

IoT Datahub Embedded SDK

by Dasudian

Contents

1 IoT Datahub Embedded SDK	1
2 更新日志	1
3 FAQ	1
4 结构体索引	2
4.1 结构体	2
5 文件索引	2
5.1 文件列表	2
6 结构体说明	2
6.1 datahub_client结构体 参考	2
6.1.1 详细描述	3
6.1.2 结构体成员变量说明	3
6.2 datahub_message结构体 参考	4
6.2.1 详细描述	4
6.2.2 结构体成员变量说明	4
6.3 datahub_options结构体 参考	4
6.3.1 详细描述	5
6.3.2 结构体成员变量说明	5
6.4 lenstring_s结构体 参考	6
6.4.1 详细描述	6
6.4.2 结构体成员变量说明	6
6.5 Network结构体 参考	7
6.5.1 详细描述	7
6.5.2 结构体成员变量说明	7
6.6 Timer结构体 参考	8
6.6.1 详细描述	8
6.6.2 结构体成员变量说明	8
7 文件说明	8
7.1 DataHubClient.h 文件参考	8
7.1.1 宏定义说明	10
7.1.2 类型定义说明	11
7.1.3 枚举类型说明	12
7.1.4 函数说明	13
7.2 DatahubNetwork.h 文件参考	16
7.2.1 宏定义说明	17
7.2.2 类型定义说明	17
7.2.3 函数说明	17
7.3 DatahubTimer.h 文件参考	19
7.3.1 类型定义说明	19

7.3.2 函数说明	20
7.4 FAQ.md 文件参考	21
7.5 release_note.txt 文件参考	21
索引	23

1 IoT Datahub Embedded SDK

SDK提供了一个简单的基于MQTT协议与服务器交互的方法

SDK基于MQTT协议,传输实时的消息到大数点IoT云服务器

你可以收集设备上的数据发送到云上;也可以订阅某个topic,来接收云服务器推送的消息

如何使用SDK:

- 1: 创建一个客户端实例
- 2: 如果想接收消息,那么就订阅某个topic
- 3: 或者发送消息到服务器
- 4: 退出时,销毁该客户端

功能:

- 1: 当连接丢失时,SDK会尝试自动重连
- 2: SDK为非线程安全,即不能在多个Task中使用客户端

2 更新日志

Author	Date	Version	Note
Eden Wang	7/4/2017	3.3.0	添加更多的调试信息; 修复重复订阅失败的bug; 使用"\n"作为换行符, 避免与GPS命令发生冲突
Eden Wang	6/17/2017	3.2.1	修复bug: 数据发送失败导致客户端永久"在线"
Eden Wang	6/15/2017	3.2.0	增加支持STM32F2,STM32F3;支持iar和keil
Eden Wang	5/17/2017	3.1.0	暴露cleansession和订阅qos; 修复自动重连不稳定的bug; 修复其他bug
Eden Wang	4/6/2017	3.0.0	根据SDK的标准修改API
Eden Wang	3/31/2017	2.2.0	暴露订阅的qos; 修复bug: 服务器无法推送QOS2消息
Eden Wang	3/24/2017	2.1.0	增加支持STM32F4
Eden Wang	3/23/2017	2.0.1	暴露datahub_yield()接口;修复不发送数据也无法接收数据的bug
Eden Wang	2/23/2017	2.0.0	删除与操作系统有关的源码, 适配不同的操作系统;新增定时器接口;不再支持自动重连
Eden Wang	2/10/2017	1.1.1	修复无法重连的bug
Eden Wang	1/18/2017	1.1.0	新增网络驱动接口
Jack Liu	1/4/2017	1.0.0	第一版本

3 FAQ

Q: ..axf: Error: L6218E: Undefined symbol datahub_printf (referred from xxx.o).

A: 未实现datahub_printf().datahub_printf()用于输出调试信息,对于开发工作十分重要.请将下列代码加入源文件中,即可链接成功.如果您重定向了输出,可以使能vprintf()

```
#include <stdarg.h>
```

```
void datahub_printf(const char *format, ...)
{
    va_list ap;

    va_start(ap, format);
    /* 如果重定向了输出,可以取消注释 */
    // vprintf(format, ap);
    va_end(ap);
}
```

Q: Error: L6367E: datahubclient.o attributes are not compatible with the provided attributes . Tag_CPU_arch = ARMv7E-M (=13)

A: 这是因为SDK与您的CPU架构不兼容.请在lib目录下找到合适的SDK库.如果没有提供,请联系我们

Q: 出现HardFault_Handler的错误

A: 很可能是定义的堆栈大小不够, 堆栈大小至少需要256B, 否则堆栈指针可能会溢出, 出现HardFault_Handler的错误

4 结构体索引

4.1 结构体

这里列出了所有结构体, 并附带简要说明:

datahub_client	描述客户端的结构体, 由SDK维护, 用户无需关心	2
datahub_message	消息结构体, 用于描述一个消息	4
datahub_options	选项结构体, 用于设置MQTT协议的相关选项	4
lenstring_s	字符串, 不以'\0'结尾	6
Network	描述网络的结构体	7
Timer	描述定时器的结构体	8

5 文件索引

5.1 文件列表

这里列出了所有文件, 并附带简要说明:

DataHubClient.h	8
DatahubNetwork.h	16
DatahubTimer.h	19

6 结构体说明

6.1 datahub_client结构体 参考

描述客户端的结构体, 由SDK维护, 用户无需关心

```
#include <DataHubClient.h>
```

成员变量

- char * [instance_id](#)
- char * [instance_key](#)
- char * [client_name](#)
- char * [client_id](#)
- [datahub_options](#) * [options](#)
- [Network](#) * [network](#)
- Client [c](#)

6.1.1 详细描述

描述客户端的结构体, 由SDK维护, 用户无需关心

由于SDK非线程安全, 不能在多个Task中使用datahub_client实例, 只能在一个Task中 使用

6.1.2 结构体成员变量说明

6.1.2.1 instance_id

```
char* instance_id
```

6.1.2.2 instance_key

```
char* instance_key
```

6.1.2.3 client_name

```
char* client_name
```

6.1.2.4 client_id

```
char* client_id
```

6.1.2.5 options

```
datahub\_options* options
```

6.1.2.6 network

```
Network* network
```

6.1.2.7 c

Client c

该结构体的文档由以下文件生成:

- [DataHubClient.h](#)

6.2 datahub_message结构体 参考

消息结构体,用于描述一个消息

```
#include <DataHubClient.h>
```

成员变量

- void * [payload](#)
- unsigned int [payload_len](#)

6.2.1 详细描述

消息结构体,用于描述一个消息

6.2.2 结构体成员变量说明

6.2.2.1 payload

void* payload

指向待发送的消息

6.2.2.2 payload_len

unsigned int payload_len

消息的长度

该结构体的文档由以下文件生成:

- [DataHubClient.h](#)

6.3 datahub_options结构体 参考

选项结构体,用于设置MQTT协议的相关选项

```
#include <DataHubClient.h>
```

成员变量

- char * [host](#)
大数点MQTT服务器域名或地址
- int [port](#)
大数点MQTT服务器开放端口号
- int [cleansession](#)
是否保存客户端的会话信息.
- void * [context](#)
传递给回调函数message_received和connection_status_changed,即回调函数的第一个参数context

- `MESSAGE_RECEIVED * message_received`
收到消息后的回调函数
- `CONNECTION_STATUS_CHANGED * connection_status_changed`
当网络状态发生变化时, 通知用户的回调函数
- `unsigned char * sendbuf`
发送缓冲区的首地址
- `int sendbuf_size`
发送缓冲区的大小
- `unsigned char * readbuf`
接收缓冲区的首地址
- `int readbuf_size`
接收缓冲区的大小

6.3.1 详细描述

选项结构体,用于设置MQTT协议的相关选项

6.3.2 结构体成员变量说明

6.3.2.1 host

`char* host`

大数点MQTT服务器域名或地址

默认为 `try.iotdatahub.net`

6.3.2.2 port

`int port`

大数点MQTT服务器开放端口号

默认为1883

6.3.2.3 cleansession

`int cleansession`

是否保存客户端的会话信息.

`cleansession`为`DATAHUB_FALSE`, 当客户端断线或下线后, 保存客户端订阅的 `topic`和发送给客户端的所有消息.

`cleansession`为`DATAHUB_TRUE`, 当客户端断线或下线后, 不保留客户端订阅的 `topic`和发送给客户端的任何消息.

6.3.2.4 context

`void* context`

传递给回调函数`message_received`和`connection_status_changed`, 即回调函数的第一个参数`context`

默认为`NULL`

6.3.2.5 message_received

`MESSAGE_RECEIVED* message_received`

收到消息后的回调函数

默认为`NULL`

6.3.2.6 connection_status_changed

[CONNECTION_STATUS_CHANGED](#)* connection_status_changed

当网络状态发生变化时, 通知用户的回调函数

默认为NULL

6.3.2.7 sendbuf

unsigned char* sendbuf

发送缓冲区的首地址

必须由用户手动设置, 默认为NULL

6.3.2.8 sendbuf_size

int sendbuf_size

发送缓冲区的大小

必须由用户手动设置, 默认为0

6.3.2.9 readbuf

unsigned char* readbuf

接收缓冲区的首地址

必须由用户手动设置, 默认为NULL

6.3.2.10 readbuf_size

int readbuf_size

接收缓冲区的大小

必须由用户手动设置, 默认为0

该结构体的文档由以下文件生成:

- [DataHubClient.h](#)

6.4 lenstring_s结构体 参考

字符串, 不以'\0'结尾

```
#include <DataHubClient.h>
```

成员变量

- char * [data](#)
- int [len](#)

6.4.1 详细描述

字符串, 不以'\0'结尾

6.4.2 结构体成员变量说明

6.4.2.1 data

char* data

字符串的起始位置

6.4.2.2 len

int len

字符串的长度

该结构体的文档由以下文件生成:

- [DataHubClient.h](#)

6.5 Network结构体 参考

描述网络的结构体

```
#include <DatahubNetwork.h>
```

成员变量

- void * [data](#)
一个指向用户自定义的结构体指针.
- int(* [read](#))([Network](#) *Net, unsigned char *buffer, int len, int timeout_ms)
读取网络数据的函数指针.
- int(* [write](#))([Network](#) *Net, unsigned char *buffer, int len, int timeout_ms)
发送网络数据的函数指针.
- void(* [disconnect](#))([Network](#) *Net)
断开网络的函数指针.

6.5.1 详细描述

描述网络的结构体

注解

如果网络接口和client实例在不同的Task中运行, 则网络接口任务优先级要高于调用client实例的Task; 否则, 调用所有接口都会超时, 数据却可能发送成功

6.5.2 结构体成员变量说明

6.5.2.1 data

void* data

一个指向用户自定义的结构体指针.

由函数NewNetwork初始化. 函数ConnectNetwork, read, write, 和disconnect可使用

6.5.2.2 read

int(* read)([Network](#) *Net, unsigned char *buffer, int len, int timeout_ms)

读取网络数据的函数指针.

SDK将调用该接口从网络收数据. 需要用户实现并赋值

6.5.2.3 write

```
int(* write) (Network *Net, unsigned char *buffer, int len, int timeout_ms)
```

发送网络数据的函数指针.

SDK将调用该接口向网络发送数据. 需要用户实现并赋值

6.5.2.4 disconnect

```
void(* disconnect) (Network *Net)
```

断开网络的函数指针.

SDK将调用该接口断开与服务器的连接. 需要用户实现并赋值

该结构体的文档由以下文件生成:

- [DatahubNetwork.h](#)

6.6 Timer结构体 参考

描述定时器的结构体

```
#include <DatahubTimer.h>
```

成员变量

- unsigned long [end](#)

6.6.1 详细描述

描述定时器的结构体

6.6.2 结构体成员变量说明

6.6.2.1 end

```
unsigned long end
```

用于存储定时器的超时时刻

该结构体的文档由以下文件生成:

- [DatahubTimer.h](#)

7 文件说明

7.1 DataHubClient.h 文件参考

```
#include "MQTTClient.h"
```

结构体

- struct [datahub_message](#)

消息结构体,用于描述一个消息

- struct [lenstring_s](#)
字符串,不以'\0'结尾
- struct [datahub_options](#)
选项结构体,用于设置MQTT协议的相关选项
- struct [datahub_client](#)
描述客户端的结构体,由SDK维护,用户无需关心

宏定义

- #define [DATAHUB_FALSE](#) 0
- #define [DATAHUB_TRUE](#) 1
- #define [DEFAULT_HOST](#) "try.iotdatahub.net"
- #define [DEFAULT_PORT](#) 1883
- #define [DEFAULT_CLEAN_SESSION](#) [DATAHUB_FALSE](#)
- #define [DEFAULT_CONTEXT](#) NULL
- #define [DEFAULT_CALLBACK](#) NULL
- #define [DEFAULT_BUF](#) NULL
- #define [DEFAULT_BUF_SIZE](#) 0
- #define [DATAHUB_OPTIONS_INITIALIZER](#)
- #define [DATAHUB_MESSAGE_INITIALIZER](#)

类型定义

- typedef struct [datahub_message](#) [datahub_message](#)
消息结构体,用于描述一个消息
- typedef struct [lenstring_s](#) [lenstring](#)
字符串,不以'\0'结尾
- typedef void [MESSAGE_RECEIVED](#)(void *context, [lenstring](#) *topic, [datahub_message](#) *msg)
收到消息后的回调函数
- typedef void [CONNECTION_STATUS_CHANGED](#)(void *context, int isconnected)
当网络状态发生变化时,通知用户的回调函数
- typedef struct [datahub_options](#) [datahub_options](#)
选项结构体,用于设置MQTT协议的相关选项
- typedef struct [datahub_client](#) [datahub_client](#)
描述客户端的结构体,由SDK维护,用户无需关心

枚举

- enum [datahub_error_code_s](#) {
[ERROR_NONE](#) = 0, [ERROR_ILLEGAL_PARAMETERS](#) = -1, [ERROR_DISCONNECTED](#) = -2, [ERROR_UNACCEPT_PROTOCOL_VERSION](#) = -3,
[ERROR_IDENTIFIER_REJECTED](#) = -4, [ERROR_SERVER_UNAVAILABLE](#) = -5, [ERROR_BAD_USERNAME_OR_PASSWD](#) = -6, [ERROR_UNAUTHORIZED](#) = -7,
[ERROR_AUTHORIZED_SERVER_UNAVAILABLE](#) = -8, [ERROR_OPERATION_FAILURE](#) = -9, [ERROR_MESSAGE_TOO_BIG](#) = -10, [ERROR_NETWORK_UNREACHABLE](#) = -11,
[ERROR_TIMEOUT](#) = -12, [ERROR_ILLEGAL_OPTION](#) = -200 }
 错误码

函数

- int `datahub_create` (`datahub_client` *client, char *instance_id, char *instance_key, char *client_name, char *client_id, `Network` *network, `datahub_options` *options)
该函数创建一个客户端实例,该实例可用于连接大数点MQTT服务器
- int `datahub_sendrequest` (`datahub_client` *client, char *topic, `datahub_message` *message, int qos, int timeout)
同步发送消息
- int `datahub_subscribe` (`datahub_client` *client, char *topic, int qos, int timeout)
同步订阅主题
- int `datahub_unsubscribe` (`datahub_client` *client, char *topic, int timeout)
同步取消订阅topic
- void `datahub_destroy` (`datahub_client` *client)
销毁客户端,销毁后连接不存在,不能收发数据
- int `datahub_yield` (`datahub_client` *client, int timeout_ms)
被动接收消息情况下保持和服务器的连接

7.1.1 宏定义说明

7.1.1.1 DATAHUB_FALSE

```
#define DATAHUB_FALSE 0
```

7.1.1.2 DATAHUB_TRUE

```
#define DATAHUB_TRUE 1
```

7.1.1.3 DEFAULT_HOST

```
#define DEFAULT_HOST "try.iotdatahub.net"
```

7.1.1.4 DEFAULT_PORT

```
#define DEFAULT_PORT 1883
```

7.1.1.5 DEFAULT_CLEAN_SESSION

```
#define DEFAULT_CLEAN_SESSION DATAHUB_FALSE
```

7.1.1.6 DEFAULT_CONTEXT

```
#define DEFAULT_CONTEXT NULL
```

7.1.1.7 DEFAULT_CALLBACK

```
#define DEFAULT_CALLBACK NULL
```

7.1.1.8 DEFAULT_BUF

```
#define DEFAULT_BUF NULL
```

7.1.1.9 DEFAULT_BUF_SIZE

```
#define DEFAULT_BUF_SIZE 0
```

7.1.1.10 DATAHUB_OPTIONS_INITIALIZER

```
#define DATAHUB_OPTIONS_INITIALIZER
```

值:

```
{\
    DEFAULT_HOST,\
    DEFAULT_PORT,\
    DEFAULT_CLEAN_SESSION,\
    DEFAULT_CONTEXT,\
    DEFAULT_CALLBACK,\
    DEFAULT_CALLBACK,\
    DEFAULT_BUF,\
    DEFAULT_BUF_SIZE,\
    DEFAULT_BUF,\
    DEFAULT_BUF_SIZE,\
}
```

选项的初始化宏

7.1.1.11 DATAHUB_MESSAGE_INITIALIZER

```
#define DATAHUB_MESSAGE_INITIALIZER
```

值:

```
{\
    "",\
    1\
}
```

消息的初始化宏,默认为""

7.1.2 类型定义说明

7.1.2.1 datahub_message

```
typedef struct datahub_message datahub_message
```

消息结构体,用于描述一个消息

7.1.2.2 lenstring

```
typedef struct lenstring_s lenstring
```

字符串,不以'\0'结尾

7.1.2.3 MESSAGE_RECEIVED

```
typedef void MESSAGE_RECEIVED(void *context, lenstring *topic, datahub_message *msg)
```

收到消息后的回调函数

参数

<i>context</i>	在选项中设置的context
<i>topic</i>	消息对应的主题
<i>msg</i>	描述消息的结构体

7.1.2.4 CONNECTION_STATUS_CHANGED

```
typedef void CONNECTION_STATUS_CHANGED(void *context, int isconnected)
```

当网络状态发生变化时,通知用户的回调函数

注解

该函数只用于通知用户网络状态,请不要在函数中进行耗时操作或者阻塞

参数

<i>context</i>	在选项中设置的context
<i>isconnected</i>	DATAHUB_TRUE表示已连接, DATAHUB_FALSE表示已断开

7.1.2.5 datahub_options

```
typedef struct datahub_options datahub_options
```

选项结构体,用于设置MQTT协议的相关选项

7.1.2.6 datahub_client

```
typedef struct datahub_client datahub_client
```

描述客户端的结构体,由SDK维护,用户无需关心

由于SDK非线程安全,不能在多个Task中使用datahub_client实例,只能在一个Task中使用

7.1.3 枚举类型说明

7.1.3.1 datahub_error_code_s

```
enum datahub_error_code_s
```

错误码

枚举值

ERROR_NONE	成功
ERROR_ILLEGAL_PARAMETERS	某些参数不合法
ERROR_DISCONNECTED	客户端已断开
ERROR_UNACCEPT_PROTOCOL_VERSION	MQTT服务器不支持当前使用的协议版本号,请联系开发人员
ERROR_IDENTIFIER_REJECTED	client_id不可用,可能使用了不支持的字符

枚举值

ERROR_SERVER_UNAVAILABLE	服务器不可用
ERROR_BAD_USERNAME_OR_PASSWD	instance_id 或者instance_key不正确,请检查
ERROR_UNAUTHORIZED	未被授权
ERROR_AUTHORIZED_SERVER_UNAVAILABLE	验证服务器不可用
ERROR_OPERATION_FAILURE	操作失败
ERROR_MESSAGE_TOO_BIG	消息过长
ERROR_NETWORK_UNREACHABLE	网络不可用
ERROR_TIMEOUT	超时
ERROR_ILLEGAL_OPTION	非法的选项

7.1.4 函数说明

7.1.4.1 datahub_create()

```
int datahub_create (
    datahub_client * client,
    char * instance_id,
    char * instance_key,
    char * client_name,
    char * client_id,
    Network * network,
    datahub_options * options )
```

该函数创建一个客户端实例,该实例可用于连接大数点MQTT服务器

参数

<i>client</i>	如果函数成功调用,则客户端实例可用于发送数据和订阅主题等 注意: 不能为空
<i>instance_id</i>	用于连接大数点服务器的唯一标识,由大数点提供 注意: 不能为空
<i>instance_key</i>	用于连接大数点服务器的密码,由大数点提供 注意: 不能为空
<i>client_name</i>	设备的名字 注意: 不能为空
<i>client_id</i>	设备的id 注意: 不能为空
<i>network</i>	网络接口,由用户实现网络数据的收发,SDK会调用这些函数 注意: 不能为空且传入前需要调用NewNetwork初始化,否则会出现未定义的错误
<i>options</i>	可选的选项.具体包含的选项可以查看datahub_options结构体 注意:不能为空

返回

ERROR_NONE 表示成功,其他表示错误.
错误码请查看开发文档

7.1.4.2 datahub_sendrequest()

```
int datahub_sendrequest (
    datahub_client * client,
    char * topic,
    datahub_message * message,
    int qos,
    int timeout )
```

同步发送消息

参数

client	由函数datahub_create成功创建的客户端实例 注意: 不能为空
topic	消息对应的topic.如果消息发送前另一个客户端订阅该topic,则另一个客户端就会收到消息. 注意: 不能为空
message	发送的消息 注意: 不能为空
qos	消息的服务质量 0: 消息可能到达,也可能不到达 1: 消息一定会到达,但可能会重复,当然,前提是返回ERROR_NONE 2: 消息一定会到达,且只到达一次,当然,前提是返回ERROR_NONE
timeout	函数阻塞的最大时间,单位为秒 注意: 必须 > 0

返回

ERROR_NONE 表示成功,其他表示错误.
错误码请查看开发文档

7.1.4.3 datahub_subscribe()

```
int datahub_subscribe (
    datahub_client * client,
    char * topic,
    int qos,
    int timeout )
```

同步订阅主题

参数

<i>client</i>	由函数datahub_create成功创建的客户端实例 注意: 不能为空
<i>topic</i>	订阅的topic 注意: 不能为空
<i>qos</i>	订阅消息的服务质量(发送消息的qos和订阅消息的qos共同决定服务器下发消息的qos) 0: 消息可能到达,也可能不到达 1: 消息一定会到达,但可能会重复,当然,前提是返回ERROR_NONE 2: 消息一定会到达,且只到达一次,当然,前提是返回ERROR_NONE 注意: 有效值为0, 1, 2
<i>timeout</i>	函数阻塞的最大时间,单位为秒 注意: 必须 > 0

返回

ERROR_NONE 表示成功,其他表示错误.
错误码请查看开发文档

7.1.4.4 datahub_unsubscribe()

```
int datahub_unsubscribe (
    datahub_client * client,
    char * topic,
    int timeout )
```

同步取消订阅topic

参数

<i>client</i>	由函数datahub_create成功创建的客户端实例 注意: 不能为空
<i>topic</i>	取消的topic 注意: 不能为空
<i>timeout</i>	函数阻塞的最大时间,单位为秒 注意: 必须 > 0

返回

ERROR_NONE 表示成功,其他表示错误.
错误码请查看开发文档API.md

7.1.4.5 datahub_destroy()

```
void datahub_destroy (
    datahub_client * client )
```

销毁客户端,销毁后连接不存在,不能收发数据

参数

<i>client</i>	由函数datahub_create成功创建的客户端实例 注意: 不能为空
---------------	---

返回

无

7.1.4.6 datahub_yield()

```
int datahub_yield (
    datahub_client * client,
    int timeout_ms )
```

被动接收消息情况下保持和服务器的连接

注解

在只订阅主题的情况下, 由于是被动接收服务器发送的消息, 为了维持和服务器的连接, 需要定时调用datahub_yield; 如果客户端一直调用datahub_sendrequest, 而该函数内嵌了与服务器维持连接的功能, 则不需要调用datahub_yield

参数

<i>client</i>	由函数datahub_create成功创建的客户端实例 注意: 不能为空
<i>timeout_ms</i>	函数阻塞的最大时间 注意: 单位为毫秒, 必须大于0

返回

ERROR_NONE 表示操作成功,其他表示错误.
错误码请查看开发文档

7.2 DatahubNetwork.h 文件参考

结构体

- struct [Network](#)
描述网络的结构体

宏定义

- #define [mqttread](#) read
- #define [mqttwrite](#) write

类型定义

- typedef struct Network Network

函数

- int ucos_read (Network *Net, unsigned char *buffer, int len, int timeout_ms)
从网络中读取数据
- int ucos_write (Network *Net, unsigned char *buffer, int len, int timeout_ms)
向网络中发送数据
- void ucos_disconnect (Network *Net)
断开与服务器的连接
- void NewNetwork (Network *Net, void *data)
初始化Network结构体
- int ConnectNetwork (Network *Net, char *addr, int port)
连接服务器

7.2.1 宏定义说明

7.2.1.1 mqttread

#define mqttread read

7.2.1.2 mqttwrite

#define mqttwrite write

7.2.2 类型定义说明

7.2.2.1 Network

typedef struct Network Network

7.2.3 函数说明

7.2.3.1 ucos_read()

```
int ucos_read (  
    Network * Net,  
    unsigned char * buffer,  
    int len,  
    int timeout_ms )
```

从网络中读取数据

参数

Net	一个指向Network的结构体指针
buffer	读缓冲区的首地址
len	SDK需要读取的数据长度

参数

<i>timeout_ms</i>	读操作的最大阻塞时间, 单位为毫秒
-------------------	-------------------

返回

返回读取的数据长度, 正常情况下为参数*len*表示的大小, 其他情况表示失败

注解

当SDK还未读取数据而4G模块已经接收到数据, 用户需要保存数据, 等待 SDK读取
此外, SDK读取的数据长度是通过参数*len*指定的, 而不是返回所有接收到的数据

7.2.3.2 ucosh_write()

```
int ucosh_write (
    Network * Net,
    unsigned char * buffer,
    int len,
    int timeout_ms )
```

向网络中发送数据

参数

<i>Net</i>	一个指向Network的结构体指针
<i>buffer</i>	发送缓冲区的首地址
<i>len</i>	发送数据的大小
<i>timeout_ms</i>	最大阻塞时间, 单位为毫秒

返回

返回写入网络中的数据长度, 正常情况下为参数*len*表示的大小, 其他情况表示失败

注解

成功是指用户将数据全部发送到网络上, 而不是仅仅交付给传输层而未发送到网络上;
比如说通过串口发送给4G模块后, 4G模块只是保留了用户发送的数据且还未完全传输到网络上, 用户需要等待4G模块发送完成后才能返回

7.2.3.3 ucosh_disconnect()

```
void ucosh_disconnect (
    Network * Net )
```

断开与服务器的连接

参数

<i>Net</i>	一个指向Network的结构体指针
------------	-------------------

返回

无

7.2.3.4 NewNetwork()

```
void NewNetwork (
    Network * Net,
    void * data )
```

初始化Network结构体

参数

Net	一个指向Network的结构体指针
data	一个指向用户自定义的结构体指针

返回

无

7.2.3.5 ConnectNetwork()

```
int ConnectNetwork (
    Network * Net,
    char * addr,
    int port )
```

连接服务器

参数

Net	一个指向Network的结构体指针
addr	服务器的ip或者域名
port	服务器的端口号

返回

0 表示成功, 其他表示错误

7.3 DatahubTimer.h 文件参考

结构体

- struct Timer
描述定时器的结构体

类型定义

- typedef struct Timer Timer
描述定时器的结构体

函数

- void `InitTimer` (`Timer` *timer)
初始化定时器
- char `expired` (`Timer` *timer)
判断定时器是否超时
- void `countdown_ms` (`Timer` *timer, unsigned int timeout)
增加定时器的时间
- void `countdown` (`Timer` *timer, unsigned int timeout)
增加定时器的时间
- int `left_ms` (`Timer` *timer)
定时器从当前时间开始，剩余多少时间超时

7.3.1 类型定义说明

7.3.1.1 Timer

```
typedef struct Timer Timer
```

描述定时器的结构体

7.3.2 函数说明

7.3.2.1 InitTimer()

```
void InitTimer (
    Timer * timer )
```

初始化定时器

参数

<i>timer</i>	指向待初始化的Timer结构体
--------------	-----------------

返回

无返回值

7.3.2.2 expired()

```
char expired (
    Timer * timer )
```

判断定时器是否超时

参数

<i>timer</i>	定时器指针
--------------	-------

返回

1 表示超时, 0 表示未超时

7.3.2.3 countdown_ms()

```
void countdown_ms (
    Timer * timer,
    unsigned int timeout )
```

增加定时器的时间

参数

<i>timer</i>	指向Timer结构体
<i>timeout</i>	增加的时间, 单位为毫秒

返回

无返回值

7.3.2.4 countdown()

```
void countdown (
    Timer * timer,
    unsigned int timeout )
```

增加定时器的时间

参数

<i>timer</i>	指向Timer结构体
<i>timeout</i>	增加的时间, 单位为秒

返回

无返回值

7.3.2.5 left_ms()

```
int left_ms (
    Timer * timer )
```

定时器从当前时间开始, 剩余多少时间超时

参数

<i>timer</i>	指向Timer结构体
--------------	------------

返回

剩余的时间，单位为毫秒

7.4 **FAQ.md** 文件参考

7.5 **release_note.txt** 文件参考

Index

c

- [datahub_client](#), [3](#)
- [CONNECTION_STATUS_CHANGED](#)
 - [DataHubClient.h](#), [11](#)
- [cleansession](#)
 - [datahub_options](#), [5](#)
- [client_id](#)
 - [datahub_client](#), [3](#)
- [client_name](#)
 - [datahub_client](#), [3](#)
- [ConnectNetwork](#)
 - [DatahubNetwork.h](#), [19](#)
- [connection_status_changed](#)
 - [datahub_options](#), [5](#)
- [context](#)
 - [datahub_options](#), [5](#)
- [countdown](#)
 - [DatahubTimer.h](#), [21](#)
- [countdown_ms](#)
 - [DatahubTimer.h](#), [20](#)
- [DATAHUB_FALSE](#)
 - [DataHubClient.h](#), [10](#)
- [DATAHUB_MESSAGE_INITIALIZER](#)
 - [DataHubClient.h](#), [11](#)
- [DATAHUB_OPTIONS_INITIALIZER](#)
 - [DataHubClient.h](#), [10](#)
- [DATAHUB_TRUE](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_BUF_SIZE](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_BUF](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_CALLBACK](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_CLEAN_SESSION](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_CONTEXT](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_HOST](#)
 - [DataHubClient.h](#), [10](#)
- [DEFAULT_PORT](#)
 - [DataHubClient.h](#), [10](#)
- [data](#)
 - [lenstring_s](#), [6](#)
 - [Network](#), [7](#)
- [DataHubClient.h](#), [8](#)
 - [CONNECTION_STATUS_CHANGED](#), [11](#)
 - [DATAHUB_FALSE](#), [10](#)
 - [DATAHUB_MESSAGE_INITIALIZER](#), [11](#)
 - [DATAHUB_OPTIONS_INITIALIZER](#), [10](#)
 - [DATAHUB_TRUE](#), [10](#)
 - [DEFAULT_BUF_SIZE](#), [10](#)
 - [DEFAULT_BUF](#), [10](#)
 - [DEFAULT_CALLBACK](#), [10](#)
 - [DEFAULT_CLEAN_SESSION](#), [10](#)
 - [DEFAULT_CONTEXT](#), [10](#)
 - [DEFAULT_HOST](#), [10](#)
 - [DEFAULT_PORT](#), [10](#)
 - [datahub_client](#), [2](#)
 - [c](#), [3](#)
 - [client_id](#), [3](#)
 - [client_name](#), [3](#)
 - [DataHubClient.h](#), [12](#)
 - [instance_id](#), [3](#)
 - [instance_key](#), [3](#)
 - [network](#), [3](#)
 - [options](#), [3](#)
 - [datahub_create](#)
 - [DataHubClient.h](#), [13](#)
 - [datahub_destroy](#)
 - [DataHubClient.h](#), [15](#)
 - [datahub_error_code_s](#)
 - [DataHubClient.h](#), [12](#)
 - [datahub_message](#), [4](#)
 - [DataHubClient.h](#), [11](#)
 - [payload](#), [4](#)
 - [payload_len](#), [4](#)
 - [datahub_options](#), [4](#)
 - [cleansession](#), [5](#)
 - [connection_status_changed](#), [5](#)
 - [context](#), [5](#)
 - [DataHubClient.h](#), [12](#)
 - [host](#), [5](#)
 - [message_received](#), [5](#)
 - [port](#), [5](#)
 - [readbuf](#), [6](#)
 - [readbuf_size](#), [6](#)
 - [sendbuf](#), [5](#)
 - [sendbuf_size](#), [6](#)
 - [datahub_sendrequest](#)
 - [DataHubClient.h](#), [13](#)
 - [datahub_subscribe](#)
 - [DataHubClient.h](#), [14](#)
 - [datahub_unsubscribe](#)
 - [DataHubClient.h](#), [15](#)
 - [datahub_yield](#)
 - [DataHubClient.h](#), [15](#)

- DatahubNetwork.h, 16
 - ConnectNetwork, 19
 - mqttread, 17
 - mqttwrite, 17
 - Network, 17
 - NewNetwork, 18
 - ucos_disconnect, 18
 - ucos_read, 17
 - ucos_write, 17
- DatahubTimer.h, 19
 - countdown, 21
 - countdown_ms, 20
 - expired, 20
 - InitTimer, 20
 - left_ms, 21
 - Timer, 19
- disconnect
 - Network, 7
- end
 - Timer, 8
- expired
 - DatahubTimer.h, 20
- FAQ.md, 21
- host
 - datahub_options, 5
- InitTimer
 - DatahubTimer.h, 20
- instance_id
 - datahub_client, 3
- instance_key
 - datahub_client, 3
- left_ms
 - DatahubTimer.h, 21
- len
 - lenstring_s, 6
- lenstring
 - DataHubClient.h, 11
- lenstring_s, 6
 - data, 6
 - len, 6
- MESSAGE_RECEIVED
 - DataHubClient.h, 11
- message_received
 - datahub_options, 5
- mqttread
 - DatahubNetwork.h, 17
- mqttwrite
 - DatahubNetwork.h, 17
- Network, 7
 - data, 7
 - DatahubNetwork.h, 17
 - disconnect, 7
 - read, 7
 - write, 7
- network
 - datahub_client, 3
- NewNetwork
 - DatahubNetwork.h, 18
- options
 - datahub_client, 3
- payload
 - datahub_message, 4
- payload_len
 - datahub_message, 4
- port
 - datahub_options, 5
- read
 - Network, 7
- readbuf
 - datahub_options, 6
- readbuf_size
 - datahub_options, 6
- release_note.txt, 21
- sendbuf
 - datahub_options, 5
- sendbuf_size
 - datahub_options, 6
- Timer, 8
 - DatahubTimer.h, 19
 - end, 8
- ucos_disconnect
 - DatahubNetwork.h, 18
- ucos_read
 - DatahubNetwork.h, 17
- ucos_write
 - DatahubNetwork.h, 17
- write
 - Network, 7