

# IoT Datahub C SDK

by Dasudian

## Contents

<b>1 IoT Datahub C SDK</b>	<b>1</b>
<b>2 更新日志</b>	<b>1</b>
<b>3 FAQ</b>	<b>2</b>
<b>4 结构体说明</b>	<b>2</b>
4.1 datahub_message_s结构体 参考	2
4.1.1 详细描述	3
4.1.2 结构体成员变量说明	3
4.2 datahub_options_s结构体 参考	3
4.2.1 详细描述	3
4.2.2 结构体成员变量说明	3
<b>5 文件说明</b>	<b>4</b>
5.1 datahub_sdk_c.h 文件参考	5
5.1.1 宏定义说明	6
5.1.2 类型定义说明	7
5.1.3 枚举类型说明	8
5.1.4 函数说明	9
5.2 FAQ.md 文件参考	13
5.3 release_note.txt 文件参考	13
<b>索引</b>	<b>15</b>

## 1 IoT Datahub C 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。如果连接失败,下一次重连将会在1s,2s,4s,8s, 16s,1s,2s,4s,8s,16s,1s,...后再次尝试,最大重连间隔为16秒
- 2: SDK为线程安全
- 3: 一个进程只能创建一个客户端,如果需要多个客户端,需要创建多个进程

## 2 更新日志

Author	Date	Version	Note
Eden Wang	10/27/2017	3.0.0	增加数据类型;增加设备类型;修改验证方式
Eden Wang	6/05/2017	2.2.0	取消异步发送接口;API文档修改为pdf格式
Eden Wang	6/16/2017	2.1.0	支持ARM
Eden Wang	5/05/2017	2.0.0	根据标准修改
Eden Wang	2/16/2017	1.2.0	限制消息大小, 不超过512K
Eden Wang	2/14/2017	1.1.0	添加选项debug
Eden Wang	1/13/2017	1.0.0	第一个版本

### 3 FAQ

默认使用gcc编译器

Q:

```
main.c:(.text+0x49): undefined reference to `datahub_create'
main.c:(.text+0x86): undefined reference to `datahub_connect'
main.c:(.text+0xf1): undefined reference to `datahub_sendrequest'
main.c:(.text+0x124): undefined reference to `datahub_disconnect'
main.c:(.text+0x137): undefined reference to `datahub_disconnect'
```

A: 需要添加链接选项 -ldasudian-datahub-sdk

Q:

```
/usr/bin/ld: cannot find -ldasudian-datahub-sdk
```

A: 需要指明SDK库所在的路径: -L lib/x86-64 或者 -L lib/arm

Q:

```
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_subscribe'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_disconnect'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_setCallbacks'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_publishMessage'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_isConnected'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_connect'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_unsubscribe'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_destroy'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_free'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_waitForCompletion'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_create'
lib//x86-64//libdasudian-datahub-sdk.so: undefined reference to `MQTTClient_freeMessage'
```

A: 需要链接库paho-mqtt3cs: -lpaho-mqtt3cs

Q:

```
./main: error while loading shared libraries: libddasudian-datahub-sdk.so: cannot open shared object file:
No such file or directory
```

A: 需要指明运行时SDK库所在的路径: -Wl,-rpath lib/x86-64 或者 -Wl,-rpath lib/arm

## 4 结构体说明

### 4.1 datahub\_message\_s结构体 参考

消息的结构体类型

```
#include <datahub_sdk_c.h>
```

成员变量

- unsigned int [payload\\_len](#)
- void \* [payload](#)

#### 4.1.1 详细描述

消息的结构体类型

#### 4.1.2 结构体成员变量说明

##### 4.1.2.1 payload\_len

unsigned int payload\_len

消息长度,必须大于0

##### 4.1.2.2 payload

void\* payload

发送消息的起始地址

该结构体的文档由以下文件生成:

- [datahub\\_sdk\\_c.h](#)

## 4.2 datahub\_options\_s结构体 参考

设置选项

```
#include <datahub_sdk_c.h>
```

成员变量

- char \* [server\\_url](#)  
设置使用哪种协议(*ssl*, 普通的*tcp*)连接服务器, 以及设置服务器的地址和端口号.
- int [debug](#)  
开启调试选项
- 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()*的 参数, 对应回调函数的第一个参数*context*
- [MESSAGE\\_RECEIVED](#) \* [message\\_received](#)  
接收到消息后的回调函数
- [CONNECTION\\_STATUS\\_CHANGED](#) \* [connection\\_status\\_changed](#)  
当客户端的状态发生改变(连接上服务器或者从服务器断开)的时候,SDK会通知用户

#### 4.2.1 详细描述

设置选项

#### 4.2.2 结构体成员变量说明

#### 4.2.2.1 server\_url

char\* server\_url

设置使用哪种协议(ssl, 普通的tcp)连接服务器, 以及设置服务器的地址和端口号.

"协议: //服务器地址: 端口号". 协议支持普通的tcp协议和加密的ssl协议; 服务器地址和端口号 由大数点提供.

默认值为DEFAULT\_SERVER\_URL

#### 4.2.2.2 debug

int debug

开启调试选项

DATAHUB\_TRUE表示开启调试选项

DATAHUB\_FALSE表示关闭调试选项

默认值为DATAHUB\_FALSE

#### 4.2.2.3 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.

默认值为DATAHUB\_FALSE

#### 4.2.2.4 context

void\* context

传递给回调函数message\_received()和connection\_status\_changed()的 参数, 对应回调函数的第一个参数context

默认值为NULL

#### 4.2.2.5 message\_received

MESSAGE\_RECEIVED\* message\_received

接收到消息后的回调函数

NULL或者函数指针, 默认值为DEFAULT\_CALLBACK

#### 4.2.2.6 connection\_status\_changed

CONNECTION\_STATUS\_CHANGED\* connection\_status\_changed

当客户端的状态发生改变(连接上服务器或者从服务器断开)的时候,SDK会通知用户

NULL或者函数指针, 默认值为DEFAULT\_CALLBACK

该结构体的文档由以下文件生成:

- [datahub\\_sdk\\_c.h](#)

## 5 文件说明

## 5.1 datahub\_sdk\_c.h 文件参考

```
#include <stdlib.h>
#include <string.h>
```

### 结构体

- struct [datahub\\_message\\_s](#)  
消息的结构体类型
- struct [datahub\\_options\\_s](#)  
设置选项

### 宏定义

- #define [DATAHUB\\_TRUE](#) 1
- #define [DATAHUB\\_FALSE](#) 0
- #define [DEFAULT\\_SERVER\\_URL](#) "tcp://try.iotdatahub.net:1883"
- #define [DEFAULT\\_DEBUG\\_OPT](#) [DATAHUB\\_FALSE](#)
- #define [DEFAULT\\_CLEANSESSION](#) [DATAHUB\\_FALSE](#)
- #define [DEFAULT\\_CONTEXT](#) NULL
- #define [DATAHUB\\_OPTIONS\\_INITIALIZER](#)
- #define [DATAHUB\\_MESSAGE\\_INITIALIZER](#) {1, ""}

### 类型定义

- typedef void \* [datahub\\_client](#)
- typedef struct [datahub\\_message\\_s](#) [datahub\\_message](#)  
消息的结构体类型
- typedef enum [datahub\\_data\\_type\\_s](#) [datahub\\_data\\_type](#)
- typedef void [MESSAGE\\_RECEIVED](#)(void \*context, char \*topic, [datahub\\_message](#) \*msg)  
接收到消息后的回调函数
- typedef void [CONNECTION\\_STATUS\\_CHANGED](#)(void \*context, int isconnected)  
当客户端的状态发生改变(连接上服务器或者从服务器断开)的时候,SDK会通知用户
- typedef struct [datahub\\_options\\_s](#) [datahub\\_options](#)  
设置选项

### 枚举

- enum [qos\\_s](#) { [QoS0](#) = 0, [QoS1](#) = 1, [QoS2](#) = 2 }
- 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\\_MEMORY\\_ALLOCATE](#) = -100, [ERROR\\_MESSAGE\\_INVALID\\_JSON](#) = -101, [ERROR\\_INVALID\\_CLIENT\\_TYPE](#) = -102 }
- enum [datahub\\_data\\_type\\_s](#) { [JSON](#) = 0, [TEXT](#) = 1, [BINARY](#) = 2, [DATA\\_TYPE\\_END](#) }

## 函数

- int `datahub_create` (`datahub_client` \*client, char \*instance\_id, char \*instance\_key, char \*client\_type, char \*client\_id, `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` \*msg, `datahub_data_type` data\_type, int qos, int timeout)  
同步发送消息
- int `datahub_subscribe` (`datahub_client` \*client, char \*topic, int qos, int timeout)  
同步订阅某一个topic
- int `datahub_unsubscribe` (`datahub_client` \*client, char \*topic, int timeout)  
同步取消订阅某一个topic
- void `datahub_destroy` (`datahub_client` \*client)  
销毁客户端并断开连接
- void `datahub_callback_free` (char \*topic, `datahub_message` \*msg)  
接收函数中,主题和消息占用的内存需要用户手动释放

## 5.1.1 宏定义说明

## 5.1.1.1 DATAHUB\_TRUE

```
#define DATAHUB_TRUE 1
```

真值

## 5.1.1.2 DATAHUB\_FALSE

```
#define DATAHUB_FALSE 0
```

假值

## 5.1.1.3 DEFAULT\_SERVER\_URL

```
#define DEFAULT_SERVER_URL "tcp://try.iotdatahub.net:1883"
```

## 5.1.1.4 DEFAULT\_DEBUG\_OPT

```
#define DEFAULT_DEBUG_OPT DATAHUB_FALSE
```

## 5.1.1.5 DEFAULT\_CLEANSESSION

```
#define DEFAULT_CLEANSESSION DATAHUB_FALSE
```

## 5.1.1.6 DEFAULT\_CONTEXT

```
#define DEFAULT_CONTEXT NULL
```

## 5.1.1.7 DATAHUB\_OPTIONS\_INITIALIZER

```
#define DATAHUB_OPTIONS_INITIALIZER
```

值:

```
{\
DEFAULT_SERVER_URL,\
```

```

DEFAULT_DEBUG_OPT,\
DEFAULT_CLEANSSESSION,\
DEFAULT_CONTEXT,\
}

```

选项的初始化宏

#### 5.1.1.8 DATAHUB\_MESSAGE\_INITIALIZER

```
#define DATAHUB_MESSAGE_INITIALIZER {1, ""}
```

消息的初始化宏,只包含字符串结尾符'\0'

### 5.1.2 类型定义说明

#### 5.1.2.1 datahub\_client

```
typedef void* datahub_client
```

客户端的类型

#### 5.1.2.2 datahub\_message

```
typedef struct datahub_message_s datahub_message
```

消息的结构体类型

#### 5.1.2.3 datahub\_data\_type

```
typedef enum datahub_data_type_s datahub_data_type
```

数据类型

#### 5.1.2.4 MESSAGE\_RECEIVED

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

接收到消息后的回调函数

例子:

```

void message_received_callback(void *context, char *topic,
    datahub_message *msg)
{
    //做一些操作
    //...

    //释放topic和msg占用的内存
    datahub_callback_free(topic, msg);
}

```

参数

<i>context</i>	即datahub_options_s中的context
<i>topic</i>	本次消息所属的主题,需要调用datahub_callback_free()手动释放内存
<i>msg</i>	存放消息的结构体,需要调用datahub_callback_free()手动释放内存

#### 5.1.2.5 CONNECTION\_STATUS\_CHANGED

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



当客户端的状态发生改变(连接上服务器或者从服务器断开)的时候,SDK会通知用户

参数

<code>context</code>	即datahub_options_s中的context
<code>isconnected</code>	连接状态, DATAHUB_FALSE 表示从服务器断开,DATAHUB_TRUE 表示连接上服务器

### 5.1.2.6 datahub\_options

```
typedef struct datahub_options_s datahub_options
```

设置选项

### 5.1.3 枚举类型说明

#### 5.1.3.1 qos\_s

```
enum qos_s
```

消息的服务质量

枚举值

QoS0	客户端最多发送一次, 服务器最多接收一次
QoS1	客户端至少发送一次, 服务器可能接收多次
QoS2	客户端至少发送一次, 但保证服务器只接收到一次消息

#### 5.1.3.2 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	客户端未连接服务器
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或者client_id的长度太大
ERROR_SERVER_UNAVAILABLE	服务器不可用
ERROR_BAD_USERNAME_OR_PASSWD	instance_id 或者instance_key不正确,请检查或者联系客服人员
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

枚举值

ERROR_TIMEOUT	同步超时
ERROR_MEMORY_ALLOCATE	内存申请失败
ERROR_MESSAGE_INVALID_JSON	不合法的JSON字符串
ERROR_INVALID_CLIENT_TYPE	不合法的设备类型字符;设备类型不能包含竖线" ",也不能以下划线"_"开头

### 5.1.3.3 datahub\_data\_type\_s

enum datahub\_data\_type\_s

数据类型

枚举值

JSON	数据为JSON格式
TEXT	数据为文本/字符串
BINARY	数据为二进制
DATA_TYPE_END	

## 5.1.4 函数说明

### 5.1.4.1 datahub\_create()

```
int datahub_create (
    datahub_client * client,
    char * instance_id,
    char * instance_key,
    char * client_type,
    char * client_id,
    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,则会返回一个客户端实例. 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_type</i>	设备类型. 如传感器"sensor", 充电桩"charging_pile", 车载电池"car_battery" 注意:可以为空

## 参数

<i>client_id</i>	设备的id, 用于服务器唯一标记一个设备 注意: 不同的设备client id必须不同,如果两个设备有相同的id,服务器会关掉其中一个连接 可以使用设备的mac地址,或者第三方账号系统的id(比如qq号,微信号);如果没有自己的账号系统,则可以随机生成一个不会重复的id.或者自己指定设备的 client id,只要能保证不同客户端id不同即可. client_id不能为空
<i>options</i>	MQTT的选项.具体包含的选项可以查看datahub_options_s结构体.如果不想设置选项,请传递NULL.如果你想设置某些选项,先使用 DATAHUB_OPTIONS_INITIALIZER初始化. 注意:可以为空

## 返回

ERROR\_NONE 表示成功,其他表示错误.  
其他错误码请查看开发文档

## 5.1.4.2 datahub\_sendrequest()

```
int datahub_sendrequest (
    datahub_client * client,
    char * topic,
    datahub_message * msg,
    datahub_data_type data_type,
    int qos,
    int timeout )
```

## 同步发送消息

注意: 程序会阻塞

## 参数

<i>client</i>	由函数datahub_create()成功返回的客户端实例. 注意:不能为空.
<i>topic</i>	消息对应的topic.如果消息发送前有另一个客户端已经订阅该topic,则 另一个客户端就会收到消息. 注意:不能为空
<i>msg</i>	发送的消息,使用前请使用DATAHUB_MESSAGE_INITIALIZER初始化. 注意: 消息的长度必须小于512K,否则会发生错误. 注意:不能为空
<i>data_type</i>	数据类型, 只能为JSON或TEXT或BINARY
<i>qos</i>	消息的服务质量. <b>0:</b> 消息可能到达,也可能不到达. <b>1:</b> 消息一定会到达,但可能会重复,当然,前提是返回ERROR_NONE. <b>2:</b> 消息一定会到达,且只到达一次,当然,前提是返回ERROR_NONE. 注意: 只能为0,1,2三者中的一个,其他为非法参数

参数

<i>timeout</i>	函数阻塞的最大时间. 注意：这是函数阻塞的最大时间,不是消息的超时时间
----------------	--

返回

**ERROR\_NONE:** 表示成功,消息一定发送出去.

**ERROR\_TIMEOUT:** 表示阻塞等待时间的最大值已到,但是消息可能发送给服务器,也可能未发送.如果想确保消息一定发送出去,请根据消息大小和网络状况设置较大的阻塞等待时间.

其他错误码请查看开发文档

#### 5.1.4.3 datahub\_subscribe()

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

同步订阅某一个topic

注意：程序会阻塞

参数

<i>client</i>	由函数datahub_create()成功返回的客户端实例 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) <b>0:</b> 消息可能到达,也可能不到达 <b>1:</b> 消息一定会到达,但可能会重复,当然,前提是返回ERROR_NONE <b>2:</b> 消息一定会到达,且只到达一次,当然,前提是返回ERROR_NONE
<i>timeout</i>	函数阻塞的最大时间. 注意：这是函数阻塞的最大时间

返回

**ERROR\_NONE:** 表示成功,其他表示错误.

**ERROR\_TIMEOUT:** 表示阻塞等待时间的最大值已到,但是可能订阅成功,也可能订阅失败.如果想确保订阅一定成功,请根据消息大小和网络状况设置较大的阻塞等待时间.

其他错误码请查看开发文档

#### 5.1.4.4 datahub\_unsubscribe()

```
int datahub_unsubscribe (
```

```

datahub_client * client,
char * topic,
int timeout )

```

同步取消订阅某一个topic

参数

<i>client</i>	由函数datahub_create()成功返回的客户端实例 注意:不能为空
<i>topic</i>	取消订阅的topic 注意:不能为空
<i>timeout</i>	函数阻塞的最大时间. 注意:这是函数阻塞的最大时间

返回

**ERROR\_NONE:** 表示成功,其他表示错误.

**ERROR\_TIMEOUT:** 表示阻塞等待时间的最大值已到,但是可能取消成功,也可能取消失败.如果想确保取消一定成功,请根据消息大小和网络状况设置较大的阻塞等待时间.  
其他错误码请查看开发文档

#### 5.1.4.5 datahub\_destroy()

```

void datahub_destroy (
    datahub_client * client )

```

销毁客户端并断开连接

参数

<i>client</i>	由函数datahub_create()成功返回的客户端实例 注意:不能为空
---------------	--

返回

无

#### 5.1.4.6 datahub\_callback\_free()

```

void datahub_callback_free (
    char * topic,
    datahub_message * msg )

```

接收函数中,主题和消息占用的内存需要用户手动释放

参数

<i>topic</i>	返回的主题
<i>msg</i>	返回的消息

返回

无

5.2 FAQ.md 文件参考

5.3 release\_note.txt 文件参考



## Index

CONNECTION\_STATUS\_CHANGED

datahub\_sdk\_c.h, [7](#)

cleansession

datahub\_options\_s, [4](#)

connection\_status\_changed

datahub\_options\_s, [4](#)

context

datahub\_options\_s, [4](#)

DATAHUB\_FALSE

datahub\_sdk\_c.h, [6](#)

DATAHUB\_MESSAGE\_INITIALIZER

datahub\_sdk\_c.h, [7](#)

DATAHUB\_OPTIONS\_INITIALIZER

datahub\_sdk\_c.h, [6](#)

DATAHUB\_TRUE

datahub\_sdk\_c.h, [6](#)

DEFAULT\_CLEANSSESSION

datahub\_sdk\_c.h, [6](#)

DEFAULT\_CONTEXT

datahub\_sdk\_c.h, [6](#)

DEFAULT\_DEBUG\_OPT

datahub\_sdk\_c.h, [6](#)

DEFAULT\_SERVER\_URL

datahub\_sdk\_c.h, [6](#)

datahub\_callback\_free

datahub\_sdk\_c.h, [12](#)

datahub\_client

datahub\_sdk\_c.h, [7](#)

datahub\_create

datahub\_sdk\_c.h, [9](#)

datahub\_data\_type

datahub\_sdk\_c.h, [7](#)

datahub\_data\_type\_s

datahub\_sdk\_c.h, [9](#)

datahub\_destroy

datahub\_sdk\_c.h, [12](#)

datahub\_error\_code\_s

datahub\_sdk\_c.h, [8](#)

datahub\_message

datahub\_sdk\_c.h, [7](#)

datahub\_message\_s, [2](#)

payload, [3](#)

payload\_len, [3](#)

datahub\_options

datahub\_sdk\_c.h, [8](#)

datahub\_options\_s, [3](#)

cleansession, [4](#)

connection\_status\_changed, [4](#)

context, [4](#)

debug, [4](#)

message\_received, [4](#)

server\_url, [3](#)

datahub\_sdk\_c.h, [5](#)

CONNECTION\_STATUS\_CHANGED, [7](#)

DATAHUB\_FALSE, [6](#)

DATAHUB\_MESSAGE\_INITIALIZER, [7](#)

DATAHUB\_OPTIONS\_INITIALIZER, [6](#)

DATAHUB\_TRUE, [6](#)

DEFAULT\_CLEANSSESSION, [6](#)

DEFAULT\_CONTEXT, [6](#)

DEFAULT\_DEBUG\_OPT, [6](#)

DEFAULT\_SERVER\_URL, [6](#)

datahub\_callback\_free, [12](#)

datahub\_client, [7](#)

datahub\_create, [9](#)

datahub\_data\_type, [7](#)

datahub\_data\_type\_s, [9](#)

datahub\_destroy, [12](#)

datahub\_error\_code\_s, [8](#)

datahub\_message, [7](#)

datahub\_options, [8](#)

datahub\_sendrequest, [10](#)

datahub\_subscribe, [11](#)

datahub\_unsubscribe, [11](#)

MESSAGE\_RECEIVED, [7](#)

qos\_s, [8](#)

datahub\_sendrequest

datahub\_sdk\_c.h, [10](#)

datahub\_subscribe

datahub\_sdk\_c.h, [11](#)

datahub\_unsubscribe

datahub\_sdk\_c.h, [11](#)

debug

datahub\_options\_s, [4](#)

FAQ.md, [13](#)

MESSAGE\_RECEIVED

datahub\_sdk\_c.h, [7](#)

message\_received

datahub\_options\_s, [4](#)

payload

datahub\_message\_s, [3](#)

payload\_len

datahub\_message\_s, [3](#)

qos\_s

datahub\_sdk\_c.h, [8](#)

release\_note.txt, [13](#)

server\_url

datahub\_options\_s, [3](#)