Pluto API for ESP8266

V1.0	2020.01.01	lort

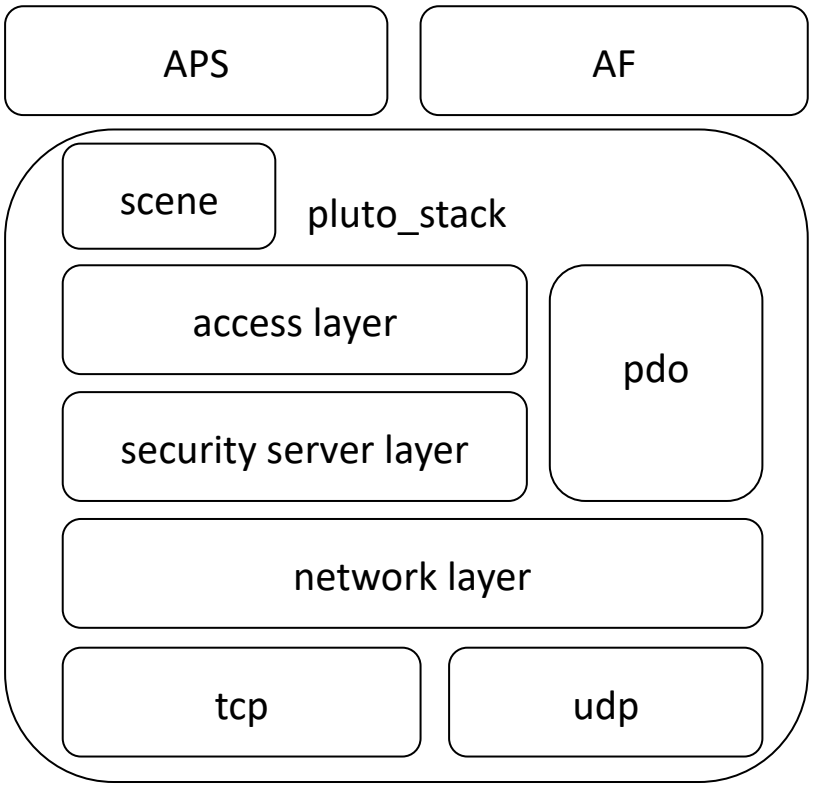
目录

一、简介

Pluto 是一个完全开放连接的物联网标准通信系统，它能实现真正意义上的万物互联。它具有低延迟、安全、可靠、使用简单等特点。

Pluto 项目包含一个开放的服务器应用程序、开放的设备端 SDK、开放的手机端 SDK 三个部分构成，任何人都可以选择基于其中的一部分或多个部分开发并应用于自己的项目中并且无需费用。

二、框架组成



三、项目模块构成

目录与来源：

1， ESP8266_RTOS_SDK：https://github.com/espressif

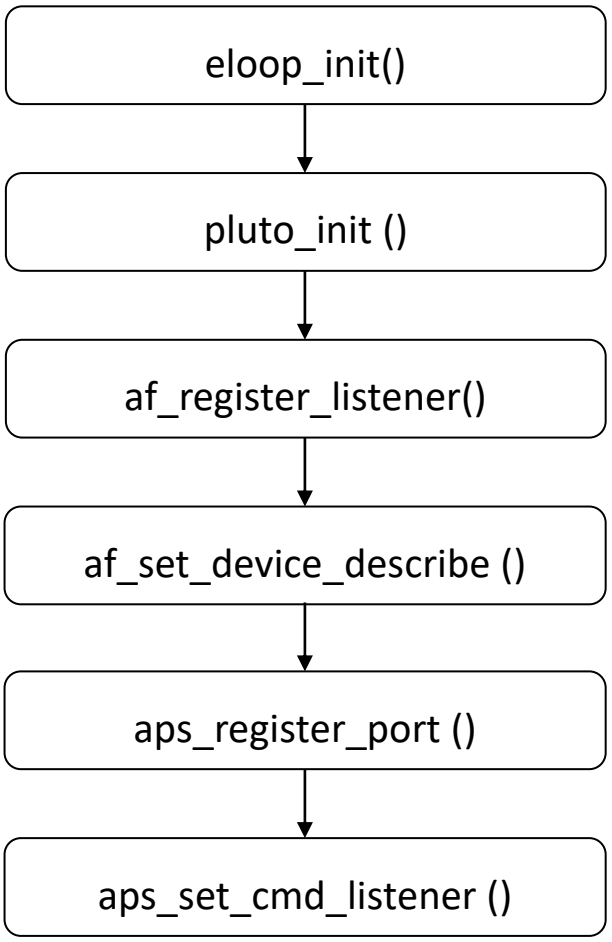
2， eloop:https://github.com/bingoiot

3， pluto_stack & app

依赖的模块:

FREE_RTOS，pthread，spiffs， nvs， eloop，pluto

四、初始化流程



eloop 内核初始化

Pluto 服务初始化

注册 application framework 回调函数

设置设备描述信息，用于识别设备等操作

设置端口描述信息，用于识别逻辑设备数量，逻辑设备的操作方式，每个 port 都表示一个逻辑设备

设置命令接收函数，发送状态回调函数

五、eloop API 描述

I、eloop 移植相关 API

1、void eloop_update_tick(u_int32 tick);

描述：更新系统时钟，单片机或系统在每个节拍需要更新一次 eloop 时钟以维持 eloop 定时任务的正常运行。仅在移植时处理，用户层无需调用。

Parameter :

tick: 系统在上次调用到本次调用的实际时间间隔，单位 ms。

Return : none。

2、s_int8 eloop_task_poll(void);

描述：循环调度并处理可运行的任务。仅在移植时处理，用户层无需调用。

Parameter : none

Return : ES_TASK_IDLE：空闲， ES_TASK_BUSSY: 任务繁忙。

II、eloop 核心 api

1、int eloop_init(void)

描述：系统初始化。

Parameter : none

Return : 0 表示成功，其它表示失败。

2、void eloop_enter_critical(void)

描述：进入临界区。

Parameter : none

Return : none。

3、void eloop_exit_critical (void)

描述：退出临界区。

Parameter : none

Return : none。

4、void　eloop_sleep(uint32 tim)

描述：使用内核进入休眠，进入休眠后所有 eloop 创建的任务都将暂停。

Parameter :

tim: 休眠时间，单位：ms

Return : none。

5、void　eloop_set_power_conserve(u_int8 enable)

描述： 使能低功耗模式，在功耗模式下，任务处理完成之后回调注册的休眠函数，并给出允许休眠的时间让芯片进入休眠。

Parameter :

enable: 1 全能低功耗模式，0 禁止低功耗模式。

Return : none。

6、s_int8　eloop_create_task(eloop_task_t task,u_int8 priority);

描述：创建任务，不经过创建的任务，所有与任务相关的操作将失败。

Parameter :

task: 任务函数名。

Priority: 任务优先级，最高 254，最低 1，0 无效。255 预留给系统定时器。

Return : 0 成功，其它失败。

7、s_int8　　eloop_delete_task(eloop_task_t task);

描述：删除任务。

Parameter :

task: 任务函数名。

Return : 0 成功，其它失败。

8、void*　eloop_malloc(s_int32 size);

描述：内存分配。

Parameter :

size: 内存大小，单位：字节。

Return : 非 0 指针：成功，返回 NULL：失败。申请的内存需要用 eloop_free()释放。

9、void　eloop_free(void* p);

描述：释放分配的内存。

Parameter :

p: 内存首地址。

Return : none。

10、s_int32 eloop_get_free_heap(void);

描述：获取剩余可用内存大小。

Parameter : none

Return : 可用内存大小，单位：字节。

III、eloop 事件与消息

1、s_int8 eloop_set_event(eloop_task_t task, u_int32 event);

描述：给指定任务通知事件发生。

Parameter :

task: 任务函数名。

event: 事件掩码。按位操作，每个任务最仅只有 32 个整件。

Return : ES_FAILED：失败， ES_SUCCEED: 成功。

2、s_int8 loop_clear_event (eloop_task_t task, u_int32 event);

描述：清除指定任务的事件。

Parameter :

task: 任务函数名。

event: 事件掩码。按位操作，每个任务最仅只有 32 个整件。

Return : ES_FAILED：失败， ES_SUCCEED: 成功。

3、s_int8 eloop_list_insert(void **list, void *msg);

描述： 消息插入链表之后。

Parameter :

list： 表头指针。

msg: 消息指针。

Return : ES_SUCCEED 成功。

4、s_int8 eloop_list_insert_front(void **list, void *msg);

描述： 消息插入链表前头。

Parameter :

list： 表头指针。

msg: 消息指针。

Return : ES_SUCCEED 成功。

5、void * eloop_list_detach(void **list);

描述： 分离链表最后面的消息。

Parameter :

list： 表头指针。

Return : 非 0 指针：分离出的消息，NULL 分离失败。分离出的消息需要 free()释放。

6、void * eloop_list_detach_front (void **list);

描述： 分离链表最前面的消息。

Parameter :

list： 表头指针。

Return : 非 0 指针：分离出的消息，NULL 分离失败。分离出的消息需要 free()释放。

7、s_int8 eloop_list_delete(void **list, void *msg);

描述： 从链表中删除与消息地址一样的消息。

Parameter :

list： 表头指针。

msg: 消息指针。

Return : ES_FAILED：失败， ES_SUCCEED: 成功。

8、s_int8 eloop_list_delete_list(void **list);

描述： 删除链表。

Parameter :

list： 表头指针。

Return : ES_SUCCEED 成功。

9、s_int8 eloop_list_delete_list(void **list);

描述： 删除链表。

Parameter :

list： 表头指针。

Return : ES_SUCCEED 成功。

10、s_int8 eloop_send_msg(eloop_task_t task, void *msg, s_int32 len);

描述： 发送消息到任务中，并把消息添加到队列末尾。

Parameter :

task： 目标任务函数名。

msg：消息指针。

len: 消息长度。

Return : ES_SUCCEED 成功。

11、s_int8 eloop_send_msg_front (eloop_task_t task, void *msg, s_int32 len);

描述： 发送消息到任务中，并把消息添加到队列前面。

Parameter :

task： 目标任务函数名。

msg：消息指针。

len: 消息长度。

Return : ES_SUCCEED 成功。

12、s_int8 eloop_send_stream (eloop_task_t task, void *msg, s_int32 len);

描述： 把消息复制后发送到任务中，并把消息添加到队列末尾。

Parameter :

task： 目标任务函数名。

msg：消息指针。

len: 消息长度。

Return : ES_SUCCEED 成功。

13、s_int8 eloop_send_stream_front (eloop_task_t task, void *msg, s_int32 len);

描述： 把消息复制后发送到任务中，并把消息添加到队列前面。

Parameter :

task： 目标任务函数名。

msg：消息指针。

len: 消息长度。

Return : ES_SUCCEED 成功。

IV、eloop 定时与任务定时

1、s_int8 **eloop_start_timer(eloop_timer_cb_t timer_cb, u_int8 signal, u_int32 time, u_int8 reload);**

描述： 启动一个定时器。

Parameter :

timter_cb： 定时回调函数名。

signal： 定时器信号名，用于区别同一个定时任务时不同的定时策略。

time: 定时时间，单位:ms。

reload: ES_FALSE:单次定时，ES_TRUE：自动重载定时。

Return : ES_SUCCEED 成功。

2、s_int8　eloop_stop_timer(eloop_timer_cb_t timer_cb,u_int8 signal);

描述： 停止一个定时器。

Parameter :

timter_cb： 定时回调函数名。

signal： 定时器信号名，用于区别同一个定时任务时不同的定时策略。

Return : ES_SUCCEED 成功。

3、s_int8　eloop_start_timer_task(eloop_task_t task,u_int32 event, u_int32 time, u_int8 reload);

描述： 为任务启动一个定时器，定时时间到期将发送预设设置的整件通知任务。

Parameter :

task： 任务函数名。

event: 事件掩码。按位操作，每个任务最仅只有 32 个整件。

time: 定时时间，单位:ms。

reload: ES_FALSE:单次定时，ES_TRUE：自动重载定时。

Return : ES_SUCCEED 成功。

4、s_int8　eloop_stop_timer_task(eloop_task_t timer_cb,u_int32 event);

描述： 停止一个定时器。

Parameter :

task： 任务函数名。

event: 事件掩码。按位操作，每个任务最仅只有 32 个整件。

Return : ES_SUCCEED 成功。

5、s_int32　eloop_get_next_timeout(void);

描述： 获取距离下次最近的一次定时时间。

Parameter : none

Return : 返回时间。

6、s_int32　eloop_get_timestamp (void);

描述： 获取时间戳，单位 ms。

Parameter : none

Return : 返回时间。

7、s_int32　eloop_get_unixtime (void);

描述： 获取时间，单位 s。

Parameter : none

Return： 返回时间。

8、s_int32 eloop_set_unixtime (void);

描述： 设置时间，单位 s。

Parameter : none

Return： 返回时间。

9、s_int8 eloop_timer_start_measure(eTMeasure_t *pm,u_int32 timeout);

描述： 启动时间测量工具。

Parameter :

pm: 测量参数，由函数内部初始化。

timeout: 超时时间。

Return： 返回 ES_SUCCEED。

10、s_int8 eloop_timer_expire(eTMeasure_t *pm);

描述： 测试本次测量是否到期。

Parameter :

pm: 测量参数。

Return： 返回 ES_TRUE:时间到期，ES_FALSE:时间未到期。

11、u_int64 eloop_local_time_to_unix(int zone,eTime_t *date_time);

描述： 根据时区转换成 unix 时间。

Parameter :

zone: 时区偏移时间。

date_time:需要转换的日期。

Return： 返回 ES_TRUE:时间到期，ES_FALSE:时间未到期。

12、int eloop_unix_to_local_time (int zone, eTime_t *tp,u_int64 t);

描述： 根据时区转换成本地时间。

Parameter :

zone: 时区偏移时间。

tp: 转换输出的日期指针。

t: unix 时间。

Return： ES_SUCCEED。

六、pluto　API 描述

I、初始化 API

1、void　pluto_init(FirmwareInfo_t *info);

描述：　pluto　初始化。

Parameter :

info: 固件信息。用于固件更新时提示当前软件版本的信息

Return :　none。

2、void　pluto_deinit(void);

描述：　释放 pluto 资源。

Parameter : none

Return :　none。

4、sint8　pluto_write_server(PlutoServer_t *pserver);

描述：　写入服务器信息，用于远程通信。

Parameter :

pserver: 服务器信息

Return :　ES_SUCCEED:成功，ES_FAILED:失败。

II、AF 框架基础 API

1、void　af_register_listener(AfListener_t *listener);

描述：　af 监听回调函数，用于监听登录状态，逻辑设备删减，通知等应用层基础回调函数。

Parameter :

listener:　af 监听回调函数结构指针。

Return :　none。

2、sint8　af_set_device_describe(AfDescribe_t *desc);

描述：　af 层设备描述，用于描述设备支持的功能。

Parameter :

desc:　设备描述。

Return :　ES_SUCCEED 成功。

3、char* af_get_device_describe(void);

描述： 获取 af 设备描述。

Parameter : none

Return : 返回 cJSON 字符串格式的设备描述。使用完之后必须使用 free 释放()。

4、sint8 af_req_send_annouce(uint8 *pdata, uint32 len);

描述： 广播声明设备特征，用于发现局域网设备。

Parameter :

pdata:特征数据。

len:数据长度。

Return : ES_SUCCEED 成功。

5、sint8 af_req_send_beacon(uint8 keyID, uint8 *addr, uint8 *pdata, uint32 len);

描述：发送信标 。

Parameter :

keyID:密钥 ID，密码有三种，管理员，成员，临时成员，对应目标设备也有三种密钥，通信前双方必须知道其中一方的对应的 ID 密钥才能使用此 ID 密钥通信。

addr: 当 addr 所有位全为 1 时为局域网广播，局域网内所有设备都可以接收到此消息，远程设备不能收到广播消息.

pdata:发送的数据。

len:数据长度。

Return : ES_SUCCEED 成功。

6、char* af_read_firmware_info(void);

描述：获取当前软件版本信息。

Parameter : none

Return : 返回 JSON 格式的字符串，使用完之后必须使用 free 释放()。

III、AF 登录操作 API

1、void af_login_set_user(char *user, char *psw);

描述： 设置登录帐号与密码。

Parameter :

user:登录帐号。固定长度 8 字节。

psw:登录密码。固定长度 16 字节。

Return : none。

2、void　af_login_get_user (char *user, char *psw);

描述：　获取登录帐号与密码。

Parameter :

user:登录帐号。

psw:登录密码。

Return :　none。

3、uint8　af_login_start(void);

描述：　开始登录。

Parameter : none

Return :　ES_SUCCEED 成功。

4、uint8　login_stop (void);

描述：　停止登录调度。

Parameter : none

Return :　ES_SUCCEED。

5、uint8　af_login_get_state(void);

描述：　获取登录状态。

Parameter : none

Return :　返回登录状态。

6、uint8　af_logout(void);

描述：　退出登录。

Parameter : none

Return :　ES_SUCCEED。

III、AF 本地设备操作 API

1、uint8* af_get_local_addr(void);

描述：　获取本设备地址。

Parameter : none

Return :　设备地址首地址，固定长度 8 字节。不得直接修改与释放返回的内容。

2、uint8* af_get_local_key(uint8 keyID);

描述： 获取本设备密钥，设备的 3 种密钥由设备恢复出厂设置后第一次上电随机生成。

Parameter :

keyID：密钥 ID。

Return : 返回对应密钥 ID 的设备密钥，固定长度 16 字节。不得直接修改与释放返回的内容。

3、void af_registe_led(led_blink_cb_t cb)

描述： 注册 LED 操作函数，用于内部通过 LED 提示当前状态。

Parameter :

cb：LED 闪烁函数。

Return : none。

4、void af_led_blink(int num, int htime, int ltime);

描述： LED 闪烁函数。

Parameter :

num：LED 闪烁次数。

htime:亮的持续时间，单位 ms

ltime:灭的持续时间，单位 ms

Return : none。

IV、APS 接收与发送 API

1、void aps_set_cmd_listener(ApsCmdListener_t *listener);

描述： 设置 Aps 监听回调函数，用于接收远程命令与监听发送状态。

Parameter :

listener：监听函数结构指针。

Return : none。

2、sint8 aps_req_send(Address_t *dst,uint8 seq,Command_t* cmd, uint8 *pdata, uint32 len,uint8 send_option);

描述： 命令发送函数。

Parameter :

dst：目标地址。

seq: 序列号，要求每使用一次加 1。

cmd: 操作命令。

pdata: 发送数据。

len: 数据长度。

send_option: 发送选项，可选择跳过触发场景等操作。

Return： ES_SUCCEED 成功，其它失败。

3、sint8 aps_req_send_state(Address_t *dst,uint8 seq,Command_t* cmd,uint8 state, uint8 *pdata, uint32 len,uint8 send_option);

描述： 命令发送操作状态函数。

Parameter :

dst：目标地址。

seq: 序列号，要求每使用一次加 1。

cmd: 操作命令。

state: 发送操作状态。

pdata: 发送数据。

len: 数据长度。

send_option: 发送选项，可选择跳过触发场景等操作。

Return： ES_SUCCEED 成功，其它失败。

4、sint8 aps_req_report(uint8 port, uint8 seq,Command_t *cmd,uint8 *pdata, uint32 len ,uint8 send_option);

描述： 向所有通信过的用户、设备返回数据。

Parameter :

port：报告源应用端口号。

seq: 序列号，要求每使用一次加 1。

cmd: 操作命令。

pdata: 发送数据。

len: 数据长度。

send_option: 发送选项，可选择跳过触发场景等操作。

Return： ES_SUCCEED 成功，其它失败。

5、sint8 aps_req_report_state(uint8 port,uint8 seq,Command_t* cmd,uint8 state, uint8 *pdata, uint32 len,uint8 send_option);

描述： 向所有通信过的用户、设备返回状态（无需提供通信地址）。

Parameter :

port：报告源应用端口号。

seq: 序列号，要求每使用一次加 1。

cmd: 操作命令。

state: 返回操作状态。

pdata: 发送数据。

len: 数据长度。

send_option: 发送选项，可选择跳过触发场景等操作。

Return： ES_SUCCEED 成功，其它失败。

6、sint8　aps_mcmd_req_send(Address_t *dst,uint8 seq,MixCommand_t* cmd, int cmd_num,uint8 send_option);

描述： 多命令发送函数，能够把多个设备应用命令同时发送出去。

Parameter :

dst：目标地址。

seq: 序列号，要求每使用一次加 1。

cmd: 操作命令。

cmd_num:命令数量。

send_option: 发送选项，可选择跳过触发场景等操作。

Return : ES_SUCCEED 成功，其它失败。

7、sint8　aps_mcmd_req_report(uint8 port, uint8 seq,MixCommand_t* cmd, int cmd_num ,uint8 send_option);

描述：向所有通信过的用户、设备返回多个应用的状态（无需提供通信地址）。

Parameter :

port：报告源应用端口号。

seq: 序列号，要求每使用一次加 1。

cmd: 操作命令。

cmd_num:命令数量。

send_option: 发送选项，可选择跳过触发场景等操作。

Return : ES_SUCCEED 成功，其它失败。

8、uint16　aps_get_seq(void);

描述： 获取通信序号，每获取一次自加 1。

Parameter :none

Return : seq。

IV、APS 远程设备文件操作 API

1、void　aps_set_fs_listener(ApsFsListener_t *listener);

描述： 设置远程设备文件操作回函函数。

Parameter :

listener：监听函数结构指针。

Return : none。

2、sint8　af_req_write_file(Address_t *dst,uint8 seq,char *name, uint8 *pdata, int len);

描述： 向远程设备写入文件。

Parameter :

dst:　目标设备。

seq: 序列号

name: 文件名称。

len: 文件长度。

Return :　ES_SUCCEED。

3、sint8　aps_req_rename_file(Address_t *dst,uint8 seq,char *name,char *new_name)

描述：　向远程设备更改文件名。

Parameter :

dst:　目标设备。

seq: 序列号

name: 文件名称。

new_name: 新文件名。

Return :　ES_SUCCEED。

4、sint8　aps_req_read_file(Address_t *dst,uint8 seq,char *name);

描述：　向远程设备读取文件。

Parameter :

dst:　目标设备。

seq: 序列号

name: 文件名称，或通配符。

Return :　ES_SUCCEED。

5、sint8　aps_req_read_name(Address_t *dst,uint8 seq,char *name);

描述：　向远程设备读取文件名。

Parameter :

dst:　目标设备。

seq: 序列号

name: 文件名称，或统配符。

Return :　ES_SUCCEED。

6、sint8　aps_req_delete_file(Address_t *dst,uint8 seq,char *name);

描述：　向远程设备删除文件。

Parameter :

dst:　目标设备。

seq: 序列号

name: 文件名称，或统配符。

Return :　ES_SUCCEED。

V、APS 设备密钥操作 API

1、sint8　aps_add_device_key(uint8 *addr,uint8 keyID,uint8 *key);

描述：　添加目标设备密钥。

Parameter :

addr:目标设备地址。

keyID：密钥 ID。

key：设备密钥，固定长度 16 字节。

Return :　ES_SUCCEED 成功，ES_FAILED 失败。

2、uint8*　aps_get_device_key(uint8 *addr, uint8 keyID);

描述：　获取目标设备密钥。

Parameter :

addr:目标设备地址。

keyID：密钥 ID。

Return :　设备密钥，固定长度 16 字节，不得直接修改与释放返回的内容。

3、sint8　aps_remove_device_keys(void);

描述：移除所有设备密钥。

Parameter :none。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

4、sint8　aps_remove_device_key(uint8 *addr);

描述：移除当前设备所有密钥。

Parameter :

addr:目标设备地址

Return : ES_SUCCEED 成功，ES_FAILED 失败。

5、sint8　aps_remove_key(uint8 *addr,uint8 keyID);

描述：移除当前设备所有密钥。

Parameter :

addr:目标设备地址

keyID：密钥 ID。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

VI、APS 设备应用端口操作 API

1、sint8　aps_register_port(uint8 port_num,uint32 applicationID, uint32 *attr_list, uint8 attr_num, uint8 force);

描述：注册端口描述。

Parameter :

port: 端口号

applicationID:应用 ID

attr_list:支持的操作属性。

attr_num:属性数量。

force:是否强制注册，1 强制注册并删除除原有的端口所有信息

Return : ES_SUCCEED 成功，ES_FAILED 失败。

2、uint8　aps_read_port_num(void);

描述：读取注册的端口数目。

Parameter :none

Return : 返回端口数目。

3、uint8*　aps_read_port_list(void);

描述：读取所有注册的端口并以列表方式返回，数量使用 aps_read_port_num()获取。

Parameter :none

Return : 返回端口列表，使用后，必须使用 free()释放。

4、uint8　aps_get_free_port(void);

描述：获取没有使用过的端口号。

Parameter :none

Return : 返回端口号。

5、uint8　aps_port_is_exist(uint8 port);

描述：测试端口是否已经注册。

Parameter :

port:端口号。

Return : ES_FALSE 没有注册，ES_TRUE 已使用。

6、char*　aps_read_port_describe (uint8 port);

描述：读取端口信息。

Parameter :

port:端口号。

Return : 端口描述，JSON 格式，使用后必须使用 free()释放。

7、sint8 aps_update_port_describe (char *describe);

描述：更新端口描述信息。

Parameter :

describe:端口描述。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

8、sint8 aps_delete_port (uint8 port_num);

描述：删除端口。

Parameter :

port:端口号。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

9、sint8 aps_port_set_value(uint8 port,char *obj, int value);

描述：向端口中写入值。

Parameter :

port:端口号。

obj:写入的信息名称。

value:写入的信息内容。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

10、sint8 aps_port_set_string(uint8 port, char *obj, char *str);

描述：向端口中写入字符串。

Parameter :

port:端口号。

obj:写入的信息名称。

str:写入的信息内容。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

11、sint8 aps_port_set_string(uint8 port, char *obj, char *str);

描述：向端口中写入字符串值。

Parameter :

port:端口号。

obj:写入的信息名称。

str:写入的信息内容。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

12、sint8 aps_port_set_string(uint8 port, char *obj, char *str);

描述：向端口中写入字符串值。

Parameter :

port:端口号。

obj:写入的信息名称。

str:写入的信息内容。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

13、uint32 aps_port_get_value(uint8 port, char *obj);

描述：向端口中读取指定值。

Parameter :

port:端口号。

obj:读取的信息名称。

Return : 返回值。

14、char* aps_port_get_string(uint8 port, char *obj);

描述：向端口中读取指定值。

Parameter :

port:端口号。

obj:读取的信息名称。

Return : 返回字符串,不得释放与直接修改返回的信息。

15、uint8 aps_count_port_by_value(char *obj,int value);

描述：计算匹配目标值的端口数量。

Parameter :

obj:匹配的信息名称。

value:匹配的信息值

Return : 返回匹配的端口数量。

16、uint8 aps_count_port_by_string(char *obj,char *str);

描述：计算匹配目标值的端口数量。

Parameter :

obj:匹配的信息名称。

str:匹配的信息值

Return： 返回匹配的端口数量。

17、uint8 aps_get_port_by_value_index(char *obj,int value, uint8 index);

描述：获取匹配信息的端口号。

Parameter：

obj:匹配的信息名称。

str:匹配的信息值。

index: 匹配的信息索引。

Return： 返回匹配的端口号。

18、uint8 aps_get_port_by_string_index(char *obj,char *str, uint8 index);

描述：获取匹配信息的端口号。

Parameter：

obj:匹配的信息名称。

str:匹配的信息值。

index: 匹配的信息索引。

Return： 返回匹配的端口号。

19、uint8aps_get_port_by_value2(char *obj1,int value1, char *obj2, int value2);

描述：获取匹配信息的端口号。

Parameter：

obj1:匹配的信息名称。

value1:匹配的信息值。

obj2:匹配的信息名称。

value2:匹配的信息值。

Return： 返回匹配的端口号。

VII、APS 设备场景操作 API

1、sint8 aps_run_scene(char *task,char *name);

描述：运行场景。

Parameter：

task: 场景内容。

name: 场景名称。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

2、uint16　aps_count_scene(void);

描述：获取运行中场景的数量。

Parameter :none

Return : 场景数量。

3、char*　aps_get_scene_by_name(char *name);

描述：获取运行中的场景。

Parameter :

name: 场景名称。

Return : 场景内容，JSON 格式，使用完必须用 free()释放。

4、char*　aps_get_scene_name_by_id(uint16 id);

描述：获取运行场景的名字。

Parameter :

id: 场景索引。

Return : 场景名称,不得修改与释放返回的内容。

5、sint8　aps_delete_scene(char *name);

描述：删除场景。

Parameter :

name: 场景名称。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

6 、sint8　aps_join_scene(char *tsk0, char *tsk1);

描述： 合并场景，场景 1 加入场景 2 之后执行。

Parameter :

task0: 场景 1 名称。

task1: 场景 2 名称。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

7、cJSON*　aps_make_scene_head(int version, char *boot_state);

描述： 生成场景头信息。

Parameter :

version: 场景版本控制。

boot_state: 设备上电时场景状态。

Return： 返回场景，JSON 格式。

8、cJSON*　aps_make_scene_loop(cJSON *body,char *reason);

描述：　在场景中添加循环块。

Parameter：

body: 添加的语句体位置。

reason: 循环控制条件，当为 1 时为死循环。

Return： 返回循环语句体当前位置，JSON 格式。

9、cJSON*　aps_make_scene_if(cJSON *body, char *what, char *reason);

描述：　在场景中添加条件。

Parameter：

body: 添加的语句体位置。

reason: 条件。

Return： 返回 if 语句体当前位置，JSON 格式。

10、cJSON*　aps_make_scene_else(cJSON *body);

描述：　在 if 语句后添加 else 执行语句。

Parameter：

body: 添加的语句体位置。

Return： 返回 else 语句体当前位置，JSON 格式。

11、cJSON*　aps_make_scene_read_for(cJSON *body, char *reason);

描述：读设备，等待设备返回数据后并匹配返回的内容，条件为 true 执行语句块内的内容。

Parameter：

body: 添加的语句体位置。

reason: 判断条件

Return： 返回语句体当前位置，JSON 格式。

12、void　　aps_make_scene_action(cJSON *body, char *fuc, char *str);

描述：　执行一个动作，可以是简单的数学表达式，也可以是一个控制设备的命令。

Parameter：

body: 添加的语句体位置。

fuc：功能。可以是”send”,可以是”do”

str: 表达式。

Return： 返回语句体当前位置，JSON 格式。

13、cJSON*　aps_make_scene_exception(cJSON *body);

描述：暂不支持。

Parameter :

body: 添加的语句体位置。

Return : 返回语句体当前位置，JSON 格式。

14、sint8　aps_save_scene(char *name,cJSON *psc);

描述：保存并运行场景。

Parameter :

name: 场景名称。

psc：场景执行体。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

15、sint8　aps_release_scene(cJSON *psc);

描述：释放创建的场景执行体，创建场景使用完成后必须使用此函数释放。

Parameter :

psc：场景执行体。

Return : ES_SUCCEED 成功。

六、EFS　API 描述

I、文件基本操作

1、char *　efs_read_text(const char *fname);

描述：读取文本文件。

Parameter :

fname：文件名称。

Return :文件内容，字符串，使用完必须使用 free()释放。

2、int　efs_write_text(const char *fname, char *str);

描述：写入文本文件。

Parameter :

fname：文件名称。

str: 文件内容。

Return : 返回 0 写入成功。

3、int efs_read_file(const char *fname, int offset, u_int8 *buf, int len);

描述：读取二进制文件内容。

Parameter :

fname：文件名称。

offset: 读取位置

buf: 输出内容指针。

len: 读取长度。

Return : 返回实际读取长度。

4、int efs_write_file(const char *fname, int offset, u_int8 *pdata, int len);

描述：写入二进制文件内容。

Parameter :

fname：文件名称。

offset: 写入位置

pdata: 写入内容指针。

len: 写入长度。

Return : 返回实际读取长度。

5、int efs_get_file_length(const char *fname);

描述：获取文件长度。

Parameter :

fname：文件名称。

Return : 返回文件内容长度。

6、char * efs_get_file_name(const char *filter, int id);

描述：根据通配符与索引获取文件名称。

Parameter :

filter：通配符。

id：索引。

Return : 返回文件名称，如果返回 NULL 表示匹配的文件名已经读取完整，使用后需要 free()释放。

7、int efs_delete_file(const char *fname);

描述：删除文件。

Parameter :

fname：文件名称。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

8、int　efs_delete_all_file(void);

描述：删除所有文件。

Parameter：none

Return : ES_SUCCEED 成功，ES_FAILED 失败。

9、int　efs_disk_format(void);

描述：格式化文件系统。

Parameter：none

Return : ES_SUCCEED 成功，ES_FAILED 失败。

10、int efs_rename(const char *old_name, const char *new_name);

描述：文件重命名。

Parameter：

old_name:旧文件名。

new_name:新文件名

Return : ES_SUCCEED 成功，ES_FAILED 失败。

II、简易的安全数据库操作

数据库使用了文件备份，是相对安全的数据保存操作。

1、 int　efs_db_save_string(const char *db, char *key, char *value);

描述：保存字符串到数据库中。

Parameter：

db：数据库名字。

key：数据键值。

value：数据字符串。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

2、 int　efs_db_save_block(const char *db, char *key, void *block, int block_size);

描述：保存二进制数据、结构体到数据库中。

Parameter：

db：数据库名字。

key：数据键值。

block: 数据块指针。

block_size: 数据块大小。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

3、int　efs_db_save_number(const char *db, char *key, u_int32 value);

描述：保存数值到数据库中。

Parameter：

db：数据库名字。

key：数据键值。

value: 数值。

Return : ES_SUCCEED 成功，ES_FAILED 失败。

4、char *　efs_db_read_string(const char *db, char *key);

描述：从数据库中读取字符串。

Parameter：

db：数据库名字。

key：数据键值。

Return : 返回字符串指针，不得修改，不得使用 free 释放。

5、int　efs_db_read_block(const char *db, char *key, void *block, int block_size);

描述：读取二进制数据、结构体。

Parameter：

db：数据库名字。

key：数据键值。

block: 数据块指针。

block_size: 数据块大小。

Return :返回读取的数据字节大小。

6、u_int32　efs_db_read_number(const char *db,char *key);

描述：从数据库中读取数值。

Parameter：

db：数据库名字。

key：数据键值。

Return : 返回数值。

7、int　efs_db_count_element(const char *db);

描述：计算数据库中的元素个数。

Parameter：

db：数据库名字。

Return： 返回元素个数。

8、char *　efs_db_get_key(const char *db, int id);

描述：根据索引获取数据库中的键名称。

Parameter：

db：数据库名字。

id：索引

Return： 返回键名称。不得修改，不得使用 free 释放。

9、int　　efs_db_delete_value(const char *db, char *key);

描述：删除数据库中的键与值。

Parameter：

db：数据库名字。

key： 键名称

Return : ES_SUCCEED 成功，ES_FAILED 失败。

10、int　efs_db_flush(void);

描述： 立即更新数据数据库。数据库更改之后会延迟写入，将可能造成掉电丢失，使用此函数可立即写入文件系统中。

Parameter：none

Return : ES_SUCCEED 成功，ES_FAILED 失败。

七、ATC　API 描述

1、uint8 ATC_TaskInit(uint8 dev_type);

描述：ATC 初始化，用于设备的出厂快速产测，工厂升级等。

Parameter :

dev_type：ATC_TYPE_AS_DEVICE 为设备模式, ATC_TYPE_AS_DONGLE 为 dongle 模式通过串口与计算相连接,

Return : ES_SUCCEED 成功。