
- 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，该值越大，单帧发送并成功接收的成功率会越高，但是相应速率会下降，以及单帧的发送时间会增加。如果接收到的基站信号很好，那就没必要使用512/1024的配置，因为128的配置成功率就很高了，使用512/1024会降低速率又不能提高多少有效的成功率，白白降低了传输速率。
- 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1时，代表使用的是低阶mcs组，即低码率传输组。bt_value为0时，代表使用的是高mcs组，即高码率传输组（symbol_length为2或者3时，bt_value不能配置为0）。
- 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；group_number取值为0时，显然dlul_ratio不能为1；为了保证帧长不会过长，在symbol_length为0/1/2/3时，group_number最高限制分别为3/2/1/0。
- spectrum_idx：频段序列号，默认为3，即470-510M(具体见下图)。
- old_subsys_v：默认为0，如果v2.4版本（包含）的终端与v2.3（包含）之前版本的AP通信，需要将该值设置为1。
- bitscb：bit加扰功能，默认为1，如果v2.4版本（包含）的终端与v2.3（包含）之前版本的AP通信，需要将该值设置为0。
- freq_idx：频点，详见频点配置接口。
- system_id：系统id，预留值，必须设置，但是不起作用。sync_V2.5以及以后的版本不存在此字段。
- 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）。 V2.9版本开始，subsystem_id 的高12bit，内部默认固定为0x214, 32bit的整体默认值仍为0x21456981，举例：如果配置subsystem_id为0x12345678，则会被内部修改为 0x21445678。
- freq_list：用于自动管理功能。
- subsystemid_list: 子系统ID列表，在自动运行打开后，会配合频点列表扫频连接试用。在2.5以及以后的版本才有此字段。
- na：28个字节预留位。2.5以及以后的版本不存在此字段。

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（v3.2版本之前）	430	432	431	2	0.2	0~10	11
1（v3.2版本以及之后）	400	440	420	40	0.2	0~200	201
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
  (1) 子系统配置表需要与ap一样才能同步。
  (2) 暂不支持BT_1.2，即btvalue=0。
  (3) V2.9版本开始，subsystem_id 的高12bit，内部默认固定为0x214, 32bit的整体默认值仍为0x21456981，举例：如果配置subsystem_id为 0x12345678，则会被内部修改为 0x21445678。

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，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。
- 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
  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。
  如果需要在启动之后修改子系统ID，则需要先disconnect，再配置subsystemid，再重新connect。

6. 功率设置

6.1 设置当前功率

- 目的
  设置固定功率或者自动功率(自动/手动切换)。
- 语法

```
void uc_wiota_set_cur_power(signed char power);
```

- 描述
  设置功率值，如果功率值为正常范围值，则设置成该功率，如果超出范围，则设置为对应的最大或最小功率，并且关闭自动功率模式。
  如果功率值为107，则代表恢复自动功率模式 。
- 返回值
  无。
- 参数
  power，范围-16 ~ 21dbm。 (V2.7版本更新为 -18 ~ 22 dbm)
- 注意
  无。

6.2 设置最大功率

- 目的
  设置最大功率。
- 语法

```
void uc_wiota_set_max_power(signed char power);
```

- 描述
  设置最大功率值，在自动功率模式情况下会用到最大功率值。
- 返回值
  无。
- 参数
  输入power，范围-16 ~ 21dbm。 (V2.7版本更新为 -18 ~ 22 dbm)
- 注意
  无。

7. 频点相关

7.1 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，目前的频点范围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，系统启动之前调用，否则无法生效。
  如果需要在启动之后修改频点，则需要先disconnect，再配置频点，再重新connect。

7.2 查询频点

- 目的
  获取频点idx。
- 语法

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

- 描述
  查询频点，目前频点范围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。
- 注意
  无。

7.3 扫频

- 目的
  扫频，获取可接入频点的RSSI和SNR（本小节末尾有详细介绍），用于判断接入哪个频点。
- 语法

```
void uc_wiota_scan_freq(unsigned char* data, unsigned short len, unsigned char
mode, unsigned int 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

```
  typedef struct {
    u8_t    result;  //如果这个类型为超时，则data数据无意义
    u8_t    type;  // UC_RECV_DATA_TYPE
    u16_t   data_len;
    u8_t*   data;  //是uc_freq_scan_result_t这个结构体数组的头指针，本小节后边有介绍
  }uc_recv_back_t,*uc_recv_back_p;
```

```
typedef enum {
  UC_OP_SUCC = 0,  //上一个结构体result这个参数类型
  UC_OP_TIMEOUT,
  UC_OP_FAIL,
}UC_OP_RESULT;

typedef enum {
  UC_RECV_MSG = 0,
  UC_RECV_BC,
  UC_RECV_OTA,
  UC_RECV_SCAN_RESULT,  // 第一个结构体中的type这个参数类型
  UC_RECV_SYNC_LOST,
}UC_RECV_DATA_TYPE;

typedef void (*uc_recv)(uc_recv_back_p recv_data);
```

- 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，mode为0时，数据内容为uc_freq_scan_req_t的结构体数组，mode为1时，数据内容为uc_freq_scan_req_dyn_t的结构体数组。
  len：数据长度，如果len为0并且data为空，则代表需要全频带扫频，此刻的timeout建议设置为60000ms。
  mode：扫频模式，模式0，使用已配置的子系统id，统一扫频，模式1，需要填写频点数相同个数的子系统id，一一对应。
  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 int   sub_sys_id;
    unsigned char  freq_idx;
    unsigned char  reserved0;
    unsigned short reserved1;
  }uc_freq_scan_req_dyn_t,*uc_freq_scan_req_dyn_p;

  typedef struct {
    unsigned char  freq_idx;
    signed char    snr;
    signed char    rssi;
    unsigned char  is_synced;
    unsigned int   sub_sys_id;  // v2.5 add
  }uc_freq_scan_result_t,*uc_freq_scan_result_p;
```

- 参数介绍
  - freq_idx：频点。
  - snr：该频点的信噪比，范围-25 ~ 30。
  - rssi：该频点的接收信号强度指示, 范围 -128 ~ 0。
  - is_synced：表示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。

- 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。
  API的使用可以参考AT扫频接口。

8. 用户ID相关

8.1 设置用户id

- 目的
  设置用户id。
- 语法

```
int uc_wiota_set_userid(unsigned int* id, unsigned char 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: 用户id的地址指针。
  例：

```
unsigned int uid_list[1] = {0x12345678};
uc_wiota_set_userid(uid_list,4);
```

id_len: id长度，取值范围1~4字节。

- 注意
  目前支持最大4字节长度的user id。请勿使用0值。
  如果需要在启动之后修改用户ID，则需要先disconnect，再配置userid，再重新connect。

8.2 获取用户id

- 目的
  获取用户id。
- 语法

```
void uc_wiota_get_userid(unsigned int* id, unsigned char* 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。

8.3 获取模组ID

- 目的
  获取芯片中不可擦除部分的模组ID，每颗芯片的ID全球唯一。 v2.9新增接口。
- 语法

```
void uc_wiota_get_module_id(unsigned char *module_id);
unsigned char uc_wiota_get_module_id(unsigned char *module_id); // v3.4新增返回值，表示模组ID是否有效
```

- 描述
  获取模组ID的字符串。
- 返回值
  模组ID，以出参形式返回。
  v3.4新增模组ID是否有效的返回值。
- 参数
  module_id： 模组ID长度为18个字符，第19个字符为'\0'，作为字符串的固定结束符。
  模组ID的规则请联系FAE获取相应文档。
- 举例

```
unsigned char module_id[19] = {0};
uc_wiota_get_module_id(module_id);
rt_kprintf("module id %s\n", module_id);
```

9. 同步及WIoTa状态

9.1 连接同步ap

- 目的
  iote同步到ap。
- 语法

```
void uc_wiota_connect(void);
```

- 描述
  同步到ap的同步帧结构后，WIoTa协议栈处于进入同步状态，即UC_STATUS_SYNC，此时可发起随机接入。
- 返回值
  无。
- 参数
  无。
- 注意
  在WIoTa启动之后调用。

9.2 快速连接同步ap

- 目的
  在被sync paging信号唤醒后，iote快速同步到ap。 v2.7版本更新接口。
- 语法

```
void uc_wiota_connect_quick(void);
// v2.8更新接口如下：
void uc_wiota_connect_quick(u16_t is_force_active);
```

- 描述
  在WIoTa启动之后，当判断paging唤醒原因为sync paging信号唤醒时，可调用该接口进行快速同步。
- 返回值
  无。
- 参数
  is_force_active：v2.8新增参数，是否在同步上后强制自动进入连接态。 0，不需要，1，自动进入连接态。
- 注意
  在WIoTa启动之后调用，需要判断唤醒原因为sync paging信号唤醒才能使用。

9.3 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的同步连接态，回到NULL状态。
- 返回值
  无。
- 参数
  无。

9.4 暂停与ap的同步

- 目的
  暂停同步状态。
- 语法

```
void uc_wiota_suspend_connect(void);
```

- 描述
  暂停与AP的同步连接，用于读写flash时不与底层冲突。不建议用在其他情况下！暂停时间在2~3帧长，不建议太长！
- 返回值
  无。
- 参数
  无。

9.5 恢复与ap的同步

- 目的
  恢复同步状态。
- 语法

```
void uc_wiota_recover_connect(void);
```

- 描述
  在暂停同步之后，恢复与AP的连接状态。

- 返回值
  无。
- 参数
  无。

9.6 查询WIoTa当前状态

- 目的
  查询WIoTa协议栈的状态，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_SYNC_LOST,
    UC_STATUS_SLEEP,
    UC_STATUS_ERROR,
}UC_WITOA_STATUS;
```

UC_STATUS_NULL：初始化或者关闭协议栈后，处于该状态。
UC_STATUS_SYNC：同步成功后，处于该状态。
UC_STATUS_SYNC_LOST：同步失败后，或者在SYNC状态时出现异常失步之后，处于该状态。
UC_STATUS_SLEEP：协议栈休眠时，处于SLEEP状态，该状态暂未支持。
UC_STATUS_ERROR：其他状态。

- 参数
  无。

9.7 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。
- 出参
  无线信道参数表：
  rssi: 信号强度，范围0~150，实际表示0 ~ -150dbm。
  ber: 误码率，暂不支持。
  snr: 信噪比，范围 -25dB ~ 30dB。
  cur_pow: 当前发射功率，范围 -16~21dBm。 (V2.7版本更新为 -18 ~ 22 dbm)
  min_pow: 最小发射功率，范围 -16~21dBm。 (V2.7版本更新为 -18 ~ 22 dbm)
  max_pow: 最大发射功率，范围 -16~21dBm。 (V2.7版本更新为 -18 ~ 22 dbm)

cur_mcs: 当前数据发送速率级别，范围 0~7，越大速率越高。
max_mcs: 截止目前最大数据发送速率级别，范围 0~7。
frac_offset: 基带同步频偏，仅供参考，可判断此时同步是否正常，-1500 ~ 1500都属于正常。

- 结构体

```
typedef struct {
  unsigned char   rssi;  // absolute value, 0~150 means 0 ~ -150
  unsigned char   ber;
  signed char     snr;
  signed char     cur_pow;
  signed char     min_pow;
  signed char     max_pow;
  unsigned char   cur_mcs;
  unsigned char   max_mcs;
  signed int      frac_offset;
}radio_info_t;
```

- 注意
  无。

9.8 获取当前帧号

- 目的
  获取当前帧号，可与其他终端协调工作。v2.8新增接口
- 语法

```
u32_t uc_wiota_get_frame_num(void);
u8_t uc_wiota_get_is_frame_valid(void);
```

- 描述
  查询当前帧号，查询当前帧号是否有效，有效的意思是指收到AP的帧号广播后，会更新为AP的有效帧号。
- 返回值
  当前帧号，当前帧号是否有效。
- 参数
  无。

9.9 强制回到IDLE状态

- 目的
  强制wiota协议回到IDLE状态，此时再发送上行一定会先接入，避免临界情况导致发送失败或延后帧才发送。v3.1新增接口
- 语法

```
void uc_wiota_back_to_idle(void);
```

- 描述
  强制wiota协议回到IDLE状态，并清除数据队列等，如果已经在IDLE，则接口会直接返回。
- 返回值
  无。
- 参数
  无。

9.10 等待同步状态

- 目的
  设定超时时间，在该时间内等待协议状态为sync。v3.2新增接口
- 语法

```
int uc_wiota_wait_sync(int timeout, unsigned char fn_num);
```

- 描述
  检测WIoTa协议状态是否为sync
- 返回值
  0：表示处于sync状态
  1：不处于sync状态
- 参数
  timeout：检测时间，在该时间内，一直为非sync，则超时退出，返回1
  fn_num：连续检测帧数，最小可为0，即只检测一次，如果本次为sync，则直接返回0；如果设置为2，则第一次检测为sync，延迟一帧时长，再次检测，如果仍为sync，继续延迟一帧时长，再测检测，如果仍为sync，此时才会返回0，中间如果某次不为sync，则计数清零，重新开始连续检测。
- 注意
  连续帧数，主要是为了保证协议确实处于sync状态，建议设成2，为了避免虚警等临界问题，避免协议只是临时处于sync状态。

10. 频偏及dcxo

10.1 设置DCXO

- 目的
  设置频偏，无源晶体才需要设置dcxo。
  获取频偏值，v2.8新增接口。
- 语法

```
void uc_wiota_set_dcxo(unsigned int dcxo);
unsigned int uc_wiota_get_dcxo(void);  // v2.8新增接口
unsigned char uc_wiota_set_dcxo(unsigned int dcxo); // v3.4 新增是否配置成功的返回值
```

- 描述
  每块芯片的频偏不同，在协议栈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
  set接口返回结果：v3.4 新增是否配置成功的返回值

无。

- 参数
  dcxo：频偏。 dcxo_index 范围为0~63（十进制），dcxo = (dcxo_index << 12); 典型值为0x20000（16进制），即dcxo_index为32
- 注意
  在协议栈初始化之后，启动之前调用，否则无法生效。

10.2 设置有源晶体

- 目的
  设置有源晶体，查询有源晶体。
- 语法

```
void uc_wiota_set_is_osc(unsigned char is_osc);
unsigned char uc_wiota_get_is_osc(void);
```

- 描述
  硬件如果是有源晶体，需要设置为有源晶体。此项设置与DCXO设置互斥，如果设置了有源晶体，就不能再设置DCXO。第二个函数获取第一个函数设置下去的值。
- 返回值
  无。
- 参数
  is_osc：是否有源晶体。0: 设置非有源晶体， 1: 设置有源晶体。如果设置大于1，内部也处理成有源晶体。

10.3 设置外部32K晶振模式

- 目的
  设置外部32K晶振。模组硬件如果有外部32K晶振，可根据需要设置为外部32K晶振模式.
- 语法

```
void uc_wiota_set_outer_32K(unsigned char is_open);
```

- 描述
  msp; 模组硬件如果有外部32K晶振，可根据需要设置为外部32K晶振模式。
- 返回值
  无。
- 参数
  is_open：是否使用外部32K晶振，默认为0时，使用的是内部32K晶振。
- 注意
  可在系统启动后配置，与协议栈运行与否无关。

10.4 获取校准参数

- 目的
  获取校准参数。 v2.8新增接口。
- 语法

```
void uc_wiota_get_adjust_result(unsigned char mode, signed char* temp, unsigned char* dir, unsigned int* offset);
unsigned char uc_wiota_get_adjust_result(unsigned char mode, signed char *temp, unsigned char *dir, unsigned int *offset);  // V3.4版本新增返回值，表示校准参数是否有效
```

- 描述
  msp; 获取校准参数。

- 返回值
  temp：温度校准偏移
  dir：校准偏移方向，1：正，2：负
  offset：校准偏移量
- 参数
  mode：0，dcxo晶体，1，tcxo晶体。

10.5 设置是否使用温度计算dcxo

- 目的
  设置是否使用温度计算dcxo。 v3.0新增接口。
- 语法

```
void uc_wiota_set_is_use_temp(unsigned char is_use_temp);
unsigned char uc_wiota_set_is_use_temp(unsigned char is_use_temp); // v3.4版本新增返回值，表示是否设置成功
unsigned char uc_wiota_get_is_use_temp(void);
```

- 描述
  msp; 设置是否使用温度计算dcxo。dcxo模组，每次同步完成或者同步失败，都会重新计算初始dcxo值，一种是使用默认校准dcxo值，是量产时在室内温度下测出的dcxo值，一种是根据温度曲线和当前温度计算出当前的dcxo值，但是读取温度每次会耗时16ms，如不需要根据温度计算dcxo，可通过该接口关闭。
- 返回值
  is_use_temp：是否使用温度计算dcxo
- 参数
  is_use_temp：是否使用温度计算dcxo

11. 连接态时间

11.1 设置终端连接态时间

- 目的
  设置终端接入后连接态保持的时间。
- 语法

```
void uc_wiota_set_active_time(unsigned int 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。系统配置中symbol_length为0/1/2/3时默认连接时间是(2/3/4/8)*(下行group数)秒，也就是说ap侧应用层在收到终端接入后，在该时间内下发下行数据，不需要再走寻呼流程。
- 返回值
  无。
- 参数
  active_s：连接态时间，单位秒。
- 注意
  1. 需要跟AP侧同步设置，否则终端状态会不同步。 默认设置已经匹配。

2. 当配置系统配置后(即调用uc_wiota_set_system_config)，连接态时间会恢复成默认值！ 需要重新配置连接态时间！

11.2 获取终端连接态时间

- 目的
  获取终端接入后保持连接态的时间。
- 语法

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

- 描述
  同上。
- 返回值
  active_s，单位秒。
- 参数
  无。
- 注意
  无。

12. 低功耗相关

12.1 开关gating省电模式

- 目的
  开关gating省电模式。
- 语法

```
void uc_wiota_set_is_gating(unsigned char is_gating, unsigned char
is_phy_gating);
```

- 描述
  设置gating开关标志。
- 返回值
  无。
- 参数
  is_gating：0，关闭gating功能；1，打开gating功能。
  is_phy_gating：0，关闭物理层gating功能；1，打开物理层gating功能。
- 注意
  该功能在协议栈开启时才有效，需初始化协议栈之后再打开该功能，关闭协议栈则自动关闭gating功能。

12.2 设置中断唤醒源

- 目的
  设置gating省电模式下的中断唤醒源。
- 语法

```
void uc_wiota_set_gating_event(unsigned char action, unsigned char event_id);
```

- 描述
  设置唤醒源。

- 返回值
  无。
- 参数
  action：0，清除该event_id唤醒源；1，设置该event_id唤醒源。
  event_id：对应于中断向量表，将某一个中断作为唤醒源，参考代码interrupt_handle.c。
- 注意
  (1) 该接口在协议栈开启时才有效，需初始化协议栈之后再设置。
  (2) 不支持修改和配置event_id为0/1/23/24/29的唤醒源，分别为RTC/CCE/UART0/UART1/SYSTIMER。

12.3 设置闹钟

- 目的
  设置闹钟定时，定时触发RTC中断。
- 语法

```
void uc_wiota_set_alarm_time(unsigned int sec);
```

- 描述
  设置闹钟定时，例如定时20秒后，会触发RTC中断，在sleep之前设置，可在sleep之后唤醒系统。
- 返回值
- 参数
  sec：定时时间，秒
- 注意

12.4 进入sleep模式

- 目的
  进入sleep模式
- 语法

```
void uc_wiota_sleep_enter(unsigned char is_need_ex_wk, unsigned char
is_need_32k_div);
```

- 描述
  系统进入sleep
- 返回值
  无。
- 参数
  is_need_ex_wk：0，不需要外部唤醒源；1，需要外部唤醒源。
  外部唤醒源一般是指串口，当前硬件，串口悬空是不定态，必须关闭外部唤醒源，即is_need_ex_wk设为0，否则可能无法真正进入sleep
  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，可降低sleep时0.3uA左右的电流。
  降低32K时钟频率会导致32K定时不准。
- 注意
  不需要关闭协议栈再sleep，可以直接sleep

12.5 paging接收检测

- 目的
  进入paging接收检测模式，降低终端功耗。
- 语法

```
void uc_wiota_set_paging_rx_cfg(uc_lpm_rx_cfg_t *config);
void uc_wiota_get_paging_rx_cfg(uc_lpm_rx_cfg_t *config);
void uc_wiota_paging_rx_enter(unsigned char is_need_32k_div);
void uc_wiota_get_awaken_id_limit(unsigned char symbol_length);
```

v2.7版本更新，增加最大次数后强制醒来配置：

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

v3.0版本更新，增加set cfg的返回值，判断是否设置成功：

```
unsigned char uc_wiota_set_paging_rx_cfg(uc_lpm_rx_cfg_t *config);
```

- 描述
  进入超低功耗检测模式。
- 返回值
  无。
- 参数
  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。（v2.9版本更新）
  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。(PS: v2.7及之前版本，extra_period均不能小于等于 （detect_period + 10）ms，v2.8版本已修复)
  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。（v2.9版本新增）
  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，倍数的最大值限制为255，一般不建议设置太大。（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。（v3.1版本新增参数）

- 结构体

```
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;  // v2.5更新参数，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; // v2.5新增参数，ms, extra new period before
wake up
    unsigned char mode; // v3.1修改，将v2.9新增的reserved1改成mode，0: old id
range(narrow), 1: extend id range(wide)
    unsigned char period_multiple;    // v2.9新增，the multiples of detect_period
using awaken_id_ano, if 0, no need
    unsigned short awaken_id_another; // v2.9新增，another awaken_id
  }uc_lpm_rx_cfg_t;
```

- 注意
  （1） 初始化协议栈之后，如果不设置，则为内部缺省值。
  （2） 与系统配置类似，如果不想改变默认值，则先get，再修改，最后set。
  （3） 进入paging rx模式前，也可以设置闹钟进行定时唤醒，与进sleep前配置类似，但是不建议用此方法，目前可能会出现异常，建议配置最大检测次数来定时强制唤醒。
  （4） paging rx模式下不能打开外部唤醒寄存器，只能通过拉低spi cs引脚来强制唤醒。
  （5） 扩展ID模式，第二个唤醒ID仅支持与第一个唤醒ID相同周期，进休眠不能32K时钟降频，paging tx的send time必须与paging rx的detect time相同！

12.6 获取被唤醒原因

v2.6版本新增接口。

- 目的
  获取唤醒前状态和唤醒原因。
- 语法

```
unsigned char uc_wiota_get_awakened_cause(unsigned char *is_cs_awakened); //
UC_AWAKENED_CAUSE
```

- 描述
  获取唤醒前状态和唤醒原因。
  在系统启动后，main函数里可调用该接口，获取当前系统启动是从什么情况下启动的，可能是之前还在paging或者在sleep，或者是硬件复位，watchdog复位，或者spi cs拉低复位，特别的，如果是spi cs拉低复位，还有is_cs_awakened标志位。
  该接口无法判断出在paging下是被rtc时钟唤醒，还是被paging tx信号唤醒，统一为AWAKENED_CAUSE_PAGING。
  同样，该接口无法判断出在sleep下是被rtc时钟唤醒，还是被外部唤醒源唤醒，统一为AWAKENED_CAUSE_SLEEP。

- 返回值

```
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;
```

- 参数
  is_cs_awakened：出参（返回值）， 是否被spi cs唤醒。
- 注意
  与sleep不同的是，在paging下无法使用外部唤醒源唤醒，只能使用spi cs拉低的方式唤醒。

12.7 获取paging唤醒原因

v2.7版本新增接口。

- 目的
  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。
- 语法

```
unsigned char uc_wiota_get_paging_awaken_cause(void); //
UC_LPM_PAGING_WAKEN_CAUSE_E
```

v2.8版本接口中新增返回值如下：

```
unsigned char uc_wiota_get_paging_awaken_cause(unsigned int* detected_times); //
UC_LPM_PAGING_WAKEN_CAUSE_E
```

v2.9版本接口中新增返回值如下：

```
unsigned char uc_wiota_get_paging_awaken_cause(unsigned int* detected_times,
unsigned char *detect_idx); // UC_LPM_PAGING_WAKEN_CAUSE_E
```

- 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。
- 返回值

```
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;
```

- 参数
  detected_times：v2.8版本新增返回参数，用于获取当前被唤醒时的检测次数。
  detect_idx：v2.9版本新增返回参数，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。
- 注意
  无

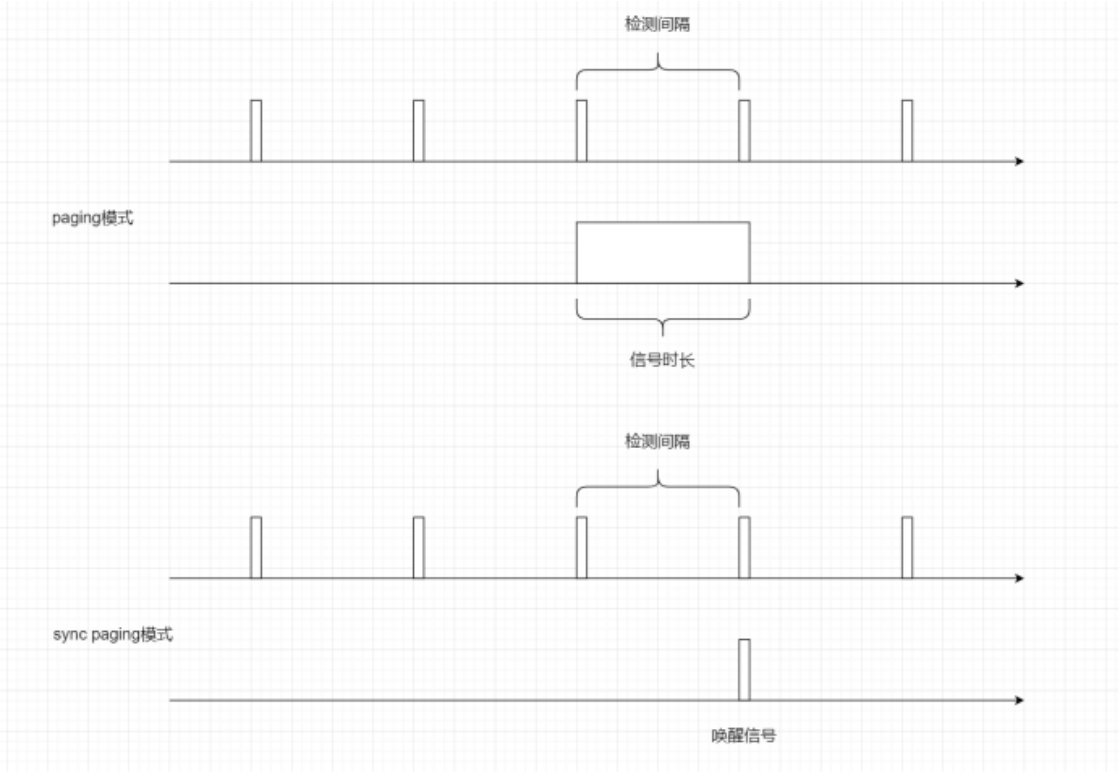
12.8 sync paging接收检测

v2.8版本新增接口。

- 目的
  进入sync paging接收检测模式，降低终端功耗。该模式精度要求较高，需要使用接口uc_wiota_set_outer_32K配置外部32K晶振模式。
- 语法

```
void uc_wiota_sync_paging_enter(u8_t mode, u32_t period, u32_t times, u32_t timeout_max);
```

- 描述
  模式0时，times和timeout_max有效，举例times为9，timeout_max为100，则每次间隔（9+1）*frame_len时长，基带醒来检测一次；
  当基带第100次醒来后，会强制唤醒系统，此时通过uc_wiota_get_paging_awaken_cause获取到的原因为PAGING_WAKEN_CAUSE_SYNC_PG_TIMEOUT；
  如果在第15次基带检测到信号并唤醒系统，则获取到原因为PAGING_WAKEN_CAUSE_SYNC_PG_SIGNAL；
  模式1时，period和timeout_max有效，每隔period时长，基带会起来计数并立即再次休眠，当计数达到timeout_max时，会强制唤醒系统，此时获取到paging原因为PAGING_WAKEN_CAUSE_SYNC_PG_TIMING。
  模式1与进入sleep并且使用rtc唤醒的情况类似，但是定时比rtc更加精准，也能明确知道唤醒原因。timeout_max次数一般设为1，timeout_max越大，误差越大。
  模式2与模式0类似，区别在于，模式0会先发一个上行控制包给网关，告诉网关马上要睡的帧号，确认发送成功后再真正进入休眠，此时网关可以根据休眠间隔和进入休眠的帧号，精确计算出之后需要在具体哪一帧发送唤醒信号；而模式2，终端会直接在下一帧进入休眠，如果检测间隔是N帧，则网关如果需要唤醒该终端，则必须在连续N+1帧都发送唤醒信号。
  v2.9版本将原来的模式0换成了模式2，同时改变了现在模式0的含义，需要网关使用对应的v2.9版本。
  sync paging和paging超低功耗唤醒模式的区别，如下图所示：



- 返回值
  无。
- 参数
  mode：唤醒模式，如果为0或2，则为正常sync paging模式，如果为1，则直接进入sleep。
  period：检测周期，单位为微妙(us)，最大值为44秒，最小值20ms，如果设为0，则为当前帧长。mode为0时该值无效，内部默认为帧长。
  times：mode为0时的检测间隔次数，如果设为0，则代表每隔1帧长基带醒来检测一次。mode为1时该值无效。
  timeout_max：最大检测次数，到达最大检测次数，不管有没有检测到信号，都会强制唤醒系统。
- 结构体
  无
- 注意
  （1） 初始化协议栈之后，如果不设置，则为内部缺省值。
  （2） 进入sync paging前，也可以设置闹钟进行定时唤醒，与进sleep前配置类似，但是不建议用此方法，目前可能会出现异常，建议配置最大检测次数来定时强制唤醒。
  （3） 模式0时，（times+1）*frame_len 不能大于44秒！
  （4） 需要使用uc_wiota_set_outer_32K配置外部32K晶振。
  （5） 不能打开外部唤醒寄存器，只能通过拉低spi cs引脚来强制唤醒。

12.9 设置外部唤醒

- 目的
  设置外部唤醒。 v2.8新增接口。
- 语法

```
void uc_wiota_set_is_ex_wk(boolean is_need_ex_wk);
```

- 描述
  设置外部唤醒。
- 返回值
  无。

- 参数
  is_need_ex_wk：0，不需要外部唤醒源；1，需要外部唤醒源。
  外部唤醒源一般是指串口，当前硬件，串口悬空是不定态，必须关闭外部唤醒源，即is_need_ex_wk设为0，否则可能无法真正进入sleep
- 注意

12.10 设置降压模式

- 目的
  降压模式设置
- 语法

```
void uc_wiota_set_vol_mode(unsigned char vol_mode);
```

- 描述
  设置是否降压。
- 返回值
  无。
- 参数
  div_mode：0，默认电压（1.82v）；1，降压（1.56v）。

```
typedef enum {
  VOL_MODE_CLOSE = 0,  // 1.82v
  VOL_MODE_OPEN = 1,   // 1.56v
  VOL_MODE_TEMP_MAX,
} uc_vol_mode_e;
```

- 注意
  该接口在不启动协议栈时使用。

12.11 设置降频

- 目的
  降频分频比设置
- 语法

```
void uc_wiota_set_freq_div(unsigned char div_mode);
```

- 描述
  设置降频分频比
- 返回值
  无。
- 参数

```
typedef enum {
  FREQ_DIV_MODE_1 = 0,  // default: 96M
  FREQ_DIV_MODE_2 = 1,  // 48M
  FREQ_DIV_MODE_4 = 2,  // 24M
  FREQ_DIV_MODE_6 = 3,  // 16M
  FREQ_DIV_MODE_8 = 4,  // 12M
  FREQ_DIV_MODE_10 = 5, // 9.6M
  FREQ_DIV_MODE_12 = 6, // 8M
  FREQ_DIV_MODE_14 = 7, // 48/7 M
  FREQ_DIV_MODE_16 = 8, // 6M
  FREQ_DIV_MODE_MAX,
} uc_freq_div_mode_e;
```

- 注意
  该接口在不启动协议栈时使用，降频后系统运行速率会变慢。

12.12 paging发送唤醒

V3.1版本新增接口，V3.0版本以及之前暂不支持。

- 目的
  唤醒处于paging检测模式下的AP。
  只能在IDLE态发送唤醒信号，即未connect或者diconnect之后。
- 语法

```
void uc_wiota_paging_tx_start(void);
unsigned char uc_wiota_set_paging_tx_cfg(uc_lpm_tx_cfg_t *config);
void uc_wiota_get_paging_tx_cfg(uc_lpm_tx_cfg_t *config);
```

- 描述
  设置并发送唤醒信号。
- 返回值
  tx配置是否成功返回值。
- 参数
  config：配置结构体指针
  freq：频点
  spectrum_idx： 频谱idx，与系统配置相同
  symbol_length：取值0,1,2,3代表128,256,512,1024
  awaken_id：唤醒ID，根据symbol length不同，最大值不同，当mode为0时，最大值限制为[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！）
  send_time：最小值为接收端检测周期（单位ms），如果小于周期，可能无法唤醒，使用扩展ID模式时，该时间必须与paging rx的detect_time相同。
  bandwidth：暂时只支持200K带宽，即数值1
  mode：为0时，为默认模式，mode为1时，为扩展ID模式，ID范围更大。
- 结构体

```
typedef struct
{
  unsigned char freq;
  unsigned char spectrum_idx;
  unsigned char bandwidth;
  unsigned char symbol_length;
  unsigned short awaken_id; // indicate which id should send
  unsigned char mode;       // 0: old id range(narrow), 1: extend id
range(wide)
  unsigned char reserved;   // re
  unsigned int send_time;   // ms, at least rx detect period
} uc_lpm_tx_cfg_t, *uc_lpm_tx_cfg_p;
```

- 注意
  （1） 初始化协议栈之后，如果不设置，则为内部缺省值。
  （2） 与系统配置类似，如果不想改变默认值，则先get，再修改，最后set。
  （3） 扩展ID模式，第二个唤醒ID仅支持与第一个唤醒ID相同周期，进休眠不能32K时钟降频，paging tx的send time必须与paging rx的detect time相同！

12.13 唤醒输出电平初始化

- 目的
  本终端被唤醒后输出高电平，通知上位机已成功唤醒。
- 语法

```
int wake_out_pulse_init(void);
```

- 描述
  本终端被唤醒后，通过GPIO输出电平信号，通知上位机，默认为`GPIO7`，当被唤醒时输出高电平，并保持设定的脉宽后输出低电平。GPIO及电平脉宽可通过静态数据或AT指令（AT+WAKEOUTPIN）配置。
- 返回值
  0 成功，-1 失败。
- 参数

  无。
- 注意
  本函数于同步3.0和异步3.03后在main函数中初始化。

13. 数据收发

13.1 设置数据传输速率

- 目的
  根据应用需求设置数据传输速率。
- 语法

```
void uc_wiota_set_data_rate(unsigned char rate_mode, unsigned short
rate_value);
unsigned char uc_wiota_set_data_rate(unsigned char rate_mode, unsigned short
rate_value); // v3.4新增返回值，表示是否设置成功
```

V3.0版本新增功能。

```
typedef struct
{
  unsigned short data_len;
  unsigned char is_right; // for recv data, not use now
  unsigned char reserved;
  unsigned char *data;
} uc_subf_data_t, *uc_subf_data_p;

unsigned char uc_wiota_add_subframe_data(uc_subf_data_p subf_data);
unsigned int uc_wiota_get_subframe_data_num(void);
void uc_wiota_set_subframe_data_limit(unsigned int num_limit);
```

- 描述
  设置最大速率模式和级别，三种模式，与枚举UC_DATA_RATE_MODE里对应。
  (1) 第一种基本模式UC_RATE_NORMAL，是基本速率设置，有9档mcs速率级别（包括自动mcs），详见UC_MCS_LEVEL，默认为自动mcs，设置非自动mcs时同时关闭自动速率匹配功能。
  (2) 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UC_RATE_MID，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。
  (3) 在第一种模式的基础上，打开第三种模式UC_RATE_HIGH，能够提升（SUBFRAME_NUM_IN_GOUP*(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。
  (4) 在第一种模式的基础上，打开子帧发送模式，会在子帧队列中有数据时，并且没有正常上行数据时，发送子帧数据，子帧模式为1时，每帧发送1个子帧数据，子帧模式为2时，并且在系统配置中dlul_ratio为1:2时，每帧发送2个子帧数据。（V3.0版本新增功能）

  备注：group_number为系统配置中的参数。
  备注：上述（3）中的SUBFRAME_NUM_IN_GOUP为8
- 返回值
  无。
- 参数
  rate_mode：枚举UC_DATA_RATE_MODE。
  rate_value： 当rate_mode为UC_RATE_NORMAL时，rate_value为枚举UC_MCS_LEVEL。
  当rate_mode为UC_RATE_MID时，rate_value为0或1，表示关闭或打开。
  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。

```
typedef enum {
  UC_RATE_NORMAL = 0,
  UC_RATE_MID,
  UC_RATE_HIGH,
  UC_RATE_SUBFRAME, // subframe mode， v3.0新增
}UC_DATA_RATE_MODE;

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_AUTO = 8,
}UC_MCS_LEVEL;
```

BT_0.3时在不同symbol length和不同MCS时，对应每帧传输的应用数据量（byte）。
(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	不支持	不支持	不支持
256	6	14	21	51	107	156	191	不支持
512	6	14	30	41	72	135	254	296
1024	6	14	30	62	107	219	450	618

初始化协议栈时默认打开自动速率匹配功能，调用该接口入参为0~7时，设置最大速率级别，同时关闭自动速率匹配功能，再次调用该接口入参为UC_MCS_AUTO（或者不是0~7）时，会打开自动速率匹配功能。
为了保证接入成功率，接入短消息暂只使用mcs0~3，由于其中需要携带user id，正常会再减去4个字节空间，实际给应用的数据量会比正常短消息少。
接入短消息的MCS还有其他限制（应用层可不关注），symbol length为128/256/512/1024时，接入短消息的MCS最高分别为1/2/3/3。

每帧时间长度（frameLen）的粗略计算表格（单位微妙，该表格并不绝对准确）：
计算公式暂不公开，如需要可使用接口uc_wiota_get_frame_len获取（v0.13版本及之后提供该接口）

dlul_ratio	group_number	symbol_length	frameLen(us)
0	0	0	73216
0	0	1	146432
0	0	2	292864
0	0	3	585728
0	1	0	138752
0	1	1	277504
0	1	2	555008
0	2	0	269824
0	2	1	539648
0	3	0	531968
1	0	0	105984
1	0	0	211968
1	0	0	423936

dlul_ratio	group_number	symbol_length	frameLen(us)
1	0	0	208576
1	0	0	408576
1	0	0	400896

举例： 系统配置中group_number为0，dlul_ratio为0，symbol_length为1，则frameLen为146432 us

在此帧结构配置情况下，如果选择MCS2，则应用数据速率为 8*21/0.146432 = 1147 bps
(计算上行数据速率时，一般不考虑第一个包即随机接入包)。

- 注意
  一味提高速率，可能导致上行始终无法成功。

13.2 发送数据

- 目的
  发送数据给ap。
- 语法

```
UC_OP_RESULT uc_wiota_send_data(unsigned char* data, unsigned short len,
unsigned short 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。
  len：数据长度，数据最长为310字节。
  callback：回调函数，非阻塞时处理返回结果。
  timeout：超时时间，单位ms，范围1~65534ms
- 结构体

```
typedef struct {
  unsigned int    result;
  unsigned char*  oriPtr;
}uc_send_back_t,*uc_send_back_p;

typedef void (*uc_send)(uc_send_back_p send_result);
```

result：返回结果，UC_OP_RESULT。
oriPtr：返回原数据的地址，方便应用确认对应数据。

- 注意
（1） 在收到返回结果之前不能释放data内存，并且需要预留2字节的空间给底层CRC使用，比如数据len为101，则申请data_buffer大小为103，可参考at_wiotasend_setup的代码实现。
（2） 不能在callback函数里释放内存。
（3） 数据最长为310字节，数据超过310将被丢掉。如果应用层需要传超过310字节的数据，建议自己先分包。
（4） 在返回结果之前（包括回调函数结果之前），该函数不支持连续调用，否则会直接返回UC_OP_FAIL（回调函数也一样）。

13.3 被动接收数据接口注册

- 目的
被动接收数据。
- 语法

```
void uc_wiota_register_recv_data_callback(uc_recv
callback,UC_CALLBACK_DATA_TYPE type);
```

- 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（包括广播、OTA，即UC_RECV_DATA_TYPE中前6种消息）时，会调用该回调函数上报数据，接收UC_RECV_MSG时，才可能出现result为UC_OP_FAIL，此时表示CRC错误，同时data为NULL。
注册一个接收协议栈状态信息的回调函数，只需要系统启动后注册一次即可，目前只有2种状态信息（即UC_RECV_DATA_TYPE中的UC_RECV_SYNC_LOST和UC_RECV_IDLE_PAGING）。结果中只有type有效，其余参数均为0。
- 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;  // UC_OP_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_CALLBACK_NORAMAL_MSG, normal msg from ap
    UC_RECV_BC,         // UC_CALLBACK_NORAMAL_MSG, broadcast msg from ap
    UC_RECV_OTA,        // UC_CALLBACK_NORAMAL_MSG, ota msg from ap
    UC_RECV_MULT0,      // UC_CALLBACK_NORAMAL_MSG, multcast0 msg from ap
    UC_RECV_MULT1,      // UC_CALLBACK_NORAMAL_MSG, multcast1 msg from ap
    UC_RECV_MULT2,      // UC_CALLBACK_NORAMAL_MSG, multcast2 msg from ap
    UC_RECV_SCAN_FREQ, // UC_CALLBACK_NORAMAL_MSG, result of freq scan by
riscv
    UC_RECV_SYNC_LOST, // UC_CALLBACK_STATE_INFO, sync lost notify by riscv,
need scan freq
```

```
    UC_RECV_IDLE_PAGING, // v2.8新增 UC_CALLBACK_STATE_INFO, when idle
state, recv ap's paging signal
    UC_RECV_VOICE = 9,       // v3.0新增 UC_CALLBACK_NORAMAL_MSG, voice msg
from ap
    UC_RECV_PG_TX_DONE = 10, // v3.1新增 UC_CALLBACK_STATE_INFO, when pg tx
end, tell app
    UC_RECV_FN_CHANGE = 11,  // v3.2新增 UC_CALLBACK_NORAMAL_MSG, when frame
num change, tell app
    UC_RECV_PHY_ERROR = 12,  // v3.4新增 UC_CALLBACK_STATE_INFO, if phy not
response timeout, tell app
}uc_recv_data_type_e;

typedef enum {
  UC_CALLBACK_NORMAL_MSG = 0,
  UC_CALLBACK_STATE_INFO,
  UC_CALLBACK_INTERRUPT_HANDLER, // v3.0新增 when rf interrupt come, will
check app handler
  UC_CALLBACK_TEMPERATURE,       // v3.4新增 use app's func to read
temperature
  UC_CALLBACK_PA_CONTROL,        // v3.4新增 pa control, return
uc_recv_back_p, result: 1 open pa, 2, close pa
}uc_callback_data_type_e;

typedef void (*uc_recv)(uc_recv_back_p recv_data);
```

13.4 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, unsigned short 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。
  该回调函数与被动接收的回调注册函数不冲突，应用可根据自身需求设置。
- 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。

13.5 设置CRC校验开关

- 目的
  设置CRC校验开关。
- 语法

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

- 描述
  开关协议层的CRC，并设置校验长度的标准。
  如果crc_limit为0，表示关闭CRC校验功能。
  如果crc_limit大于0，表示数据长度大于等于crc_limit时，才打开CRC校验功能，所以crc_limit设置为1，则可表示任意长度的数据均加CRC。
- 返回值
  无。
- 参数
  crc_limit：校验长度限制。
- 注意
  终端和AP的crc_limit设置需要一致！

13.6 设置组播ID列表

- 目的
  初始化之后设置组播ID，系统启动后会自动开始组播接收。
- 语法

```
void uc_wiota_set_multcast_id_list(unsigned int *multcast_id_list);
```

- 描述
  设置0~3个组播ID，系统启动后会自动接收该ID对应的组播数据。
  组播ID和userID类似，也是32bit长度。
- 返回值
  无。
- 参数
  multcast_id_list：数组指针头，数组个数只能为3，编号依次为0/1/2，如果不需要，对应id设为0即可。
- 注意
  如果设置为0，则代表取消该编号的组播ID配置。

13.7 设置上行数据包重发次数

- 目的
  协议分包后，每包发送时，默认重发尝试次数为4，达到该次数后，认为发送失败，停止发送并上报给上层。 可配置该发送次数。 v2.8版本新增接口。
- 语法

```
u8_t uc_wiota_get_sm_resend_times(void);
void uc_wiota_set_sm_resend_times(u8_t resend_times);
```

- 描述
  设置上行数据包重发次数。
- 返回值
  无。

- 参数
  resend_times：重发次数，默认值为4。可配置最小值为0，表示发送失败一次后不重发，最大不建议超过6次。
- 注意

13.8 计算CRC

- 目的
  计算CRC函数。（v3.0新增接口）
- 语法

```
#define CRC8_LEN 1
#define CRC16_LEN 2
#define CRC32_LEN 4

unsigned char uc_wiota_crc8_calc(unsigned char *data, unsigned int
data_len);
unsigned short uc_wiota_crc16_calc(unsigned char *data, unsigned int
data_len);
unsigned int uc_wiota_crc32_calc(unsigned char *data, unsigned int
data_len);
```

- 描述
  计算不同长度的CRC。不支持修改多项式和初始值。
- 返回值
  CRC。
- 参数
  data：数据指针。
  data_len：数据长度。
- 注意
  无。

13.9 计算随机数

- 目的
  计算随机数，伪随机数。（v3.0新增接口）
- 语法

```
#define UC_ALGO_MULTIPLIER 0x15a4e35
#define UC_ALGO_INCREMENT 0x1

void uc_wiota_algo_srand(unsigned int seedSet);
unsigned int uc_wiota_algo_rand(void);
```

- 描述
  计算伪随机数。
- 返回值
  随机数，32位。

- 参数
  seedSet：初始设置种子，如果种子一样，则获取的随机数是一样的，实现原理与C语言中的rand函数一致。
- 注意
  无。

13.10 数据暂存初始化

- 目的
  用于启用数据暂存功能初始化。
- 语法

```
int at_wiota_gpio_report_init(void);
```

- 描述
  用于启用数据暂存功能初始化。该功能启用后，当连接本模块的MCU或上位机处于休眠状态，可通过本函数指定的GPIO作为唤醒信号（默认GPIO2），数据暂存默认可以存5条WIOTA数据。GPIO及电平脉宽可通过静态数据或AT指令（AT+PULSEWIDTH）配置。
- 返回值
  0 成功，-1 失败。
- 参数
  无。
- 注意
  本函数于同步3.0和异步3.03后在main函数中初始化。

14. 时间相关

14.1 获取当前rf cnt

- 目的
  获取当前rf cnt
- 语法

```
unsigned int uc_wiota_get_curr_rf_cnt(void);
```

- 描述
  获取当前rf cnt
- 返回值
  curr rf cnt：单位us，重新初始化协议栈时，rf会重置，rf会重新从0开始，范围32bit，大约4295秒也会翻转。
- 参数
  无。
- 结构体
  无。

14.2 获取帧头时间

- 目的
  获取帧头和帧中下行上行切换时间点。 （v3.1新增）
- 语法

```
void uc_wiota_get_frame_head_rf_cnt(uc_frame_head_p frame_head_info);
typedef struct
{
  unsigned char is_valid;
  unsigned char reserved0;
  unsigned short reserved1;
  unsigned int frame_head;
  unsigned int frame_mid;
} uc_frame_head_t, *uc_frame_head_p;
```

- 描述
  获取当前帧的帧头，根据帧长可计算出下一帧或下下帧的帧头时刻，之后再使用uc_wiota_get_curr_rf_cnt查询，可设timer准确定时到想要的帧头时间，完成例如控制LNA等射频控制的需求。
- 返回值
  is_valid：时间是否有效
  frame_head：当前帧的帧头rf cnt
  frame_mid：当前帧的下行和上行切换的rf cnt
- 参数
  无
- 结构体
  参见上述接口。

14.3 获取世界时间

- 目的
  获取世界时间（gps）
- 语法

```
void uc_wiota_get_gps_info(uc_gps_time_p gps_info);
typedef struct
{
  unsigned char is_valid;   // if info valid
  unsigned char reserved0;  // re
  unsigned short reserved1; // re1
  unsigned int gps_time_us; // gps time us remainder of second, 0~999999 us
  unsigned int gps_time_s;  // gps time second
  unsigned int rf_cnt_us;   // frame head
  unsigned int rf_cnt_curr; // curr dfe
} uc_gps_time_t, *uc_gps_time_p;
```

- 描述
  世界时间与rf_cnt_us的时间戳对应，之后再使用 uc_wiota_get_curr_rf_cnt， 可换算出当前世界时间。
- 返回值
  is_valid：gps时间是否有效
  gps_time_us：gps时间的微妙 （模1秒的余数）
  gps_time_s：gps时间的秒
  rf_cnt_us：与gps对应的rf cnt时间戳，单位us
  rf_cnt_curr：当前rf cnt，一般比rf_cnt_us大

- 参数
  无
- 结构体
  参见上述接口。

15. 调试相关

15.1 设置WIoTa log开关

- 目的
  设置协议层的log开关。
- 语法

```
void uc_wiota_log_switch(unsigned char log_type, unsigned char is_open);
typedef enum {
  UC_LOG_UART = 0,
  UC_LOG_SPI,
}UC_LOG_TYPE;
```

- 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。
  is_open：是否开启该log。
- 结构体
  参见上述接口。

15.2 WIoTa统计信息获取

- 目的
  获取WIoTa的统计信息。
- 语法

```
unsigned int uc_wiota_get_stats(unsigned char type);
void uc_wiota_get_all_stats(uc_stats_info_p stats_info_ptr);
void uc_wiota_reset_stats(unsigned char type);
```

- 描述
  获取/重置/增加某个/所有统计信息的计数。
- 返回值
  uc_wiota_get_stats：返回对应type的统计计数。
  stats_info_ptr：本地统计信息表，用来获取所有统计信息。
- 参数
  type：UC_STATS_TYPE，与uc_stats_info_t的参数一一对应。
  注意，在uc_wiota_get_stats中type为0，则返回无效值0。
- 结构体

```
typedef struct {
  unsigned int rach_fail;
  unsigned int active_fail;
  unsigned int ul_succ;
  unsigned int dl_fail;
  unsigned int dl_succ;
  unsigned int bc_fail;
  unsigned int bc_succ;
  unsigned int ul_sm_succ;
  unsigned int ul_sm_total;
}uc_stats_info_t,*uc_stats_info_p;

typedef enum {
  UC_STATS_READ = 0,
  UC_STATS_WRITE,
}UC_STATS_MODE;

typedef enum {
  UC_STATS_TYPE_ALL = 0,
  UC_STATS_RACH_FAIL,
  UC_STATS_ACTIVE_FAIL,
  UC_STATS_UL_SUCC,
  UC_STATS_DL_FAIL,
  UC_STATS_DL_SUCC,
  UC_STATS_BC_FAIL,
  UC_STATS_BC_SUCC,
  UC_STATS_UL_SM_SUCC,
  UC_STATS_UL_SM_TOTAL,
  UC_STATS_TYPE_MAX,
}UC_STATS_TYPE;
```

- 结构体描述
  UC_STATS_RACH_FAIL：接入失败次数。
  UC_STATS_ACTIVE_FAIL：连接态发送失败次数。
  UC_STATS_UL_SUCC：上行发送成功次数。
  UC_STATS_DL_FAIL：下行接收失败次数（收完整段数据校验CRC错误）。
  UC_STATS_DL_SUCC：下行接收成功次数。
  UC_STATS_BC_FAIL：广播接收失败次数。
  UC_STATS_BC_SUCC：广播接收成功次数。
  UC_STATS_UL_SM_SUCC：上行短消息成功次数。
  UC_STATS_UL_SM_TOTAL：上行短消息总发送次数。

15.3 设置指示灯开关

- 目的
  开关指示灯，在二次开发版本中，可关闭指示灯，即停止协议栈对相应GPIO（2/3/7/16/17）的操作，避免冲突。
- 语法

```
void uc_wiota_light_func_enable(unsigned char func_enable);
```

- 描述
  开启或关闭协议栈运行状态及上下行数据的指示灯，默认关闭。

- 返回值
  无。
- 参数
  func_enable：开关指示灯功能。
- 注意