User Manual

接口函数使用手册

CAN/CANFD 接口卡系列产品

UM01010101 V1.05 Date: 2019/11/19

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。

修订历史

版本	日期	原因
V1.00	2019/01/09	创建文档
V1.01	2019/03/18	更新文档页眉页脚、“销售与服务网络”内容和新增“免责声明”内容
V1.02	2019/09/06	调整云设备数据结构
V1.03	2019/09/24	统一波特率设置，添加代码示例
V1.04	2019/10/14	USBCAN-2E-U 属性表的滤波项添加说明
V1.05	2019/11/19	属性表添加调用顺序说明

目 录

第 1 章 ZLGCAN接口编程……1
1.1 资料包简介……1
1.2 开发流程图……1
1.2.1 普通CAN卡开发流程……1
1.2.2 云设备……3
1.3 数据结构定义……5
1.3.1 ZCAN_DEVICE_INFO……5
1.3.2 ZCAN_CHANNEL_INIT_CONFIG……6
1.3.3 ZCAN_CHANNEL_ERROR_INFO……9
1.3.4 ZCAN_CHANNEL_STATUS……9
1.3.5 can_frame……10
1.3.6 canfd_frame……11
1.3.7 ZCAN_Transmit_Data……11
1.3.8 ZCAN_TransmitFD_Data……12
1.3.9 ZCAN_Receive_Data……12
1.3.10 ZCAN_ReceiveFD_Data……13
1.3.11 ZCAN_AUTO_TRANSMIT_OBJ……13
1.3.12 ZCANFD_AUTO_TRANSMIT_OBJ……14
1.3.13 ZCLOUD_DEVINFO……14
1.3.14 ZCLOUD_USER_DATA……15
1.3.15 ZCLOUD_GPS_FRAME……16
1.3.16 USBCANFDTxTimeStamp……16
1.3.17 IProperty……17
1.3.18 ZCAN_LIN_MSG……17
1.3.19 ZCAN_LIN_INIT_CONFIG……18
1.4 接口库函数说明……19
1.4.1 ZCAN_OpenDevice……19
1.4.2 ZCAN_CloseDevice……19
1.4.3 ZCAN_GetDeviceInf……19
1.4.4 ZCAN_IsDeviceOnLine……20
1.4.5 ZCAN_InitCAN……20
1.4.6 ZCAN_StartCAN……20
1.4.7 ZCAN_ResetCAN……20
1.4.8 ZCAN_ClearBuffer……21
1.4.9 ZCAN_ReadChannelErrInfo……21
1.4.10 ZCAN_ReadChannelStatus……21
1.4.11 ZCAN_Transmit……22
1.4.12 ZCAN_TransmitFD……22
1.4.13 ZCAN_GetReceiveNum……22
1.4.14 ZCAN_Receive……23
1.4.15 ZCAN_ReceiveFD……23

1.4.16 GetIProperty……23
1.4.17 ReleaseIProperty……24
1.4.18 ZCLOUD_SetServerInfo……24
1.4.19 ZCLOUD_ConnectServer……24
1.4.20 ZCLOUD_IsConnected……25
1.4.21 ZCLOUD_DisconnectServer……25
1.4.22 ZCLOUD_GetUserData……25
1.4.23 ZCLOUD_ReceiveGPS……25
1.4.24 ZCAN_InitLIN……25
1.4.25 ZCAN_StartLIN……26
1.4.26 ZCAN_ResetLIN……26
1.4.27 ZCAN_TransmitLIN……26
1.4.28 ZCAN_GetLINReceiveNum……27
1.4.29 ZCAN_ReceiveLIN……27
1.4.30 ZCAN_SetLINSlaveMsg……27
1.4.31 ZCAN_ClearLINSlaveMsg……28
1.5 属性表……28
1.5.1 USBCANFD系列……28
1.5.2 PCIECANFD系列……30
1.5.3 USBCAN-xE-U PCI-50x0-U系列……31
1.5.4 USBCAN-4E-U……32
1.5.5 USBCAN-8E-U……32
1.5.6 CANDTU-x00UR……33
1.5.7 NET-TCP系列……34
1.5.8 NET-UDP系列……34
1.5.9 其他接口卡……34
1.6 错误码定义……36

第1章 ZLGCAN接口编程

✍本章导读

为满足市场发展的需要，广州致远电子有限公司推出了各式各样的 CAN(FD)接口卡，例如 USBCANFD 系列、PCIECANFD 系列和 USBCAN 系列等等。除了必要的硬件支持，更是配备了功能完善的分析软件 ZCANPro，给 CAN(FD)开发和诊断带来了很大的便利。

为满足接口卡接入集成系统的需要，公司推出了统一的编程接口，同时支持 CAN 和 CANFD。除了简单易用的接口，还配以接口使用例程和接口使用说明。本章将对编程接口的使用作详尽的描述，务必带给您更好的体验。

1.1 节为资料包简介，1.2 节为开发流程图，1.3 节为数据结构定义，1.4 节为接口库函数说明，1.5 节为属性表，1.6 节为错误码定义。

1.1 资料包简介

接口库以基于 window 系统的动态链接库(DLL)的方式提供，可实现设备打开、配置、报文收发、关闭等功能。接口库采用 visual studio 2008 开发，依赖运行库 2008 版本，需要确保计算机中已包含该运行库，否则可到微软官方网站下载安装。

如图 1.1 所示，资料包中包含了 zlgcan.dll、kerneldlls 和 zlgcan 文件夹，其中 kerneldlls 文件夹包含具体接口卡的操作库，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。

开发编程直接加载 zlgcan.dll 即可，zlgcan.h 为接口描述头文件，zlgcan.dll 和 kerneldlls 文件夹需要放在可执行程序生成目录下。

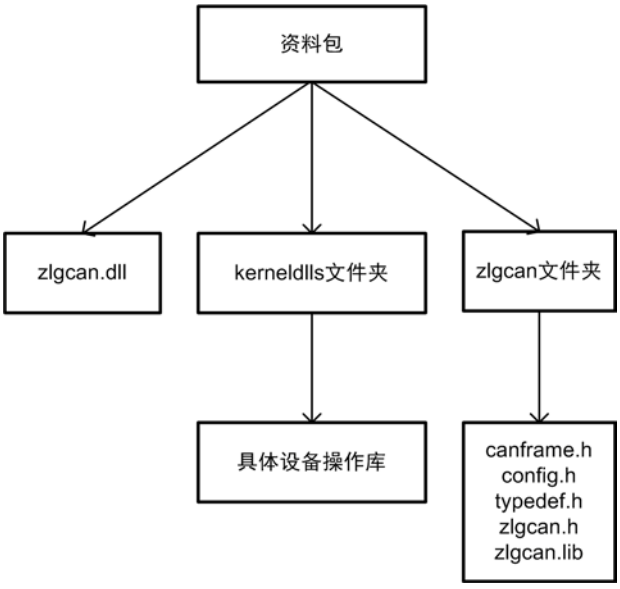

图 1.1 资料包结构

1.2 开发流程图

1.2.1 普通CAN卡开发流程

该开发流程适合我司出品的所有 CAN/CANFD 接口卡，请按照开发流程进行二次开发，

流程如图 1.2 所示，开发代码示例如程序清单 1.1 所示。

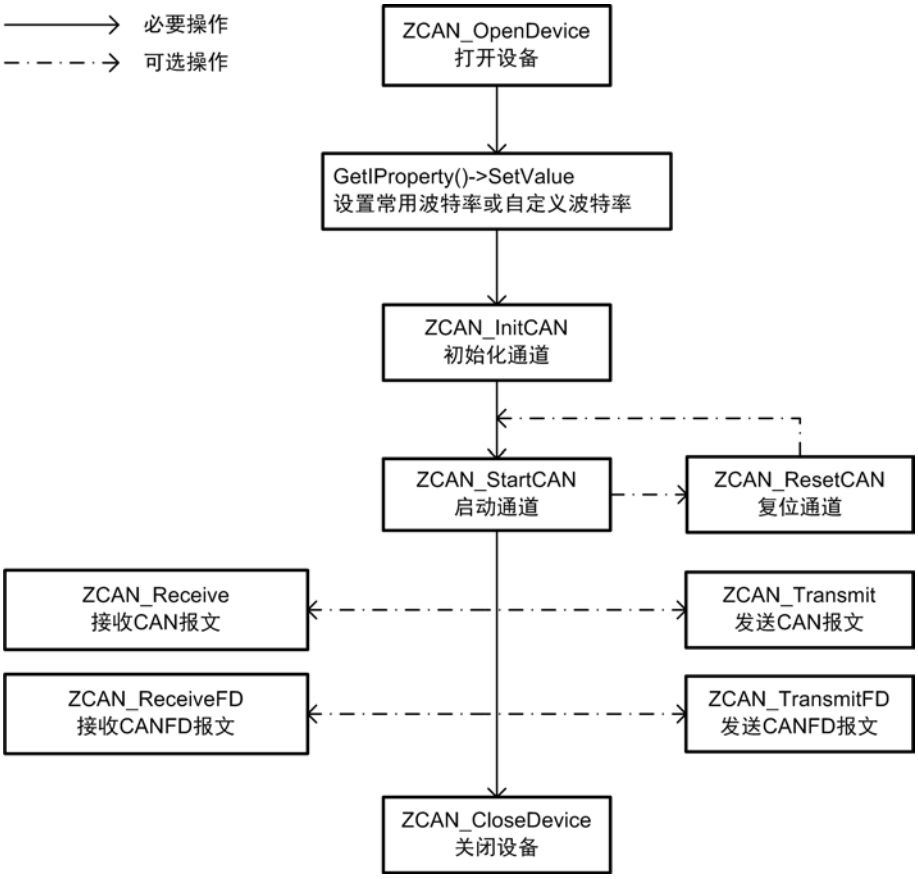

图 1.2　CAN 卡开发流程图

程序清单 1.1　开发代码示例

```
    //ZCAN_USBCAN_2E_U 为设备类型，请根据实际修改
    DEVICE_HANDLE dhandle = ZCAN_OpenDevice(ZCAN_USBCAN_2E_U, 0, 0);
    if (INVALID_DEVICE_HANDLE == dhandle)
    {
        std::cout << "打开设备失败" << std::endl;
        return 0;
    }
    IProperty* property = GetIProperty(dhandle);
    if (NULL == property)
    {
        std::cout << "属性指针为空" << std::endl;
        goto end;
    }
    //CAN 设备设置波特率的 key 为 baud_rate，值 1000000 为 1000kbps, 800000 为 800kbps, 其它请查看属性表
    //若为 CANFD 设备，设置冲裁域波特率的 key 为 canfd_abit_baud_rate，数据域波特率为 canfd_dbit_baud_rate，请注意区分 CAN 和 CANFD 设备设置波特率的区别。
    if (property->SetValue("0/baud_rate", "1000000") != STATUS_OK)
    {
```

```
        std::cout << "设置波特率失败" << std::endl;
        goto end;
    }
    ZCAN_CHANNEL_INIT_CONFIG cfg;
    memset(&cfg, 0, sizeof(cfg));
    cfg.can_type = TYPE_CAN;//CANFD 设备为 TYPE_CANFD
    cfg.can.filter = 0;
    cfg.can.mode = 0; //正常模式, 1 为只听模式
    cfg.can.acc_code = 0;
    cfg.can.acc_mask = 0xffffffff;
    CHANNEL_HANDLE chHandle = ZCAN_InitCAN(dhandle, 0, &cfg);
    if (INVALID_CHANNEL_HANDLE == chHandle)
    {
        std::cout << "初始化通道失败" << std::endl;
        goto end;
    }
    if (ZCAN_StartCAN(chHandle) != STATUS_OK)
    {
        std::cout << "启动通道失败" << std::endl;
        goto end;
    }
    ZCAN_Transmit_Data frame;
    memset(&frame, 0, sizeof(frame));
    frame.frame.can_id = MAKE_CAN_ID(0x100, 1, 0, 0);
    frame.frame.can_dlc = 8;
    BYTE data[] = {1, 2, 3, 4, 5, 6, 7, 8};
    memcpy(frame.frame.data, data, sizeof(data));
    if (ZCAN_Transmit(chHandle, &frame, 1) != 1)
    {
        std::cout << "发送数据失败" << std::endl;
        goto end;
    }
end:
    ReleaseIProperty(property);
    ZCAN_CloseDevice(dhandle);
```

1.2.2 云设备

本系列的开发流程图如图 1.9 所示，适用的设备有：ZCAN_CLOUD (46)。

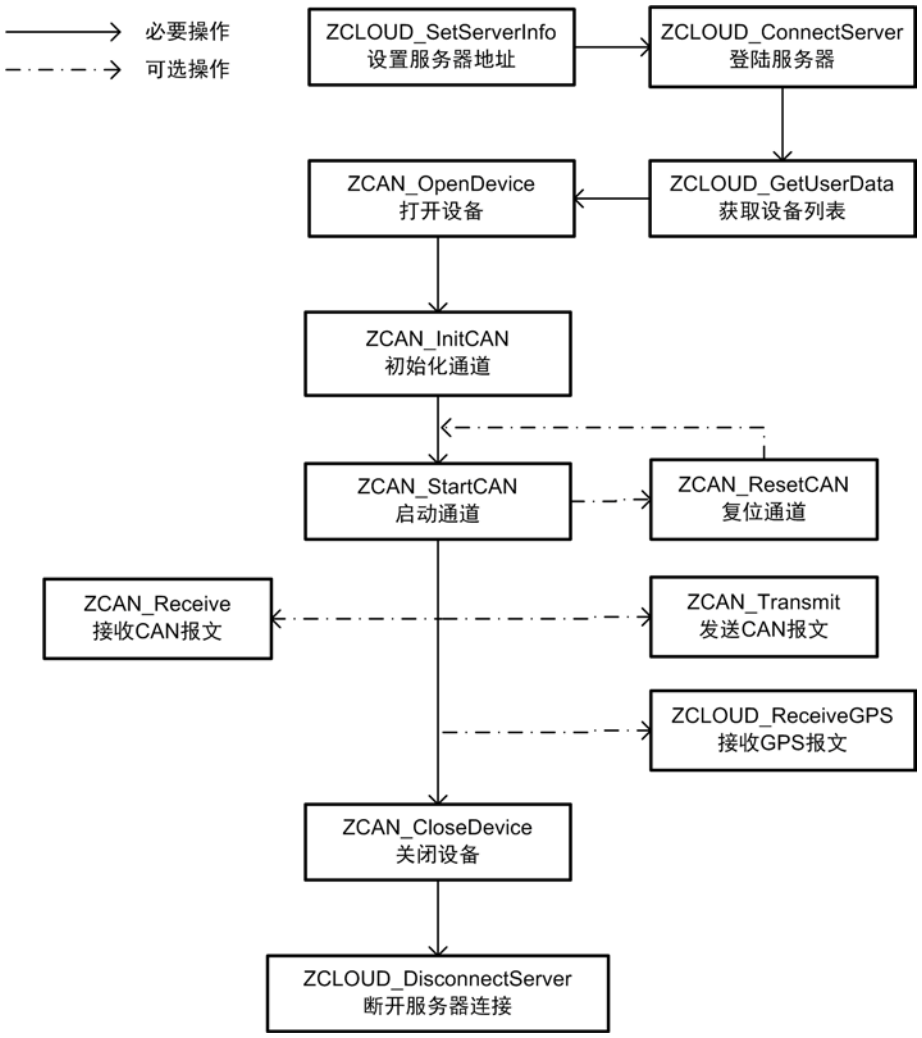
必要操作
可选操作
ZCLOUD_SetServerInfo
设置服务器地址
ZCLOUD_ConnectServer
登陆服务器
ZCAN_OpenDevice
打开设备
ZCLOUD_GetUserData
获取设备列表
ZCAN_InitCAN
初始化通道
ZCAN_StartCAN
启动通道
ZCAN_ResetCAN
复位通道
ZCAN_Receive
接收CAN报文
ZCAN_Transmit
发送CAN报文
ZCLOUD_ReceiveGPS
接收GPS报文
ZCAN_CloseDevice
关闭设备
ZCLOUD_DisconnectServer
断开服务器连接

图 1.3　云设备流程图

1.3 数据结构定义

1.3.1 ZCAN_DEVICE_INFO

结构体详情见程序清单 1.1，包含设备的一些基本信息，在函数 ZCAN_GetDeviceInf 中被填充。

程序清单 1.2 ZCAN_DEVICE_INFO 结构体成员

```
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，16 进制，比如 0x0100 表示 V1.00。

fw_Version

固件版本号，16 进制。

dr_Version

驱动程序版本号，16 进制。

in_Version

接口库版本号，16 进制。

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

仅作保留，不设置。

1.3.2 ZCAN_CHANNEL_INIT_CONFIG

结构体详情见程序清单 1.2，定义了初始化配置的参数，调用 ZCAN_InitCAN 之前，要先初始化该结构体。

程序清单 1.3 ZCAN_CHANNEL_INIT_CONFIG 结构体成员

```
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。

- **CAN 设备**

acc_code

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，此报文可以被接收，否则不接收。推荐设置为 0。

acc_mask

SJA1000 的帧过滤屏蔽码，对接收的 CAN 帧 ID 进行过滤，位为 0 的是“有关位”，

位为 1 的是“无关位”。推荐设置为 0xFFFFFFFF，即全部接收。

reserved

仅作保留，不设置。

filter

滤波方式，=1 表示单滤波，=0 表示双滤波。

timing0

忽略，不设置。

timing1

忽略，不设置。

mode

工作模式，=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）。

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，帧过滤（acc_code 和 acc_mask 忽略）采用 GetIProperty 设置，详见 1.4.16。

CANFD 设备

acc_code

验收码，同 CAN 设备。

acc_mask

屏蔽码，同 CAN 设备。

abit_timing

忽略，不设置。

dbit_timing

忽略，不设置。

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，不设置。

reserved

仅作保留，不设置。

注意：当设备类型为 USBCANFD-100U、USBCANFD-200U、USBCANFD-MINI 时，帧过滤(acc_code 和 acc_mask 忽略)采用 GetIProperty 设置，详见 1.4.16。

1.3.3 ZCAN_CHANNEL_ERROR_INFO

结构体详情见程序清单 1.3，包含总线错误信息，在函数 ZCAN_ReadChannelErrInfo 中被填充。

程序清单 1.4　ZCAN_CHANNEL_ERROR_INFO 结构体成员

```
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，详见 1.6。

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。

1.3.4 ZCAN_CHANNEL_STATUS

结构体详情见程序清单 1.4，包含控制器状态信息，在函数 ZCAN_ReadChannelStatus 中被填充。

程序清单 1.5　ZCAN_CHANNEL_STATUS 结构体成员

```
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 时，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 时，为错误被动状态；为 255 时，为总线关闭状态。

reserved

仅作保留，不设置。

1.3.5 can_frame

结构体详情见程序清单 1.5，包含了 CAN 报文信息。

程序清单 1.6　can_frame 结构体成员

```
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，高 3 位属于标志位，标志位含义如下：

第 31 位(最高位)代表扩展帧标志，=0 表示标准帧，=1 代表扩展帧，宏 IS_EFF 可获取该标志；

第 30 位代表远程帧标志，=0 表示数据帧，=1 表示远程帧，宏 IS_RTR 可获取该标志；

第 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，使用宏 MAKE_CAN_ID 构造 ID，使用宏 GET_ID 获取 ID。

can_dlc

数据长度。

__pad

对齐，忽略。

__res0

仅作保留，不设置。

__res1

仅作保留，不设置。

data

报文数据，有效长度为 can_dlc。

1.3.6 canfd_frame

结构体详情见程序清单 1.6，包含了 CANFD 报文信息。

程序清单 1.7　canfd_frame 结构体成员

```
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

帧 ID，同 1.3.5。

len

数据长度。

flags

额外标志，比如使用 CANFD 加速，则设置为宏 CANFD_BRS。

__res0

仅作保留，不设置。

__res1

仅作保留，不设置。

data

报文数据，有效长度为 len。

1.3.7 ZCAN_Transmit_Data

结构体详情见程序清单 1.7，包含发送的 CAN 报文信息，在函数 ZCAN_Transmit 中使用。

程序清单 1.8　ZCAN_Transmit_Data 结构体成员

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

```
}ZCAN_Transmit_Data;
```

成员

frame

报文数据信息，详见 1.3.5。

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，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。

1.3.8 ZCAN_TransmitFD_Data

结构体详情见程序清单 1.8，包含发送的 CANFD 报文信息，在函数 ZCAN_TransmitFD 中使用。

程序清单 1.9　ZCAN_TransmitFD_Data 结构体成员

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

成员

frame

报文数据信息，详见 1.3.6。

transmit_type

发送方式，同 1.3.7。

1.3.9 ZCAN_Receive_Data

结构体详情见程序清单 1.9，包含接收的 CAN 报文信息，在函数 ZCAN_Receive 中使用。

程序清单 1.10　ZCAN_Receive_Data 结构体成员

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

```
}ZCAN_Receive_Data;
```

成员

frame

报文数据信息，详见 1.3.5。

timestamp

时间戳，单位微秒，基于设备启动时间。（如果为云设备，则基于 1970 年 1 月 1 日 0 时 0 分 0 秒）

1.3.10 ZCAN_ReceiveFD_Data

结构体详情见程序清单 1.10，包含接收的 CANFD 报文信息，在函数 ZCAN_ReceiveFD 中使用。

程序清单 1.11 ZCAN_ReceiveFD_Data 结构体成员

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

成员

frame

报文数据信息，详见 1.3.6。

timestamp

时间戳，单位微秒。

1.3.11 ZCAN_AUTO_TRANSMIT_OBJ

结构体详情见程序清单 1.11，包含定时发送 CAN 参数信息。

程序清单 1.12 ZCAN_AUTO_TRANSMIT_OBJ 结构体成员

```
typedef struct tagZCAN_AUTO_TRANSMIT_OBJ{
    USHORT enable;
    USHORT index;
    UINT   interval;//定时发送时间。单位毫秒
    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，编号相同则使用最新的一条信息。

interval

发送周期，单位毫秒。

obj

发送的报文，详见 1.3.7。

1.3.12 ZCANFD_AUTO_TRANSMIT_OBJ

结构体详情见程序清单 1.12，包含定时发送 CANFD 参数信息。

程序清单 1.13 ZCANFD_AUTO_TRANSMIT_OBJ 结构体成员

```
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，编号相同则使用最新的一条信息。

interval

发送周期，单位毫秒。

obj

发送的报文，详见 1.3.8。

1.3.13 ZCLOUD_DEVINFO

结构体详情见程序清单 1.13，包含云设备的属性信息，在 ZCLOUD_GetUserData 中被填充。

程序清单 1.14 ZCLOUD_DEVINFO 结构体成员

```
typedef struct tagZCLOUD_DEVINFO
{
    int devIndex;
    char type[64];
    char id[64];
    char owner[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。

owner

设备的拥有者

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。

1.3.14 ZCLOUD_USER_DATA

结构体详情见程序清单 1.14，包含用户信息，包含用户基本信息以及用户拥有的设备信息，通过 ZCLOUD_GetUserData 获取。

程序清单 1.15 ZCLOUD_USER_DATA 结构体成员

```
typedef struct tagZCLOUD_USER_DATA
{
    char username[64];
    char mobile[64];
    ZCLOUD_DEVINFO devices[ZCLOUD_MAX_DEVICES];
    size_t devCnt;
}ZCLOUD_USER_DATA;
```

成员

username

用户名字符串。

mobile

用户手机号。

devices

用户拥有的设备组，详见 1.3.13。

devCnt

设备个数。

1.3.15 ZCLOUD_GPS_FRAME

结构体详情见程序清单 1.15，包含设备 GPS 数据，通过 ZCLOUD_ReceiveGPS 获取。

程序清单 1.16 ZCLOUD_GPS_FRAME 结构体成员

```
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。

1.3.16 USBCANFDTxTimeStamp

结构体详情见程序清单 1.16，用于获取发送帧的时间戳。

程序清单 1.17 USBCANFDTxTimeStamp 结构体成员

```
typedef struct tagUSBCANFDTxTimeStamp
```

```
{
    UINT* pTxTimeStampBuffer; //allocated by user, size:nBufferTimeStampCount * 4,unit:100us
    UINT    nBufferTimeStampCount; //buffer size
}USBCANFDTxTimeStamp;
```

成员

pTxTimeStampBuffer

用户申请的内存，用于存放返回的时间戳，大小为 nBufferTimeStampCount*4。

nBufferTimeStampCount

用户申请内存的大小，可以存放时间戳的个数。

1.3.17 IProperty

结构体详情见程序清单 1.17，用于获取/设置设备参数信息，示例代码参考程序清单 1.18。

程序清单 1.18 IProperty 结构体成员

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

成员

SetValue

设置设备属性值，详见 1.5。

GetValue

获取属性值。

GetPropertys

用于返回设备包含的所有属性。

程序清单 1.19 IProperty 示例代码

```
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;
}
```

1.3.18 ZCAN_LIN_MSG

结构体详情见程序清单 1.19，该结构体定义了 LIN 消息的结构，在设置从站响应信息和接收 LIN 数据接口中使用此结构表示单帧 LIN 消息。

程序清单 1.20　ZCAN_LIN_MSG 结构体成员

```
typedef struct _VCI_LIN_MSG{
    BYTE       ID;
    BYTE       DataLen;
    USHORT     Flag;
    UINT       TimeStamp;
    BYTE       Data[8];
}ZCAN_LIN_MSG, *PZCAN_LIN_MSG;
```

成员

ID

消息 ID，LIN ID 中的 ID 部分，不包括校验。校验信息设备自动计算。

DataLen

消息长度，表示后续 Data 的有效长度。

Flag

消息 Flag，暂未使用。

TimeStamp

时间戳，表示消息接收时间。

Data

消息的负载数据，具体长度由 DataLen 成员给出。

1.3.19　ZCAN_LIN_INIT_CONFIG

结构体详情见程序清单 1.20，该结构体表示配置 LIN 的信息，在函数 VCI_InitLIN 函数中调用。用于设置设备 LIN 的工作模式、波特率，是否使用增强校验等信息。

程序清单 1.21　ZCAN_LIN_INIT_CONFIG 结构体成员

```
typedef struct _VCI_LIN_INIT_CONFIG
{
    BYTE       linMode;
    BYTE       linFlag;
    USHORT     reserved;
    UINT      linBaud;
}ZCAN_LIN_INIT_CONFIG, *PZCAN_LIN_INIT_CONFIG;
```

成员

LinMode

LIN 工作模式，主站为 0，从站为 1

linFlag

LIN 支持的特性，多个特性使用位或方式进行指定。

Reserved

保留位

linBaud

LIN 作为主站时使用的波特率。

标志定义

```
#define LIN_MODE_MASTER              0
#define LIN_MODE_SLAVE               1
#define LIN_FLAG_CHK_ENHANCE      0x01
#define LIN_FLAG_VAR_DLC          0x02
```

1.4 接口库函数说明

1.4.1 ZCAN_OpenDevice

该函数用于打开设备。一个设备只能被打开一次。

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

参数

device_type

设备类型，详见头文件 zlgcan.h 中的宏定义。

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

仅作保留。

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，请保存该句柄值，往后的操作需要使用。

1.4.2 ZCAN_CloseDevice

该函数用于关闭设备，关闭设备和打开设备一一对应。

```
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。

1.4.3 ZCAN_GetDeviceInf

该函数用于获取设备信息。

```
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，详见 1.3.1。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.4.4 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。

1.4.5 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

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

pInitConfig

初始化结构，详见 1.3.2。

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，请保存该句柄值，往后的操作需要使用。

1.4.6 ZCAN_StartCAN

该函数用于启动 CAN 通道。

```
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。

1.4.7 ZCAN_ResetCAN

该函数用于复位 CAN 通道，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。

1.4.8 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。

1.4.9 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，详见 1.3.3。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.4.10 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，详见 1.3.4。

ZCAN_CHANNEL_STATUS。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.4.11 ZCAN_Transmit

该函数用于发送 CAN 报文。

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

参数

channel_handle

通道句柄值。

pTransmit

结构体 ZCAN_Transmit_Data 数组的首指针。

len

报文数目

返回值

返回实际发送成功的报文数目。

1.4.12 ZCAN_TransmitFD

该函数用于发送 CANFD 报文。

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

参数

channel_handle

通道句柄值。

pTransmit

结构体 ZCAN_TransmitFD_Data 数组的首指针。

len

报文数目

返回值

返回实际发送成功的报文数目。

1.4.13 ZCAN_GetReceiveNum

获取缓冲区中 CAN 或 CANFD 报文数目。

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

参数

channel_handle

通道句柄值。

type

获取 CAN 或 CANFD 报文，0=CAN，1=CANFD。

返回值

返回报文数目。

1.4.14 ZCAN_Receive

该函数用于接收CAN报文，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

结构体 ZCAN_Receive_Data 数组的首指针。

len

数组长度（本次接收的最大报文数目，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，单位毫秒。若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的报文数目。

1.4.15 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

结构体 ZCAN_ReceiveFD_Data 数组的首指针。

len

数组长度（本次接收的最大报文数目，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，单位毫秒。若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的报文数目。

1.4.16 GetIProperty

该函数返回属性配置接口。

```
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，详见 1.3.17，空则表示操作失败。

1.4.17 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。

1.4.18 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。

1.4.19 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。

1.4.20 ZCLOUD_IsConnected

该函数用于判断是否已经连接到云服务器。

```
bool ZCLOUD_IsConnected();
```

返回值

true：已连接，false：未连接。

1.4.21 ZCLOUD_DisconnectServer

该函数用于断开云服务器连接。

```
UINT ZCLOUD_DisconnectServer()
```

返回值

0：成功，1：失败。

1.4.22 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();
```

返回值

用户数据结构指针。

1.4.23 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

结构体 ZCLOUD_GPS_FRAME 数组的首指针。

len

数组长度（本次接收的最大报文数目，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，单位毫秒，若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的报文数目。

1.4.24 ZCAN_InitLIN

该函数用于对 LIN 进行初始化，指定设备工作模式，采用经典校验方式还是增强校验等参数，如果是主站模式，需要指定 LIN 工作的波特率。

```
CHANNEL_HANDLE FUNC_CALL ZCAN_InitLIN(DEVICE_HANDLE device_handle, UINT can_index, PZCAN_LIN_INIT_CONFIG pLINInitConfig);
```

参数

device_handle

设备句柄值。

can_index

通道索引号，通道 0 的索引号为 0，通道 1 的索引号为 1，以此类推。

pLINInitConfig

初始化结构，详见 1.3.19。

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，请保存该句柄值，往后的操作需要使用。

1.4.25 ZCAN_StartLIN

该函数用于启动 LIN 通道。

```
UINT FUNC_CALL ZCAN_StartLIN(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。

1.4.26 ZCAN_ResetLIN

该函数用于复位对应的 LIN 通道，即停止此通道的数据发送和接收。复位之后如果需要继续接收或者发送数据，需要重新调用 VCI_StartLIN 来启动 LIN 通道。

```
UINT FUNC_CALL ZCAN_ResetLIN(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。

1.4.27 ZCAN_TransmitLIN

该函数用来控制 LIN 发送 LIN 消息，只有 LIN 处于主站模式下才可以使用此函数进行数据发送。

```
ULONG FUNC_CALL ZCAN_TransmitLIN(CHANNEL_HANDLE channel_handle, PZCAN_LIN_MSG pSend,
ULONG Len);
```

参数

channel_handle

通道句柄值。

pSend

结构体 ZCAN_LIN_MSG 数组的首指针。

Len

报文数目

返回值

返回实际发送成功的报文数目。

1.4.28 ZCAN_GetLINReceiveNum

该函数用于获取指定通道已经接收到的 LIN 消息数量。

```
ULONG FUNC_CALL ZCAN_GetLINReceiveNum(CHANNEL_HANDLE channel_handle);
```

参数

channel_handle

通道句柄值。

返回值

返回报文数目。

1.4.29 ZCAN_ReceiveLIN

该函数用来接收 LIN 消息，不论 LIN 处于主站还是从站模式，都可以使用该函数获取总线上的数据信息。

```
ULONG FUNC_CALL ZCAN_ReceiveLIN(CHANNEL_HANDLE channel_handle, PZCAN_LIN_MSG pReceive, ULONG Len,int WaitTime);
```

参数

channel_handle

通道句柄值。

pReceive

结构体 ZCAN_LIN_MSG 数组的首指针。

Len

数组长度（本次接收的最大报文数目，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，单位毫秒。若为-1 则表示无超时，一直等待，默认值为-1。

返回值

返回实际接收的报文数目。

1.4.30 ZCAN_SetLINSlaveMsg

该函数用来设置 LIN 作为从站时候的响应信息，设置响应信息后，从站收到对应 ID 的请求时候会将预定义的数据发送出去作为响应。

```
UINT FUNC_CALL ZCAN_SetLINSlaveMsg(CHANNEL_HANDLE channel_handle, PZCAN_LIN_MSG pSend, UINT nMsgCount);
```

参数

channel_handle

通道句柄值。

pSend

结构体 ZCAN_LIN_MSG 数组的首指针。

nMsgCount

数组长度。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.4.31 ZCAN_ClearLINSlaveMsg

该函数用来清除 LIN 作为从站时候的响应信息，设置响应信息后，从站收到对应 ID 的请求时候会将预定义的数据发送出去作为响应，如果需要控制从站不再响应对应的 ID，需要调用此函数清除对特定 ID 的响应信息，清除后，从站不会在对此 ID 进行响应。

```
UINT FUNC_CALL ZCAN_ClearLINSlaveMsg(CHANNEL_HANDLE channel_handle, BYTE* pLINID, UINT nIDCount);
```

参数

channel_handle

通道句柄值。

pLINID

LIN ID 数组的首指针。

nIDCount

数组长度。

返回值

STATUS_OK 表示操作成功，STATUS_ERR 表示操作失败。

1.5 属性表

本节列出了 CAN 接口卡的属性配置项，即 IProperty 的 SetValue 或 GetValue 的 path、value 配置项。

1.5.1 USBCANFD系列

本系列的属性表如表 1.3 所示，适用的设备：USBCANFD-100U、USBCANFD-200U、USBCANFD-MINI。

表 1.1 USBCANFD 系列属性表

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 注：在 ZCAN_InitCAN 之前设置
仲裁域波特率	n/ canfd_abit_baud_rate	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置
数据域波特率	n/ canfd_dbit_baud_rate	5000000:5Mbps 4000000:4Mbps 2000000:2Mbps 1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置
终端电阻	n/initenal_resistance	“0”=禁能 “1”=使能 注：在 ZCAN_InitCAN 之后设置
定时发送 CAN	n/auto_send	ZCAN_AUTO_TRANSMIT_OBJ 把该结构指针转换为 char* 注：在 ZCAN_StartCAN 之后设置
定时发送 CANFD	n/auto_send_canfd	ZCANFD_AUTO_TRANSMIT_OBJ 把该结构指针转换为 char* 注：在 ZCAN_StartCAN 之后设置
清空定时发送	n/clear_auto_send	“0” 注：在 ZCAN_StartCAN 之后设置
使定时发送生效	n/apply_auto_send	“0” 注：在 ZCAN_StartCAN 之后设置
设置序列号	n/set_cn	最多 128 字符
获取序列号	n/get_cn/1	最多 128 字符，后面的 1 必须，可以是任意数字
升级	n/update	文件路径

续上表

属性	path	value
时钟	n/clock	“60000000”=60M
清除滤波	n/filter_clear	“0” 注：在 ZCAN_InitCAN 之后设置
滤波模式	n/filter_mode	“0”=标准帧 “1”=扩展帧 注：在 ZCAN_InitCAN 之后设置
滤波起始帧	n/filter_start	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波结束帧	n/filter_end	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波生效	n/filter_ack	“0” 注：在 ZCAN_InitCAN 之后设置
发送重试超时时间	n/tx_timeout	“1000”，单位毫秒，最大值为 4000
发送时间戳开启关闭功能	n/set_tx_timestamp	“0”=关闭 “1”=开启
获取发送时间戳	n/get_tx_timestamp/m m 代表要获取的时间戳个数	首先要开启发送时间戳功能，参考 n/set_tx_timestamp

1.5.2 PCIECANFD系列

本系列的属性表如表 1.4 所示，适用的设备：PCIE-CANFD-100U、PCIE-CANFD-200U、PCIE-CANFD-400U。

表 1.2　PCIECANFD 系列属性表

参数	path	value
仲裁域波特率	n/ canfd_abit_baud_rate n 代表通道号，如 0 代表通道 0，1 代表通道 1，下同	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 注：在 ZCAN_InitCAN 之前设置
数据域波特率	n/ canfd_dbit_baud_rate	8000000:8Mbps 4000000:4Mbps 2000000:2Mbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置
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

续上表

属性	path	value
定时发送 CAN	n/auto_send	ZCAN_AUTO_TRANSMIT_OBJ 把该结构指针转换为 char* 注：在 ZCAN_StartCAN 之后设置
定时发送 CANFD	n/auto_send_canfd	ZCANFD_AUTO_TRANSMIT_OBJ 把该结构指针转换为 char* 注：在 ZCAN_StartCAN 之后设置
清空定时发送	n/clear_auto_send	“0” 注：在 ZCAN_StartCAN 之后设置

1.5.3 USBCAN-xE-U PCI-50x0-U系列

本系列的属性表如表 1.5 所示，适用的设备：PCI-5010-U、PCI-5020-U、USBCAN-E-U、USBCAN-2E-U、CANalyst-II+。

表 1.3　USBCAN-xE-U PCI-50x0-U 系列属性表

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	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置
清除滤波	n/filter_clear	“0” 注：USBCAN-2E-U 的清除滤波在执行滤波生效时执行，如果没有执行滤波生效，则只是清除驱动的缓存数据，不清除实际设备的数据。 注：在 ZCAN_InitCAN 之后设置
滤波模式	n/filter_mode	“0”=标准帧 “1”=扩展帧 注：在 ZCAN_InitCAN 之后设置
滤波起始帧	n/filter_start	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波结束帧	n/filter_end	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波生效	n/filter_ack	“0” 注：USBCAN-2E-U 的滤波生效是所有通

		道的滤波生效，也就是设置了两个通道的滤波，只需要执行一次滤波生效即可。 注：在 ZCAN_InitCAN 之后设置

1.5.4 USBCAN-4E-U

USBCAN-4E-U 的属性表如表 1.6 所示。

表 1.4　USBCAN-4E-U 属性表

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	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置
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” 注：在 ZCAN_InitCAN 之后设置
滤波模式	n/filter_mode	“0”=标准帧 “1”=扩展帧 注：在 ZCAN_InitCAN 之后设置
滤波起始帧	n/filter_start	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波结束帧	n/filter_end	“0x00000000”，16 进制字符 注：在 ZCAN_InitCAN 之后设置
滤波生效	n/filter_ack	“0” 注：在 ZCAN_InitCAN 之后设置

1.5.5 USBCAN-8E-U

USBCAN-8E-U 的属性表如表 1.7 所示。

表 1.5　USBCAN-8E-U 属性表

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	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置
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

1.5.6 CANDTU-x00UR

本系列的属性表如表 1.8 所示，适用的设备：CANDTU-100UR、CANDTU-200UR。

表 1.6　CANDTU-x00UR 属性表

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	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置

自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置
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

1.5.7 NET-TCP系列

本系列的属性表如表 1.9 所示，适用的设备：CANDTU-NET、CANDTU-NET-400、CANET-TCP、CANWIFI-TCP、CANFDNET-TCP。

表 1.7　NET-TCP 系列属性表

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

1.5.8 NET-UDP系列

本系列的属性表如表 1.10 所示，适用的设备：CANET-UDP、CANWIFI-UDP、CANFDNET-UDP。

表 1.8　NET-UDP 系列属性表

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

1.5.9 其他接口卡

本系列的属性表如表 1.11 所示，适用的设备：USBCAN-I、USBCAN-II、PCI9810、PCI9820、PCI5110、PCIe-9110I、PCI9820I、PCIE-9221、PCIe-9120I、PCI5121。

表 1.9　其他接口卡属性表

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	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps 注：在 ZCAN_InitCAN 之前设置
自定义波特率	n/baud_rate_custom	请使用 ZCANPro 目录下的 baudcal 计算 注：在 ZCAN_InitCAN 之前设置

1.6 错误码定义

表 1.10 错误码定义

名称	值	描述
CAN 错误码		
ZCAN_ERROR_CAN_OVERFLOW	0x0001	CAN 控制器内部 FIFO 溢出
ZCAN_ERROR_CAN_ERRALARM	0x0002	CAN 控制器错误报警
ZCAN_ERROR_CAN_PASSIVE	0x0004	CAN 控制器消极错误
ZCAN_ERROR_CAN_LOSE	0x0008	CAN 控制器仲裁丢失
ZCAN_ERROR_CAN_BUSERR	0x0010	CAN 控制器总线错误
ZCAN_ERROR_CAN_BUSOFF	0x0020	CAN 控制器总线关闭
ZCAN_ERROR_CAN_BUFFER_OVERFLOW	0x0040	CAN 缓存溢出
通用错误码		
ZCAN_ERROR_DEVICEOPENED	0x0100	设备已经打开
ZCAN_ERROR_DEVICEOPEN	0x0200	打开设备错误
ZCAN_ERROR_DEVICENOTOPEN	0x0400	设备没有打开
ZCAN_ERROR_BUFFEROVERFLOW	0x0800	缓冲区溢出
ZCAN_ERROR_DEVICENOTEXIST	0x1000	此设备不存在
ZCAN_ERROR_LOADKERNELDLL	0x2000	装载动态库失败
ZCAN_ERROR_CMDFAILED	0x4000	执行命令失败错误码
ZCAN_ERROR_BUFFERCREATE	0x8000	内存不足