API library（Secondary Development Library）

User Manual

Version：V1.3

Update Date：2022.08.08

Revision History

Version	Date	Description
V1.0	2022.01.12	First Edition
V1.1	2022.03.23	Newly added library function (chapter 3.16~3.20)
V1.2	2022.04.03	Improve filtering Configuration; Add APIs(chapter 3.21~3.25)
V1.3	2022.08.08	Improve Functionality, Compatible with CANtest/CANPro; Add Chapter 6

Contents

1 Overview

If users only use USBCANFD devices for CAN/CANFD bus debugging,you can directly use the provided CANFD Tool software to test data transmission and reception.

If users plan to write software programs for their own products. Please carefully read the following instructions and refer to the demo we provided.

Development Library Files:ControlCANFD.lib, ControlCANFD.dll

VC Platform Function Declaration File:ControlCANFD.h,config.h

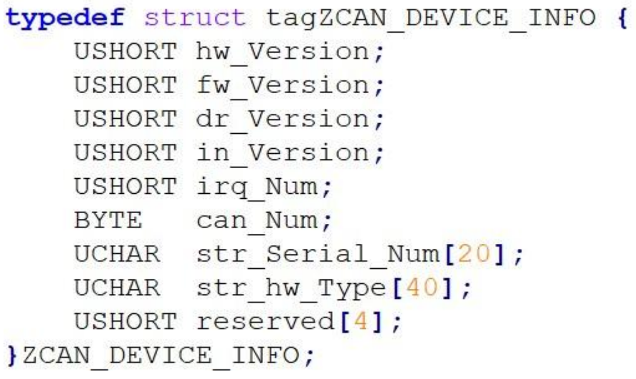
Note1: ControlCANFD.lib, ControlCANFD.dll Relying on the VC2008 runtime, which is typically included in most systems but not in very few lean systems, it needs to be installed.

Note2: The secondary development interface functions and data structures supported by this device are compatible with ZLG's interface and data structures.

2 Data Structure Definition

2.1 ZCAN_DEVICE_INFO

This structure contains some basic information about the device, which can be filled in the function ZCAN_GetDeviceInf.



```
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;
```

Member

hw_Version

Hardware version number, in hexadecimal. For example, 0x0100 represents V1.00.

fw_Version

Firmware version number, hexadecimal.

dr_Version

Driver version number, hexadecimal.

in_Version

Interface library version number, hexadecimal.

irq_Num

The interrupt number used by the board.

can_Num

Indicates how many channels there are.

str_Serial_Num

The serial number of this board, such as "USBCANFD0002" (note: including the string terminator '\0').

str_hw_Type

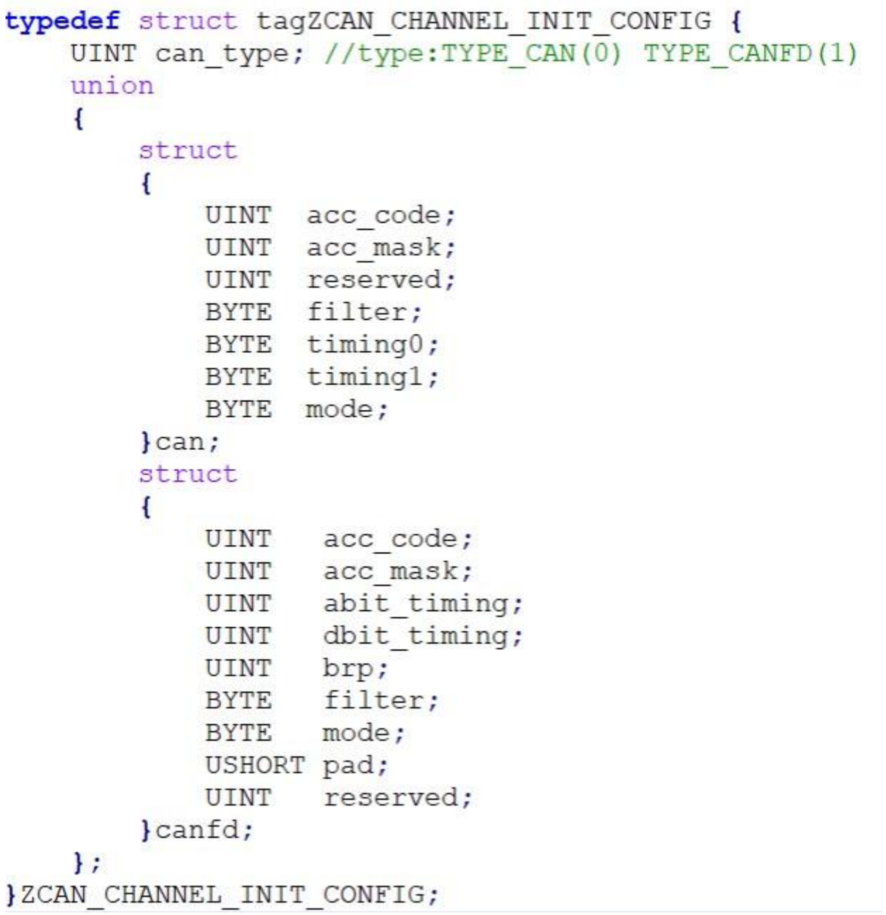
hardware type.

reserved

Reserved only, not set.

2.2 ZCAN_CHANNEL_INIT_CONFIG

This structure defines the parameters for channel initialization configuration and initialize the structure before call ZCAN_InitCAN.



```
typedef struct tagZCAN_CHANNEL_INIT_CONFIG {
    UINT can_type; //type:TYPE_CAN(0) TYPE_CANFD(1)
    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;
```

Member

can_type

Device type,=0 represents CAN device,=1 represents CANFD device.

CAN Device

acc_code

The frame filtering acceptance code of SJA1000 matches the "relevant bits" filtered by the mask code. After all matches are successful, this message can be received, otherwise it will not be received. Recommended setting is 0.

acc_mask

The frame filtering mask code of SJA1000 filters the received CAN frame ID, with bits 0 being "relevant bits" and 8 bits 1 being "irrelevant bits". It is recommended to set it to 0xFFFFFFFF, that is, receive all.

reserved

Reserved only, not set.

filter

Filtering method,=1 represents single filtering,=0 represents double filtering.

timing0

Ignore, do not set.

timing1

Ignore, do not set.

mode

Working mode,=0 represents normal mode (equivalent to a normal node),=1 represents listening only mode (only receiving, not affecting the bus).

CANFD Device

acc_code

Acceptance code, same as CAN device.

acc_mask

Shield code, same as CAN device.

abit_timing

Ignore, do not set.

dbit_timing

Ignore, do not set.

brp

Baud prescaler, set to 0.

filter

Filtering method, same as CAN device.

mode

Mode, same as CAN device.

pad

Data alignment, not set.

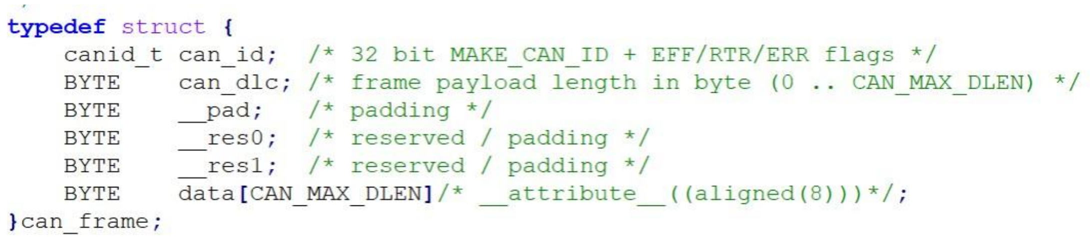
reserved

Reserved only, not set.

Note: The Baud (abit_timing and dbit_timing) of the device is set by GetIProperty. See Chapter 5.2 for details。

2.3 can_frame

This structure contains CAN message information.



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

Member

can_id

The frame ID, 32 bits, and the upper 3 bits belong to the flag bits. The meaning of the flag bits is as follows:

The 31st bit (highest bit) represents the extended frame flag,=0 represents the standard frame,=1 represents the extended frame, macro IS_ EFF can obtain this flag;

The 30th bit represents the remote frame flag,=0 represents the data frame,=1 represents the remote frame, macro IS_ RTR can obtain this flag;

The 29th digit represents the error frame standard,=0 represents the CAN frame, and=1 represents the error frame. Currently, it can only be set to 0;

The remaining bits represent the actual frame ID value, using the macro MAKE_CAN_ID Construct ID, using macro GET_ ID Get ID.

can_dlc

data length.

__pad

Align, ignore.

__res0

Reserved only, not set.

__res1

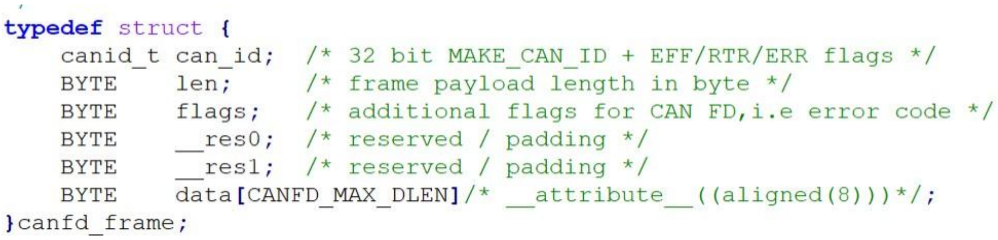
Reserved only, not set.

data

Message data, with an effective length of can_dlc.

2.4 canfd_frame

This structure contains CANFD message information.



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

Member

can_id

Frame ID, same as chapter 2.3.

len

data length.

flags

Additional flags, such as using CANFD baud rate switch, then set to macro CANFD_BRS.

__res0

Reserved only, not set.

__res1

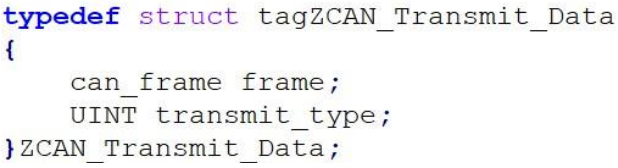
Reserved only, not set.

data

Message data, with an effective length of len.

2.5 ZCAN_Transmit_Data

Contains CAN send message information , using in function ZCAN_Transmit.



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

Member

frame

Message data information, see chapter 2.3 for details.

transmit_type

Sending type: 0=normal sending, 1=single sending, 2=spontaneous self receiving, and 3=single spontaneous self receiving.

The description of the sending type is as follows:

Normal sending: When the ID arbitration is lost or there is an error in sending, the CAN controller will automatically resend until the transmission is successful, or the transmission times out, or the bus is turned off.

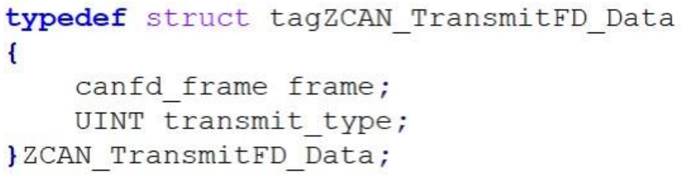
Single sending: In some applications, automatic retransmission is meaningless when partial data loss is allowed but transmission delay cannot occur. In these applications, data is generally sent at fixed time intervals, and automatic resending can cause subsequent data to be unable to be sent, resulting in transmission delays. If a single transmission is used, arbitration is lost or transmission error occurs, and the CAN controller will not resend the message.

Spontaneous self reception: generates a normal transmission with self reception characteristics, and after the transmission is completed, the sent message can be read from the receiving buffer.

Single spontaneous self reception: A single transmission with self reception characteristics is generated, and retransmission will not be executed in case of transmission error or arbitration loss. After the transmission is completed, the sent message can be read from the receive buffer.

2.6 ZCAN_TransmitFD_Data

Contains CANFD send message information , using in function ZCAN_TransmitFD.



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

Member

frame

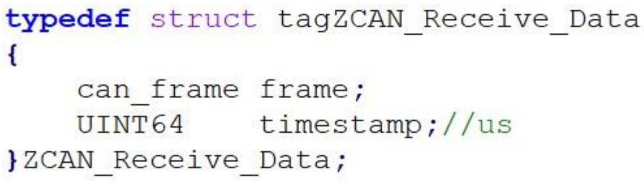
Message data information, see chapter 2.4 for details.

transmit_type

Sending type, same as chapter 2.5.

2.7 ZCAN_Receive_Data

Contains CAN recv message information , used in function ZCAN_Receive.



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

Member

frame

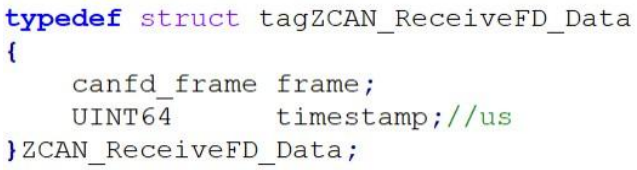
Message data information, see chapter 2.3 for details.

timestamp

Timestamp, in microseconds, based on device startup time.

2.8 ZCAN_ReceiveFD_Data

Contains CANFD recv message information , used in function ZCAN_ReceiveFD.



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

Member

frame

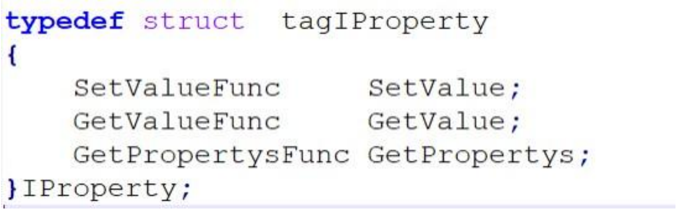
Message data information, see chapter 2.4 for details.

timestamp

Timestamp, in microseconds.

2.9 IProperty

The details of the structure are as follows, used to obtain/set device parameter information. For example code, refer to program listing 5.2.



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

Member

SetValue

Set the equipment attribute values, see Chapter 4 Property list for details.

GetValue

Get attribute values.

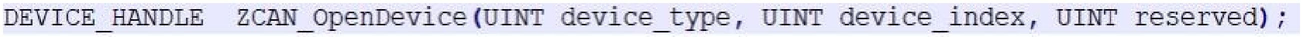
GetPropertys

Used to return all attributes contained in the device.

3 APIs Description

3.1 ZCAN_OpenDevice

This function is used to open the device. A device can only be opened once.



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

parameter

device_type

For the device type, see the macro definition in the Header file zlgcan.h.

device_index

Device index number, for example, when there is only one USBCANFD, the index number is 0. If another USBCANFD is inserted, the device index number inserted later will be 1, and so on.

reserved

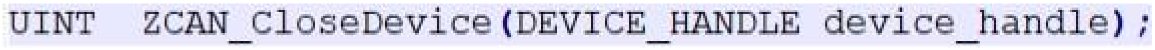
Reserved only.

return value

INVALID_DEVICE_HANDLE indicates that the operation failed, otherwise it indicates that the operation was successful. The device handle value is returned, please save the handle value. Future operations will need to use.

3.2 ZCAN_CloseDevice

This function is used to shut down the device, and the closing and opening devices correspond one by one.



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

parameter

device_handle

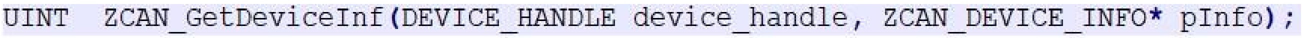
The handle value of the device that needs to be closed, i.e. the value returned by ZCAN_OpenDevice successfully execute.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.3 ZCAN_GetDeviceInf

This function is used to obtain device information.



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

parameter

device_handle

Device handle value.

pInfo

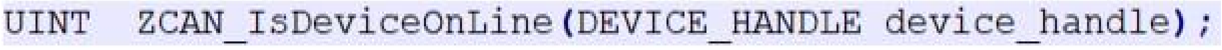
Device information structure, see chapter 2.1 for details.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.4 ZCAN_IsDeviceOnLine

This function is used to detect whether the device is online.



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

parameter

device_handle

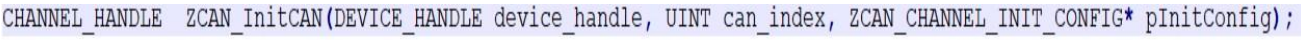
Device handle value.

return value

Device online=STATUS_ONLINE, not online= STATUS _OFFLINE.

3.5 ZCAN_InitCAN

This function is used to initialize CAN.



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

parameter

device_handle

Device handle value.

can_index

Channel index number, channel 0's index number is 0, channel 1's index number is 1, and so on.

pInitConfig

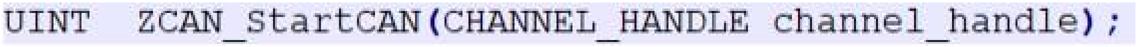
Initialization structure, see chapter 2.2 for details.

return value

INVALID_CHANNEL_HANDLE indicates that the operation failed, otherwise it indicates that the operation was successful. The channel handle value is returned. Please save the handle value for future operations.

3.6 ZCAN_StartCAN

This function is used to start the CAN channel.



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

parameter

channel_handle

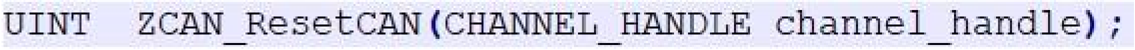
Channel handle value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.7 ZCAN_ResetCAN

This function is used to reset the CAN channel, which can be accessed through ZCAN_StartCAN to recovery.



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

parameter

channel_handle

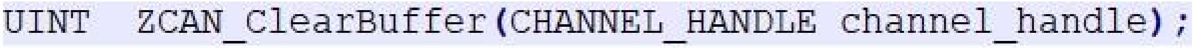
Channel handle value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.8 ZCAN_ClearBuffer

This function is used to clear the library receive buffer.



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

parameter

channel_handle

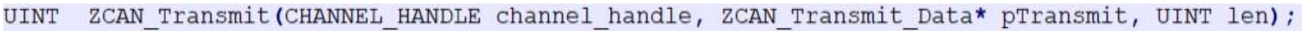
Channel handle value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.9 ZCAN_Transmit

This function is used to send CAN frame.



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

parameter

channel_handle

Channel handle value.

pTransmit

The first pointer of the Structure array ZCAN_Transmit_Data.

len

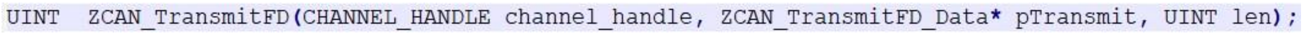
frame number.

return value

Returns the actual number of successfully sent frames.

3.10 ZCAN_TransmitFD

This function is used to send CANFD frame.



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

parameter

channel_handle

Channel handle value.

pTransmit

The first pointer of the Structure array ZCAN_TransmitFD_Data.

len

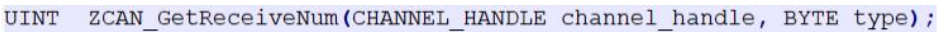
frame number.

return value

Returns the actual number of successfully sent frames.

3.11 ZCAN_GetReceiveNum

Obtain the number of CAN or CANFD messages in the buffer.



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

parameter

channel_handle

Channel handle value.

type

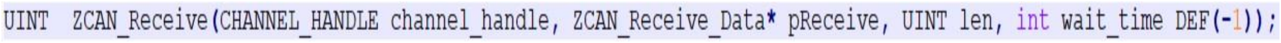
Get CAN or CANFD frame number，0=CAN，1=CANFD。

return value

Returns the frame number.

3.12 ZCAN_Receive

This function is used to receive CAN frames, it is recommended to use ZCAN_GetReceiveNum to ensures that the buffer has data before use this function.



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

parameter

channel_handle

Channel handle value.

pReceive

The first pointer of the Structure array ZCAN_Receive_Data.

len

Array length (maximum number of frames received this time, actual return value is less than or equal to this value).

wait_time

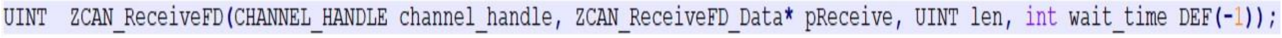
There is no data in the buffer. The waiting time for function blocking is in milliseconds. If it is -1, it indicates wait forever. The default value is -1.

return value

Returns the actual number of received frames.

3.13 ZCAN_ReceiveFD

This function is used to receive CANFD frames, it is recommended to use ZCAN_GetReceiveNum to ensures that the buffer has data before use this function.



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

parameter

channel_handle

Channel handle value.

pReceive

The first pointer of the Structure array ZCAN_ReceiveFD_Data.

len

Array length (maximum number of frames received this time, actual return value is less than or equal to this value).

wait_time

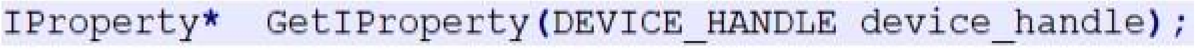
There is no data in the buffer. The waiting time for function blocking is in milliseconds. If it is -1, it indicates wait forever. The default value is -1.

return value

Returns the actual number of received frames.

3.14 GetIProperty

This function returns the property configuration interface.



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

parameter

device_handle

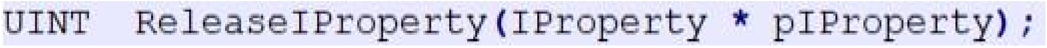
Device handle value.

return value

Returns the pointer to the property configuration interface, see chapter 2.9 for details. If it is empty, it indicates that the operation has failed.

3.15 ReleaseIProperty

Release the property interface and pair it with GetIProperty for use.



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

parameter

pIProperty

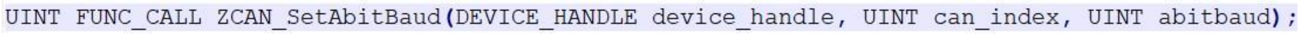
GetIProperty's return value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.16 ZCAN_SetAbitBaud

This function is used to set the baudrate of the CANFD arbitration domain. When using the attribute 'n/canfd_abit_baud_rate' to set baudrate fails, then this function can be called to set the baudrate. For example, when the development environment is VC, you can call this function interface to set the CANFD arbitration baudrate.



```
UINT FUNC_CALL ZCAN_SetAbitBaud(DEVICE_HANDLE device_handle, UINT can_index, UINT abitbaud);
```

parameter

device_handle

Device handle value.

can_index

Channel index number, channel 0's index number is 0, channel 1's index number is 1, and so on.

abitbaud

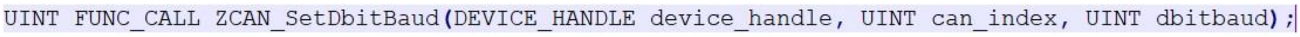
For the baudrate value of the arbitration domain, see the baudrate value of the arbitration domain in the Property list.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.17 ZCAN_SetDbitBaud

This function is used to set the baudrate of the CANFD data domain. When using the attribute 'n/canfd_dbit_baud_rate' to set baudrate fails, then this function can be called to set the baudrate. For example, when the development environment is VC, you can call this function interface to set the CANFD date baudrate.



```
UINT FUNC_CALL ZCAN_SetDbitBaud(DEVICE_HANDLE device_handle, UINT can_index, UINT dbitbaud);
```

parameter

device_handle

Device handle value.

can_index

Channel index number, channel 0's index number is 0, channel 1's index number is 1, and so on.

dbitbaud

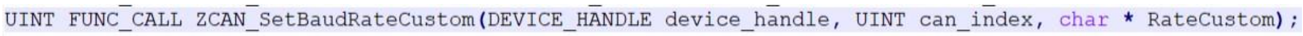
For the baudrate value of the data domain, see the baudrate value of the data domain in the Property list.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.18 ZCAN_SetBaudRateCustom

This function is used to set the CANFD custom baudrate. When using the attribute 'n/baud_rate_custom' to set the baudrate fails, then this function can be called to set the custom baudrate. For example, when the development environment is VC, you can call this function interface to set the CANFD custom baudrate.



```
UINT FUNC_CALL ZCAN_SetBaudRateCustom(DEVICE_HANDLE device_handle, UINT can_index, char * RateCustom);
```

parameter

device_handle

Device handle value.

can_index

Channel index number, channel 0's index number is 0, channel 1's index number is 1, and so on.

RateCustom

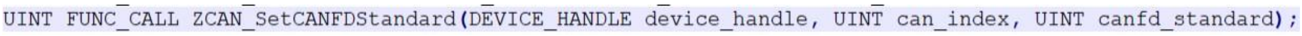
Custom baudrate string, see Property list Custom baudrate value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.19 ZCAN_SetCANFDStandard

This function is used to set the CANFD standard type. When using the attribute 'n/canfd_standard' to set the CANFD standard fails, then this function can be called to set it. If the development environment is VC, this function interface can be called to set the CANFD standard。



```
UINT FUNC_CALL ZCAN_SetCANFDStandard(DEVICE_HANDLE device_handle, UINT can_index, UINT canfd_standard);
```

parameter

device_handle

Device handle value.

can_index

Channel index number, channel 0's index number is 0, channel 1's index number is 1, and so on.

canfd_standard

CANFD standard type,0=CANFD ISO，1=CANFD BOSCH。

return value

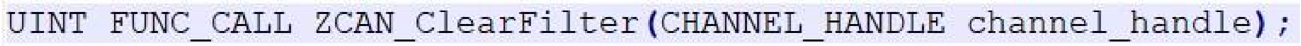
STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.20 ZCAN_SetResistanceEnable

This function do not use.

3.21 ZCAN_ClearFilter

This function is used to clear channel filtering settings. When using the attribute 'n/filter_clear' to clear filter fails, then this function can be called. e.g. when the development environment is VC, this function can be called to clear the filtering settings. This function is not called separately. Each configuration is carried out in the order of clearing filter settings, configuration mode, configuration start ID, configuration end ID, and filtering effectiveness; If you want to set multiple filters, you can set multiple filters between clearing the filter and filtering effectiveness.



```
UINT FUNC_CALL ZCAN_ClearFilter(CHANNEL_HANDLE channel_handle);
```

parameter

channel_handle

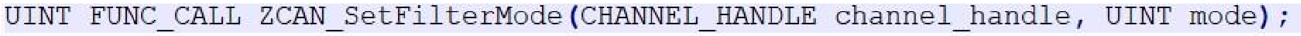
Channel handle value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.22 ZCAN_SetFilterMode

This function is used to configure the channel filtering mode. This function can be called when using the attribute 'n/filter_mode' to set the filter mode fails. e.g. this function can be called when the development environment is VC. This function is not called separately. Each configuration is carried out in the order of clearing filter settings, configuration mode, configuration start ID, configuration end ID, and filtering effectiveness; If you want to set multiple filters, you can set multiple filters between clearing the filter and the filter effectiveness.



```
UINT FUNC_CALL ZCAN_SetFilterMode(CHANNEL_HANDLE channel_handle, UINT mode);
```

parameter

channel_handle

Channel handle value.

mode

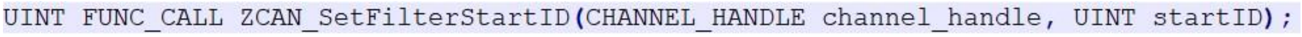
mode,0=Standard Frame, 1=Extended Frame.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.23 ZCAN_SetFilterStartID

This function is used to configure the channel filtering start ID. This function can be called when using the attribute 'n/filter_start' to set the start ID fails. e.g. this function interface setting can be called when the development environment is VC. This function is not called separately. Each configuration is carried out in the order of clearing filter settings, configuration mode, configuration start ID, configuration end ID, and filtering effectiveness; If you want to set multiple filters, you can set multiple filters between clearing the filter and the filter effectiveness.



```
UINT FUNC_CALL ZCAN_SetFilterStartID(CHANNEL_HANDLE channel_handle, UINT startID);
```

parameter

channel_handle

Channel handle value.

startID

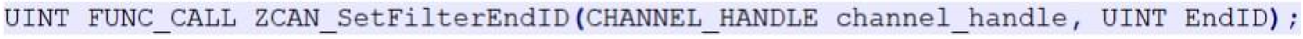
start ID value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.24 ZCAN_SetFilterEndID

This function is used to configure the channel filtering end ID. This function can be called when using the attribute 'n/filter_end' to set the end ID fails. e.g. this function interface setting can be called when the development environment is VC. This function is not called separately. Each configuration is carried out in the order of clearing filter settings, configuration mode, configuration start ID, configuration end ID, and filtering effectiveness; If you want to set multiple filters, you can set multiple filters between clearing the filter and the filter effectiveness.



```
UINT FUNC_CALL ZCAN_SetFilterEndID(CHANNEL_HANDLE channel_handle, UINT EndID);
```

parameter

channel_handle

Channel handle value.

EndID

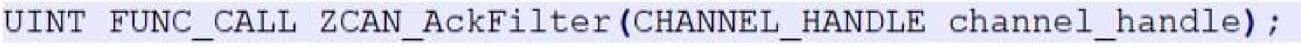
end ID value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

3.25 ZCAN_AckFilter

This function is used to validate channel filtering settings. This function can be called when the 'n/filter_ack' setting fails to take effect. e.g. this function interface setting can be called when the development environment is VC. This function is not called separately. Each configuration is carried out in the order of clearing filter settings, configuration mode, configuration start ID, configuration end ID, and filtering effectiveness; If you want to set multiple filters, you can set multiple filters between clearing the filter and the filter effectiveness.



```
UINT FUNC_CALL ZCAN_AckFilter(CHANNEL_HANDLE channel_handle);
```

parameter

channel_handle

Channel handle value.

return value

STATUS_OK indicates successful operation, STATUS_ERR indicates that the operation failed.

4 Attribute List

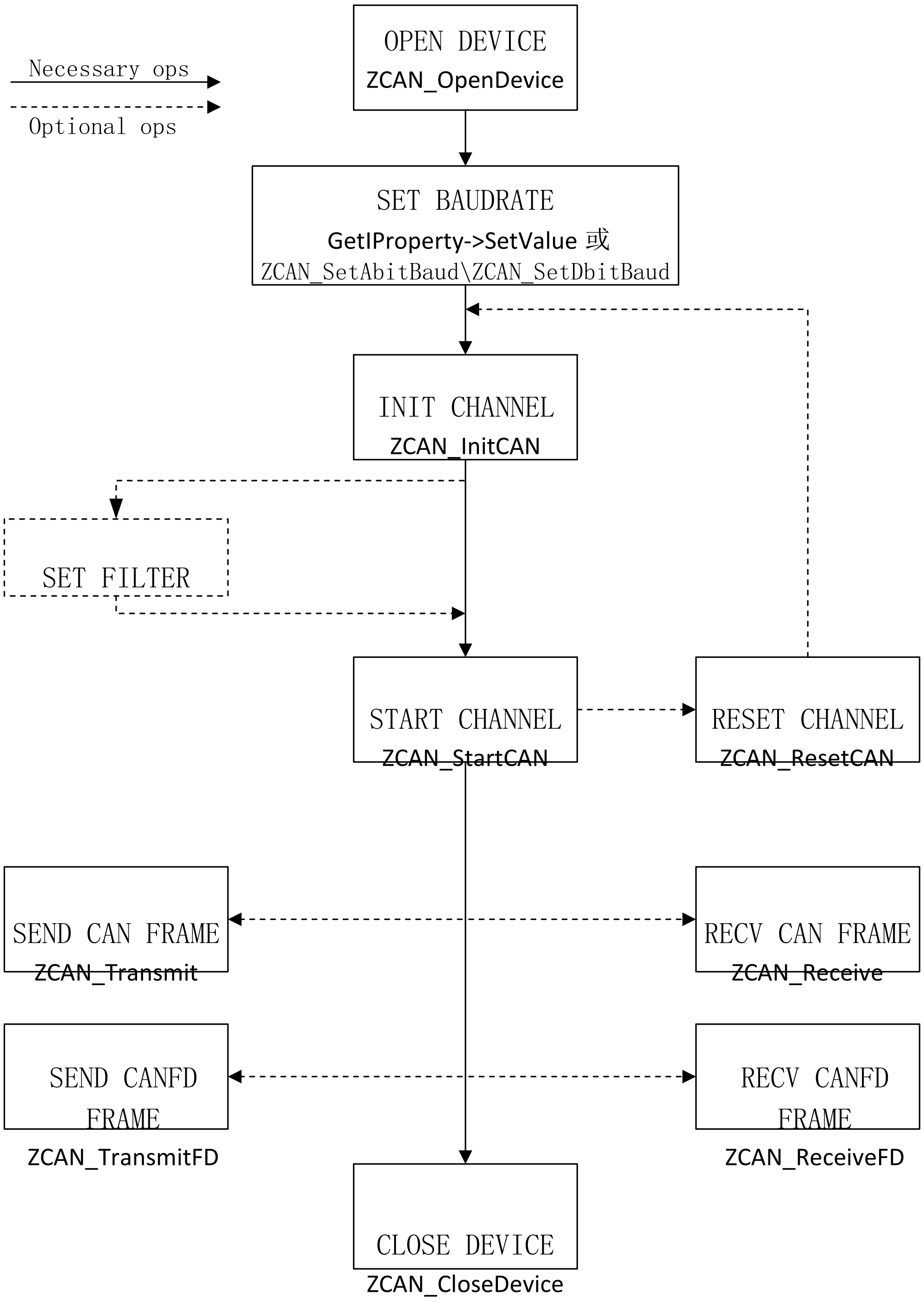
The Property list supported by this device is shown in the following table.

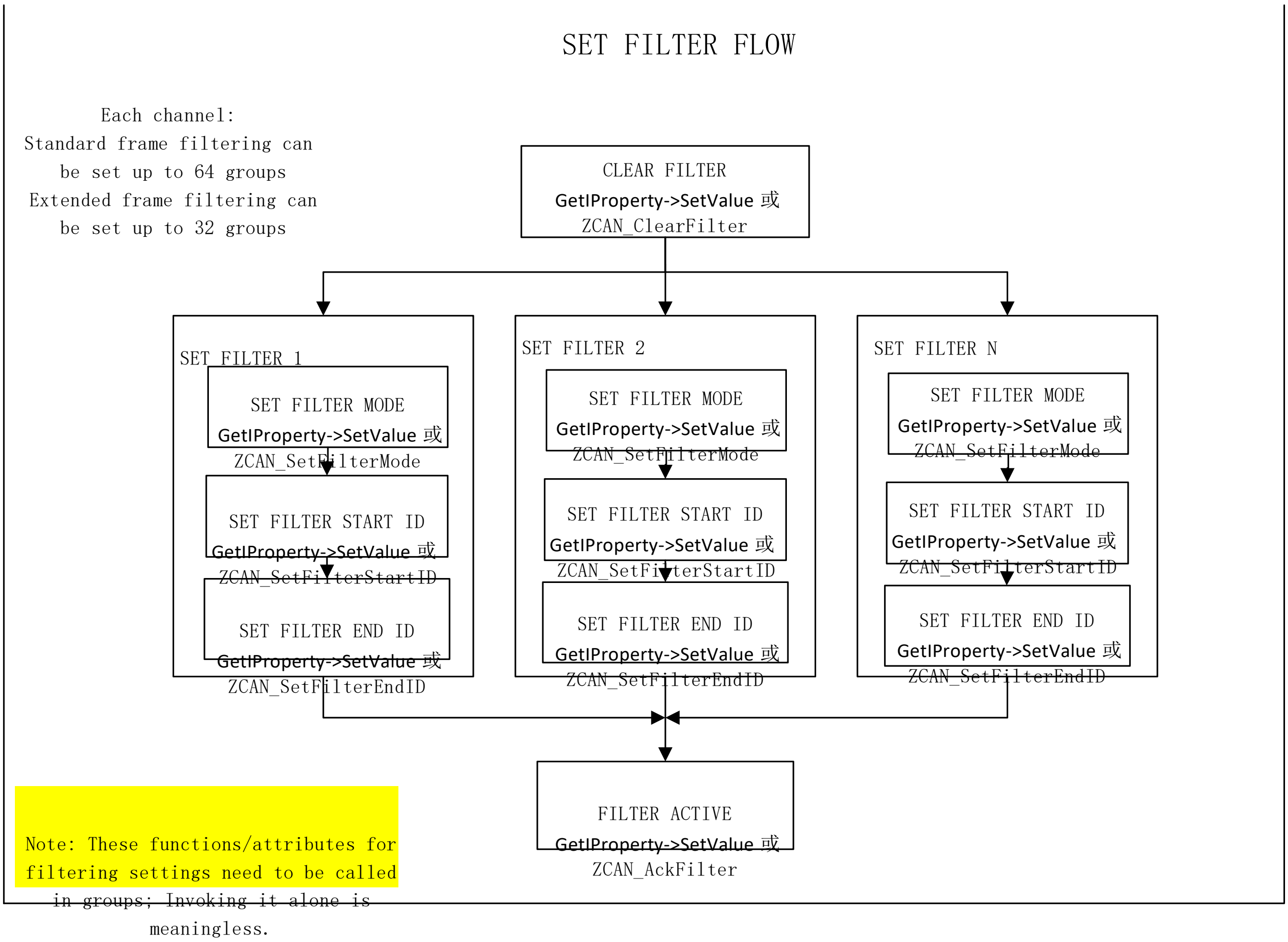
parameter	Path	value
Arbitration domain baudrate	n/ canfd_abit_baud_rate n Indicates the channel number, 0=channel 1,1=channel 2	1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps 50000:50kbps Note：set before call ZCAN_InitCAN
date domain baudrate	n/ canfd_dbit_baud_rate n Indicates the channel number, 0=channel 1,1=channel 2	5000000:5Mbps 4000000:4Mbps 2000000:2Mbps 1000000:1Mbps 800000:800kbps 500000:500kbps 250000:250kbps 125000:125kbps 100000:100kbps Note：set before call ZCAN_InitCAN
custom baudrate	n/baud_rate_custom n Indicates the channel number, 0=channel 1,1=channel 2	Note：set before call ZCAN_InitCAN
filter mode	n/filter_mode n Indicates the channel number, 0=channel 1,1=channel 2	“0”=standard frame “1”=extended frame Note：set after call ZCAN_InitCAN
Filter start frame	n/filter_start n Indicates the channel number, 0=channel 1,1=channel 2	“0x00000000”，hex char Note：set after call ZCAN_InitCAN
Filter end frame	n/filter_end n Indicates the channel number, 0=channel	“0x00000000”，hex char Note：set after call ZCAN_InitCAN

	1,1=channel 2	
Clear filter	n/filter_clear n Indicates the channel number, 0=channel 1,1=channel 2	“0” Note：set after call ZCAN_InitCAN
Filter effective	n/filter_ack n Indicates the channel number, 0=channel 1,1=channel 2	“0” Note：set after call ZCAN_InitCAN
CANFD standard type	n/canfd_standard n Indicates the channel number, 0=channel 1,1=channel 2	“0”=CANFD ISO “1”=CANFD BOSCH Note：set before call ZCAN_InitCAN

5. Flow of Using API

5.1 Flow



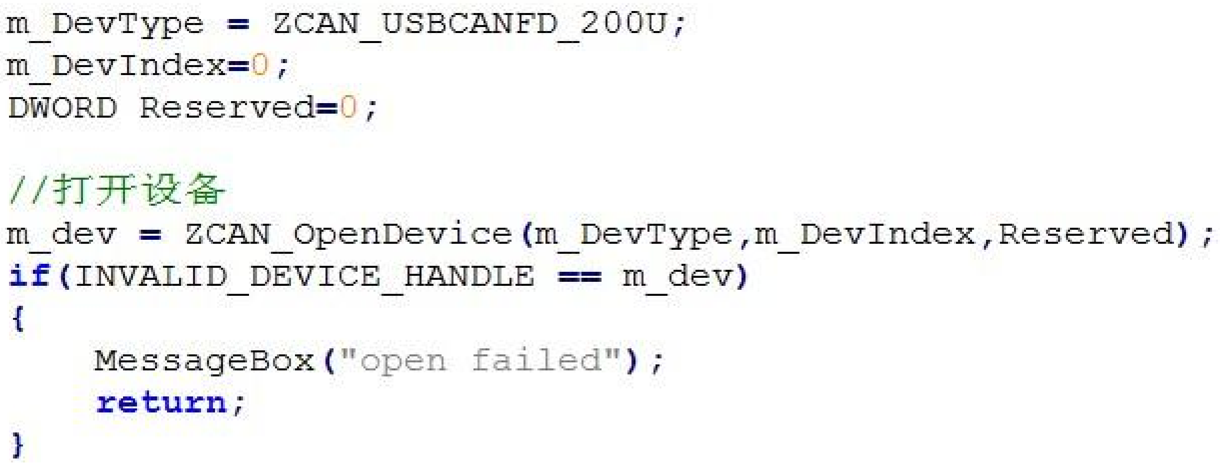


Each channel:
Standard frame filtering can be set up to 64 groups
Extended frame filtering can be set up to 32 groups

Note: These functions/attributes for filtering settings need to be called in groups; Invoking it alone is meaningless.

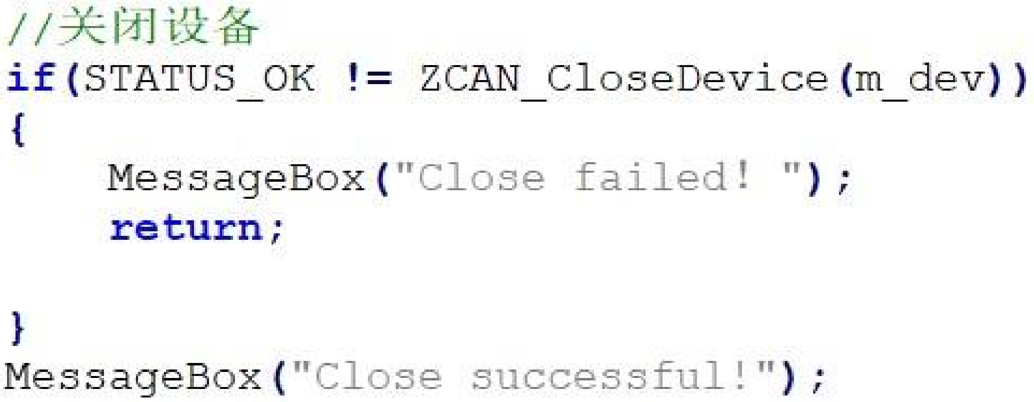
5.2 Sample Code

OPEN DEVICE



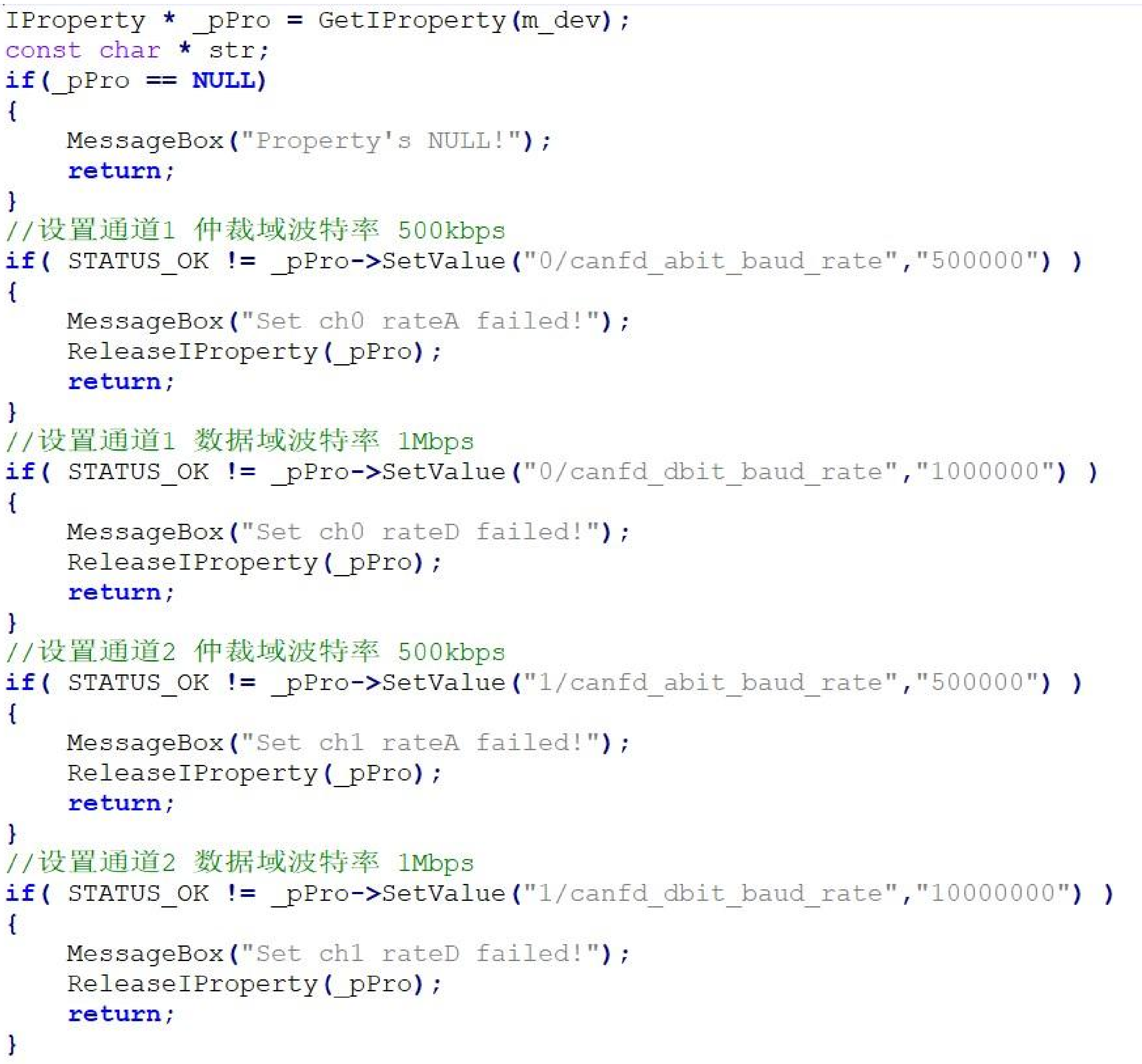
```
m_DevType = ZCAN_USBCANFD_200U;
m_DevIndex=0;
DWORD Reserved=0;

//打开设备
m_dev = ZCAN_OpenDevice(m_DevType,m_DevIndex,Reserved);
if(INVALID_DEVICE_HANDLE == m_dev)
{
    MessageBox("open failed");
    return;
}
```

CLOSE DEVICE



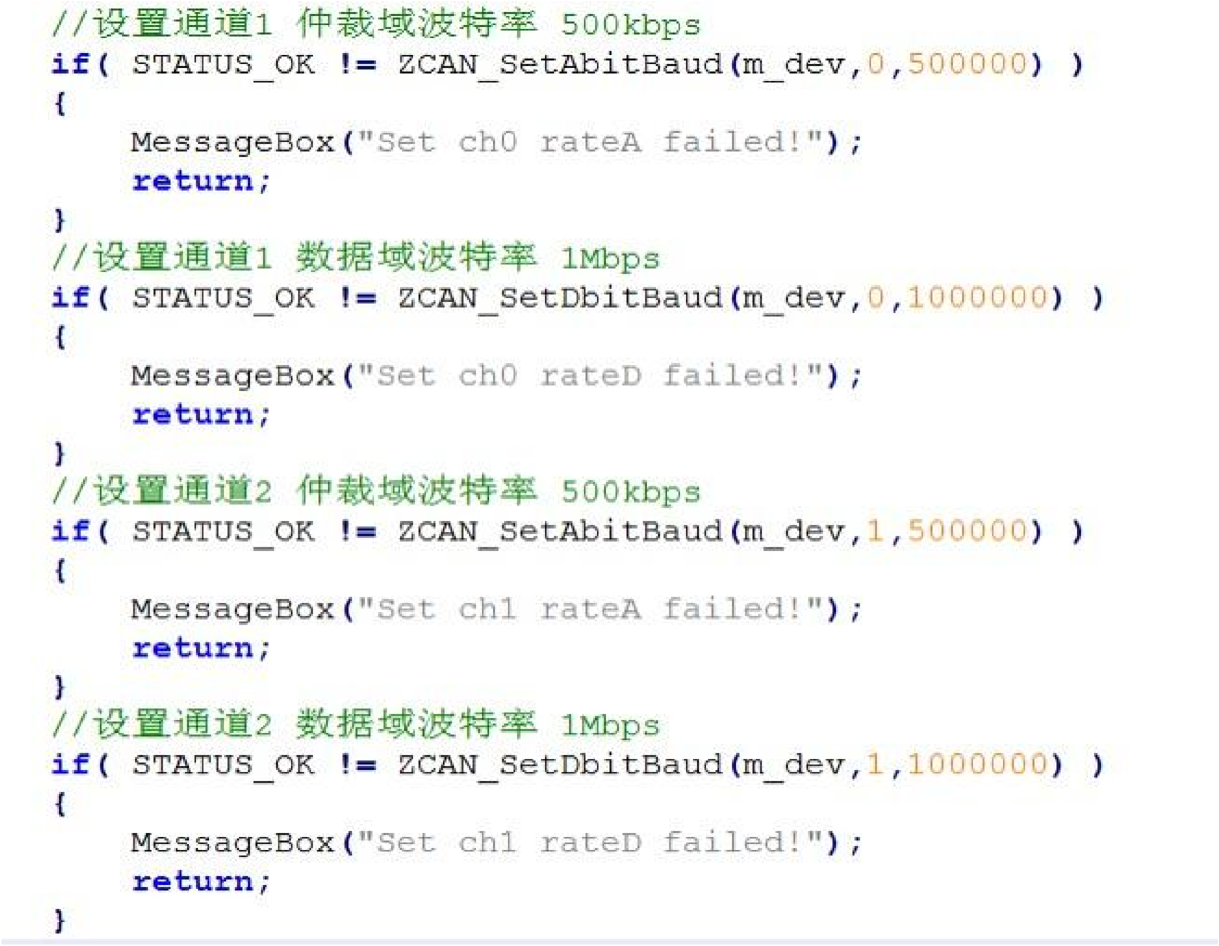
```
//关闭设备
if(STATUS_OK != ZCAN_CloseDevice(m_dev))
{
    MessageBox("Close failed! ");
    return;

}
MessageBox("Close successful!");
```

SET　BAUDRATE



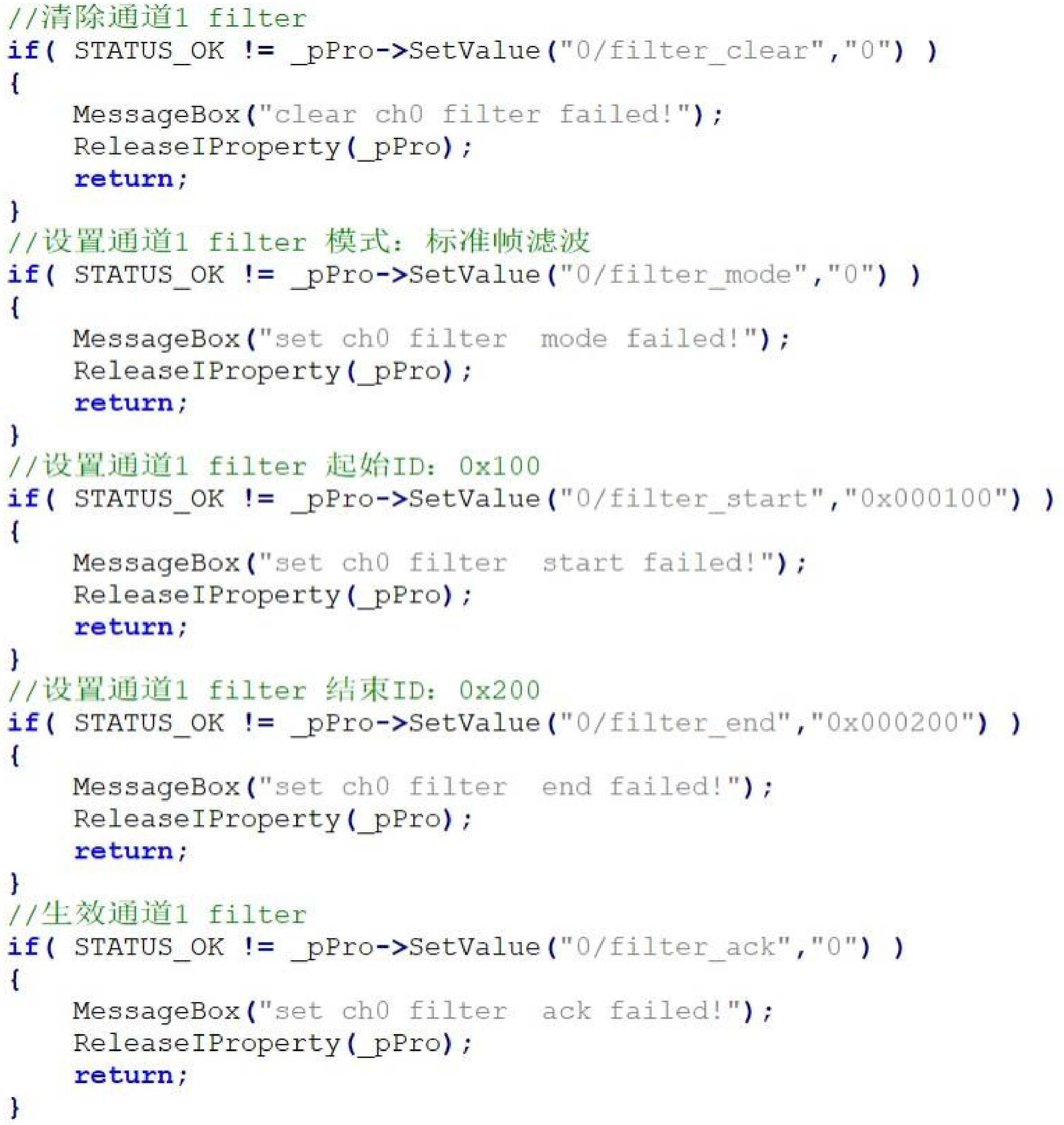
```
IProperty * _pPro = GetIProperty(m_dev);
const char * str;
if(_pPro == NULL)
{
    MessageBox("Property's NULL!");
    return;
}
//设置通道1 仲裁域波特率 500kbps
if( STATUS_OK != _pPro->SetValue("0/canfd_abit_baud_rate","500000") )
{
    MessageBox("Set ch0 rateA failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 数据域波特率 1Mbps
if( STATUS_OK != _pPro->SetValue("0/canfd_dbit_baud_rate","1000000") )
{
    MessageBox("Set ch0 rateD failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道2 仲裁域波特率 500kbps
if( STATUS_OK != _pPro->SetValue("1/canfd_abit_baud_rate","500000") )
{
    MessageBox("Set ch1 rateA failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道2 数据域波特率 1Mbps
if( STATUS_OK != _pPro->SetValue("1/canfd_dbit_baud_rate","10000000") )
{
    MessageBox("Set ch1 rateD failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

SET BAUDRATE 2



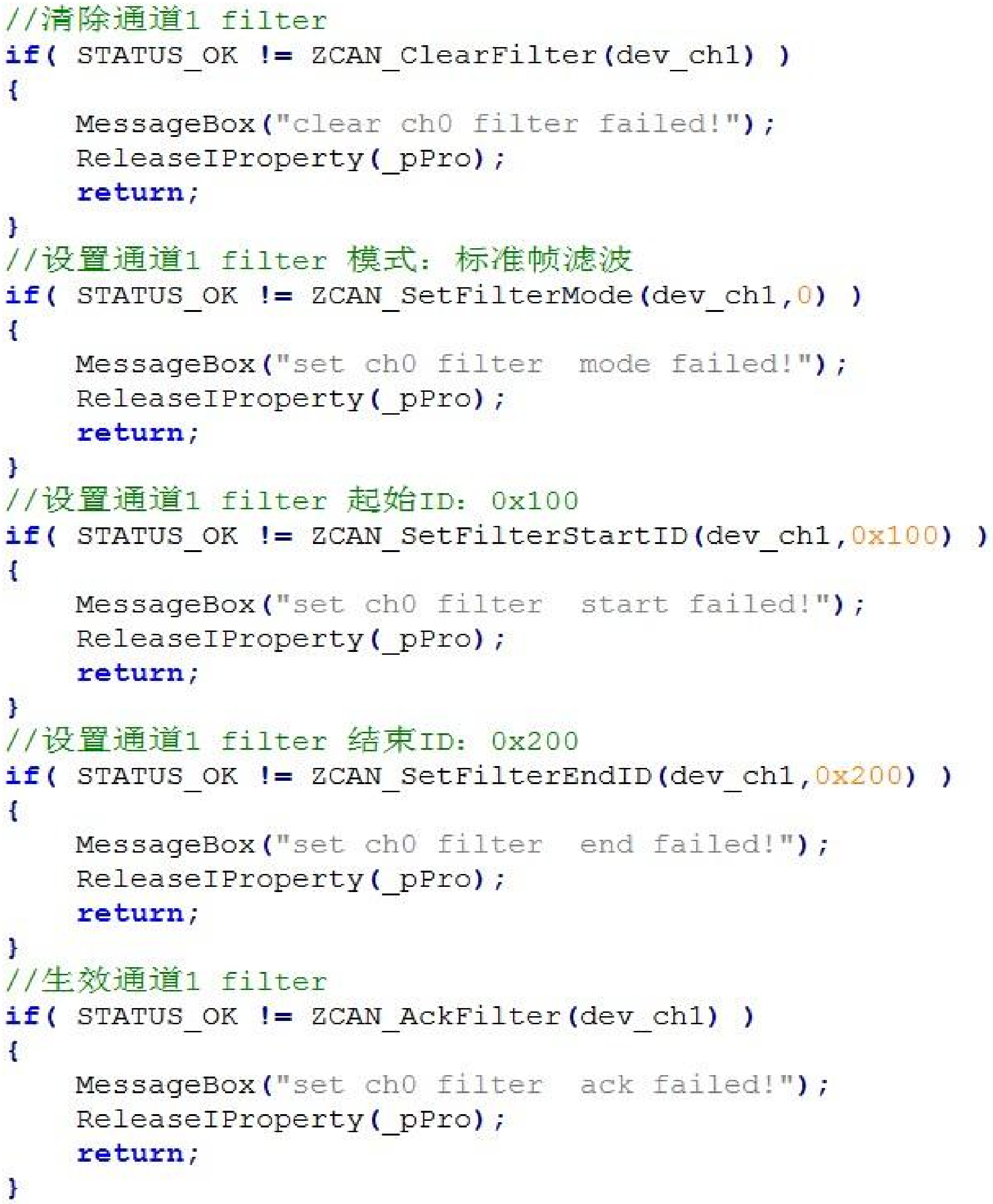
```
//设置通道1 仲裁域波特率 500kbps
if( STATUS_OK != ZCAN_SetAbitBaud(m_dev,0,500000) )
{
    MessageBox("Set ch0 rateA failed!");
    return;
}
//设置通道1 数据域波特率 1Mbps
if( STATUS_OK != ZCAN_SetDbitBaud(m_dev,0,1000000) )
{
    MessageBox("Set ch0 rateD failed!");
    return;
}
//设置通道2 仲裁域波特率 500kbps
if( STATUS_OK != ZCAN_SetAbitBaud(m_dev,1,500000) )
{
    MessageBox("Set ch1 rateA failed!");
    return;
}
//设置通道2 数据域波特率 1Mbps
if( STATUS_OK != ZCAN_SetDbitBaud(m_dev,1,1000000) )
{
    MessageBox("Set ch1 rateD failed!");
    return;
}
```

SET　CHANNEL　FILTER [after channel init, before start channel]



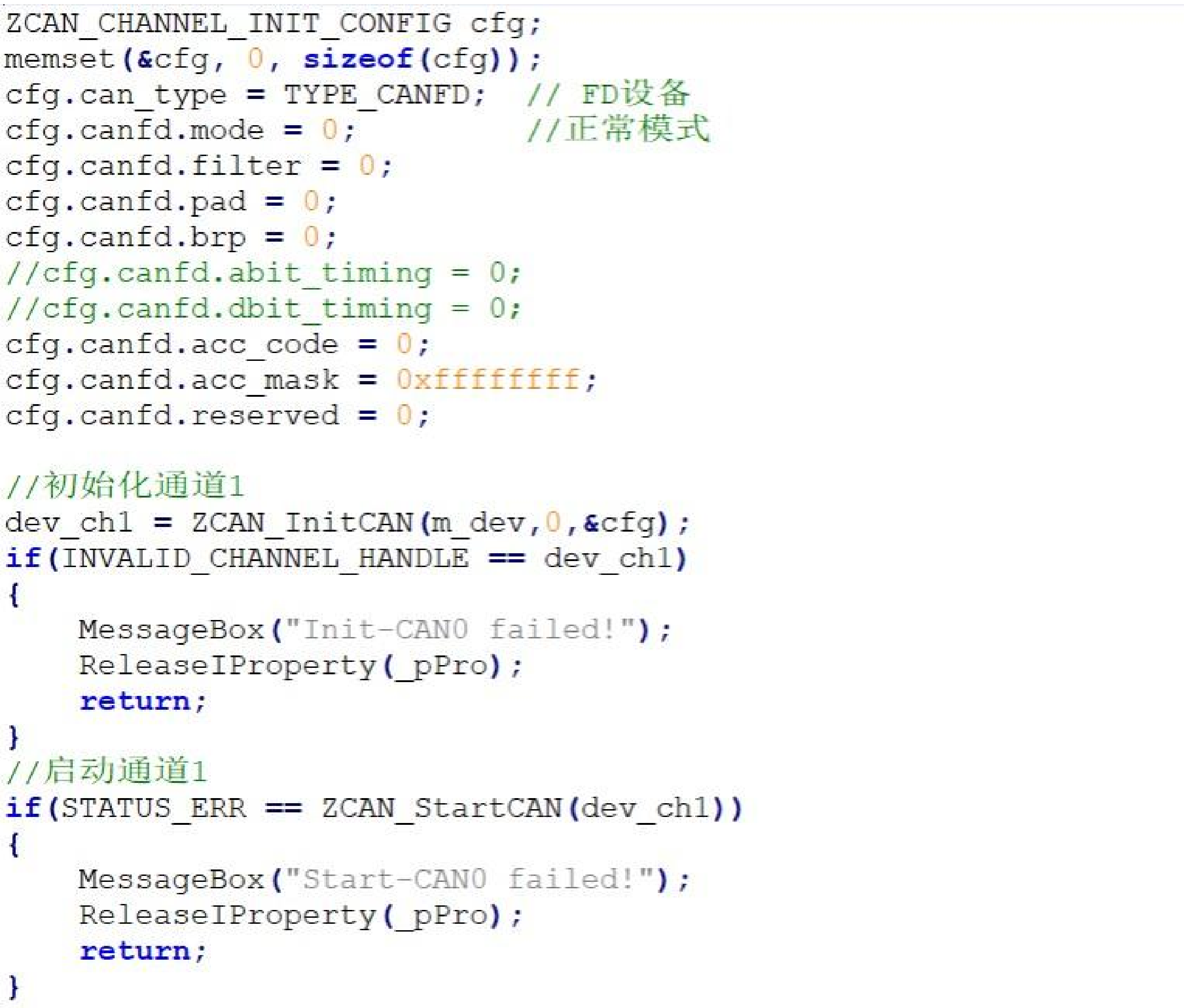
```
//清除通道1 filter
if( STATUS_OK != _pPro->SetValue("0/filter_clear","0") )
{
    MessageBox("clear ch0 filter failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 模式: 标准帧滤波
if( STATUS_OK != _pPro->SetValue("0/filter_mode","0") )
{
    MessageBox("set ch0 filter  mode failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 起始ID: 0x100
if( STATUS_OK != _pPro->SetValue("0/filter_start","0x000100") )
{
    MessageBox("set ch0 filter  start failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 结束ID: 0x200
if( STATUS_OK != _pPro->SetValue("0/filter_end","0x000200") )
{
    MessageBox("set ch0 filter  end failed!");
    ReleaseIProperty(_pPro);
    return;
}
//生效通道1 filter
if( STATUS_OK != _pPro->SetValue("0/filter_ack","0") )
{
    MessageBox("set ch0 filter  ack failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

SET CHANNEL FILTER 2 [after channel init, before start channel]



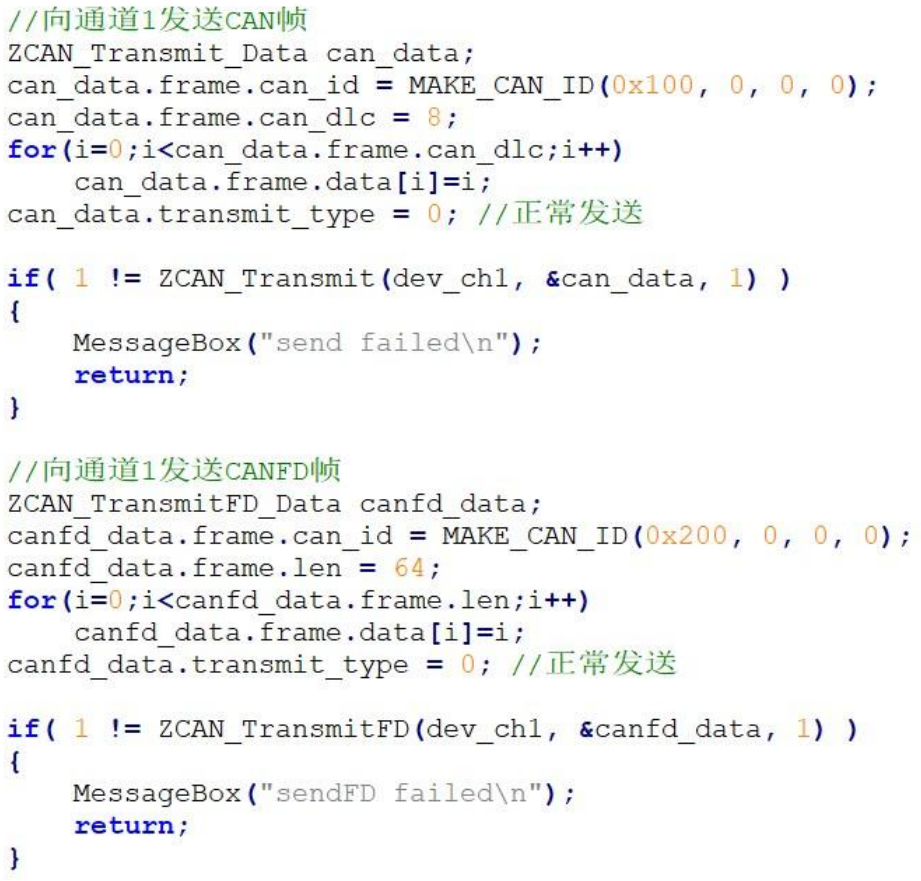
```
//清除通道1 filter
if( STATUS_OK != ZCAN_ClearFilter(dev_ch1) )
{
    MessageBox("clear ch0 filter failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 模式： 标准帧滤波
if( STATUS_OK != ZCAN_SetFilterMode(dev_ch1,0) )
{
    MessageBox("set ch0 filter  mode failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 起始ID: 0x100
if( STATUS_OK != ZCAN_SetFilterStartID(dev_ch1,0x100) )
{
    MessageBox("set ch0 filter  start failed!");
    ReleaseIProperty(_pPro);
    return;
}
//设置通道1 filter 结束ID: 0x200
if( STATUS_OK != ZCAN_SetFilterEndID(dev_ch1,0x200) )
{
    MessageBox("set ch0 filter  end failed!");
    ReleaseIProperty(_pPro);
    return;
}
//生效通道1 filter
if( STATUS_OK != ZCAN_AckFilter(dev_ch1) )
{
    MessageBox("set ch0 filter  ack failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

INIT AND START CHANNEL



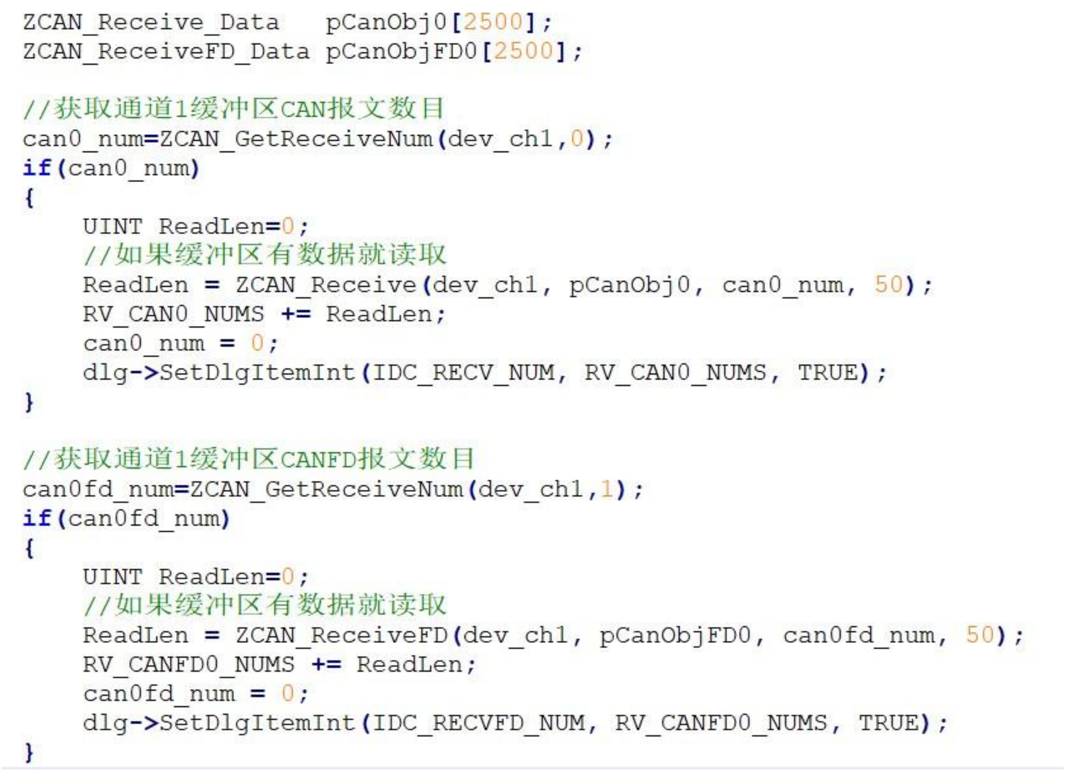
```
ZCAN_CHANNEL_INIT_CONFIG cfg;
memset(&cfg, 0, sizeof(cfg));
cfg.can_type = TYPE_CANFD;  // FD设备
cfg.canfd.mode = 0;         //正常模式
cfg.canfd.filter = 0;
cfg.canfd.pad = 0;
cfg.canfd.brp = 0;
//cfg.canfd.abit_timing = 0;
//cfg.canfd.dbit_timing = 0;
cfg.canfd.acc_code = 0;
cfg.canfd.acc_mask = 0xffffffff;
cfg.canfd.reserved = 0;

//初始化通道1
dev_ch1 = ZCAN_InitCAN(m_dev,0,&cfg);
if(INVALID_CHANNEL_HANDLE == dev_ch1)
{
    MessageBox("Init-CAN0 failed!");
    ReleaseIProperty(_pPro);
    return;
}
//启动通道1
if(STATUS_ERR == ZCAN_StartCAN(dev_ch1))
{
    MessageBox("Start-CAN0 failed!");
    ReleaseIProperty(_pPro);
    return;
}
```

SEND　FRAME



```
//向通道1发送CAN帧
ZCAN_Transmit_Data can_data;
can_data.frame.can_id = MAKE_CAN_ID(0x100, 0, 0, 0);
can_data.frame.can_dlc = 8;
for(i=0;i<can_data.frame.can_dlc;i++)
    can_data.frame.data[i]=i;
can_data.transmit_type = 0; //正常发送

if( 1 != ZCAN_Transmit(dev_ch1, &can_data, 1) )
{
    MessageBox("send failed\n");
    return;
}

//向通道1发送CANFD帧
ZCAN_TransmitFD_Data canfd_data;
canfd_data.frame.can_id = MAKE_CAN_ID(0x200, 0, 0, 0);
canfd_data.frame.len = 64;
for(i=0;i<canfd_data.frame.len;i++)
    canfd_data.frame.data[i]=i;
canfd_data.transmit_type = 0; //正常发送

if( 1 != ZCAN_TransmitFD(dev_ch1, &canfd_data, 1) )
{
    MessageBox("sendFD failed\n");
    return;
}
```

RECV　FRAME



```
ZCAN_Receive_Data   pCanObj0[2500];
ZCAN_ReceiveFD_Data pCanObjFD0[2500];

//获取通道1缓冲区CAN报文数目
can0_num=ZCAN_GetReceiveNum(dev_ch1,0);
if(can0_num)
{
    UINT ReadLen=0;
    //如果缓冲区有数据就读取
    ReadLen = ZCAN_Receive(dev_ch1, pCanObj0, can0_num, 50);
    RV_CAN0_NUMS += ReadLen;
    can0_num = 0;
    dlg->SetDlgItemInt(IDC_RECV_NUM, RV_CAN0_NUMS, TRUE);
}

//获取通道1缓冲区CANFD报文数目
can0fd_num=ZCAN_GetReceiveNum(dev_ch1,1);
if(can0fd_num)
{
    UINT ReadLen=0;
    //如果缓冲区有数据就读取
    ReadLen = ZCAN_ReceiveFD(dev_ch1, pCanObjFD0, can0fd_num, 50);
    RV_CANFD0_NUMS += ReadLen;
    can0fd_num = 0;
    dlg->SetDlgItemInt(IDC_RECVFD_NUM, RV_CANFD0_NUMS, TRUE);
}
```

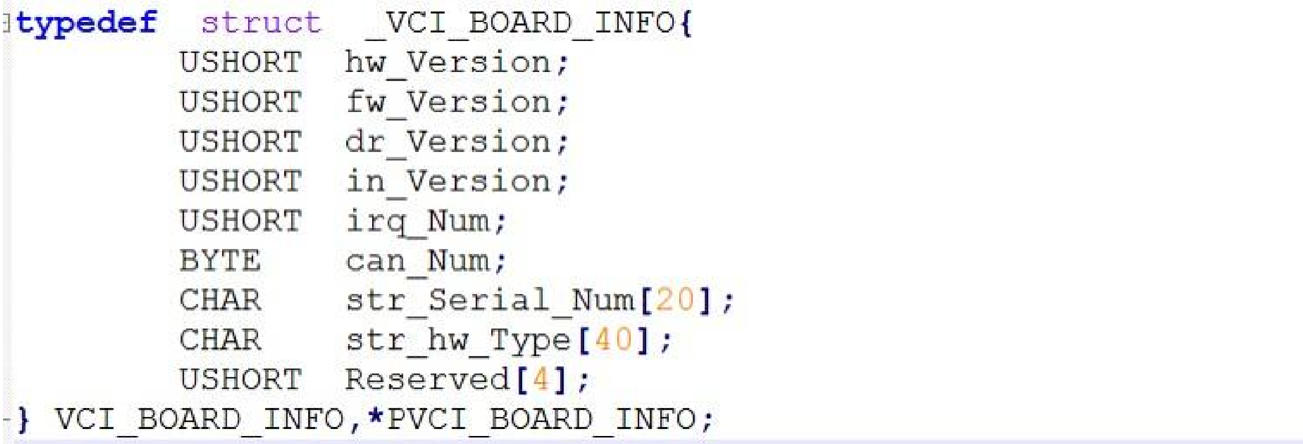
6. Compatible with ZLG ControlCAN.dll API Library Manual

If this device uses standard CAN, it is compatible with ZLG CANtest and CANPro protocol analysis software. This chapter provides an overview of its data structure and function. For detailed instructions on how to use CANtest software and CANPro software, please refer to the analyzer materials 'How to Compatible with the Use of Zhou Ligong CANTest Software' and 'How to Compatible with the Use of Zhou Ligong CANPro Protocol Analysis Platform V1.50.pdf'. To use the ControlCAN.dll, ControlCAN.lib, and ControlCAN.h files mentioned in the document, simply replace the relevant files provided by this driver library with the corresponding file names.

6.1 Data Structure Definition

6.1.1 VCI_BOARD_INFO

The structure VCI_BOARD_INFO contains the device information of the USB-CAN series interface card. The structure will be in filled in VCI_ReadBoardInfo function.



```
typedef  struct  _VCI_BOARD_INFO{
        USHORT  hw_Version;
        USHORT  fw_Version;
        USHORT  dr_Version;
        USHORT  in_Version;
        USHORT  irq_Num;
        BYTE    can_Num;
        CHAR    str_Serial_Num[20];
        CHAR    str_hw_Type[40];
        USHORT  Reserved[4];
} VCI_BOARD_INFO,*PVCI_BOARD_INFO;
```

member

hw_Version

Hardware version number, represented in hexadecimal. For example, 0x0100 represents V1.00.

fw_Version

The firmware version number, represented in hexadecimal. For example, 0x0100 represents V1.00.

dr_Version

Driver version number, represented in hexadecimal. For example, 0x0100 represents V1.00.

in_Version

The version number of the interface library, expressed in hexadecimal. For example, 0x0100 represents V1.00

irq_Num

Retention parameter.

can_Num

Indicates how many CAN channels there are.

str_Serial_Num

The serial number of this board.

str_hw_Type

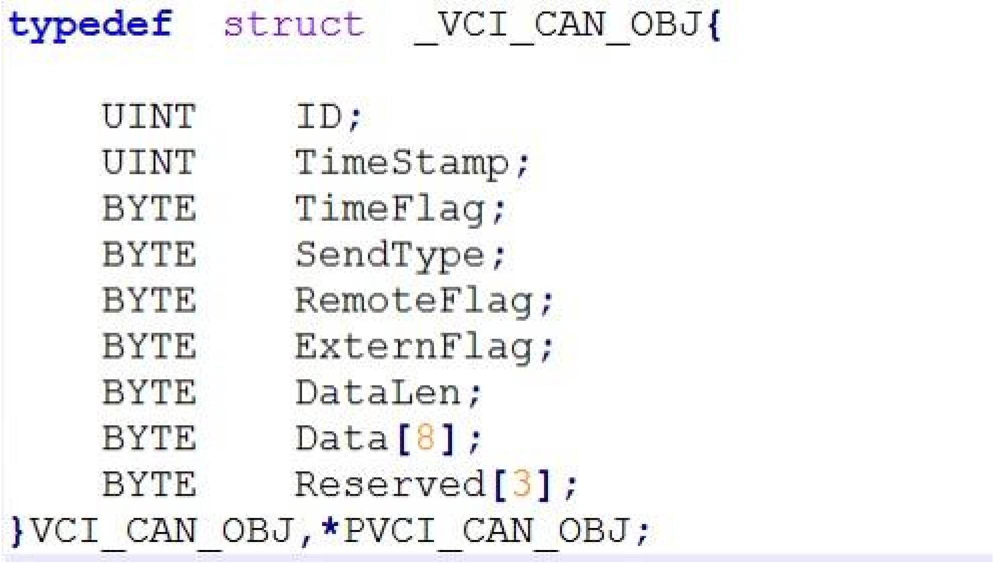
Hardware type, such as' USBCANFD0002 '(note: includes string terminator '\0').

Reserved

Reserved.

6.1.2 VCI_CAN_OBJ

The structure VCI_CAN_OBJ is CAN frame structure. One structure represents the data structure of one frame. In send function VCI_Transmit and Receive Function VCI_Receive, it is used to transmit CAN frames.



```
typedef  struct  _VCI_CAN_OBJ{

    UINT    ID;
    UINT    TimeStamp;
    BYTE    TimeFlag;
    BYTE    SendType;
    BYTE    RemoteFlag;
    BYTE    ExternFlag;
    BYTE    DataLen;
    BYTE    Data[8];
    BYTE    Reserved[3];
}VCI_CAN_OBJ,*PVCI_CAN_OBJ;
```

member

ID

Frame ID. 32-bit variable, right aligned.

TimeStamp

The time identifier of a frame received by the device. The time indicator starts counting from the device being powered on, with a timing unit of 0.1ms.

TimeFlag

It indicates a the timestamp used or not. When it is 1, TimeStamp is valid. TimeFlag and TimeStamp are only meaningful when the frame is a received frame.

SendType

Send type.

=0 indicates normal transmission (if the transmission fails, it will automatically resend for 4 seconds, and if it is not sent within 4 seconds, it will be cancelled);

= 1 indicates single transmission (only once, if the transmission fails, it will not be automatically resend, and the bus only generates one frame of data);

Other values invalid.

RemoteFlag

=0 indicates the data frame; =1 indicates the remote frame (data segment is empty)。

ExternFlag

=0 indicates standard frame(11 bits ID), =1 indicates extended frame(29 bits ID).

DataLen

The data length, DLC (<=8) refers to the number of bytes in the CAN frame Data. Constrained the valid bytes in Data[8].

Data[8]

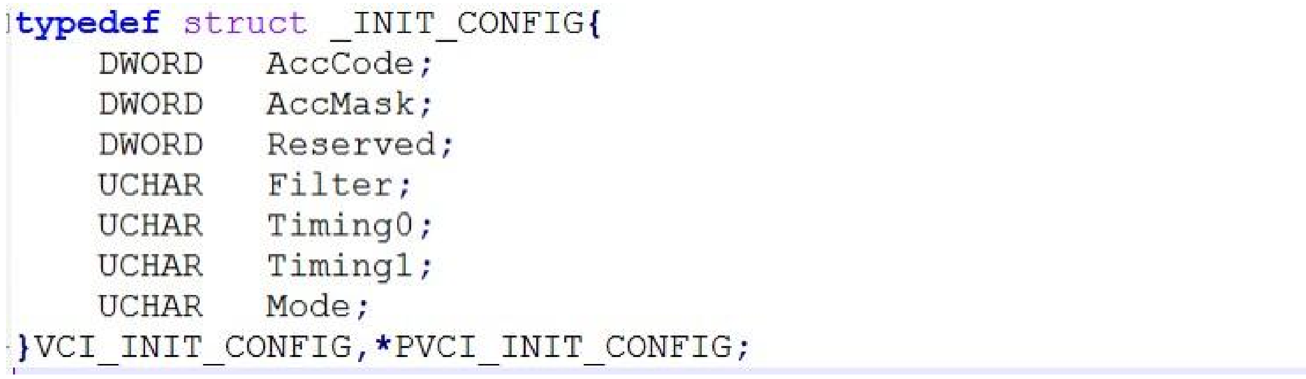
CAN frame data. Due to the CAN specification of a maximum of 8 bytes, a space of 8 bytes is reserved here, subject to DataLen constraints. If DataLen is defined as 3, that is, Data [0], Data [1], and Data [2] are valid.

Reserved

Reserved.

6.1.3 VCI_INIT_CONFIG

The structure VCI_INIT_CONFIG defines the configuration of CAN. The structure will be filled in function VCI_InitCan. Before call VCI_InitCan , init this structure.



```
typedef struct _INIT_CONFIG{
    DWORD   AccCode;
    DWORD   AccMask;
    DWORD   Reserved;
    UCHAR   Filter;
    UCHAR   Timing0;
    UCHAR   Timing1;
    UCHAR   Mode;
}VCI_INIT_CONFIG,*PVCI_INIT_CONFIG;
```

member

AccCode

Acceptance code. Frame filtering acceptance code for SJA1000. After filtering the masked code into "relevant bits" for matching, once all matches are successful, this frame can be received. Otherwise, it will not be accepted. Can be set to 0.

AccMask

Block code. Filter the received CAN frame ID using a frame filtering mask code for SJA1000. Use a corresponding bit of 0 for the 'relevant bit' and a corresponding bit of 1 for the 'irrelevant bit'. It is recommended to set the blocking code to 0xFFFFFF to receive all frames. The blocking code can also be set to 0.

Reserved

Reserved.

Filter

Not used for this device.

Timing0

This device sets the baudrate to use VCI_SetReference interface.

Timing1

This device sets the baudrate to use VCI_SetReference interface.

Mode

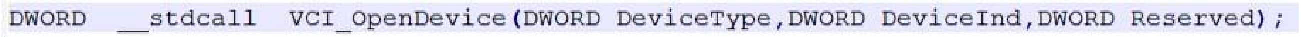
mode

=0 represents normal mode (equivalent to a normal node),=1 represents listening mode (only receiving without affecting the bus), and=2 represents spontaneous self collection mode (loopback mode).

6.2 API illustrate

6.2.1 VCI_OpenDevice

This function is used to open the device. Note that one device can only be opened once.



```
DWORD  __stdcall  VCI_OpenDevice(DWORD DeviceType,DWORD DeviceInd,DWORD Reserved);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

The device index number is assigned based on the order of insertion. For instance, if there is only one USBCANFD adapter, its index number is 0. When another USBCANFD adapter is inserted, the device index number assigned to it will be 1, and so on.

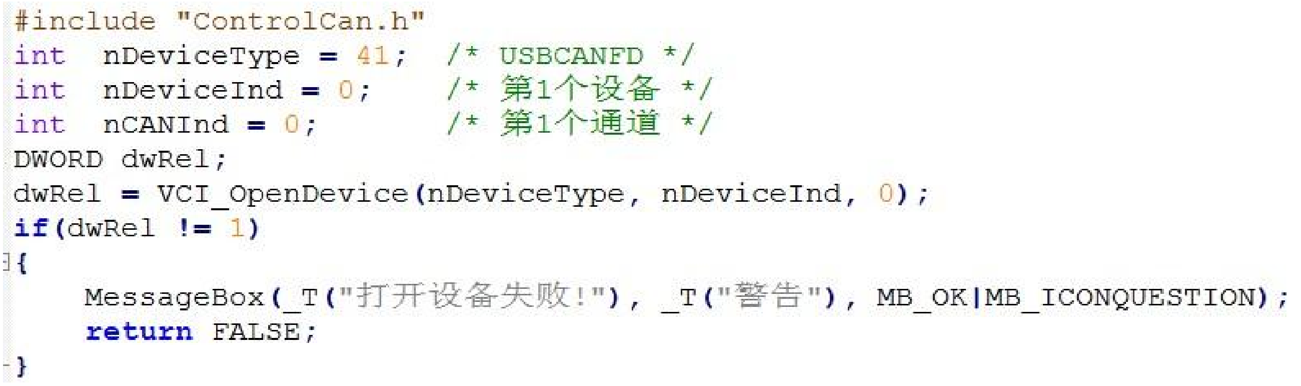
Reserved

Reserved parameter, usually.

return value

Return 1 indicates success and 0 failed.

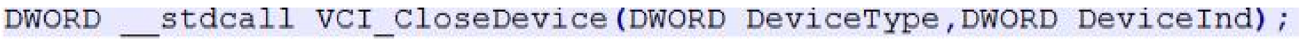
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, 0);
if(dwRel != 1)
{
    MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

6.2.2 VCI_CloseDevice

This function is used to close the device.



```
DWORD __stdcall VCI_CloseDevice(DWORD DeviceType,DWORD DeviceInd);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

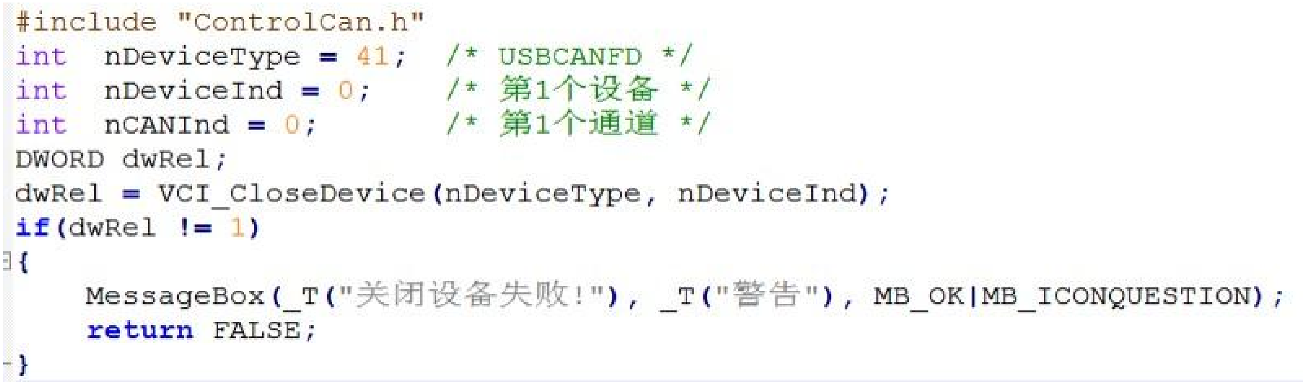
DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

return value

Return 1 indicates success and 0 failed.

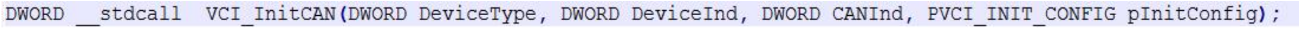
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_CloseDevice(nDeviceType, nDeviceInd);
if(dwRel != 1)
{
    MessageBox(_T("关闭设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

6.2.3 VCI_InitCAN

This function is used to initialize the specified CAN channel. When there are multiple CAN channels, multiple calls are required.



```
DWORD __stdcall  VCI_InitCAN(DWORD DeviceType, DWORD DeviceInd, DWORD CANInd, PVCI_INIT_CONFIG pInitConfig);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

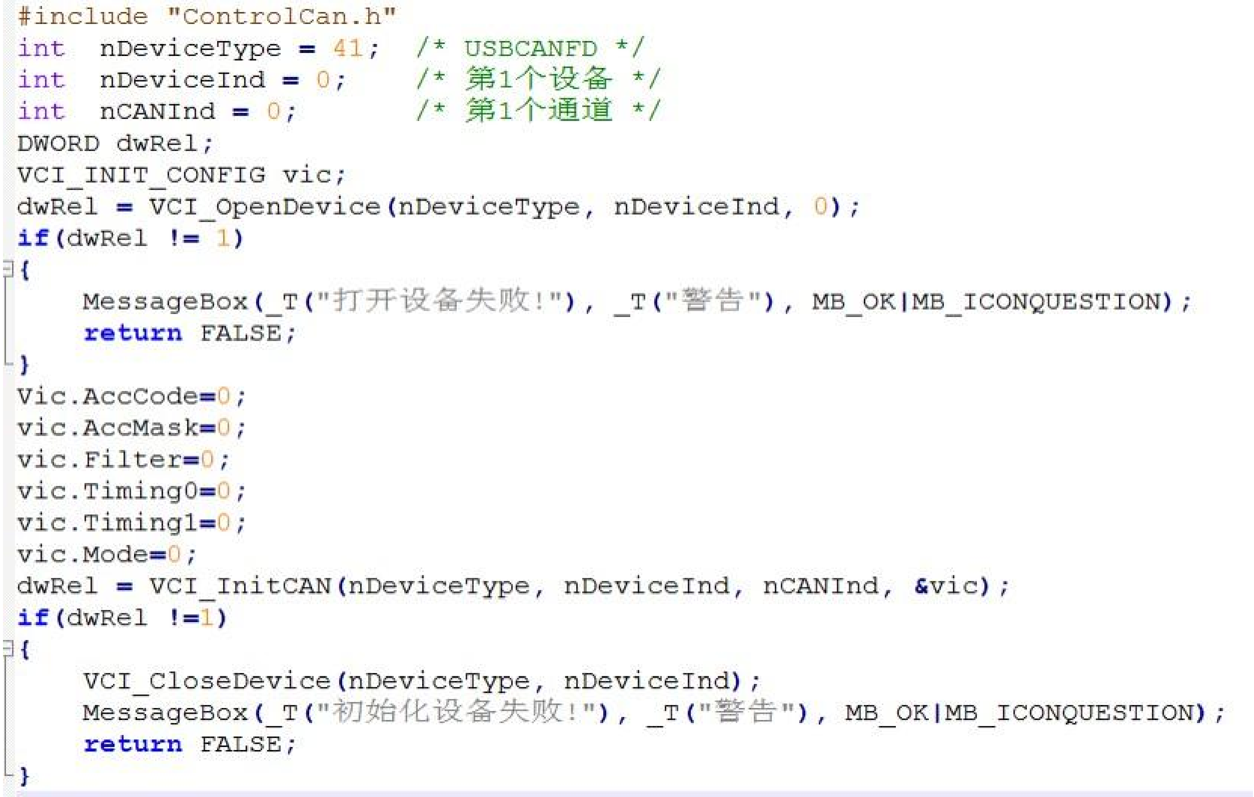
pInitConfig

Initialize parameter structure.

return value

Return 1 indicates success and 0 failed.

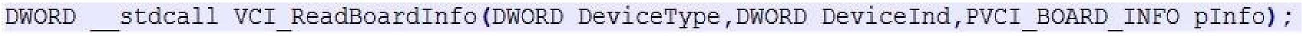
example



```
#include "ControlCan.h"
int  nDeviceType = 41;   /* USBCANFD */
int  nDeviceInd = 0;     /* 第1个设备 */
int  nCANInd = 0;        /* 第1个通道 */
DWORD dwRel;
VCI_INIT_CONFIG vic;
dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, 0);
if(dwRel != 1)
{
    MessageBox(_T("打开设备失败！"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
Vic.AccCode=0;
vic.AccMask=0;
vic.Filter=0;
vic.Timing0=0;
vic.Timing1=0;
vic.Mode=0;
dwRel = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if(dwRel !=1)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败！"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

6.2.4 VCI_ReadBoardInfo

This function is used to obtain device information.



```
DWORD __stdcall VCI_ReadBoardInfo(DWORD DeviceType,DWORD DeviceInd,PVCI_BOARD_INFO pInfo);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

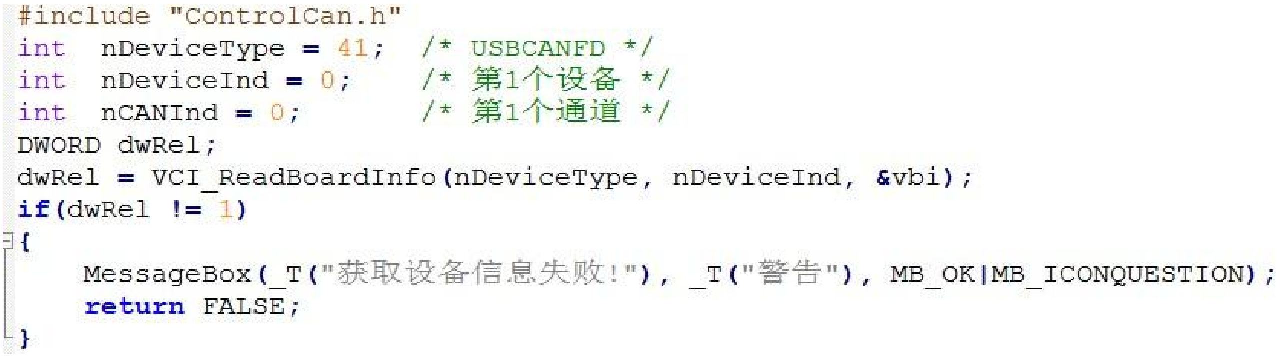
pInfo

The structure pointer of VCI_BOARD_ INFO used to store device information.

return value

Return 1 indicates success and 0 failed.

example

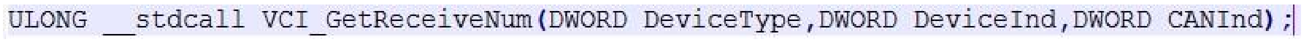


```
#include "ControlCan.h"
int   nDeviceType = 41;   /* USBCANFD */
int   nDeviceInd = 0;     /* 第1个设备 */
int   nCANInd = 0;        /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_ReadBoardInfo(nDeviceType, nDeviceInd, &vbi);
if(dwRel != 1)
{
    MessageBox(_T("获取设备信息失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

6.2.5 VCI_GetReceiveNum

This function retrieves the number of frames that have been received but not yet read in the receive buffer of the specified CAN channel. Its main purpose is to work in conjunction with VCI_Receive, which assumes that the buffer contains data before it is received.

In practical applications, users can improve program efficiency by directly looping calls to VCI_Receive and ignoring this function, thereby saving PC system resources.



```
ULONG __stdcall VCI_GetReceiveNum(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

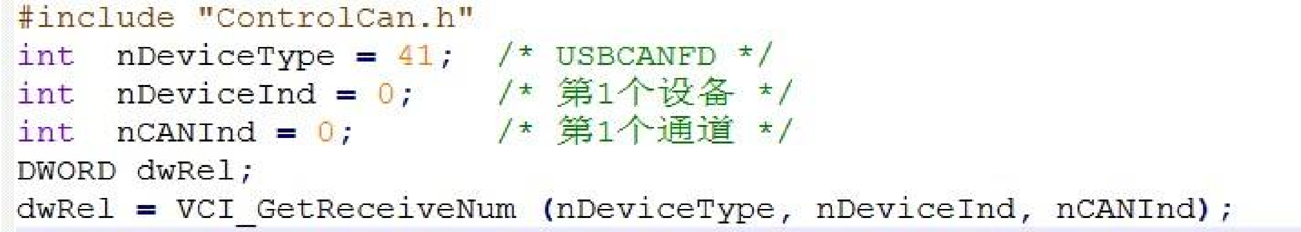
CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

return value

Returns the number of frames that have not been read yet,=-1 indicates that the USBCANFD device does not exist or the USB is disconnected.

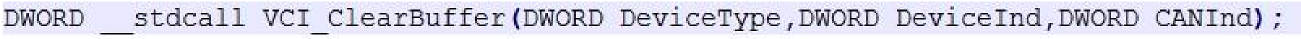
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_GetReceiveNum (nDeviceType, nDeviceInd, nCANInd);
```

6.2.6 VCI_ClearBuffer

This function is used to clear the buffer of the specified CAN channel. Mainly used in situations where receiving buffer data needs to be cleared, and sending buffer data will also be cleared.



```
DWORD __stdcall VCI_ClearBuffer(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

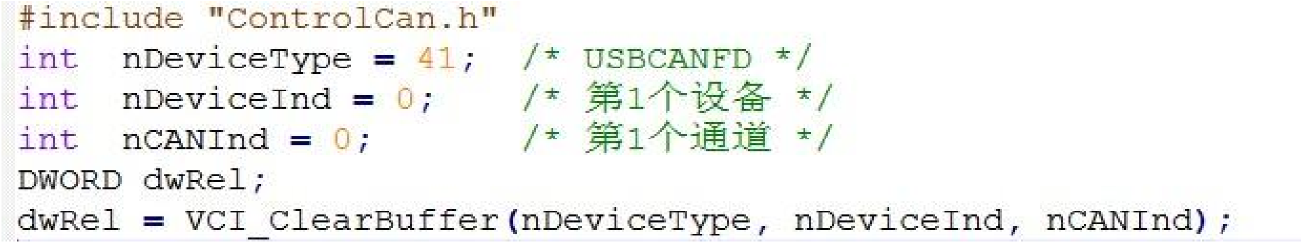
CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

return value

Return 1 indicates success and 0 failed.

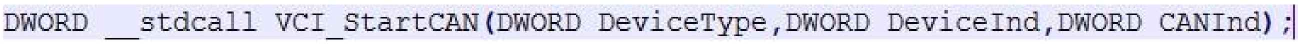
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_ClearBuffer(nDeviceType, nDeviceInd, nCANInd);
```

6.2.7 VCI_StartCAN

This function is used to activate a CAN channel of the CAN card. When there are multiple CAN channels, multiple calls are required.



```
DWORD __stdcall VCI_StartCAN(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

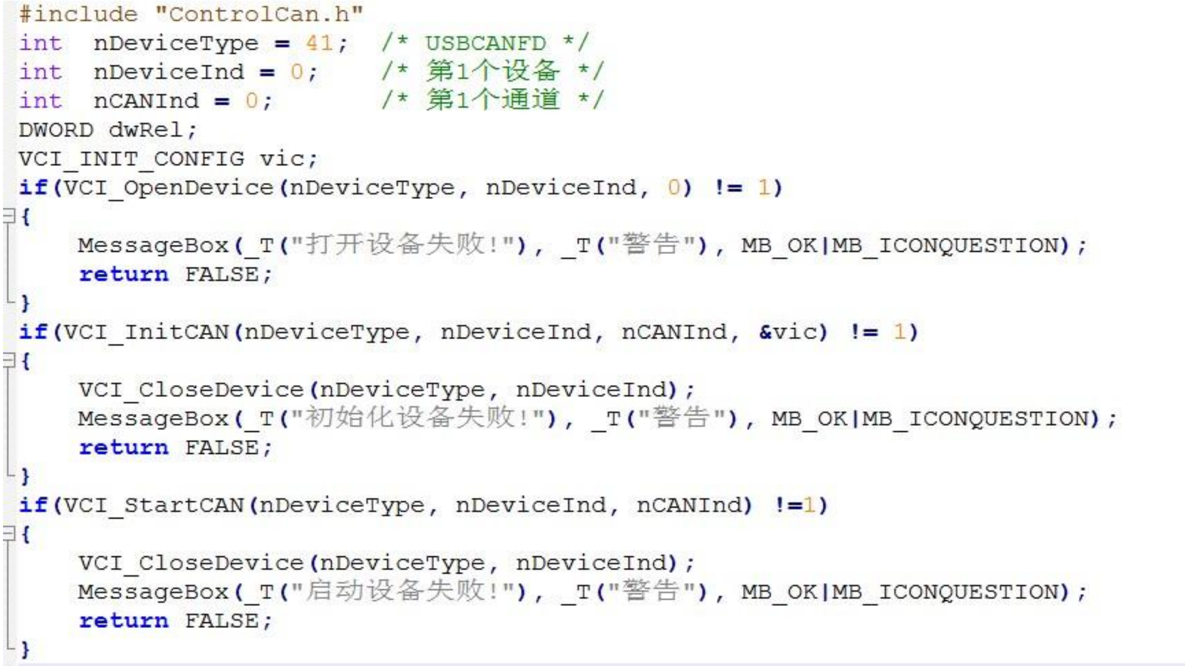
CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

return value

Return 1 indicates success and 0 failed.

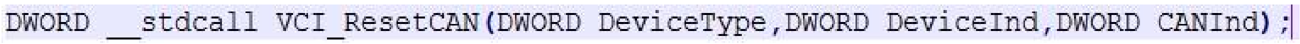
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
VCI_INIT_CONFIG vic;
if(VCI_OpenDevice(nDeviceType, nDeviceInd, 0) != 1)
{
    MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
if(VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic) != 1)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
if(VCI_StartCAN(nDeviceType, nDeviceInd, nCANInd) !=1)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("启动设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

6.2.8 VCI_ResetCAN

This function is used to reset CAN. Mainly used in conjunction with VCI_StartCAN, the normal state of the CAN card can be restored without further initialization. For example, when the CAN card enters the bus off state, this function can be called.



```
DWORD __stdcall VCI_ResetCAN(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

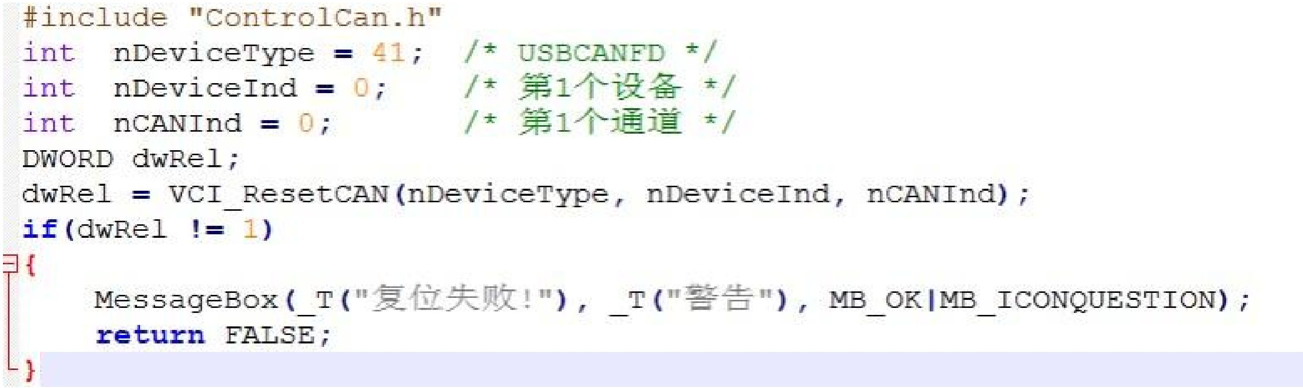
CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

return value

Return 1 indicates success and 0 failed.

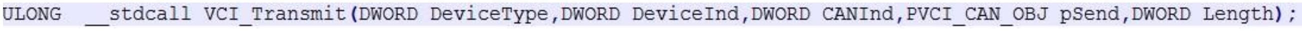
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_ResetCAN(nDeviceType, nDeviceInd, nCANInd);
if(dwRel != 1)
{
    MessageBox(_T("复位失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

6.2.9 VCI_Transmit

Send function. The return value is the actual number of frames successfully sent.



```
ULONG  __stdcall VCI_Transmit(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,PVCI_CAN_OBJ pSend,DWORD Length);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

pSend

The first pointer of structure array VCI_CAN_OBJ to be sent.

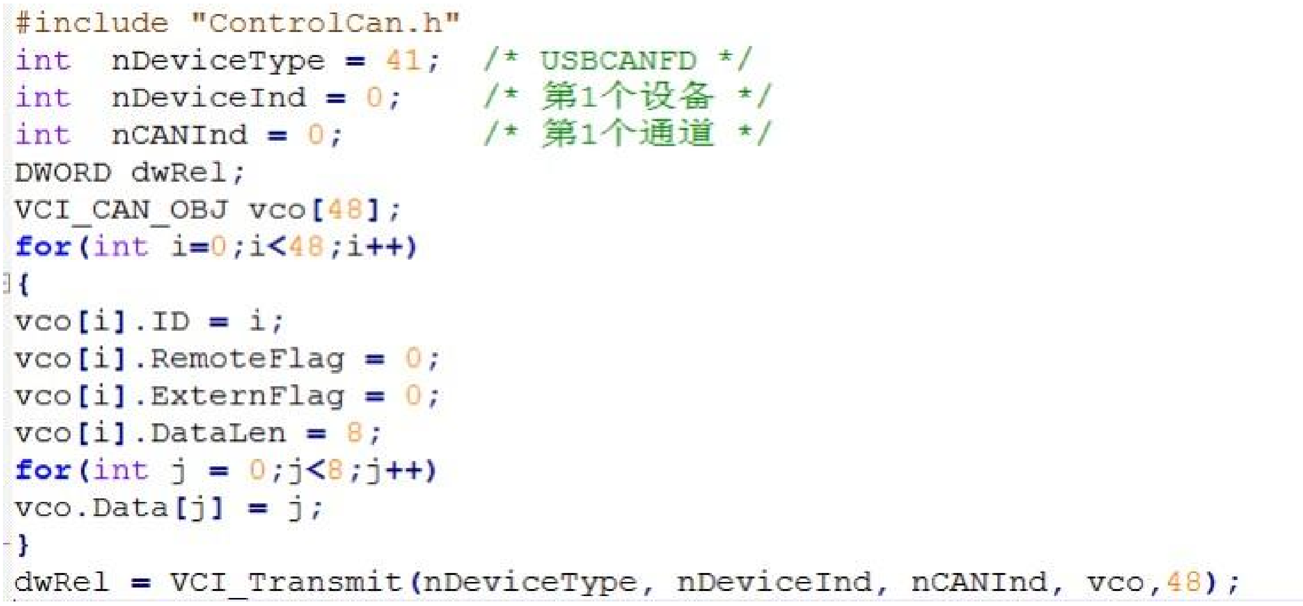
Length

The length of the frame structure array to be sent (the number of frames sent). The maximum is 1000, it is recommended to set it to 1 and send a single frame each time to improve transmission efficiency.

return value

Returns the actual number of frames sent,=-1 indicates that the USBCANFD device does not exist or the USB is disconnected.

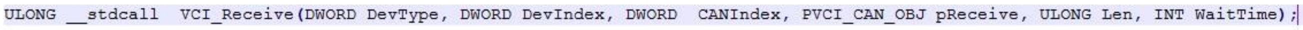
example



```
#include "ControlCan.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */
DWORD dwRel;
VCI_CAN_OBJ vco[48];
for(int i=0;i<48;i++)
{
vco[i].ID = i;
vco[i].RemoteFlag = 0;
vco[i].ExternFlag = 0;
vco[i].DataLen = 8;
for(int j = 0;j<8;j++)
vco.Data[j] = j;
}
dwRel = VCI_Transmit(nDeviceType, nDeviceInd, nCANInd, vco,48);
```

6.2.10 VCI_Receive

Receive function. This function reads data from the receive buffer of the specified device CAN channel.



```
ULONG __stdcall  VCI_Receive(DWORD DevType, DWORD DevIndex, DWORD  CANIndex, PVCI_CAN_OBJ pReceive, ULONG Len, INT WaitTime);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

pReceive

The first pointer of structure array VCI_CAN_OBJ which used for reception frames. Note: The size of the array must be larger than the len parameter below, otherwise memory read and write errors may occur。

Len

The length of the frame structure array used to receive (the maximum number of frames received this time, and the actual return value is less than or equal to this value). This value is the size of the provided storage space. The device has set a receive cache area of around 2000 frames for each channel. Users can choose an appropriate receive array length between 1 and 2000 based on their own system and working environment requirements. Generally, the size of the pReceive array and Len are set to be greater than 2000, such as 2500, which can effectively prevent address conflicts caused by data overflow. Simultaneously call VCI_Receive every 30ms is appropriate. While meeting the timeliness of the application, try to reduce the frequency of calling VCI_Receive as much as possible. As long as the internal cache is not overflowed and more frames are read and processed each time, it can improve operational efficiency.

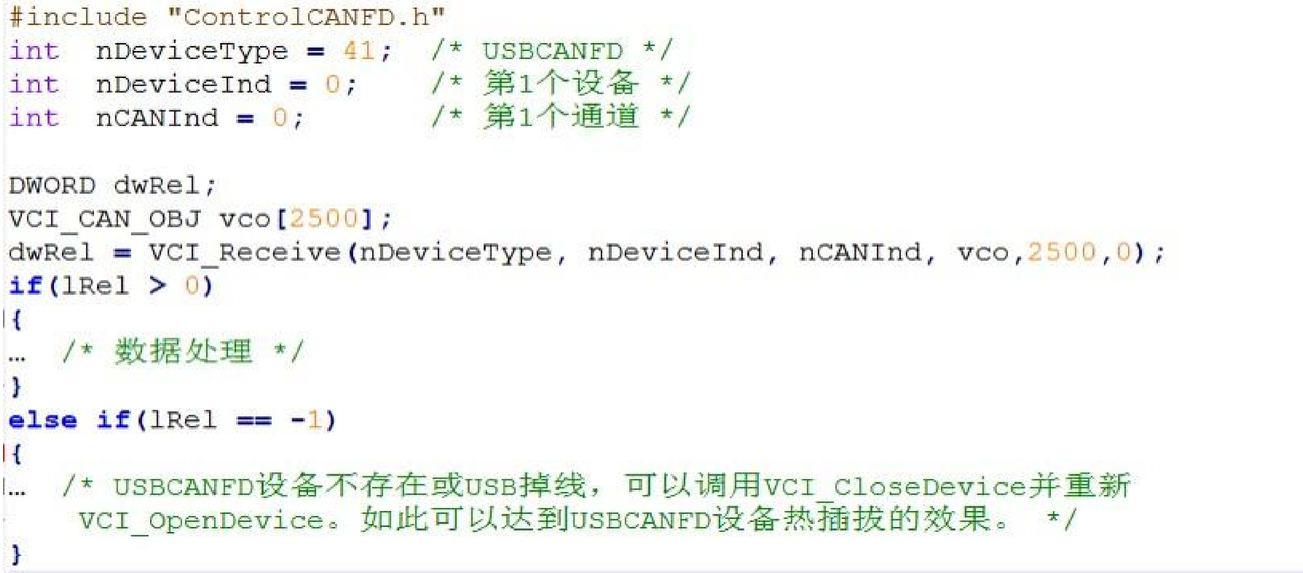
WaitTime

Retention parameter.

return value

Returns the actual number of frames read,=-1 indicates that the USBCANFD device does not exist or the USB is disconnected.

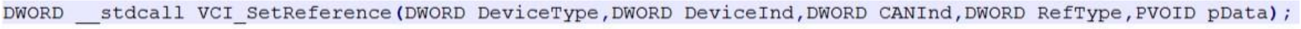
example



```
#include "ControlCANFD.h"
int  nDeviceType = 41;  /* USBCANFD */
int  nDeviceInd = 0;    /* 第1个设备 */
int  nCANInd = 0;       /* 第1个通道 */

DWORD dwRel;
VCI_CAN_OBJ vco[2500];
dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd, vco,2500,0);
if(lRel > 0)
{
…  /* 数据处理 */
}
else if(lRel == -1)
{
…  /* USBCANFD设备不存在或USB掉线, 可以调用VCI_CloseDevice并重新
    VCI_OpenDevice。如此可以达到USBCANFD设备热插拔的效果。 */
}
```

6.2.11 VCI_SetReference

Attribute setting function, which can be used to set Baud and filter.



```
DWORD __stdcall VCI_SetReference(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,DWORD RefType,PVOID pData);
```

parameter

DevType

Device type. Please refer to the definition of adapter device type for different product models.

DevIndex

Device index, for example, when there is only one USBCANFD adapter, the index number is 0. If another USBCANFD adapter is inserted, the device index number inserted later is 1, and so on.

CANIndex

CAN channel index. Which CAN channel is it. The CAN channel number of the corresponding card, with CAN1 being 0 and CAN2 being 1.

RefType

Attribute types, as shown in the table below.

pData

The data pointer corresponding to the attribute type is shown in the table below.

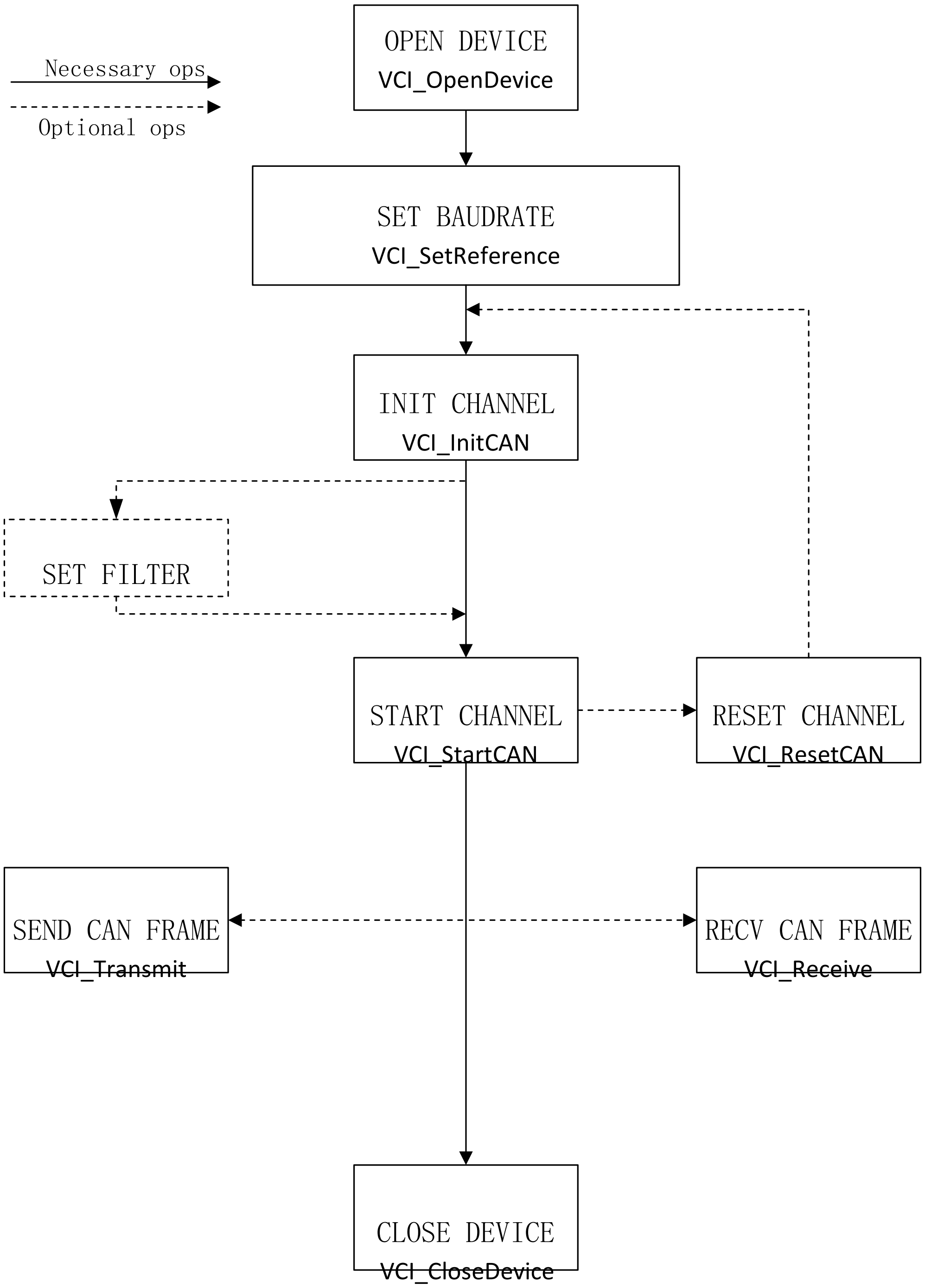
Attribute Type	RefType	pData
Baudrate setting (Because it supports the function of ordinary CAN, here only the arbitration domain baudrate of this CANFD device is set, and its data domain baudrate is fixed to 1Mbps)	0	Pointer pointing to the value of the address （*(DWORD*)pData） Corresponding relationship with baudrate setting is as follows： 0x060003 : baudrate set to 1Mbps 0x060004: baudrate set to 800kbps 0x060007 : baudrate set to 500kbps 0x1C0008: baudrate set to 250kbps 0x1C0011: baudrate set to 125kbps 0x160023: baudrate set to 100kbps 0x1C002C: baudrate set to 50kbps 0x1600B3: baudrate set to 20kbps 0x1C00E0: baudrate set to 10kbps 0x1C01C1: baudrate set to 5kbps

Filter setting
(The order of filtering settings in this set of interfaces is: 3->1->2. RefType is 3 to clear filtering, 1 to add filtering items, and 2 to start setting. Please refer to Chapter 6.3 for the filtering setting process)
	1	PData is the pointer which pointing to structure VCI_FILTER_RECORD,which is defined as：
		typedef struct _VCI_FILTER_RECORD{
		DWORD ExtFrame; // extended frame or not
		DWORD Start;
		DWORD End;
		}VCI_FILTER_RECORD,*PVCI_FILTER_RECORD;

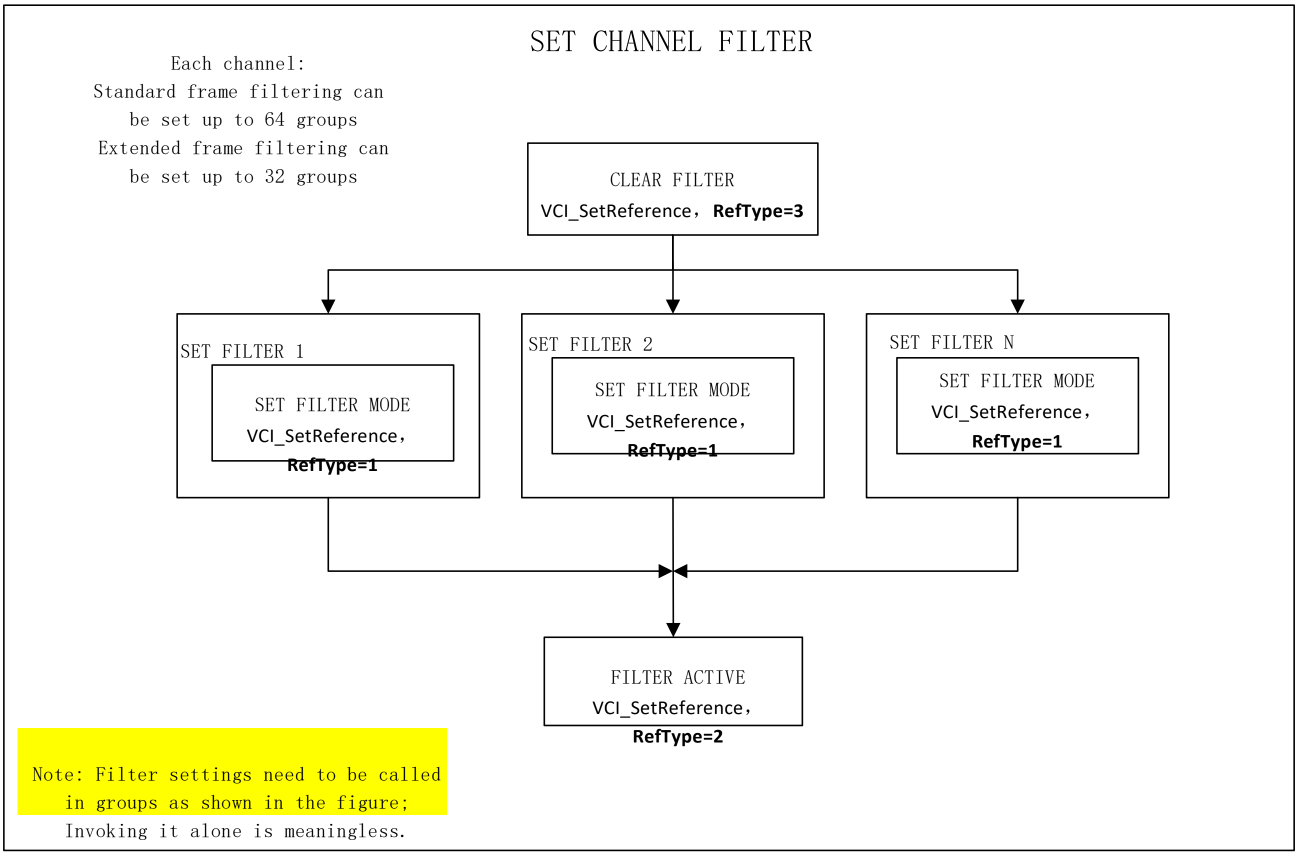
	2	No requirements, can be any value
	3	No requirements, can be any value

6.3 Flow of Using API

6.3.1 Flow



6.3.2 Filter Setup Flow


SET CHANNEL FILTER
Each channel:
Standard frame filtering can be set up to 64 groups
Extended frame filtering can be set up to 32 groups
CLEAR FILTER
VCI_SetReference，RefType=3
SET FILTER 1
SET FILTER MODE
VCI_SetReference，RefType=1
SET FILTER 2
SET FILTER MODE
VCI_SetReference，RefType=1
SET FILTER N
SET FILTER MODE
VCI_SetReference，RefType=1
FILTER ACTIVE
VCI_SetReference，RefType=2
Note: Filter settings need to be called in groups as shown in the figure;
Invoking it alone is meaningless.