C SDK

HAL列表

HAL列表

基础HAL

HAL_Free

原型

```
void HAL_Free(_IN_ void *ptr);
```

接口说明

释放参数ptr指向的一块堆内存, 当传入的参数为NULL时不执行任何操作

参数说明

参数	数据类型	方向	说明
ptr	void *	输入	指向将要释放的堆内存的指针

返回值说明

void

HAL_GetChipID

原型

```
char *HAL_GetChipID(_OU_ char cid_str[HAL_CID_LEN]);
```

接口说明

获取唯一的芯片ID字符串, 字符串长度不能超过HAL_CID_LEN定义的数值。

注：该HAL只需要芯片商进行适配，如果用户不是芯片商，该HAL返回空字符串即可

参数说明

参数	数据类型	方向	说明
cid_str	char[]	输出	存放芯片ID的字符串缓冲区

返回值说明

指向字符串缓冲区的指针

HAL_GetDeviceID

(在2.3.1及以后版本中不需要实现）

原型

```
int HAL_GetDeviceID(_OU_ char device_id[DEVICE_ID_LEN]);
```

接口说明

获取设备的DeviceID, 用于标识设备单品的ID

参数说明

参数	数据类型	方向	说明
device_id	char[]	输出	存放DeviceID的字符串缓冲区

返回值说明

实际获取到的DeviceID字符串长度

HAL_GetDeviceName

原型

```
int HAL_GetDeviceName(_OU_ char device_name[DEVICE_NAME_LEN]);
```

接口说明

获取设备的DeviceName, 用于唯一标识单个设备的名字, 三元组之一, 在云端控制台注册得到并烧写到设备中

参数说明

参数	数据类型	方向	说明
device_name	char[]	输出	存放DeviceName的字符串缓冲区

返回值说明

实际获取到的DeviceName字符串长度

HAL_GetDeviceSecret

原型

```
int HAL_GetDeviceSecret(_OU_ char device_secret[DEVICE_SECRET_LEN]);
```

接口说明

获取设备的DeviceSecret, 用于标识单个设备的密钥, 三元组之一, 在云端控制台注册得到并烧写到设备中

参数说明

参数	数据类型	方向	说明
device_secret	char[]	输出	存放DeviceSecret的字符串缓冲区

返回值说明

实际获取到的DeviceSecret字符串长度

HAL_GetFirmwareVersion

原型

```
int HAL_GetFirmwareVersion(_OU_ char version[FIRMWARE_VERSION_MAXLEN]);
```

接口说明

获取设备的固件版本字符串, 此固件版本号将会用于OTA升级的版本上报。如果设备不准备支持OTA，该函数返回空串即可。

参数说明

参数	数据类型	方向	说明
version	char[]	输出	存放FirmwareVesion的字符串缓冲区

返回值说明

实际获取到的FirmwareVesion字符串长度

HAL_GetModuleID

原型

```
int HAL_GetModuleID(_OU_ char mid_str[MID_STR_MAXLEN]);
```

接口说明

获取设备的Module ID, 仅用于紧密合作伙伴。该函数用于模组商上报模组型号，其它角色的用户返回空串即可。

参数说明

参数	数据类型	方向	说明
mid_str	char[]	输出	存放Module ID的字符串缓冲区

返回值说明

实际获取到的Module ID字符串长度

HAL_GetPartnerID

原型

```
int HAL_GetPartnerID(_OU_ char pid_str[PID_STR_MAXLEN]);
```

接口说明

获取设备的Partner ID, 仅用于紧密合作伙伴。

参数说明

参数	数据类型	方向	说明
pid_str	char[]	输出	存放Partner ID的字符串缓冲区

返回值说明

实际获取到的Partner ID字符串长度

HAL_GetProductKey

原型

```
int HAL_GetProductKey(_OU_ char product_key[PRODUCT_KEY_LEN]);
```

接口说明

获取设备的ProductKey, 用于标识设备的品类, 三元组之一, 在云端控制台注册得到并烧写到设备中

参数说明

参数	数据类型	方向	说明
product_key	char[]	输出	存放ProductKey的字符串缓冲区

返回值说明

实际获取到的ProductKey字符串长度

HAL_GetProductSecret

原型

```
int HAL_GetProductSecret(_OU_ char product_secret[DEVICE_SECRET_LEN]);
```

接口说明

获取设备的ProductSecret, 用于标识品类的密钥, 在云端控制台注册得到并烧写到设备中, 在一型一密的场景下将会使用到此字符串

参数说明

参数	数据类型	方向	说明
product_secret	char[]	输出	存放ProductSecret的字符串缓冲区

返回值说明

实际获取到的ProductSecret字符串长度

HAL_GetTimeStr

(在2.3.1及以后版本中不需要实现)

原型

```
char *HAL_GetTimeStr(_OU_ char *buf, _IN_ int len);
```

接口说明

获取当前时间字符串

参数说明

参数	数据类型	方向	说明
buf	char *	输出	指向时间字符串缓冲区的指针
len	int	输入	字符串缓冲区的字节长度

返回值说明

指向时间字符串缓冲区的指针

HAL_Kv_Del

原型

```
int HAL_Kv_Del(const char *key);
```

接口说明

删除指定KV数据, 删除key对应的KV对数据, 可以通过擦除Flash或修改文件数据的方式实现持久化数据的删除

参数说明

参数	数据类型	方向	说明
key	const char *	输入	指向key字符串的指针

buffer	void *	输出	指向存放获取数据的指针
buffer_len	int *	输出	指向存放获取

返回值说明

值	说明
0	删除成功
-1	删除失败

HAL_Kv_Erase_All

原型

```
int HAL_Kv_Erase_All(void);
```

接口说明

擦除所有的KV数据, 可以通过擦除Flash或修改文件数据的方式实现持久化数据的删除

参数说明

void

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Kv_Get

原型

```
```

```
int HAL_Kv_Get(const char *key, void *buffer, int *buffer_len);
```

接口说明

获取KV数据, 获取key对应的KV对数据, 可以通过读取Flash或读取文件的方式实现持久化数据的读取

参数说明

参数	数据类型	方向	说明
key	const char *	输入	指向key字符串的指针
buffer	void *	输出	指向存放获取数据的指针
buffer_len	int *	输出	指向存放获取

返回值说明

值	说明
0	获取成功
-1	获取失败

HAL_Kv_Set

原型

```
int HAL_Kv_Set(const char *key, const void *val, int len, int sync);
```

接口说明

设置KV数据接口, 可通过写flash或写文件的方式实现数据持久化

参数说明

参数	数据类型	方向	说明
key	const char *	输入	指向key字符串的指针
val	const void *	输入	指向待设置数据的指针
len	int	输入	待设置数据的字节长度

sync	int	输入	0: 异步接口. 1: 同步接口

返回值说明

值	说明
0	设置成功
-1	设置失败

HAL_Malloc

原型

```
void *HAL_Malloc(_IN_ uint32_t size);
```

接口说明

申请一块堆内存

参数说明

参数	数据类型	方向	说明
size	uint32_t	输入	申请的堆内存大小

返回值说明

值	说明
NULL	内存申请失败
!NULL	指向堆内存首地址的指针

HAL_Printf

原型

```
void HAL_Printf(_IN_ const char *fmt, ...);
```

接口说明

打印函数, 用于向串口或其它标准输出打印日志或调试信息, 可参考C99的printf()函数实现

参数说明

参数	数据类型	方向	说明
fmt	const char *	输入	格式化字符串
...	可变类型	输入	可变参数列表

返回值说明

void

HAL_Random

HAL_Random

原型

```
uint32_t HAL_Random(_IN_ uint32_t region);
```

接口说明

随机数函数, 接受一个无符号数作为范围, 返回0到region范围内的一个随机数

参数说明

参数	数据类型	方向	说明
region	uint32_t	输入	用于指定随机数范围的无符号数

返回值说明

在指定范围的随机数

HAL_Reboot

原型

```
void HAL_Reboot(void);
```

接口说明

设备重启, 调用该接口能实现复位功能

参数说明

void

返回值说明

void

HAL_SetDeviceName

原型

```
int HAL_SetDeviceName(_IN_ char *device_name);
```

接口说明

设置设备的DeviceName, 用于标识设备单品的名字, 三元组之一

参数说明

参数	数据类型	方向	说明
device_name	char *	输出	指向待传入DeviceName字符串的指针

返回值说明

待设置DeviceName字符串的长度

HAL_SetDeviceSecret

原型

```
int HAL_SetDeviceSecret(_IN_ char *device_secret);
```

接口说明

设置设备的DeviceSecret, 用于标识设备单品的密钥, 三元组之一

参数说明

参数	数据类型	方向	说明
device_secret	char *	输出	指向待传入DeviceSecret字符串的指针

返回值说明

待设置DeviceSecret字符串的长度

HAL_SetProductKey

原型

```
int HAL_SetProductKey(_IN_ char *product_key);
```

接口说明

设置设备的ProductKey, 用于标识设备的品类, 三元组之一

参数说明

参数	数据类型	方向	说明
product_key	char *	输入	指向待设置ProductKey字符串的指针

返回值说明

待设置ProductKey字符串的长度

HAL_SetProductSecret

原型

```
int HAL_SetProductSecret(_IN_ char *product_secret);
```

接口说明

设置设备的ProductSecret, 用于标识品类的密钥, 在一型一密场景中会使用到此字符串

参数说明

参数	数据类型	方向	说明
product_secret	char *	输出	指向待传入ProductSecret字符串的指针

返回值说明

待设置ProductSecret字符串的长度

HAL_SleepMs

原型

```
void HAL_SleepMs(_IN_ uint32_t ms);
```

接口说明

睡眠函数, 使当前执行线程睡眠指定的毫秒数

参数说明

参数	数据类型	方向	说明
ms	uint32_t	输入	线程挂起的时间, 单位 ms

返回值说明

void

HAL_Snprintf

原型

```
int HAL_Snprintf(_OU_ char *str, _IN_ const int len, _IN_ const char *fmt, ...);
```

接口说明

打印函数, 向内存缓冲区格式化构建一个字符串, 参考C99标准库函数

参数说明

参数	数据类型	方向	说明
str	char *	输入	指向字符缓冲区的指针
len	int	输入	缓冲区的字符长度
fmt	const char *	输入	格式化字符串
...		输入	可变参数列表

返回值说明

实际写入缓冲区的字符串长度

HAL_Srandom

原型

```
void HAL_Srandom(_IN_ uint32_t seed);
```

接口说明

随机数播种函数, 使 HAL_Random 的返回值每个随机序列各不相同, 类似C标准库中的srand

参数说明

参数	数据类型	方向	说明
seed	uint32_t	输入	用于产生新随机序列的种子

返回值说明

void

HAL_Sys_reboot

原型

```
void HAL_Sys_reboot(void);
```

接口说明

系统立即重启

参数说明

void

返回值说明

void

HAL_Timer_Create

原型

```
void *HAL_Timer_Create(const char *name, void (*func)(void *), void *user_data);
```

接口说明

创建指定名称的定时器, 同时注册用户回调函数和用户数据

参数说明

参数	数据类型	方向	说明
name	const char *	输入	定时器名称字符串
func	void (*func)(void *)	输入	用户回调函数
user_data	void *	输入	指向用户数据的指针

返回值说明

值	说明
NULL	创建失败
!NULL	创建成功, 返回定时器句柄

HAL_Timer_Delete

原型

```
int HAL_Timer_Delete(void *timer);
```

接口说明

删除由HAL_Timer_Create()创建的定时器, 释放资源

参数说明

参数	数据类型	方向	说明
timer	void *	输入	定时器句柄, 此句柄由调用 HAL_Timer_Create() 时返回

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Timer_Start

原型

```
int HAL_Timer_Start(void *t, int ms);
```

接口说明

启动定时器

参数说明

参数	数据类型	方向	说明
timer	void *	输入	定时器句柄, 此句柄由调用 HAL_Timer_Create() 时返回
ms	int	输入	定时器定时时间, 单位 ms

返回值说明

值	说明
0	操作成功

-1	操作失败

HAL_Timer_Stop

原型

```
int HAL_Timer_Stop(void *t);
```

接口说明

关闭定时器

参数说明

参数	数据类型	方向	说明
timer	void *	输入	定时器句柄, 此句柄由调用 HAL_Timer_Create() 时返回

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_UptimeMs

原型

```
uint64_t HAL_UptimeMs(void);
```

接口说明

获取设备从上电到当前时刻所经过的毫秒数

参数说明

void

返回值说明

设备从上电到当前时刻所经过的毫秒数

HAL_UTC_Get

原型

```
long long HAL_UTC_Get(void);
```

接口说明

获取UTC时间, 数值为从Epoch(1970年1月1日00:00:00 UTC)开始所经过的秒数单位

参数说明

void

返回值说明

单位为ms的UTC时间

HAL_UTC_Set

原型

```
void HAL_UTC_Set(long long ms);
```

接口说明

设置UTC时间, 设置参数为从Epoch(1970年1月1日00:00:00 UTC)开始所经过的秒数单位

参数说明

参数	数据类型	方向	说明
ms	long long	输入	单位为ms的UTC时间

返回值说明

void

HAL_Vsnprintf

原型

```
int HAL_Vsnprintf(_OU_ char *str, _IN_ const int len, _IN_ const char *fmt, _IN_ va_list ap);
```

接口说明

格式化输出字符串到指定buffer中, 可参考C标准库的vsnprintf()实现

参数说明

参数	数据类型	方向	说明
str	char *	输出	用于存放写入字符串的buffer
len	const int	输入	允许写入的最大字符串长度
fmt	const char	输入	格式化字符串
ap	va_list	输入	可变参数列表

返回值说明

成功写入的字符串长

MQTT连云HAL

HAL_SSL_Destroy

原型

```
int32_t HAL_SSL_Destroy(_IN_ uintptr_t handle);
```

接口说明

销毁由参数handle指定的TLS连接

参数说明

参数	数据类型	方向	说明
handle	uintptr_t	输入	TLS连接句柄

返回值说明

值	说明
<0	操作失败
= 0	操作成功

HAL_SSL_Establish

原型

```
uintptr_t HAL_SSL_Establish(
_IN_ const char *host,
_IN_ uint16_t port,
_IN_ const char *ca_crt,
_IN_ size_t ca_crt_len);
```

接口说明

根据指定的服务器网络地址, 服务器端口号和证书文件建立TLS连接, 返回对应的连接句柄

参数说明

参数	数据类型	方向	说明
host	const char	输入	指定的TLS服务器网络地址
port	uint16_t	输入	指定的TLS服务器端口
ca_crt	const char	输入	指向PEM编码的X.509证书的指针
ca_crt_len	size_t	输入	证书字节长度

返回值说明

值	说明
NULL	创建失败
!NULL	创建成功, 返回TLS连接句柄

HAL_SSL_Read

原型

```
int32_t HAL_SSL_Read(_IN_ uintptr_t handle, _OU_ char *buf, _OU_ int len, _IN_ int timeout_ms);
```

接口说明

从指定的TLS连接中读取数据, 此接口为同步接口, 如果在超时时间内读取到参数len指定长度的数据则立即返回, 否则在超时时间到时才解除阻塞返回

参数说明

参数	数据类型	方向	说明
handle	uintptr_t	输入	TLS连接句柄
buf	char *	输出	指向数据接收缓冲区的指针
len	int	输入	数据接收缓冲区的字节大小
timeout_ms	int	输入	超时时间

返回值说明

值	说明
-2	TLS连接发生错误
-1	TLS连接被远程设备关闭
0	TLS读超时, 且没有接收到任何数据
>0	TLS读取到的字节数, TLS读取成功

HAL_SSL_Write

原型

```
int32_t HAL_SSL_Write(_IN_ uintptr_t handle, _IN_ const char *buf, _IN_ int len, _IN_ int timeout_ms);
```

接口说明

从指定的TLS连接中写入数据, 此接口为同步接口, 如果在超时时间内写入了参数len指定长度的数据则立即返回, 否则在超时时间到时才解除阻塞返回

参数说明

参数	数据类型	方向	说明
handle	uintptr_t	输入	TLS连接句柄
buf	char *	输入	指向数据发送缓冲区的指针
len	int	输入	数据发送缓冲区的字节大小
timeout_ms	int	输入	超时时间

返回值说明

值	说明
<0	TLS连接发生错误
0	TLS写超时, 且没有写入任何数据
>0	TLS写入的字节数, TLS写入成功

HAL_TCP_Destroy

原型

```
int32_t HAL_TCP_Destroy(_IN_ uintptr_t fd);
```

接口说明

销毁由参数fd指定的TCP连接, 释放资源

参数说明

参数	数据类型	方向	说明
fd	uintptr_t	输入	TCP连接句柄

返回值说明

值	说明
<0	操作失败
= 0	操作成功

HAL_TCP_Establish

原型

```
uintptr_t HAL_TCP_Establish(_IN_ const char *host, _IN_  uint16_t port);
```

接口说明

根据指定的服务器网络地址和端口号建立TCP连接, 并返回对应连接句柄

参数说明

参数	数据类型	方向	说明
host	const char *	输入	指定TCP服务器的网络

			地址
port	uint16_t	输入	指定TCP服务器的端口号

返回值说明

值	说明
NULL	TCP连接建立失败
!NULL	TCP连接建立成功, 返回对应的连接句柄

HAL_TCP_Read

原型

```
int32_t HAL_TCP_Read(_IN_ uintptr_t fd, _OU_ char *buf, _IN_ uint32_t len, _IN_ uint32_t timeout_ms);
```

接口说明

从指定的TCP连接中读取数据, 此接口为同步接口, 如果在超时时间内读取到参数len指定长度的数据则立即返回, 否则在超时时间到时才解除阻塞返回

参数说明

参数	数据类型	方向	说明
fd	uintptr_t	输入	TCP连接句柄
buf	char *	输出	指向数据接收缓冲区的指针
len	int	输入	数据接收缓冲区的字节大小
timeout_ms	int	输入	超时时间

返回值说明

值	说明
-2	TCP连接发生错误
-1	TCP连接被远程设备关闭

0	TCP读超时, 且没有接收到任何数据
>0	TCP读取成功, 返回读取到的字节数

HAL_TCP_Write

原型

```
int32_t HAL_TCP_Write(_IN_ uintptr_t fd, _IN_ const char *buf, _IN_ uint32_t len, _IN_ uint32_t timeout_ms);
```

接口说明

从指定的TCP连接中写入数据, 此接口为同步接口, 如果在超时时间内写入了参数len指定长度的数据则立即返回, 否则在超时时间到时才解除阻塞返回

参数说明

参数	数据类型	方向	说明
fd	uintptr_t	输入	TCP连接句柄
buf	char *	输入	指向数据发送缓冲区的指针
len	int	输入	数据发送缓冲区的字节大小
timeout_ms	int	输入	超时时间

返回值说明

值	说明
<0	TCP连接发生错误
0	TCP写超时, 且没有写入任何数据
>0	TCP入成功, 返回TCP写入的字节数

线程HAL

HAL_MutexCreate

原型

```
void *HAL_MutexCreate(void);
```

接口说明

创建一个互斥量对象, 返回指向所创建互斥量的指针, 用于同步访问, 对于仅支持单线程应用, 可实现为空函数

参数说明

void

返回值说明

void

HAL_MutexDestroy

原型

```
void HAL_MutexDestroy(_IN_ void *mutex);
```

接口说明

销毁一个互斥量对象, 释放资源

参数说明

参数	数据类型	方向	说明
mutex	void *	输入	互斥量指针

返回值说明

void

HAL_MutexLock

原型

```
void HAL_MutexLock(_IN_ void *mutex);
```

接口说明

锁住一个互斥量

参数说明

参数	数据类型	方向	说明
mutex	void *	输入	互斥量指针

返回值说明

void

HAL_MutexUnlock

原型

```
void HAL_MutexUnlock(_IN_ void *mutex);
```

接口说明

解锁一个互斥量

参数说明

参数	数据类型	方向	说明
mutex	void *	输入	互斥量指针

返回值说明

void

HAL_SemaphoreCreate

原型

```
void *HAL_SemaphoreCreate(void);
```

接口说明

创建一个计数信号量, 此接口实现必须为原子操作, 对于仅支持单线程应用, 可实现为空函数

参数说明

void

返回值说明

值	说明
NULL	创建失败
!NULL	创建成功, 返回信号量句柄

HAL_SemaphoreDestroy

原型

```
void HAL_SemaphoreDestroy(_IN_ void *sem);
```

接口说明

销毁一个由参数sem指定的信号量, 此接口实现必须为原子操作, 此函数无返回值

参数说明

参数	数据类型	方向	说明
sem	void *	输入	信号量指针

返回值说明

void

HAL_SemaphorePost

原型

```
void HAL_SemaphorePost(_IN_ void *sem);
```

接口说明

在指定的计数信号量上做自增操作, 解除其它线程的等待, 此接口实现必须为原子操作, 对于仅支持单线程应用, 可实现为空函数

参数说明

参数	数据类型	方向	说明
sem	void *	输入	信号量句柄

返回值说明

void

HAL_SemaphoreWait

原型

```
int HAL_SemaphoreWait(_IN_ void *sem, _IN_ uint32_t timeout_ms);
```

接口说明

在指定的计数信号量上等待并做自减操作, 对于仅支持单线程应用, 此接口实现必须为原子操作, 可实现为空函数

参数说明

参数	数据类型	方向	说明
sem	void *	输入	信号量句柄
timeout_ms	uint32_t	输入	信号量等待超时时间, 单位ms, 如果参数为PLATFORM_WAIT_INFINITE, 则函数返回只能由获取信号量触发

返回值说明

值	说明
0	函数返回是由信号量触发
-1	函数返回是由超时触发

HAL_ThreadCreate

原型

```
int HAL_ThreadCreate(
_OU_ void **thread_handle,
_IN_ void *(*work_routine)(void *),
_IN_ void *arg,
_IN_ hal_os_thread_param_t *hal_os_thread_param,
_OU_ int *stack_used);
```

接口说明

按照指定入参创建一个线程

参数说明

参数	数据类型	方向	说明
thread_handle	void **	输出	指向线程句柄变量的指

			针
work_routine	void (work_routine)(void *)	输入	指向线程执行函数的函数指针
arg	void *	输入	传递给运行程序的单个参数
hal_os_thread_param	hal_os_thread_param_t *	输入	指向线程初始化参数的指针
stack_used	int *	输出	指示平台是否使用栈缓冲区, 0: 未使用. 1: 使用

线程初始化参数定义:

```
typedef struct _hal_os_thread {
hal_os_thread_priority_t priority; /* initial thread priority */
void *stack_addr; /* thread stack address malloced by caller, use system stack by . */
size_t stack_size; /* stack size requirements in bytes; 0 is default stack size */
int detach_state; /* 0: not detached state; otherwise: detached state. */
char *name; /* thread name. */
} hal_os_thread_param_t;
```

线程优先级定义:

```
typedef enum {
os_thread_priority_idle = -3, /* priority: idle (lowest) */
os_thread_priority_low = -2, /* priority: low */
os_thread_priority_belowNormal = -1, /* priority: below normal */
os_thread_priority_normal = 0, /* priority: normal (default) */
os_thread_priority_aboveNormal = 1, /* priority: above normal */
os_thread_priority_high = 2, /* priority: high */
os_thread_priority_realtime = 3, /* priority: realtime (highest) */
os_thread_priority_error = 0x84, /* system cannot determine priority or thread has illegal priority */
} hal_os_thread_priority_t;
```

返回值说明

值	说明
-1	创建失败
0	创建成功

HAL_ThreadDelete

原型

```
void HAL_ThreadDelete(_IN_ void *thread_handle);
```

接口说明

删除指定的线程

参数说明

参数	数据类型	方向	说明
thread_handle	void *	输入	线程句柄, NULL表示当前线程

返回值说明

void

HAL_ThreadDetach

原型

```
void HAL_ThreadDetach(_IN_ void *thread_handle);
```

接口说明

将指定线程设置为分离状态

参数说明

参数	数据类型	方向	说明
thread_handle	void *	输入	线程句柄, NULL表示当前线程

返回值说明

void

OTA HAL

OTA固件下载相关HAL接口详解

HAL_Firmware_Persistence_Start

原型

```
void HAL_Firmware_Persistence_Start(void);
```

接口说明

固件持久化功能开始

参数说明

void

返回值说明

void

HAL_Firmware_Persistence_Stop

原型

```
int HAL_Firmware_Persistence_Stop(void);
```

接口说明

固件持久化功能结束

参数说明

void

返回值说明

void

HAL_Firmware_Persistence_Write

原型

```
int HAL_Firmware_Persistence_Write(_IN_ char *buffer, _IN_ uint32_t length);
```

接口说明

固件持久化写入固件

参数说明

参数	数据类型	方向	说明
buffer	char *	输入	指向写入缓冲区的指针
length	uint32_t	输入	写入的字节长度

返回值说明

实际写入的字节长度

WiFi配网HAL

HAL_Aes128_Cbc_Decrypt

原型

```
typedef void *p_HAL_Aes128_t;

int HAL_Aes128_Cbc_Decrypt(
_IN_ p_HAL_Aes128_t aes,
_IN_ const void *src,
_IN_ size_t blockNum,
_OU_ void *dst);
```

接口说明

以AES-CBC-128的加解密模式, 用HAL_Aes128_Init()被调用时传入的密钥, 解密从src位置起长度为blockNum块数的密文, 并把明文结果存放到dst起始的内存缓冲区中

参数说明

参数	数据类型	方向	说明
aes	void *	输入	这是一个句柄, 是用户调用HAL_Aes128_Init()成功时所得到的返回值, 之后需要作为必选的入参传给所有AES加解密相关的HAL接口
src	const void *	输入	指定被解密的源数据的缓冲区首地址, 也就是AES密文的起始地址
blockNum	size_t	输入	指定被解密的源数据的缓冲区长度, 以16字节为一个Block, 此参数表达密文总长度是多少个Block, 或者说是16字节的多少倍
dst	void *	输出	指定被解密的目的数据的缓冲区首地址, 也就是AES解密后, 存放明文的首地址

返回值说明

值	说明
0	解密成功
-1	解密失败

HAL_Aes128_Cbc_Encrypt

原型

```
int HAL_Aes128_Cbc_Encrypt(
_IN_ p_HAL_Aes128_t aes,
_IN_ const void *src,
_IN_ size_t blockNum,
_OU_ void *dst);
```

接口说明

以AES-CBC-128的加解密模式, 用HAL_Aes128_Init()被调用时传入的密钥, 加密从src位置起长度为blockNum块数的明文, 并把密文结果存放到dst起始的内存缓冲区中

参数说明

参数	数据类型	方向	说明
aes	void *	输入	这是一个句柄, 是用户调用HAL_Aes128_Init()成功时所得到的返回值, 之后需要作为必选的入参传给所有AES加解密相关的HAL接口
src	const void *	输入	指定被加密的源数据的缓冲区首地址, 也就是明文的起始地址
blockNum	size_t	输入	指定被加密的源数据的缓冲区长度, 以16字节为一个Block, 此参数表达明文总长度是多少个Block, 或者说是16字节的多少倍
dst	void *	输出	指定被加密的目的数据的缓冲区首地址, 也就是AES加密后, 存放AES密文的首地址

返回值说明

值	说明
0	加密成功

-1	加密失败

HAL_Aes128_Cfb_Decrypt

原型

```
int HAL_Aes128_Cfb_Decrypt(
_IN_ p_HAL_Aes128_t aes,
_IN_ const void *src,
_IN_ size_t length,
_OU_ void *dst);
```

接口说明

CFB模式的AES128解密接口函数, 使用此接口前必须先调用HAL_Aes128_Init()建立AES上下文数据结构体

参数length为分组数量(AES128一个分组的长度为128bits, 也就是16bytes), 而非字节数

用户在实现此函数时无需考虑padding问题, 因为SDK在调用此接口前已完成padding处理

参数说明

参数	数据类型	方向	说明
aes	void *	输入	调用HAL_Aes128_Init()时返回的上下文结构体指针
src	const void *	输入	指向密文数据缓冲区的指针
blockNum	size_t	输入	密文数据的分组数量, AES128一个分组长度为16bytes
dst	void *	输出	指向密文数据缓冲区的指针

返回值说明

值	说明
0	解密成功
-1	解密失败

HAL_Aes128_Cfb_Encrypt

原型

```
int HAL_Aes128_Cfb_Encrypt(
_IN_ p_HAL_Aes128_t aes,
_IN_ const void *src,
_IN_ size_t length,
_OU_ void *dst);
```

接口说明

CFB模式的AES128加密接口函数, 使用此接口前必须先调用HAL_Aes128_Init()建立AES上下文数据结构体

参数length为分组数量(AES128一个分组的长度为128bits, 也就是16bytes), 而非字节数

用户在实现此函数时无需考虑padding问题, 因为SDK在调用此接口前已完成padding处理

参数说明

参数	数据类型	方向	说明
aes	p_HAL_Aes128_t	输入	调用HAL_Aes128_Init()时返回的上下文结构体指针
src	const void *	输入	指向明文数据缓冲区的指针
length	size_t	输入	明文数据的分组数量, AES128一个分组长度为16bytes
dst	void *	输出	指向密文数据缓冲区的指针

返回值说明

值	说明
0	加密成功
-1	加密失败

HAL_Aes128_Destroy

原型

```
int HAL_Aes128_Destroy(_IN_ p_HAL_Aes128_t aes);
```

接口说明

销毁AES加解密算法的上下文结构体, 释放内存资源

参数说明

参数	数据类型	方向	说明
aes	p_HAL_Aes128_t	输入	此参数应调用HAL_Aes128_Init()时返回的上下文结构体指针

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Aes128_Init

原型

```
p_HAL_Aes128_t HAL_Aes128_Init(
_IN_ const uint8_t *key,
_IN_ const uint8_t *iv,
_IN_ AES_DIR_t dir);
```

接口说明

初始化AES加解密算法的上下文结构体, 并根据dir参数完成AES算法的初始化. 用户可自定义结构体类型, 但结构体中应包含key, iv等上下文数据

参数说明

参数	数据类型	方向	说明
key	const uint8_t *	输入	AES秘钥数组首地址, 秘钥数组长度必须>=16bytes
iv	const uint8_t *	输入	AES初始向量数组首地址, 初始向量数组的长度必须>=16bytes
dir	AES_DIR_t	输入	指定AES算法用途 HAL_AES_ENCRYPTION: 表示用于加密 HAL_AES_DECRYPTION: 表示用于解密

```
typedef enum {
HAL_AES_ENCRYPTION = 0, // 用于加密
HAL_AES_DECRYPTION = 1, // 用于解密
} AES_DIR_t;
```

返回值说明

指向所初始化的AES加解密上下文结构体的指针. p_HAL_Aes128_t的类型定义如下所示:

```
typedef void *p_HAL_Aes128_t;
```

HAL_Awss_Close_Monitor

原型

```
void HAL_Awss_Close_Monitor(void);
```

接口说明

设置Wi-Fi网卡离开监听(Monitor)模式, 并开始以站点(Station)模式工作，并不再以Sniffer抓包

参数说明

void

返回值说明

void

HAL_Awss_Connect_Ap

原型

```
int HAL_Awss_Connect_Ap(
_IN_ uint32_t connection_timeout_ms,
_IN_ char ssid[HAL_MAX_SSID_LEN],
_IN_ char passwd[HAL_MAX_PASSWD_LEN],
_IN_OPT_ enum AWSS_AUTH_TYPE auth,
_IN_OPT_ enum AWSS_ENC_TYPE encry,
_IN_OPT_ uint8_t bssid[ETH_ALEN],
_IN_OPT_ uint8_t channel);
```

接口说明

要求Wi-Fi网卡连接指定热点(Access Point)的函数，bssid指定特定AP，另外bssid也可能为空或无效值（全0或全0xff）

参数说明

参数	数据类型	方向	说明
connection_timeout_ms	uint32_t	输入	连接AP的超时时间
ssid	char	输入	目的AP的SSID
passwd	char	输入	目的AP的PASSWORD
auth	enum	输入	目的AP的加密方式, HAL可以忽略
encry	enum	输入	目的AP的认证方式, HAL可以忽略
bssid	uint8_t	输入	目的AP的BSSID, 该字段可能为NULL或设置为全0
channel	uint8_t	输入	目的AP的信道, 该字段可以忽略

返回值说明

值	说明
0	连接AP和DHCP成功
-1	连接AP和DHCP失败

HAL_Awss_Get_Channelscan_Interval_Ms

原型

```
int HAL_Awss_Get_Channelscan_Interval_Ms(void);
```

接口说明

获取在每个信道(channel)上扫描的时间长度, 单位是毫秒，建议200ms~400ms,默认250ms

参数说明

void

返回值说明

时间长度, 单位是毫秒

HAL_Awss_Get_Connect_Default_Ssid_Timeout_Interval_Ms

原型

```
int HAL_Awss_Get_Connect_Default_Ssid_Timeout_Interval_Ms(void);
```

接口说明

获取配网服务(AWSS)超时时长到达之后, 去连接默认SSID时的超时时长, 单位是毫秒（该接口已经废弃，不再使用，不用对接）

参数说明

void

返回值说明

时时长, 单位是毫秒

HAL_Awss_Get_Conn_Encrypt_Type

原型

```
int HAL_Awss_Get_Conn_Encrypt_Type(void);
```

接口说明

获取零配, 热点配网和路由器配网的安全等级

参数说明

void

返回值说明

值	说明
3	aes128cfb with aes-key per product and aes-iv = random
4	aes128cfb with aes-key per device and aes-iv = random
5	aes128cfb with aes-key per manufacture and aes-iv = random
others	无效

HAL_Awss_Get_Encrypt_Type

原型

```
int HAL_Awss_Get_Encrypt_Type(void);
```

接口说明

获取smartconfig服务的安全等级

参数说明

void

返回值说明

值	说明
0	open (no encrypt)
1	aes256cfb with default aes-key and aes-iv
2	aes128cfb with default aes-key and aes-iv
3	aes128cfb with aes-key per product and aes-iv = 0
4	aes128cfb with aes-key per device and aes-iv = 0
5	es128cfb with aes-key per manufacture and aes-iv = 0
others	invalid

HAL_Awss_Get_Timeout_Interval_Ms

原型

```
int HAL_Awss_Get_Timeout_Interval_Ms(void);
```

接口说明

获取配网服务(AWSS)的超时时间长度, 单位是毫秒，建议60s或60000ms

参数说明

void

返回值说明

超时时长, 单位是毫秒

HAL_Awss_Open_Monitor

原型

```
void HAL_Awss_Open_Monitor(_IN_ awss_recv_80211_frame_cb_t cb);
```

接口说明

设置Wi-Fi网卡工作在监听(Monitor)模式, 并在收到802.11帧的时候调用被传入的回调函数

参数说明

参数	数据类型	方向	说明
cb	awss_recv_80211_frame_cb_t	输入	回调函数指针, 当WiFi接收到帧时会调用此函数

```
/**
* @brief 802.11帧的处理函数, 可以将802.11 Frame传递给这个函数
*
* @param[in] buf @n 80211 frame buffer, or pointer to struct ht40_ctrl
* @param[in] length @n 80211 frame buffer length
* @param[in] link_type @n AWSS_LINK_TYPE_NONE for most rtos HAL,
* and for linux HAL, do the following step to check
* which header type the driver supported.
* @param[in] with_fcs @n 80211 frame buffer include fcs(4 byte) or not
* @param[in] rssi @n rssi of packet, range of `[-127, -1]`
*/
typedef int (*awss_recv_80211_frame_cb_t)(char *buf, int length,
enum AWSS_LINK_TYPE link_type, int with_fcs, signed char rssi);
```

返回值说明

void

HAL_Awss_Switch_Channel

原型

```
void HAL_Awss_Switch_Channel(
_IN_ char primary_channel,
_IN_OPT_ char secondary_channel,
_IN_OPT_ uint8_t bssid[ETH_ALEN]);
```

接口说明

设置Wi-Fi网卡切换到指定的信道(channel)上

参数说明

参数	数据类型	方向	说明
primary_channel	char	输入	首选信道
secondary_channel	char	输入	辅助信道(信道带宽为40MHz时才会使用, 信道宽度为20MHz是可以忽略该参数)
bssid	uint8_t	输入	次参数已废弃, 可以忽略

返回值说明

void

HAL_RF433_Get_Rssi_Dbm

原型

```
int HAL_RF433_Get_Rssi_Dbm(void);
```

接口说明

获取RF433的接收信号强度(RSSI)(该API已经废弃，不再使用，不用对接)

参数说明

void

返回值说明

信号强度数值，单位是dBm

HAL_Sys_Net_Is_Ready

原型

```
int HAL_Sys_Net_Is_Ready(void);
```

接口说明

检查系统网络是否可用（设备是否已经成功获得IP地址并且当前IP地址可用）

参数说明

void

返回值说明

值	说明
0	网络不可用
1	网络可用

HAL_Wifi_Enable_Mgmt_Frame_Filter

原型

```
int HAL_Wifi_Enable_Mgmt_Frame_Filter(
_IN_ uint32_t filter_mask,
_IN_OPT_ uint8_t vendor_oui[3],
_IN_ awss_wifi_mgmt_frame_cb_t callback);
```

接口说明

在站点(Station)模式下使能或禁用对特定管理帧的过滤（只接受包含特定OUI的管理帧）

参数说明

参数	数据类型	方向	说明
filter_mask	uint32_t	输入	帧过滤参数
vendor_oui	uint8_t	输入	WiFi联盟分配的厂商OUI, 如果OUI为NULL, 表示不对OUI过滤, 反之要根据OUI过滤
callback	awss_wifi_mgmt_frame_cb_t	输入	用于接收802.11帧或者信息元素(IE)的回调函数

返回值说明

值	说明
= 0	发送成功
= -1	发送失败
= -2	不支持

HAL_Wifi_Get_Ap_Info

原型

```
int HAL_Wifi_Get_Ap_Info(
_OU_ char ssid[HAL_MAX_SSID_LEN],
_OU_ char passwd[HAL_MAX_PASSWD_LEN],
_OU_ uint8_t bssid[ETH_ALEN]);
```

接口说明

获取所连接的热点(Access Point)的信息

参数说明

参数	数据类型	方向	说明
ssid	char	输出	AP的SSID, 该参数可

			能为NULL
passwd	char	输出	AP的Password, 该参数为NULL
bssid	uint8_t	输出	AP的BSSID, 该参数可能为NULL，如果bssid不对将导致零配设备发现失败

返回值说明

值	说明
= 0	操作成功
= -1	操作失败

HAL_Wifi_Get_IP

原型

```
uint32_t HAL_Wifi_Get_IP(_OU_ char ip_str[HAL_IP_LEN], _IN_ const char *ifname);
```

接口说明

获取Wi-Fi网口的IP地址, 点分十进制格式保存在字符串数组出参, 二进制格式则作为返回值, 并以网络字节序(大端)表达

参数说明

参数	数据类型	方向	说明
ip_str	char[]	输出	存放点分十进制格式的IP地址字符串的数组
ifname	const char*	输入	指定Wi-Fi网络接口的名字（如果只有一个网口，可以忽略此参数，该参数可能为NULL）

返回值说明

二进制形式的IP地址, 以网络字节序(大端)组织

HAL_Wifi_Get_Mac

原型

```
char *HAL_Wifi_Get_Mac(_OU_ char mac_str[HAL_MAC_LEN]);
```

接口说明

获取Wi-Fi网口的MAC地址, 格式应当是”XX:XX:XX:XX:XX:XX”

参数说明

参数	数据类型	方向	说明
mac_str	char	输出	指向缓冲区数组起始位置的字符指针

返回值说明

指向缓冲区数组起始位置的字符指针

HAL_Wifi_Low_Power

原型

```
int HAL_Wifi_Low_Power(_IN_ int timeout_ms);
```

接口说明

使WiFi模组进入省电模式, 并持续一段时间（该API未用，可以暂时不用对接）

参数说明

参数	数据类型	方向	说明
timeout_ms	int	输入	指定在多长时间内, WiFi模组都处于省电模式, 单位是毫秒

返回值说明

值	说明
0	设置成功
-1	设置失败

HAL_Wifi_Scan

原型

```
int HAL_Wifi_Scan(awss_wifi_scan_result_cb_t cb);
```

接口说明

启动一次WiFi的空中扫描, 该API是一个阻塞操作, 扫描没有完成不能结束. 所有的AP列表收集完成后, 一个一个通过回调函数告知AWSS，最好不要限制AP的数量，否则可能导致中文GBK编码的SSID热点配网失败

参数说明

参数	数据类型	方向	说明
cb	awss_wifi_scan_result_cb_t	输入	扫描通知回调函数

返回值说明

值	说明
0	扫描正常结束
-1	其他情况

HAL_Wifi_Send_80211_Raw_Frame

原型

```
int HAL_Wifi_Send_80211_Raw_Frame(_IN_ enum HAL_Awss_Frame_Type type,
_IN_ uint8_t *buffer, _IN_ int len);
```

接口说明

在当前信道(channel)上以基本数据速率(1Mbps)发送裸的802.11帧(raw 802.11 frame)

参数说明

参数	数据类型	方向	说明
type	enum HAL_Awss_Frame_Type	输入	查看HAL_Awss_Frame_Type_t定义, 目前只支持FRAME_BEACON和FRAME_PROBE_REQ
buffer	uint8_t *	输入	80211裸数据帧, 包括完整的MAC头和FCS域
len	int	输入	80211裸帧字节长度

```
/* 80211帧类型定义 */
typedef enum HAL_Awss_Frame_Type {
FRAME_ACTION,
FRAME_BEACON,
FRAME_PROBE_REQ,
FRAME_PROBE_RESPONSE,
FRAME_DATA
} HAL_Awss_Frame_Type_t;
```

返回值说明

值	说明
= 0	发送成功
= -1	发送失败

HAL_Awss_Open_Ap

原型

```
int HAL_Awss_Open_Ap(const char *ssid, const char *passwd, int beacon_interval, int hide);
```

接口说明

开启设备热点（SoftAP模式）

参数说明

参数	数据类型	方向	说明
ssid	const char *	输入	热点的ssid字符
passwd	const char *	输入	热点的passwd字符
beacon_interval	int	输入	热点的Beacon广播周期（广播间隔）
hide	int	输入	0, 非隐藏, 其它值：隐藏；

返回值说明

值	说明
0	success
-1	unsupported
-2	failure with system error
-3	failure with no memory
-4	failure with invalid parameters

HAL_Awss_Close_Ap

原型

```
int HAL_Awss_Close_Ap(void);
```

接口说明

关闭设备热点

参数说明

void

返回值说明

值	说明
0	success
-1	unsupported
-2	failure

本地通信HAL

本地通信相关HAL接口详解

HAL_UDP_bindtodevice

原型

```
int HAL_UDP_bindtodevice(_IN_ intptr_t fd,
_IN_ const char *ifname);
```

接口说明

绑定UDP socket到指定接口, 只接收来自该接口的数据包

参数说明

参数	数据类型	方向	说明
fd	intptr_t	输入	指定用来绑定的UDP socket
ifname	const char *	输入	指定用来绑定socket的网络接口名字

返回值说明

值	说明
<0	绑定异常或失败
= 0	绑定成功

HAL_UDP_close

原型

```
void HAL_UDP_close(_IN_ intptr_t p_socket);
```

接口说明

销毁指定的UDP连接, 释放资源

参数说明

参数	数据类型	方向	说明
p_socket	intptr_t	输入	UDP socket句柄

返回值说明

void

HAL_UDP_close_without_connect

原型

```
int HAL_UDP_close_without_connect(_IN_ intptr_t sockfd);
```

接口说明

销毁指定的UDP连接, 释放资源

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	UDP socket句柄

返回值说明

值	说明
<0	操作失败
= 0	操作成功

HAL_UDP_create_without_connect

原型

```
intptr_t HAL_UDP_create_without_connect(_IN_ const char *host, _IN_ unsigned short port);
```

接口说明

创建一个本地的UDP socket, 但并不发起任何网络交互

参数说明

参数	数据类型	方向	说明
host	const char *	输入	UDP目的地址
port	unsigned short	输入	UDP目的端口

返回值说明

值	说明
<0	创建失败
>= 0	创建成功, 返回值为UDP socket句柄

HAL_UDP_joinmulticast

原型

```
int HAL_UDP_joinmulticast(_IN_ intptr_t sockfd,
_IN_ char *p_group);
```

接口说明

在指定的UDP socket上发送加入组播组的请求

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	指定用来发送组播请求的UDP socket
p_group	char *	输入	指定要加入的组播组名称

返回值说明

值	说明
<0	发送过程中出现错误或异常
= 0	发送成功

HAL_UDP_recvfrom

原型

```
int HAL_UDP_recvfrom(_IN_ intptr_t sockfd,
_OU_ NetworkAddr *p_remote,
_OU_ unsigned char *p_data,
_IN_ unsigned int datalen,
_IN_ unsigned int timeout_ms);
```

接口说明

从指定的UDP句柄接收指定长度数据到缓冲区, 阻塞时间不超过指定时长, 且指定长度若接收完需提前返回, 源地址保存在p_remote参数中

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	UDP socket句柄
p_remote	NetworkAddr *	输出	指向存放源网络地址的指针
p_data	unsigned char *	输出	指向数据接收缓冲区的指针
datalen	unsigned int	输入	接收缓冲区的字节大小
timeout_ms	unsigned int	输入	阻塞的超时时间, 单位ms

返回值说明

值	说明
<0	接收过程中出现错误或异常
= 0	在指定的timeout_ms时间内, 没有接收到任何数据
>0	在指定的timeout_ms时间内, 实际接收到的数据字节数

HAL_UDP_sendto

原型

```
int HAL_UDP_sendto(_IN_ intptr_t          sockfd,
_IN_ const NetworkAddr *p_remote,
_IN_ const unsigned char *p_data,
_IN_ unsigned int datalen,
_IN_ unsigned int timeout_ms);
```

接口说明

向指定UDP句柄发送指定长度的数据, 阻塞时间不超过指定时长, 且指定长度若发送完需提前返回

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	UDP socket句柄

p_remote	const NetworkAddr *	输入	指向目标网络地址的指针
p_data	const unsigned char *	输入	指数据发送缓冲区的指针
datalen	unsigned int	输入	待发送数据的字节长度
timeout_ms	unsigned int	输入	阻塞的超时时间, 单位 ms

返回值说明

值	说明
<0	发送过程中出现错误或异常
= 0	在指定的timeout_ms时间内, 没有任何数据发送成功
>0	在指定的timeout_ms时间内, 实际发送的数据字节数

CoAP连云HAL

CoAP上云相关HAL接口详解

HAL_DTLSSession_create

原型

```
DTLSContext *HAL_DTLSSession_create(coap_dtls_options_t  *p_options);
```

接口说明

根据参数p_options指定的证书, 服务器地址和端口建立DTLS连接, 并返回DTLS会话句柄

参数说明

参数	数据类型	方向	说明

p_options	coap_dtls_options_t *	输入	指向 coap_dtls_options_t 结构体类型选项数据的指针

```
typedef struct {
unsigned char *p_ca_cert_pem; // 指向PEM编码的X.509证书的指针
char *p_host; // 指向DTLS服务器网络地址的指针
unsigned short port; // DTLS服务器端口
} coap_dtls_options_t;
```

返回值说明

DTLS会话句柄

HAL_DTLSSession_free

原型

```
unsigned int HAL_DTLSSession_free(DTLSContext *context);
```

接口说明

销毁由参数context指定的DTLS会话, 释放资源

参数说明

参数	数据类型	方向	说明
context	DTLSContext *	输入	DTLS会话句柄

返回值说明

值	说明
0	操作成功
>0	操作错误码

DTLS错误码:

```
define DTLS_ERROR_BASE                (1<<24)
define DTLS_INVALID_PARAM (DTLS_ERROR_BASE | 1) // 无效参数
define DTLS_INVALID_CA_CERTIFICATE (DTLS_ERROR_BASE | 2) // 无效证书
define DTLS_HANDSHAKE_IN_PROGRESS (DTLS_ERROR_BASE | 3) // 正在握手
define DTLS_HANDSHAKE_FAILED (DTLS_ERROR_BASE | 4) // 握手失败
define DTLS_FATAL_ALERT_MESSAGE (DTLS_ERROR_BASE | 5) // 致命警告消息
define DTLS_PEER_CLOSE_NOTIFY (DTLS_ERROR_BASE | 6) // 对方打开连接
define DTLS_SESSION_CREATE_FAILED (DTLS_ERROR_BASE | 7) // 会话创建失败
define DTLS_READ_DATA_FAILED (DTLS_ERROR_BASE | 8) // 数据读取失败
```

HAL_DTLSSession_read

原型

```
unsigned int HAL_DTLSSession_read(DTLSContext       *context,
unsigned char *p_data,
unsigned int *p_datalen,
unsigned int timeout_ms);
```

接口说明

从指定DTLS连接中读取数据, 此接口为同步接口, 在超前前读取到数据则立即返回, 否则在超时时间到时才解除阻塞并返回

参数说明

参数	数据类型	方向	说明
context	DTLSContext *	输入	DTLS会话句柄
p_data	unsigned char *	输出	指向接收缓冲区的指针
p_datalen	unsigned int *	输出	指向接收数据长度变量的指针
timeout_ms	unsigned int	输入	超时时间

返回值说明

值	说明
0	操作成功
>0	操作错误码

HAL_DTLSSession_write

原型

```
unsigned int HAL_DTLSSession_write(DTLSContext *context,
const unsigned char *p_data,
unsigned int *p_datalen);
```

接口说明

向指定DTLS连接写入数据

参数说明

参数	数据类型	方向	说明
context	DTLSContext *	输入	DTLS会话句柄
p_data	unsigned char *	输入	指向发送数据缓冲区的指针
p_datalen	unsigned int *	输入	指向发送数据长度变量的指针, 用于指定发送字节长度

返回值说明

值	说明
0	操作成功
>0	操作错误码

HAL_UDP_close_without_connect

原型

```
int HAL_UDP_close_without_connect(intptr_t sockfd)
```

接口说明

销毁sockfd指定的UDP socket，释放资源

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	UDP socket句柄

返回值说明

值	说明
<0	操作失败
= 0	操作成功

HAL_UDP_create

原型

```
intptr_t HAL_UDP_create(_IN_ char *host, _IN_ unsigned short port);
```

接口说明

建立指定目的地址和目的端口的UDP连接

参数说明

参数	数据类型	方向	说明
host	const char *	输入	UDP目的地址
port	unsigned short	输入	UDP目的端口

返回值说明

值	说明
<0	创建失败
>= 0	创建成功, 返回值为UDP socket句柄

HAL_UDP_read

原型

```
int HAL_UDP_read(
_IN_ intptr_t p_socket,
_OU_ unsigned char *p_data,
_OU_ unsigned int datalen);
```

接口说明

从指定UDP连接中读取数据, 此接口为阻塞接口

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	指定用来发送组播请求的UDP socket
p_group	char *	输入	指定要加入的组播组名称

返回值说明

值	说明
<0	UDP连接出现错误
= 0	发送成功
>0	读取到的数据字节数

HAL_UDP_readTimeout

原型

```
int HAL_UDP_readTimeout(
_IN_ intptr_t p_socket,
_OU_ unsigned char *p_data,
_IN_ unsigned int datalen,
_IN_ unsigned int timeout_ms);
```

接口说明

从指定的UDP句柄读取指定长度数据到缓冲区, 阻塞时间不超过指定时长, 若读取到指定长度数据完需立即返回, 调用该接口之前需要调用HAL_UDP_connect()设置好源地址和端口

参数说明

返回值说明

HAL_UDP_recv

原型

```
int HAL_UDP_recv(_IN_ intptr_t sockfd,
_OU_ unsigned char *p_data,
_IN_ unsigned int datalen,
_IN_ unsigned int timeout_ms);
```

接口说明

从指定的UDP句柄接收指定长度数据到缓冲区, 阻塞时间不超过指定时长, 且指定长度若接收完需提前返回, 调用该接口之前需要调用HAL_UDP_connect()设置好源地址和端口

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	UDP socket句柄
p_data	unsigned char *	输出	指向数据接收缓冲区的指针
datalen	unsigned int	输入	接收缓冲区的字节大小
timeout_ms	unsigned int	输入	阻塞的超时时间, 单位ms

返回值说明

值	说明
<0	接收过程中出现错误或异常
= 0	在指定的timeout_ms时间内, 没有接收到任何数据
>0	在指定的timeout_ms时间内, 实际接收到的数据字

	节数

HAL_UDP_send

原型

```
int HAL_UDP_sendto(_IN_ intptr_t          sockfd,
_IN_ const NetworkAddr *p_remote,
_IN_ const unsigned char *p_data,
_IN_ unsigned int datalen,
_IN_ unsigned int timeout_ms);
```

接口说明

向指定的UDP句柄发送指定缓冲区的指定长度, 阻塞时间不超过指定时长, 且指定长度若发送完需提前返回, 调用该接口之前需要调用HAL_UDP_connect()设置好目的地址和端口

参数说明

参数	数据类型	方向	说明
sockfd	intptr_t	输入	UDP socket句柄
p_data	const unsigned char *	输入	指数据发送缓冲区的指针
datalen	unsigned int	输入	待发送数据的字节长度
timeout_ms	unsigned int	输入	阻塞的超时时间, 单位ms

返回值说明

值	说明
<0	发送过程中出现错误或异常
= 0	在指定的timeout_ms时间内, 没有任何数据发送成功
>0	在指定的timeout_ms时间内, 实际发送的数据字节数

HAL_UDP_write

原型

```
int HAL_UDP_write(
_IN_ intptr_t p_socket,
_IN_ const unsigned char *p_data,
_IN_ unsigned int datalen);
```

接口说明

向指定UDP句柄写入指定字节长度的数据

参数说明

参数	数据类型	方向	说明
p_socket	intptr_t	输入	用于标识连接的描述符
p_data	const unsigned char *	输入	指向数据发送缓冲区的指针
datalen	unsigned int	输入	待写入数据的字节长度

返回值说明

值	说明
<0	UDP连接发生错误
= 0	EOF, 文件已结束
>0	实际写入的字节数