User Manual

接口函数使用手册

CAN/CANFD 接口卡系列产品

UM01010101 V0.00 Date: 2018/03/01

产品用户手册

类别	内容
关键词	CAN/CANFD 接口函数库使用
摘　要	本软件可适用于广州致远电子有限公司出品的各种 CAN/CANFD 接口卡。接口函数库是提供给用户进行上位机二次开发，可以自行编程进行数据收发、处理等。

广州致远电子有限公司

修订历史

版本	日期	原因
V1.00	2018/03/01	创建文档
V1.01	2018/06/28	添加支持定时发送的设备信息
V1.02	2018/07/05	添加云设备相关接口

目 录

1. 接口库函数使用

1.1 接口库简介

为了满足您上位机二次开发的需要，所以提供该接口库。接口库以 window 动态链接库(dll)的形式提供，可实现设备打开、配置、数据收发、关闭等功能。除了文档说明，开发资料中还包含了使用例程，用户可实现快速入门。

库文件包括 zlgcan.dll、kerneldlls、zlgcan 文件夹，zlgcan 文件夹主要包含 zlgcan.lib、zlgcan.h 以及一些其它头文件，可参考使用例程使用。

zlgcan.dll 采用 visual studio 2008 开发，依赖 2008 运行时库，如果您的计算机缺少该运行时库，则需要先安装该运行时库才能正常使用，可到微软官网下载。

1.2 接口卡设备类型定义

1.1 接口卡的类型定义

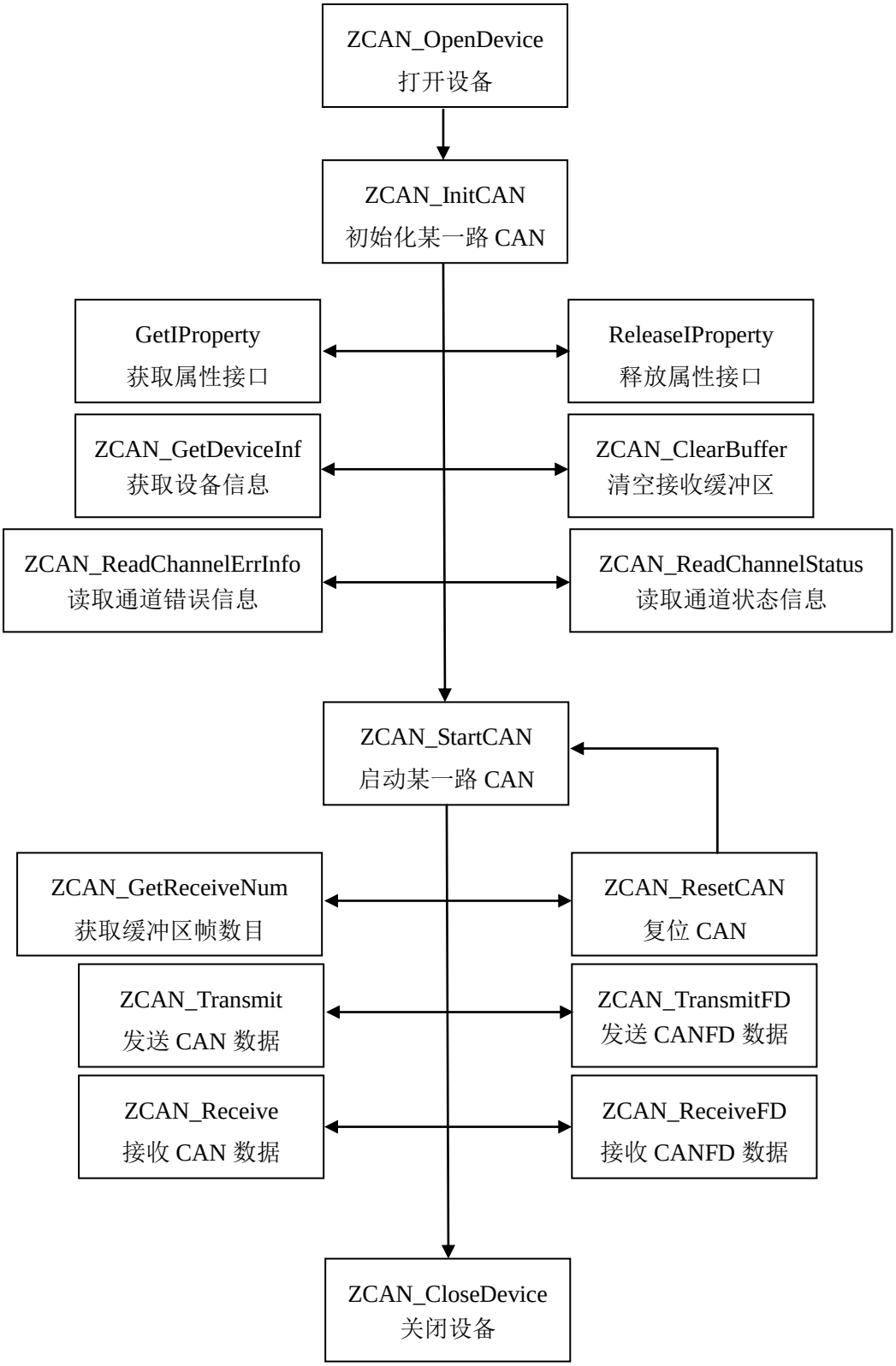
产品型号	类型号	规范
PCI-9810I	2	CAN
USBCAN-I/I+	3	CAN
USBCAN-II/II+	4	CAN
PCI-9820	5	CAN
CANET 系列的 UDP 工作方式	12	CAN
PCI-9840I	14	CAN
PC104-CAN2I	15	CAN
PCI-9820I	16	CAN
CANET 系列的 TCP 工作方式	17	CAN
PCI-5010-U	19	CAN
USBCAN-E-U	20	CAN
USBCAN-2E-U、CANalyst-II+	21	CAN
PCI-5020-U	22	CAN
PCIe-9221	24	CAN
CANWiFi-200T 的 TCP 工作方式	25	CAN
CANWiFi-200T 的 UDP 工作方式	26	CAN
PCIe-9120I	27	CAN
PCIe-9110I	28	CAN
PCIe-9140I	29	CAN
USBCAN-4E-U	31	CAN
CANDTU	32	CAN
USBCAN-8E-U	34	CAN
PCIE-CANFD-100U	38	CAN/CANFD
PCIE-CANFD-200U	39	CAN/CANFD
PCIE-CANFD-400U	40	CAN/CANFD
USBCANFD-100U	41	CAN/CANFD
USBCANFD-200U	42	CAN/CANFD
USBCANFD-MINI	43	CAN/CANFD

1.3 支持定时发送的设备

1.2 定时发送的设备

产品型号	类型号	支持的报文个数
USBCAN-2E-U	21	32
USBCAN-4E-U	31	32
USBCAN-8E-U	34	32
PCIE-CANFD-100U	38	100
PCIE-CANFD-200U	39	100
PCIE-CANFD-400U	40	100
USBCANFD-100U	41	100
USBCANFD-200U	42	100
USBCANFD-MINI	43	100

1.4 接口库函数使用流程



1.5 数据结构定义

1.5.1 ZCAN_DEVICE_INFO

ZCAN_DEVICE_INFO 结构体包含设备信息，结构体在函数 ZCAN_GetDeviceInf 中被填充。

```
typedef struct tagZCAN_DEVICE_INFO {
    USHORT hw_Version;
    USHORT fw_Version;
    USHORT dr_Version;
    USHORT in_Version;
    USHORT irq_Num;
    BYTE    can_Num;
    UCHAR   str_Serial_Num[20];
    UCHAR   str_hw_Type[40];
    USHORT reserved[4];
}ZCAN_DEVICE_INFO;
```

成员

hw_Version

硬件版本号，用 16 进制表示。比如 0x0100 表示 V1.00。

fw_Version

固件版本号，用 16 进制表示。

dr_Version

驱动程序版本号，用 16 进制表示。

in_Version

接口库版本号，用 16 进制表示。

irq_Num

板卡所使用的中断号。

can_Num

表示有几路通道。

str_Serial_Num

此板卡的序列号，比如” USBCAN V1.00”（注意：包括字符串结束符’\0’）。

str_hw_Type

硬件类型。

reserved

保留，不设置。

1.5.2 ZCAN_CHANNEL_INIT_CONFIG

ZCAN_CHANNEL_INIT_CONFIG 结构体定义了初始化配置。初始化之前，要先填充好这个结构体。

```
typedef struct tagZCAN_CHANNEL_INIT_CONFIG {
    UINT can_type; // 0:can 1:canfd
    union
    {
        struct
        {
            UINT    acc_code;
            UINT    acc_mask;
            UINT    reserved;
            BYTE    filter;
            BYTE    timing0;
            BYTE    timing1;
            BYTE    mode;
        }can;
        struct
        {
            UINT     acc_code;
            UINT     acc_mask;
            UINT     abit_timing;
            UINT     dbit_timing;
            UINT     brp;
            BYTE     filter;
            BYTE     mode;
            USHORT pad;
            UINT     reserved;
        }canfd;
    };
}ZCAN_CHANNEL_INIT_CONFIG;
```

成员

can_type

设备类型， =0 表示 CAN 设备，=1 表示 CANFD 设备，参考 1.1 接口卡的类型定义中的规范。

- **CAN 设备**

  acc_code

  验收码。SJA1000 的帧过滤验收码，对经过屏蔽码过滤为“有关位”进行匹配，全部匹配成功后，此帧可以被接收，否则不接收。推荐设置为 0。

  acc_mask

  屏蔽码。SJA1000 的帧过滤屏蔽码，对接收的 CAN 帧 ID 进行过滤，对应位为 0 的是“有关位”，对应位为 1 的是“无关位”。推荐设置为 0xFFFFFFFF，即全部接收。

reserved

保留。

filter

滤波方式。=1 表示单滤波，=0 表示双滤波。

timing0

波特率定时器 0（BTR0）。设置值见表 1.3 CAN 波特率。

timing1

波特率定时器 1（BTR1）。设置值见表 1.3 CAN 波特率。

mode

模式。=0 表示正常模式（相当于正常节点），=1 表示只听模式（只接收，不影响总线）。

timing0 和 timing1 用来设置 CAN 波特率，几种常见的波特率（采样点 87.5%，SJW 为 0）设置如下：

表 1.3 CAN 波特率

CAN 波特率	timing 0	timing 1
1Mbps 80%	0xBF	0xFF
10Kbps	0x31	0x1C
20Kbps	0x18	0x1C
40Kbps	0x87	0xFF
50Kbps	0x09	0x1C
80Kbps	0x83	0Xff
100Kbps	0x04	0x1C
125Kbps	0x03	0x1C
200Kbps	0x81	0xFA
250Kbps	0x01	0x1C
400Kbps	0x80	0xFA
500Kbps	0x00	0x1C
666Kbps	0x80	0xB6
800Kbps	0x00	0x16
1000Kbps	0x00	0x14

注意：当设备类型为 PCI-5010-U、PCI-5020-U、USBCAN-E-U、 USBCAN-2E-U、USBCAN-4E-U、CANDTU 时，波特率和帧过滤不在这里设置，参考 GetIProperty。

- **CANFD 设备**

acc_code

验收码，同 CAN 设备。

acc_mask

屏蔽码，同 CAN 设备。

abit_timing

仲裁域波特率定时器，值参考表 1.4 CANFD 默认波特率。

dbit_timing

数据域波特率定时器，值参考表 1.4 CANFD 默认波特率。

brp

波特率预分频因子，设置为 0。

filter

滤波方式，同 CAN 设备。

mode

模式，同 CAN 设备。

pad

数据对齐，不用设置。

reserved

保留，不用设置。

注意：当设备类型为 USBCANFD-100U、USBCANFD-200U、USBCANFD-MINI 时，帧过滤(acc_code、acc_mask)不在这里设置，参考 GetIProperty。

表 1.4 CANFD 默认波特率

产品型号	abit_timing	dbit_timing
PCIE-CANFD-100U PCIE-CANFD-200U PCIE-CANFD-400U	0x09133A(1Mbps) 0x0C1849(800kbps) 0x132776(500kbps) 0x274FEE(250kbps)	0x05010205(8Mbps) 0x0d02040d(4Mbps) 0x1c04091c(2Mbps)
USBCANFD-100U USBCANFD-200U USBCANFD-MINI	0x00018B2E(1Mbps) 0x00018E3A(800kbps) 0x0001975E(500kbps) 0x0001AFBE(250kbps) 0x0041AFBE(125kbps) 0x0041BBEE(100kbps) 0x00C1BBEE(50kbps)	0x00010207(5Mbps) 0x0001020A(4Mbps) 0x0041020A(2Mbps) 0x0081830E(1Mbps)

1.5.3 ZCAN_CHANNEL_ERROR_INFO

ZCAN_CHANNEL_ERROR_INFO 结构体用于装载总线错误信息，结构体将在函数 ZCAN_ReadChannelErrInfo 中被填充。

```
typedef struct tagZCAN_CHANNEL_ERROR_INFO {
    UINT error_code;
    BYTE passive_ErrData[3];
    BYTE arLost_ErrData;
} ZCAN_CHANNEL_ERROR_INFO;
```

成员

error_code

错误码，参考 1.5 错误码定义。

passive_ErrData

当产生的错误中有消极错误时表示为消极错误的错误标识数据。

arLost_ErrData

当产生的错误中有仲裁丢失错误时表示为仲裁丢失错误的错误标识数据。

1.5.4 ZCAN_CHANNEL_STATUS

ZCAN_CHANNEL_STATUS 结构体包含设备中的控制器状态信息，结构体将在函数 ZCAN_ReadChannelStatus 中调用时被填充。

```
typedef struct tagZCAN_CHANNEL_STATUS {
    BYTE errInterrupt;
    BYTE regMode;
    BYTE regStatus;
    BYTE regALCapture;
    BYTE regECCapture;
    BYTE regEWLimit;
    BYTE regRECounter;
    BYTE regTECounter;
    UINT Reserved;
}ZCAN_CHANNEL_STATUS;
```

成员

errInterrupt

中断记录，读操作会清除中断。

regMode

CAN 控制器模式寄存器值。

regStatus

CAN 控制器状态寄存器值。

regALCapture

CAN 控制器仲裁丢失寄存器值。

regECCapture

CAN 控制器错误寄存器值。

regEWLimit

CAN 控制器错误警告限制寄存器值。默认为 96。

regRECounter

CAN 控制器接收错误寄存器值。为 0-127 时，为错误主动状态，为 128-254 为错误被动状态，为 255 时为总线关闭状态。

regTECounter

CAN 控制器发送错误寄存器值。为 0-127 时，为错误主动状态，为 128-254 为错误被动状态，为 255 时为总线关闭状态。

Reserved

保留。

1.5.5 can_frame

can_frame 结构体（位于 canframe.h 中）包含了 CAN 数据信息，发送和接收 CAN 数据都使用该结构。

```
struct can_frame {
        canid_t can_id;    /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8       can_dlc; /* frame payload length in byte (0 .. CAN_MAX_DLEN) */
        __u8       __pad;      /* padding */
        __u8       __res0;    /* reserved / padding */
        __u8       __res1;    /* reserved / padding */
        __u8       data[CAN_MAX_DLEN]/* __attribute__((aligned(8)))*/;
};
```

成员

can_id

帧 ID，32 位长，高 3 位属于标志位，标志位含义如下：

31 位(最高位)代表扩展帧标志，=0 表示标准帧，=1 代表扩展帧；

30 位代表远程帧标志，=0 表示数据帧，=1 表示远程帧；

29 位代表错误帧标准，=0 表示 CAN 帧，=1 表示错误帧，目前只能设置为 0；

其余位代表实际帧 ID 值。可使用头文件中的 MAKE_CAN_ID 宏构造帧 ID，如下：

```
#define MAKE_CAN_ID(id, eff, rtr, err) (id | (!!(eff) << 31) | (!!(rtr) << 30) | (!!(err) << 29))
```

宏 IS_EFF 用于查看是否扩展帧，宏 IS_RTR 用于查看是否远程帧，宏 IS_ERR 用于查看是否错误帧，具体请查看头文件 canframe.h。

can_dlc

帧数据长度。

__pad

对齐，忽略。

__res0

保留。

__res1

保留。

data

帧数据。

1.5.6 canfd_frame

canfd_frame 结构体包含了 CANFD 数据信息，发送和接收 CANFD 数据都使用该结构。

```
struct canfd_frame {
        canid_t can_id;    /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8       len;         /* frame payload length in byte */
        __u8       flags;      /* additional flags for CAN FD,i.e error code */
        __u8       __res0;    /* reserved / padding */
        __u8       __res1;    /* reserved / padding */
        __u8       data[CANFD_MAX_DLEN]/* __attribute__((aligned(8)))*/;
};
```

成员

can_id

同**错误！未找到引用源。** 中的 can_id。

len

帧数据长度

flags

额外标志，如下：

```
#define CANFD_BRS 0x01 //用于 CANFD 加速
```

__res0

保留。

__res1

保留。

data

帧数据。

1.5.7 ZCAN_Transmit_Data

ZCAN_Transmit_Data 用于装载 CAN 数据信息，在函数 ZCAN_Transmit 中使用。

```
typedef struct tagZCAN_Transmit_Data
{
    can_frame frame;
    UINT transmit_type;
}ZCAN_Transmit_Data;
```

成员

frame

帧数据信息，参考 can_frame。

transmit_type

发送方式，0=正常发送, 1=单次发送, 2=自发自收, 3=单次自发自收。

发送方式说明如下：

- **正常发送**：在ID仲裁丢失或发送出现错误时，CAN控制器会自动重发，直到发送成功，或发送超时（超时时间1秒），或总线关闭。
- **单次发送**：在一些应用中，允许部分数据丢失，但不能出现传输延迟时，自动重发就没有意义了。在这些应用中，一般会以固定的时间间隔发送数据，自动重发会导致后面的数据无法发送，出现传输延迟。使用单次发送，仲裁丢失或发送错误，CAN控制器不会重发报文。
- **自发自收**：产生一次带自接收特性的正常发送，在发送完成后，可以从接收缓冲区中读到已发送的报文。
- **单次自发自收**：产生一次带自接收特性的单次发送，在发送出错或仲裁丢失不会执行重发。在发送完成后，可以从接收缓冲区中读到已发送的报文。

1.5.8 ZCAN_TransmitFD_Data

ZCAN_TransmitFD_Data 用于装载 CAN 数据信息，在函数 ZCAN_TransmitFD 中使用。

```
typedef struct tagZCAN_TransmitFD_Data
{
    canfd_frame frame;
    UINT transmit_type;
}ZCAN_TransmitFD_Data;
```

成员

frame

帧数据信息，参考 canfd_frame。

transmit_type

发送方式，参考 ZCAN_TransmitFD_Data。

1.5.9 ZCAN_Receive_Data

ZCAN_Receive_Data 用于装载接收到的 CAN 数据信息，在函数 ZCAN_Receive 中使用。

```
typedef struct tagZCAN_Receive_Data
{
    can_frame frame;
    UINT64    timestamp;
}ZCAN_Receive_Data;
```

成员

frame

帧数据信息，参考 can_frame。

Timestamp

时间戳，单位为微秒。

1.5.10 ZCAN_ReceiveFD_Data

ZCAN_ReceiveFD_Data 用于装载接收到的 CANFD 数据信息，在函数 ZCAN_ReceiveFD 中使用。

```
typedef struct tagZCAN_ReceiveFD_Data
{
    canfd_frame frame;
    UINT64      timestamp;
}ZCAN_ReceiveFD_Data;
```

成员

frame

帧数据信息，参考 canfd_frame。

Timestamp

时间戳，单位为微秒。

1.5.11 ZCAN_AUTO_TRANSMIT_OBJ

ZCAN_AUTO_TRANSMIT_OBJ 用于装载 CAN 定时自动发送参数，在 GetIProperty 中的 SetValue 中使用。

```
typedef struct tagZCAN_AUTO_TRANSMIT_OBJ{
    USHORT enable;
    USHORT index;
    UINT    interval;//定时发送时间。1ms 为单位
    ZCAN_Transmit_Data obj;//报文
}ZCAN_AUTO_TRANSMIT_OBJ, *PZCAN_AUTO_TRANSMIT_OBJ;
```

成员

enable

使能本条报文，0=禁能，1=使能。

index

报文编号，从 0 开始。

interval

发送周期，单位 ms。

obj

发送的报文，参考 ZCAN_Transmit_Data。

1.5.12 ZCANFD_AUTO_TRANSMIT_OBJ

ZCANFD_AUTO_TRANSMIT_OBJ 用于装载 CANFD 定时自动发送参数，在 GetIProperty 中的 SetValue 中使用。

```
typedef struct tagZCANFD_AUTO_TRANSMIT_OBJ{
    USHORT enable;
    USHORT index;
    UINT interval;
    ZCAN_TransmitFD_Data obj;//报文
}ZCANFD_AUTO_TRANSMIT_OBJ, *PZCANFD_AUTO_TRANSMIT_OBJ;
```

成员

enable

使能本条报文，0=禁能，1=使能。

index

报文编号，从 0 开始。

interval

发送周期，单位 ms。

obj

发送的报文，参考 ZCAN_TransmitFD_Data。

1.5.13 ZCLOUD_DEVINFO

ZCLOUD_DEVINFO 用于装载云设备属性信息，在 ZCLOUD_GetUserData 获取的用户数据中使用。

```
typedef struct tagZCLOUD_DEVINFO
{
    int devIndex;
    char type[64];
    char id[64];
    char model[64];
    char fwVer[16];
    char hwVer[16];
    char serial[64];
    BYTE canNum;
    int status;
    BYTE bCanUploads[16];
    BYTE bGpsUpload;
}ZCLOUD_DEVINFO;
```

成员

devIndex

设备索引号，指该设备在该用户关联的所有设备中的索引序号。

type

设备类型字符串。

id

设备唯一识别号，字符串。

model

模块型号字符串。

fwVer

固件版本号字符串，如 V1.01。

hwVer

硬件版本号字符串，如 V1.01。

serial

设备序列号字符串。

canNum

设备 CAN 通道数量。

status

设备状态，0：设备在线，1：设备离线。

bCanUploads

各通道数据云上送使能，0：不上送，1：上送。

bGpsUpload

设备 GPS 数据云上送使能，0：不上送，1：上送。

1.5.14 ZCLOUD_DEV_GROUP_INFO

ZCLOUD_DEV_GROUP_INFO 用于装载设备组信息，每个设备都有所属分组，通过 ZCLOUD_GetUserData 可以获取到相关分组信息。

```
typedef struct tagZCLOUD_DEV_GROUP_INFO
{
    char groupName[64];
    char desc[128];
    char groupId[64];
    ZCLOUD_DEVINFO *pDevices;
    size_t devSize;
}ZCLOUD_DEV_GROUP_INFO;
```

成员

groupName

设备分组名称。

desc

设备分组描述信息。

groupId

设备分组唯一识别号。

pDevices

该分组下所有设备的属性信息，参考 ZCLOUD_DEVINFO。

devSize

该分组下设备个数。

1.5.15 ZCLOUD_USER_DATA

ZCLOUD_USER_DATA 用于装载用户信息，包含用户基本信息以及用户拥有的设备信息，通过 ZCLOUD_GetUserData 可以获取到。

```
typedef struct tagZCLOUD_USER_DATA
{
    char username[64];
    char mobile[64];
    char email[64];
    ZCLOUD_DEV_GROUP_INFO *pDevGroups;
    size_t devGroupSize;
}ZCLOUD_USER_DATA;
```

成员

username

用户名字符串。

mobile

用户手机号。

email

用户邮箱。

pDevGroups

用户拥有的设备组，参考 ZCLOUD_DEV_GROUP_INFO。

devGroupSize

用户设备组个数。

1.5.16 ZCLOUD_GPS_FRAME

ZCLOUD_GPS_FRAME 用于装载设备 GPS 数据，通过 ZCLOUD_ReceiveGPS 可以获取到。

```
typedef struct tagZCLOUD_GPS_FRAME
{
    float latitude;
    float longitude;
    float speed;
    struct __gps_time {
        USHORT    year;
        USHORT    mon;
        USHORT    day;
        USHORT    hour;
        USHORT    min;
        USHORT    sec;
    }tm;
} ZCLOUD_GPS_FRAME;
```

成员

latitude

纬度。

longitude

经度。

speed

速度。

tm

时间结构。

1.6 接口库函数说明

1.6.1 ZCAN_OpenDevice

此函数用以打开设备。注意一个设备只能打开一次。

```
DEVICE_HANDLE    ZCAN_OpenDevice(UINT device_type, UINT device_index, UINT reserved);
```

参数

device_type

设备类型，具体值请参考头文件 zlgcan.h。

device_index

设备索引号，比如当只有一个 USBCANFD-200U 时，索引号为 0，这时再插入一个 USBCANFD-200U，那么后面插入的这个设备索引号就是 1，以此类推。

reserved

保留，未使用，设置为 0。

返回值

为 INVALID_DEVICE_HANDLE 表示操作失败，否则表示操作成功，返回设备句柄值，保存该句柄值，之后的操作需要使用到。

1.6.2 ZCAN_CloseDevice

此函数用以关闭设备，关闭与打开一一对应。

```
UINT ZCAN_CloseDevice(DEVICE_HANDLE device_handle);
```

参数

device_handle

需要关闭的设备的句柄值，即 ZCAN_OpenDevice 成功返回的值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.3 ZCAN_GetDeviceInf

使用该函数获取设备信息。

```
UINT ZCAN_GetDeviceInf(DEVICE_HANDLE device_handle, ZCAN_DEVICE_INFO* pInfo);
```

参数

device_handle

设备句柄值。

pInfo

设备信息结构体，参考 ZCAN_DEVICE_INFO。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.4 ZCAN_IsDeviceOnLine

该函数用于检测设备是否在线，仅支持 USB 系列设备。

```
UINT ZCAN_IsDeviceOnLine(DEVICE_HANDLE device_handle);
```

参数

device_handle

设备句柄值。

返回值

设备在线=STATUS_ONLINE，不在线=STATUS_OFFLINE。

1.6.5 ZCAN_InitCAN

该函数用于初始化 CAN。

```
CHANNEL_HANDLE ZCAN_InitCAN(DEVICE_HANDLE device_handle, UINT can_index,
ZCAN_CHANNEL_INIT_CONFIG* pInitConfig);
```

参数

device_handle

设备句柄值。

can_index

设备索引号，比如当只有一个 USBCANFD-200U 时，索引号为 0，这时再插入一个 USBCANFD-200U，那么后面插入的这个设备索引号就是 1，以此类推。

pInitConfig

初始化结构，参考 ZCAN_CHANNEL_INIT_CONFIG。

返回值

为 INVALID_CHANNEL_HANDLE 表示操作失败，否则表示操作成功，返回通道句柄值，保存该句柄值，之后的操作需要使用到。

1.6.6 ZCAN_StartCAN

该函数用于启动 CAN。

```
UINT ZCAN_StartCAN(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.7 ZCAN_ResetCAN

该函数用于复位 CAN，可通过 ZCAN_StartCAN 恢复。

```
UINT ZCAN_ResetCAN(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.8 ZCAN_ClearBuffer

该函数用于清除库接收缓冲区。

```
UINT ZCAN_ClearBuffer(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.9 ZCAN_ReadChannelErrInfo

该函数用于读取通道的错误信息。

```
UINT ZCAN_ReadChannelErrInfo(CHANNEL_HANDLE channel_handle, ZCAN_CHANNEL_ERROR_INFO* pErrInfo);
```

参数

channel_handle

通道句柄值。

pErrInfo

错误信息结构，参考 ZCAN_CHANNEL_ERROR_INFO。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.10 ZCAN_ReadChannelStatus

该函数用于读取通道的状态信息。

```
UINT ZCAN_ReadChannelStatus(CHANNEL_HANDLE channel_handle, ZCAN_CHANNEL_STATUS* pCANStatus);
```

参数

channel_handle

通道句柄值。

pCANStatus

状态信息结构，参考 ZCAN_CHANNEL_STATUS。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.11 ZCAN_Transmit

该函数用于发送 CAN 数据。

```
UINT ZCAN_Transmit(CHANNEL_HANDLE channel_handle, ZCAN_Transmit_Data* pTransmit, UINT len);
```

参数

channel_handle

通道句柄值。

pTransmit

要发送的帧结构体 ZCAN_Transmit_Data 数组的首指针。

len

帧数目

返回值

返回实际发送成功的帧数。

1.6.12 ZCAN_TransmitFD

该函数用于发送 CANFD 数据。

```
UINT ZCAN_TransmitFD(CHANNEL_HANDLE channel_handle, ZCAN_TransmitFD_Data* pTransmit, UINT len);
```

参数

channel_handle

通道句柄值。

pTransmit

要发送的帧结构体 ZCAN_TransmitFD_Data 数组的首指针。

len

帧数目

返回值

返回实际发送成功的帧数。

1.6.13 ZCAN_GetReceiveNum

获取缓冲区中 CAN 或 CANFD 帧数目。

```
UINT ZCAN_GetReceiveNum(CHANNEL_HANDLE channel_handle, BYTE type);
```

参数

channel_handle

通道句柄值。

type

获取 CAN 或 CANFD 帧数目，0=CAN，1=CANFD。

返回值

返回帧数目。

1.6.14 ZCAN_Receive

该函数用于接收 CAN 数据，建议使用 ZCAN_GetReceiveNum 确保缓冲区有数据再使用。

```
UINT ZCAN_Receive(CHANNEL_HANDLE channel_handle, ZCAN_Receive_Data* pReceive, UINT len, INT wait_time = -1);
```

参数

channel_handle

通道句柄值。

pReceive

用来接收的帧结构体 ZCAN_Receive_Data 数组的首指针。

len

用来接收的帧结构体数组的长度（本次接收的最大帧数，实际返回值小于等于这个值）。

wait_time

缓冲区无数据，函数阻塞等待时间，以毫秒为单位，若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的帧数。

1.6.15 ZCAN_ReceiveFD

该函数用于接收 CANFD 数据，建议使用 ZCAN_GetReceiveNum 确保缓冲区有数据再使用。

```
UINT ZCAN_ReceiveFD(CHANNEL_HANDLE channel_handle, ZCAN_ReceiveFD_Data* pReceive, UINT len, INT wait_time = -1);
```

参数

channel_handle

通道句柄值。

pReceive

用来接收的帧结构体 ZCAN_ReceiveFD_Data 数组的首指针。

len

用来接收的帧结构体数组的长度（本次接收的最大帧数，实际返回值小于等于这个值）。

wait_time

缓冲区无数据，函数阻塞等待时间，以毫秒为单位，若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的帧数。

1.6.16 GetIProperty

该函数返回属性配置接口，属性配置接口用于设置设备属性。

```
IProperty* GetIProperty(DEVICE_HANDLE device_handle);
```

参数

device_handle

设备句柄值。

返回值

返回属性配置接口指针，空则表示操作失败。

1.6.17 ReleaseIProperty

释放属性接口，与 GetIProperty 结对使用。

```
UINT ReleaseIProperty(IProperty * pIProperty);
```

参数

pIProperty

GetIProperty 的返回值。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.6.18 IProperty

该接口用于获取/设置设备详细参数数据。

```
typedef struct    tagIProperty
{
        SetValueFunc          SetValue;
        GetValueFunc          GetValue;
        GetPropertysFunc     GetPropertys;
}IProperty;
```

成员

SetValue

设置属性值，具体设置参考属性表。

GetValue

获取属性值，暂未实现。

GetPropertys

用于返回设备包含的所有属性。

示例

```
char path[50] = {0};
char value[50] = {0};
IProperty* property_ = GetIProperty(device_handle);// device_handle 为设备句柄
sprintf_s(path, "%d/canfd_abit_baud_rate", 0);//0 代表通道 0
sprintf_s(value, "%d", 1000000);//1Mbps 为 1000000
if (0 == property_->SetValue(path, value))
{
        return FALSE;
}
```

1.6.19 ZCLOUD_SetServerInfo

该函数用于设置云服务器相关连接信息。

```
void ZCLOUD_SetServerInfo(const char* httpSvr, unsigned short httpPort, const char* mqttSvr, unsigned short mqttPort);
```

参数

httpSvr

用户认证服务器地址，IP 地址或域名。

httpPort

用户认证服务器端口号。

mqttSvr

数据服务器地址，IP 地址或域名， 一般与认证服务器相同。

mqttPort

数据服务器端口号。

返回值

无。

1.6.20 ZCLOUD_ConnectServer

该函数用于连接云服务器，会先登录认证服务器，然后连接到数据服务器。

```
UINT ZCLOUD_ConnectServer(const char* username, const char* password);
```

参数

username

用户名。

password

密码。

返回值

0：成功，1：失败， 2：认证服务器连接错误，3：用户信息验证错误，4：数据服务器连接错误。

1.6.21 ZCLOUD_IsConnected

该函数用于判断是否已经连接到云服务器。

```
bool ZCLOUD_IsConnected();
```

参数

返回值

true：已连接，false：未连接。

1.6.22 ZCLOUD_DisconnectServer

该函数用于断开云服务器连接。

```
UINT ZCLOUD_DisconnectServer()
```

参数

返回值

0：成功，1：失败。

1.6.23 ZCLOUD_GetUserData

获取用户数据，包括用户基本信息和所拥有设备信息。

```
const ZCLOUD_USER_DATA* ZCLOUD_GetUserData();
```

参数

返回值

用户数据结构指针。

1.6.24 ZCLOUD_ReceiveGPS

该函数用于接收云设备 GPS 数据。

```
UINT ZCLOUD_ReceiveGPS(DEVICE_HANDLE device_handle, ZCLOUD_GPS_FRAME* pReceive, UINT len, int wait_time DEF(-1));
```

参数

device_handle

设备句柄值。

pReceive

用来接收的帧结构体 ZCLOUD_GPS_FRAME 数组的首指针。

len

用来接收的帧结构体数组的长度（本次接收的最大帧数，实际返回值小于等于这个值）。

wait_time

缓冲区无数据，函数阻塞等待时间，以毫秒为单位，若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的帧数。

1.7 属性表

SetValue 的 path、value 列表如下：

1、支持设备：USBCANFD-100U、USBCANFD-200U、USBCANFD-MINI

参数	path	value
CANFD 标准	n/canfd_standard， n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	“0”=CANFD ISO “1”=CANFD BOSCH
自定义波特率	n/baud_rate_custom	请使用 canmaster 目录下的 baudcal 计算
终端电阻	n/initenal_resistance	“0”=禁能 “1”=使能
定时发送 CAN	n/auto_send	**错误！未找到引用源。** 把该结构指针转换为 char*
定时发送 CANFD	n/auto_send_canfd	ZCANFD_AUTO_TRANSMIT_OBJ 把该结构指针转换为 char*
清空定时发送	n/clear_auto_send	“0”
使定时发送生效	n/apply_auto_send	“0”
设置序列号	n/set_cn	最多 128 字符
升级	n/update	文件路径
时钟	n/clock	“60000000”=60M
清除滤波	n/filter_clear	“0”
滤波模式	n/filter_mode	“0”=标准帧 “1”=扩展帧
滤波起始帧	n/filter_start	“0x00000000”，16 进制字符
滤波结束帧	n/filter_end	“0x00000000”，16 进制字符
滤波生效	n/filter_ack	“0”

发送重试超时时间	n/tx_timeout	“1000”，单位 ms，最大值为 4000

2、支持设备：PCIE-CANFD-100U、PCIE-CANFD-200U、PCIE-CANFD-400U

参数	path	value
自定义波特率	n/baud_rate_custom n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	请使用 canmaster 目录下的 baudcal 计算
发送类型	n/send_type	“0”=正常发送 “1”=自发自收
发送失败后重发次数	n/retry	“0”，整型
定时发送 CANFD	n/auto_send_canfd	ZCANFD_AUTO_TRANSMIT_OBJ 把该结构指针转换为 char*
清空定时发送	n/clear_auto_send	“0”

3、支持设备：PCI-5010-U、PCI-5020-U、USBCAN-E-U、USBCAN-2E-U

参数	path	value
波特率	n/baud_rate n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	“1000000”、“800000”、“500000”、“250000”、“125000”、“100000”、“50000”、
自定义波特率	n/baud_rate_custom	请使用 canmaster 目录下的 baudcal 计算
清除滤波	n/filter_clear	“0”
滤波模式	n/filter_mode	“0”=标准帧 “1”=扩展帧
滤波起始帧	n/filter_start	“0x00000000”，16 进制字符
滤波结束帧	n/filter_end	“0x00000000”，16 进制字符
滤波生效	n/filter_ack	“0”

4、支持设备：USBCAN-4E-U

参数	path	value
自定义波特率	n/baud_rate_custom n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	请使用 canmaster 目录下的 baudcal 计算
转发到 CAN0	n/redirect/can0	“0 1”=转发 “0 0”=不转发
转发到 CAN1	n/redirect/can1	“1 1”=转发 “1 0”=不转发
转发到 CAN2	n/redirect/can2	“2 1”=转发 “2 0”=不转发
转发到 CAN3	n/redirect/can3	“3 1”=转发 “3 0”=不转发
清除滤波	n/filter_clear	“0”
滤波模式	n/filter_mode	“0”=标准帧 “1”=扩展帧
滤波起始帧	n/filter_start	“0x00000000”，16 进制字符
滤波结束帧	n/filter_end	“0x00000000”，16 进制字符
滤波生效	n/filter_ack	“0”

5、支持设备：USBCAN-I、USBCAN-II、PCI9810、PCI9820、PCI5110、PCIe-9110I、PCI9820I、PCIE-9221、PCIe-9120I、PCI5121、CANalyst-II+

参数	path	value
自定义波特率	n/baud_rate_custom n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	请使用 canmaster 目录下的 baudcal 计算

6、支持设备：USBCAN-8E-U

参数	path	value
自定义波特率	n/baud_rate_custom n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	请使用 canmaster 目录下的 baudcal 计算
转发到 CAN0	n/redirect/can0	“0 1”=转发

		“0 0”=不转发
转发到 CAN1	n/redirect/can1	“1 1”=转发 “1 0”=不转发
转发到 CAN2	n/redirect/can2	“2 1”=转发 “2 0”=不转发
转发到 CAN3	n/redirect/can3	“3 1”=转发 “3 0”=不转发
转发到 CAN4	n/redirect/can4	“4 1”=转发 “4 0”=不转发
转发到 CAN5	n/redirect/can5	“5 1”=转发 “5 0”=不转发
转发到 CAN6	n/redirect/can6	“6 1”=转发 “6 0”=不转发
转发到 CAN7	n/redirect/can7	“7 1”=转发 “7 0”=不转发

7、支持设备：CANDTU-100UR、CANDTU-200UR

参数	path	value
波特率	n/baud_rate n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	“1000000”、“800000”、“500000”、“250000”、“125000”、“100000”、“50000”、
自定义波特率	n/baud_rate_custom	请使用 canmaster 目录下的 baudcal 计算
内置 120 欧电阻	n/internal_resistance	“0”=禁能 “1”=使能
工作模式	n/work_mode	“1”=正常模式 “0”=只听模式
验收码	n/acc_code	“0x00000000”，16 进制字符
屏蔽码	n/acc_mask	“0xFFFFFFFF”，16 进制字符
使设置生效	n/confirm	“0”，在设置最后调用

8、支持设备：CANDTU-NET、CANET-TCP、CANWIFI-TCP

参数	path	value
工作模式	n/work_mode n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	“1”=服务器 “0”=客户端
本地端口	n/local_port	“4001”，整型
Ip 地址	n/ip	“192.168.0.178”
工作端口	n/work_port	“4001”，整型

9、支持设备：CANET-UDP、CANWIFI-UDP

参数	path	value
本地端口	n/local_port n 代表通道号，如 0 代表通道 0,1 代表通道 1，下同	“4001”，整型
Ip 地址	n/ip	“192.168.0.178”
工作端口	n/work_port	“4001”，整型

1.8 错误码定义

1.5 错误码定义

名称	值	描述
CAN 错误码		
ZCAN_ERROR_CAN_OVERFLOW	0x00000001	CAN 控制器内部 FIFO 溢出
ZCAN_ERROR_CAN_ERRALARM	0x00000002	CAN 控制器错误报警
ZCAN_ERROR_CAN_PASSIVE	0x00000004	CAN 控制器消极错误
ZCAN_ERROR_CAN_LOSE	0x00000008	CAN 控制器仲裁丢失
ZCAN_ERROR_CAN_BUSERR	0x00000010	CAN 控制器总线错误
ZCAN_ERROR_CAN_BUSOFF	0x00000020	CAN 控制器总线关闭
通用错误码		
ZCAN_ERROR_DEVICEOPENED	0x00000100	设备已经打开
ZCAN_ERROR_DEVICEOPEN	0x00000200	打开设备错误
ZCAN_ERROR_DEVICENOTOPEN	0x00000400	设备没有打开
ZCAN_ERROR_BUFFEROVERFLOW	0x00000800	缓冲区溢出
ZCAN_ERROR_DEVICENOTEXIST	0x00001000	此设备不存在
ZCAN_ERROR_LOADKERNELDLL	0x00002000	装载动态库失败
ZCAN_ERROR_CMDFAILED	0x00004000	执行命令失败错误码
ZCAN_ERROR_BUFFERCREATE	0x00008000	内存不足
CANET 错误码		
ZCAN_ERROR_CANETE_PORTOPENED	0x00010000	端口已经被打开
ZCAN_ERROR_CANETE_INDEXUSED	0x00020000	设备索引号已经被占用
ZCAN_ERROR_REF_TYPE_ID	0x00030001	SetReference 或 GetReference 是传递的 RefType 是不存在
ZCAN_ERROR_CREATE_SOCKET	0x00030002	创建 Socket 时失败
ZCAN_ERROR_OPEN_CONNECT	0x00030003	打开 socket 的连接时失败，可能设备连接已经存在
ZCAN_ERROR_NO_STARTUP	0x00030004	设备没启动
ZCAN_ERROR_NO_CONNECTED	0x00030005	设备无连接
ZCAN_ERROR_SEND_PARTIAL	0x00030006	只发送了部分的 CAN 帧
ZCAN_ERROR_SEND_TOO_FAST	0x00030007	数据发得太快，Socket 缓冲

2. 销售与服务网络

广州致远电子有限公司

地址：广州市天河区车陂路黄洲工业区 7 栋 2 楼

邮编：510660

网址：www.zlg.cn

全国销售与服务电话：400-888-4005

全国服务电话：400-888-4005

销售与服务网络：

广州总公司

广州市天河区车陂路黄洲工业区 7 栋 2 楼

电话：020-28267893

上海分公司

上海市北京东路 668 号科技京城东楼 12E 室

电话：021-53865720-801

北京分公司

北京市丰台区马家堡路 180 号 蓝光云鼎 208 室

电话：010-62536178

深圳分公司

深圳市福田区深南中路 2072 号电子大厦 12 楼 1203 室

电话：0755-82941683　0755-82907445

武汉分公司

武汉市洪山区民族大道江南家园 1 栋 3 单元 602

电话：027-62436478　13006324181

南京分公司

南京市秦淮区汉中路 27 号友谊广场 17 层 F、G 区

电话：025-68123919

杭州分公司

杭州市西湖区紫荆花路 2 号杭州联合大厦 A 座 4 单元 508

电话：0571-86483297

成都分公司

成都市一环路南 2 段 1 号数码科技大厦 319 室

电话：028-85439836-805

郑州分公司

河南省郑州市中原区建设西路与百花路东南角锦绣华庭 A 座 1502

电话：400-888-4005　(0371)66868897

重庆分公司

重庆市九龙坡区石桥铺科园一路二号大西洋国际大厦（百脑会）2705 室

电话：023-68797619

西安办事处

西安市长安北路 54 号太平洋大厦 1201 室

电话：029-87881295

天津办事处

天津市河东区津塘路与十一经路交口鼎泰大厦 1004

电话：022-24216606

青岛办事处

山东省青岛市李沧区青山路 689 号宝龙公寓 3 号楼 701 室

电话：0532-58879795　17660216799

请您用以上方式联系我们，我们会为您安排样机现场演示，感谢您对我公司产品的关注！