User
Manual

CAN-bus Series Device

Common Test Software and Interface Function Library

UM04010000 V1.00 Date: 2012/12/19 **User Manual**

www.zlgmcu.com

Guangzhou ZHIYUAN Electronics Stock Co., Ltd.

Revision History

Version	Rev. Date	Modifications
V1.00	2012-12-19	Original version

Sales Information

Guangzhou ZLGMCU Technology Co., Ltd.

Address: F4 Room, 12 Floor, Everbright BANK Building, 689 Tianhe Northern Road, Guangzhou, CHINA

TEL: +86-20-38732494 38730972 38730976 38730916 38730917 38730977

FAX: +86-20-38730925

Website: www.zlgmcu.com

Guangzhou Sales Office

Address: Room 203 & 204, XinSaiGE Electronic Building, Tianhe District, Guangzhou, CHINA
TEL: +86-20-87578634, 87578842, 87569917
FAX: +86-20-87578842

Beijing Sales Office

Address: Room 1207 & 1208, Yingwang Centre, 113 Zhichun Road, Haiding District, Beijing, CHINA
TEL: +86-10-62635033, 62635573, 62635884, 62536178, 62536179, 82628073
FAX: +86-10-82614433

Hangzhou Sales Office

Address: Room 502, Jiangnan Electronics Building, 217 Tianmu Road, Hangzhou, CHINA
TEL: +86-571-89719480, 89719481, 89719482, 89719483, 89719484, 89719485
FAX: +86-571-89719494

Shenzhen Sales Office

Address: Room D, Floor 4, C Side, Dianzikeji Building, 2070 ShenNanZhong Road, Shenzhen, CHINA
TEL: +86-755-83781768, 83781788, 83782922, 82941683
FAX: +86-755-83793285

Shanghai Sales Office

Address: Room 7E, Eastern side, Kejijingcheng Building, 668 Beijingdong Road, Shanghai, CHINA
TEL: +86-21-53083452, 53083453, 53083496, 53083497
FAX: +86-21-53083491

Nanjing Sales Office

Address: Room 1501, Zhujiang Building, 280 Zhujiang Road, Nanjing, CHINA
TEL: +86-25-68123901, 68123902
FAX: +86-25-68123900

Chongqing Sales Office

Address: Room 1611, Saige electronics market, Daxiyang International Building, 2 Keyuanyi Road, Shiqiao, Chongqing, CHINA
TEL: +86-23-68796438, 68796439, 68797619
FAX: +86-23-68796439

Chengdu Sales Office

Address: Room 403, Digital Scientific Building, 2 Southern Yihuan Road, Chengdu, CHINA
TEL: +86-28-85439836, 85432683, 85437446, 85437876
FAX: +86-28-85437896

Wuhan Sales Office

Address: Room 12128, Huazhong Computer and electronics market, 158 LuoYu Road, GuangFouTun, HongShan District, Wuhan, CHINA
TEL: +86-27-87168497, 87168297, 87168397
FAX: +86-27-87163755

XiAn Sales Office

Address: Room 1201, Pacific Building, 54 Changanbei Road, XiAn, CHINA
TEL: +86-29-87881296, 83063000, 87881295
FAX: +86-29-87880865

Technical Supports

Guangzhou ZHIYUAN Electronics Stock Co., Ltd.

Address: Floor 2, Building No.3 Huangzhou Industrial Estate, Chebei Road,
Tianhe District, Guangzhou, CHINA, Post code: 510660

TEL: +86-20-22644249, 28872524, 22644399, 28872342, 28872349, 28872569, 28872573

FAX: +86-20 38601859

Website: www.embedtools.com www.embedcontrol.com www.ecardsys.com

Technical Supports

CAN-bus
TEL: +86-20-22644381, 22644382, 22644253
E-mail: can.support@embedcontrol.com

iCAN & Data collection
TEL: +86-20-28872344, 22644373
E-mail: ican@embedcontrol.com

MiniARM
TEL: +86-20-28872684, 28267813
E-mail: miniarm.support@embedtools.com

Ethernet
TEL: +86-20-22644380, 22644385
E-mail: ethernet.support@embedcontrol.com

Wireless Communication
TEL: +86-20-22644386
E-mail: wireless@embedcontrol.com

Serial Communication
TEL: +86-20-28267800, 22644385
E-mail: serial@embedcontrol.com

Programmer
TEL: +86-20-22644371
E-mail: programmer@embedtools.com

Analyze Tools & Instrument
TEL: +86-20-22644375, 28872624, 28872345
E-mail: tools@embedtools.com

ARM Embedded System Application
TEL: +86-20-28872347, 28872377,
22644383, 22644384
E-mail: arm.support@zlgmcu.com

Building Automation
TEL: +86-20-22644376, 22644389, 28267806
E-mail: mjs.support@ecardsys.com

Sales Contact
TEL: +86-20-22644249, 22644399, 22644372, 22644261, 28872524,
+86-20-28872342, 28872349, 28872569, 28872573, 38601786

Repair and rework
TEL: +86-20-22644245

Contents

Chapter 1: Test Software Usage

CAN-bus common test software is designed for testing ZLGCAN series board cards. It is featured with simple operation and easy-to-use, very convenient for user to perform testing to the board cards. The main interface is shown as following.

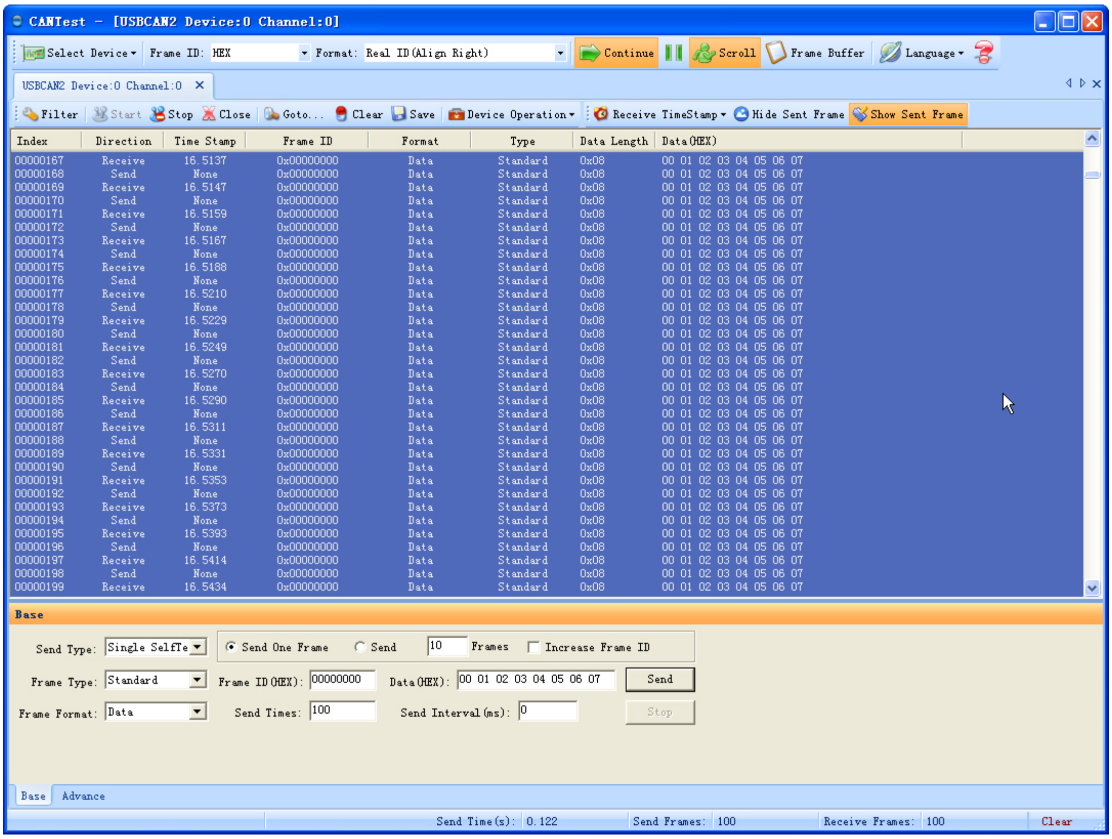


Figure 1-1: CANopen network manage software main interface

1.1 Device operation

1.1.1 Select device type

Before operations, select the device type that you want in the “Type” menu, as Figure 1-2 shows.

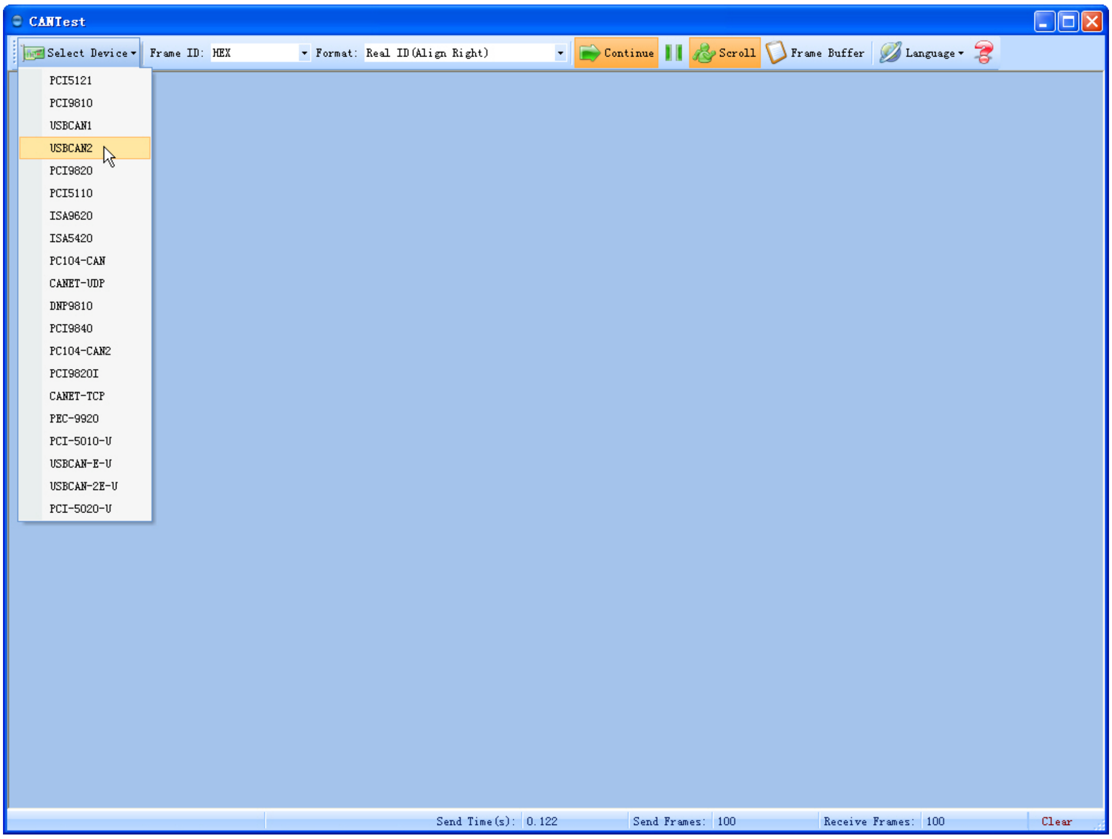


Figure 1-2: Select device type

Then, a dialog box “select device” will pop out, as Figure 1-3 shows.

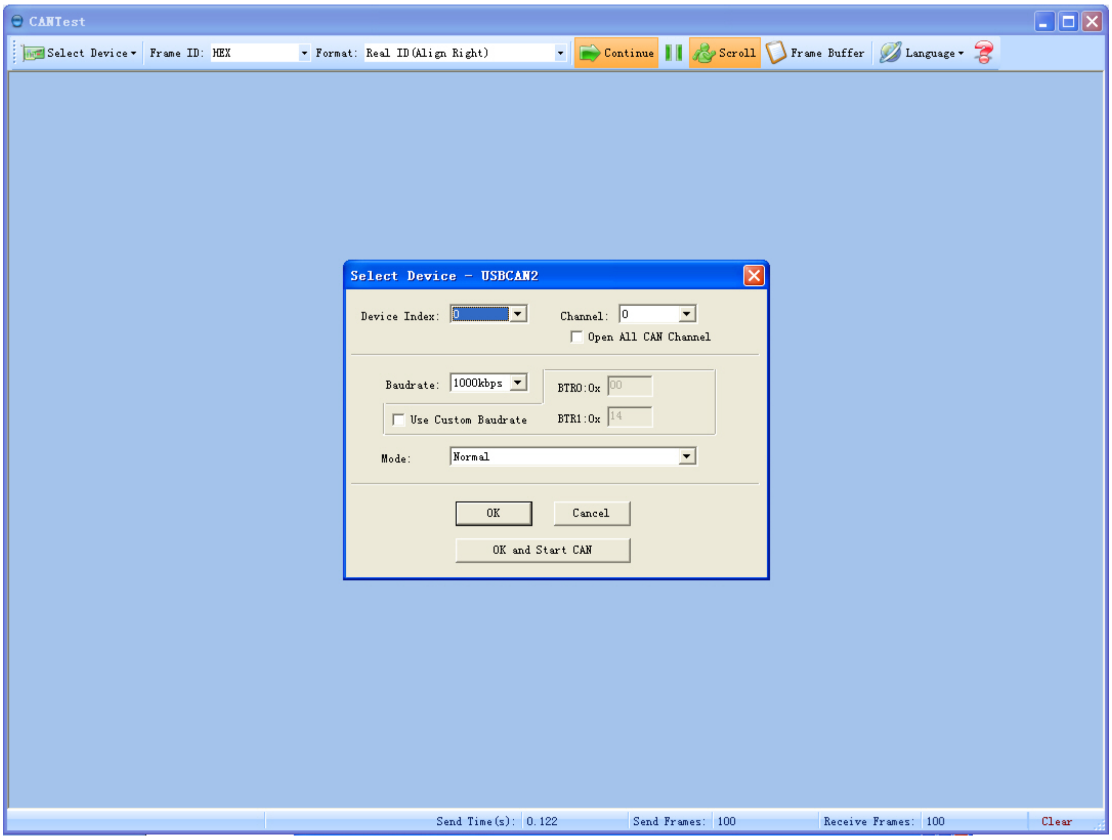


Figure 1-3: Select device

In this dialog box, you can select the device index to be open and CAN channel, and then set the initialization parameters. When finishing setting, click the OK button to open the device operation window, or click OK and Start CAN button to open the device operation window and start the device and CAN channel automatically,

1.1.2 Filter setting

In the device operation window, click the Filer setting button to perform settings, as Figure 1-3 shows. If filer setting is not required, please skip this step.

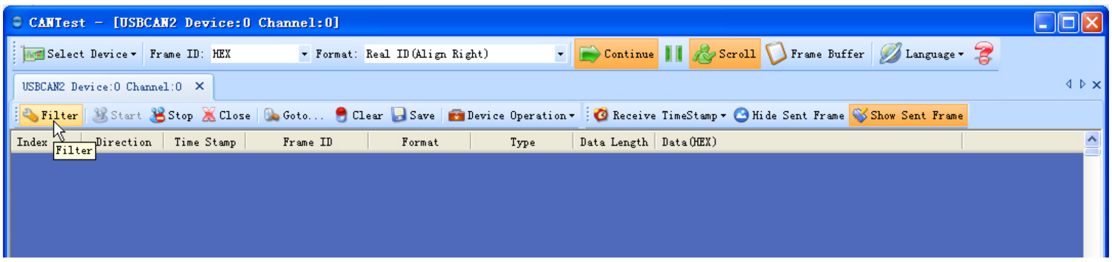


Figure 1-4: Filter setting

Then, a dialog box “Filter setting” will pop up, as Figure 1-4 shows.

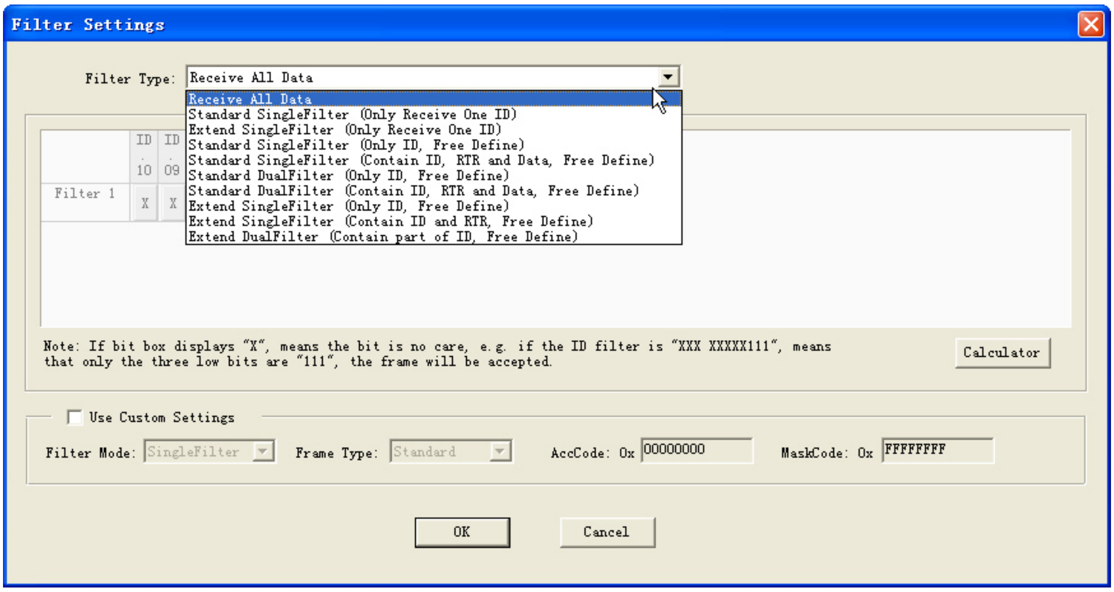


Figure 1-5: Select the filter mode

Select the filter mode, and set the CAN frame needed to be filtered by configuring the filer.

1.1.3 Start CAN

Click the “Start” button to start CAN channel, and right now the received CAN data will be shown in the data list automatically, as Figure 1-6 shows.

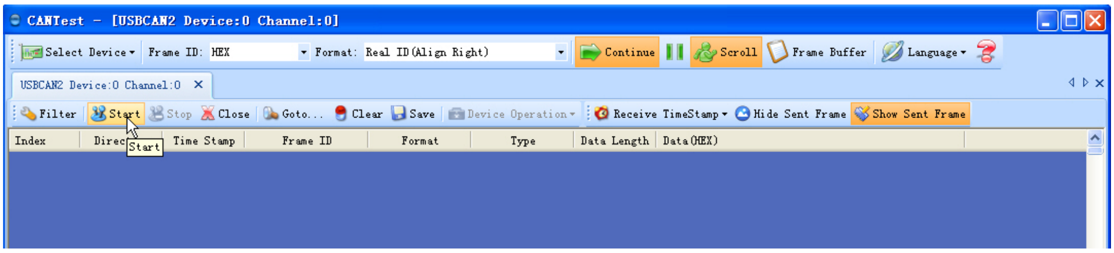


Figure 1-6: Start CAN

After analysis is finished, a dialog window will appear, as Figure 1-7 shows.

1.1.4 Obtain device information

After the CAN channel is started, you can select the “Device information” item in the “Device operation” pull-down menu to obtain the detailed information of the current device, as Figure 1-8 shows.

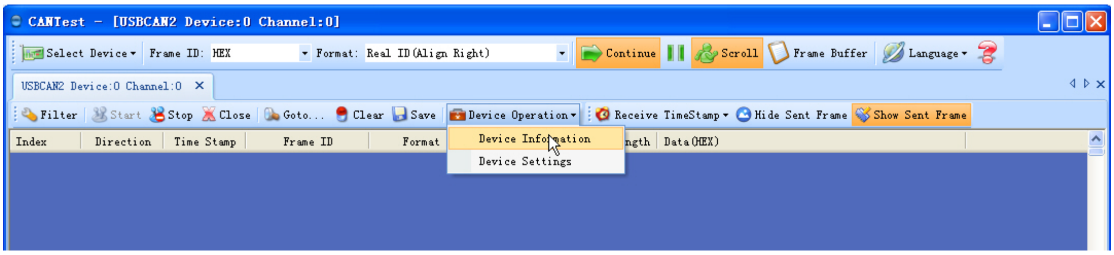


Figure 1-7: Obtain device information

1.1.5 Data transmission

When CAN channel is start successfully, set the parameters for the CAN frame to be transmitted. The item “receive own message” in the “Send Type” pull-down menu means the device can receive the CAN frame sent out by itself. What should be notice is this option is only available in test mode, so please select “Normal” for actual application.

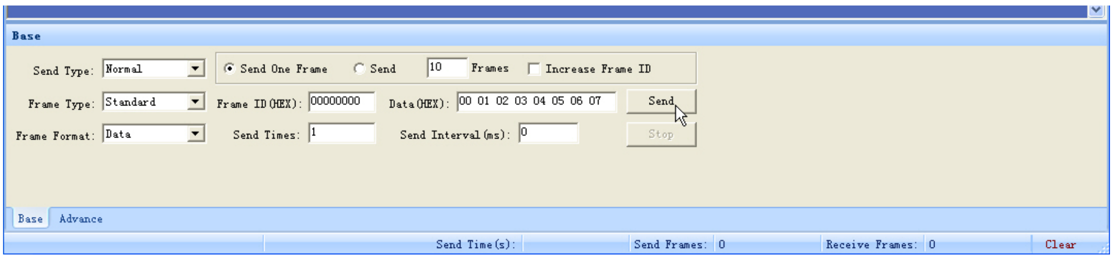


Figure 1-8: Data transmission setting

Click “Advance” option to enter the advance operation interface. In this page, you can configure up to 10 CAN frames to be sent at a same time.

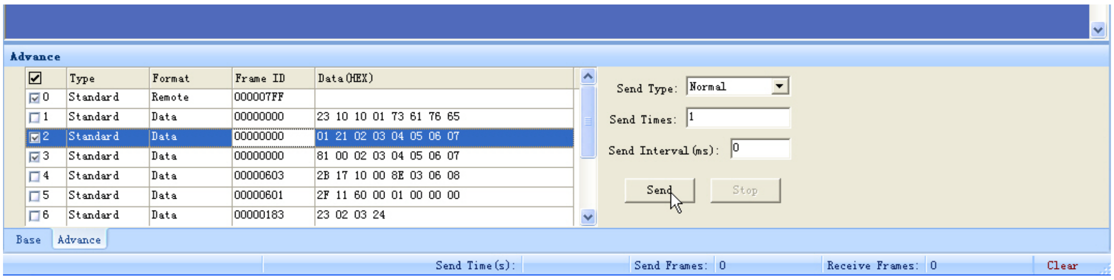


Figure 1-9: Advance operation interface

1.2 Auxiliary operation

This software also provides some auxiliary operations for users to monitor and analyze CAN data.

Figure 1-10: Auxiliary operation

1.2.1 Frame ID display mode

There are 3 frame ID display modes: Binary, Decimal and Hexadecimal.

1.2.2 Frame ID display format

There are two frame ID display formats: real ID and SJA1000 format (standard frame: the real ID left shift 3 bits; extend frame: the real ID left shift 5 bits).

1.2.3 Continue to display sent and received data

When select this option, data transmission will be processed in foreground, and all the data will be shown out.

1.2.4 Pause to display sent and received data

When select this option, data transmission will be processed in background, and all the data will be not shown out.

1.2.5 Roll

When select this option, the last line of the current data list is always visible.

1.2.6 Display frame number

This option is used to set the displayed frame number on the data list.

1.2.7 Language

This option is used to select language.

Chapter 2: Interface Function Library Usage

2.1 Interface card device type definition

The type definition of each interface card is as following:

Table 2-1: Type definition of interface card

Device Name	Type
PCI5121	1
PCI9810	2
USBCAN1	3
USBCAN2	4
PCI9820	5
CAN232	6
PCI5110	7
CANlite(CANmini)	8
ISA9620	9
ISA5420	10
PC104-CAN	11
CANET-UDP	12
DNP9810	13
PCI9840	14
PC104-CAN2	15
PCI9820I	16
CANET-TCP	17
PEC-9920	18
PCIE-9220	18
PCI-5010-U	19
USBCAN-E-U	20
USBCAN-2E-U	21
PCI-5020-U	22
EG20T-CAN	23

2.2 Error code definition

Table 2-2: The definition of error code

Name	Value	Description
CAN error code		
ERR_CAN_OVERFLOW	0x00000001	CAN controller internal FIFO overflow
ERR_CAN_ERRALARM	0x00000002	CAN controller error warning
ERR_CAN_PASSIVE	0x00000004	CAN controller passive error
ERR_CAN_LOSE	0x00000008	CAN controller arbitration lost
ERR_CAN_BUSERR	0x00000010	CAN controller bus error
ERR_CAN_BUSOFF	0x00000020	CAN controller bus off
Common error code		
ERR_DEVICEOPENED	0x00000100	Device is already open
ERR_DEVICEOPEN	0x00000200	Open device fails
ERR_DEVICENOTOPEN	0x00000400	Device is not open
ERR_BUFFEROVERFLOW	0x00000800	Buffer overflow
ERR_DEVICENOTEXIST	0x00001000	Device is not exist
ERR_LOADKERNELDLL	0x00002000	Load dynamic library fails
ERR_CMDFAILED	0x00004000	Execute command fails
ERR_BUFFERCREATE	0x00008000	Insufficient memory
CANET error code		
ERR_CANETE_PORTOPENED	0x00010000	This port is already open
ERR_CANETE_INDEXUSED	0x00020000	This device index is already in used
ERR_REF_TYPE_ID	0x00030001	SetReference or GetReference is a transferred RefType which is not exist
ERR_CREATE_SOCKET	0x00030002	Create socket fails
ERR_OPEN_CONNECT	0x00030003	Open socket connection fails, the connection of device may be exist
ERR_NO_STARTUP	0x00030004	This device is not started
ERR_NO_CONNECTED	0x00030005	This device is unconnected
ERR_SEND_PARTIAL	0x00030006	Send partial CAN frame
ERR_SEND_TOO_FAST	0x00030007	Data is sent too fast, Socket buffer is full

2.3 Function library data structure definition

2.3.1 VCI_BOARD_INFO

2.3.1.1 Description

VCI_BOARD_INFO structure contains the device information of ZLGCAN series interface card. This structure will be filled within 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;
```

2.3.1.2 Member

hw_Version

Hardware version number (Hexadecimal). For example, 0x0100 is for V1.00.

fw_Version

Firmware version number (Hexadecimal)

dr_Version

Driver version number (Hexadecimal)

in_Version

Interface library version number (Hexadecimal)

irq_Num

Interrupt number for Board card

can_Num

The quality of the CAN channel

str_Serial_Num

Board card serial number

str_hw_Type

This is for Hardware type, such as “USBCAN V1.00”. (Notice: It includes character string terminator ’\0’).

Reserved

System reserved item

2.3.2 VCI_CAN_OBJ

2.3.2.1 Description

VCI_CAN_OBJ structure is used to transfer CAN information frame in VCI_Transmit and VCI_Receive functions.

```
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;
```

2.3.2.2 Member

ID

Message ID

TimeStamp

This is the time stamp for when information frame is received. It is counted since CAN controller is initialized.

TimeFlag

This is used to indicate whether the time stamp is in used, in which 1 for TimeStamp is valid. TimeFlag and TimeStamp are effective only when the frame is a received frame.

SendType

This is used to indicate the transmission mode, in which 0 is for normal, 1 is for single transmission, 2 is for receive own message, 3 is for receive own message once. SendType is effective only when the frame is a received frame. (For EG20T-CAN device, the transmission mode is set in VCI_InitCan function. In this case, the setting on here is invalid. For receive own message setting, G20T-CAN will not receive any data from the bus, but only receive the data send by itself.)

RemoteFlag

This is used to indicate whether the frame is a remote frame.

ExternFlag

This is used to indicate whether the frame is an extend frame.

DataLen

Data length (<=8)

Data

Data in the message

Reserved

System reserved item

2.3.3 VCI_CAN_STATUS

2.3.3.1 Description

VCI_CAN_STATUS structure contains CAN controller status information. This structure will be filled within VCI_ReadCanStatus function.

```
typedef struct _VCI_CAN_STATUS {
    UCHAR     ErrInterrupt;
    UCHAR     regMode;
    UCHAR     regStatus;
    UCHAR     regALCapture;
    UCHAR     regECCapture;
    UCHAR     regEWLimit;
    UCHAR     regRECounter;
    UCHAR     regTECounter;
    DWORD     Reserved;
} VCI_CAN_STATUS, *PVCI_CAN_STATUS;
```

2.3.3.2 Member

ErrInterrupt

This is the interrupt logging. It is cleared by reading.

regMode

CAN controller mode register

regStatus

CAN controller status register

regALCapture

CAN controller arbitration lost register

regECCapture

CAN controller error register

regEWLimit

CAN controller error warning limit register

regRECounter

CAN controller receive error register

regTECounter

CAN controller transmit error register

Reserved

System reserved item

2.3.4 VCI_ERR_INFO

2.3.4.1 Description

VCI_ERR_INFO structure is used to load the VCI library running error information. This structure will be filled within VCI_ReadErrInfo function.

```
typedef struct _ERR_INFO {
    UINT ErrCode;
    BYTE Passive_ErrData[3];
    BYTE ArLost_ErrData;
} VCI_ERR_INFO, *PVCI_ERR_INFO;
```

2.3.4.2 Member

ErrCode

Error code (Refer to the definition of error code in the Section 2.2 of Chapter 2)

Passive_ErrData

When passive error occurs, it indicates the error marking data of passive error.

ArLost_ErrData

When arbitration lost error occurs, it indicates the error marking data of arbitration lost error.

2.3.5 VCI_INIT_CONFIG

2.3.5.1 Description

VCI_INIT_CONFIG structure defines the configuration for CAN initialization. This structure will be filled within VCI_InitCan function.

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

2.3.5.2 Member

AccCode

This is the verification code.

AccMask

This is the mask code.

Reserved

System reserved

Filter

This is the filter mode

Timing0

This is the Timer 0 (BTR0)

Timing1

This is the Timer 1 (BTR1)

Mode

This is the mode.

2.3.5.3 Remark

This structure is described in detailed in Table 2-3.

Timing0 and Timing1 are used to set the CAN baudrate. The tabel below lists the common baud rate settings.

CAN baud rate	Timer 0	Timer 1
5Kbps	0xBF	0xFF
10Kbps	0x31	0x1C
20Kbps	0x18	0x1C
40Kbps	0x87	0xFF
50Kbps	0x09	0x1C
80Kbps	0x83	0Xff
100Kbps	0x04	0x1C
125Kbps	0x03	0x1C
200Kbps	0x81	0xFA
250Kbps	0x01	0x1C
400Kbps	0x80	0xFA
500Kbps	0x00	0x1C
666Kbps	0x80	0xB6
800Kbps	0x00	0x16
1000Kbps	0x00	0x14

Notice: For PCI-5010-U/PCI-5020-U/USBCAN-E-U/USBCAN-2E-U device, the Baud rate is not set in this structure. It should be set by VCI_SetReference function before VCI_InitCan function is called. For more information, please refer to VCI_SetReference.

2.3.6 CHGDESIPANDPORT

2.3.6.1 Description

CHGDESIPANDPORT structure is used to load the necessary information of the target IP and port for changing CANET-E. This structure is applied in CANETE-E.

```
typedef struct _tagChgDesIPAndPort {
        char szpwd[10];
        char szdesip[20];
        int desport;
        BYTE blisten;
} CHGDESIPANDPORT;
```

2.3.6.2 Member

szpwd[10]

This is the password for changing target IP and port. The length should be less than 10, such as “11223344”.

szdesip[20]

This is the target IP to be changed, such as “192.168.0.111”.

desport

This is the target port to be changed, such as “4000”.

blisten

This is the operation mode to be changed, 0 for normal mode and 1 for listen only mode.

2.3.7 VCI_FILTER_RECORD

2.3.7.1 Description

VCI_FILTER_RECORD structure defines the filtering range of the CAN filter. This structure will be filled within VCI_SetReference function.

```
typedef struct _VCI_FILTER_RECORD{
        DWORD ExtFrame;
        DWORD Start;
        DWORD End;
} VCI_FILTER_RECORD,*PVCI_FILTER_RECORD;
```

2.3.7.2 Member

ExtFrame

This is the frame type flag for filtering, 1 for extend frame and 0 for standard frame.

Start

This is the ID of start frame in the filtering range.

End

This is the ID of end frame in the filtering range.

2.4 Interface library function specification

2.4.1 VCI_OpenDevice

2.4.1.1 Description

This function is used to open device.

```
DWORD __stdcall VCI_OpenDevice(DWORD DevType, DWORD DevIndex, DWORD Reserved);
```

2.4.1.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

Reserved

For CAN232, this parameter is used to indicate the baud rate for the serial port. It can be 2400, 4800, 9600, 14400, 19200, 28800 or 57600. For CANET-UDP, this parameter indicates the local port number to be open. The range of it can be from 5000 to 40000. For CANET-TCP, this parameter is fixed with 0. However, for other devices, this parameter is invalid.

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.1.3 Example

```
#include "ControlCan.h"


int nDeviceType = 6;       /* CAN232 */
int nDeviceInd = 0;        /* COM1 */
```



```
int nReserved = 9600;     /* Baud rate */
DWORD dwRel;

dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
      MessageBox(_T("Open device fails!"), _T("Warning"), MB_OK|MB_ICONQUESTION);
      return FALSE;
}
```

2.4.2 VCI_CloseDevice

2.4.2.1 Description

This function is used to close device.

```
DWORD __stdcall VCI_CloseDevice(DWORD DevType, DWORD DevIndex);
```

2.4.2.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.2.3 Example

```
#include "ControlCan.h"

int nDeviceType = 6;      // CAN232
int nDeviceInd = 0;       // COM1
BOOL bRel;

bRel = VCI_CloseDevice(nDeviceType, nDeviceInd);
```

2.4.3 VCI_InitCan

2.4.3.1 Description

This function is used to initialize the specific CAN channel. (For USBCAN-E-U or USBCAN-2E-U, the baud rate should be set by VCI_SetReference function before VCI_InitCan function is called)

```
DWORD __stdcall VCI_InitCan(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_INIT_CONFIG pInitConfig);
```

2.4.3.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

pInitConfig

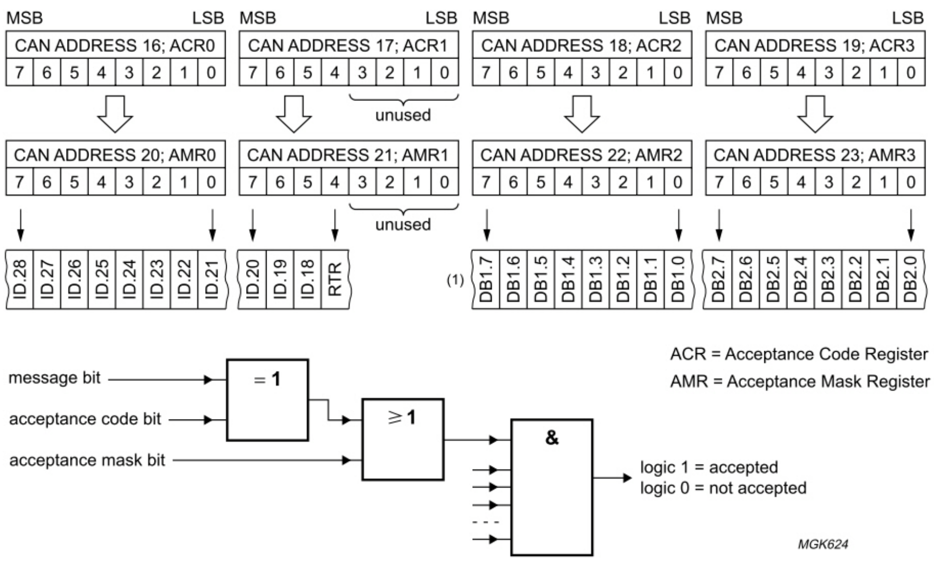
It is used to initialize parameter structure. (Notes: 1. For CAN232, this parameter should beset to NULL; 2. For PCI-5010-U/PCI-5020-U/USBCAN-E-U/USBCAN-2E-U, the settings about filtering and baud rate should be configured in VCI_SetReference. And only Mode is needed to be set in pInitConfig, while other 6 member can be ignored. For more detailed information, please refer to VCI_SetReference).

Table 2-3: Initial parameter structure

Member	Description
pInitConfig->AccCode	AccCode is corresponding to the 4 registers in SJA1000, they are ACR0, ACR1, ACR2 and ACR3, in which the higher byte is corresponding to ACR0, while the lower byte Is corresponding to ACR3. AccMask is corresponding to the 4 registers in SJA1000, they are AMR0, AMR1, AMR2 and AMR3 in which the higher byte is corresponding to AMR0, while the lower byte Is corresponding to AMR3 (For more information, please refer to the explanation below)
pInitConfig->AccMask	
pInitConfig->Reserved	Reserved
pInitConfig->Filter	Filter mode, 1 for single filtering, 0 for double filtering (For EG20T-CAN, 0 means receive standard frame and extend frame; 1 means only receive and filter standard frame, extend frame is not received; 3 means only receive and filter extend frame, standard frame is not received
pInitConfig->Timing0	Timer 0
pInitConfig->Timing1	Timer 1
pInitConfig->Mode	Mode, 0 for normal mode, and 1 for listen only mode (For EG20T-CAN, the meaning of Mode is as following:

	Bit0 is 0 for normal mode, and 1 for listen only mode (Silence mode); Bit1 is 0 for normal mode, and 1 for receive own message mode (Loop mode); Bit2 is 0 for single transmission, and 1 for automatic resend mode (Auto resend mode) For receive own message setting, G20T-CAN will not receive any data from the bus, but only receive the data send by itself.

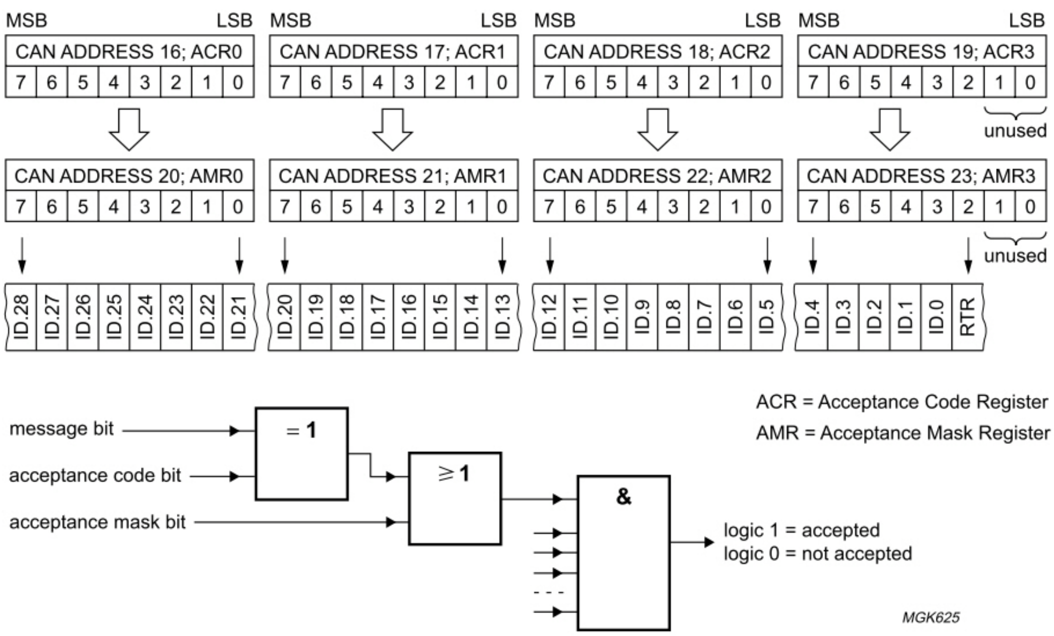
For single filtering mode, when the received frame is set to be a standard frame:



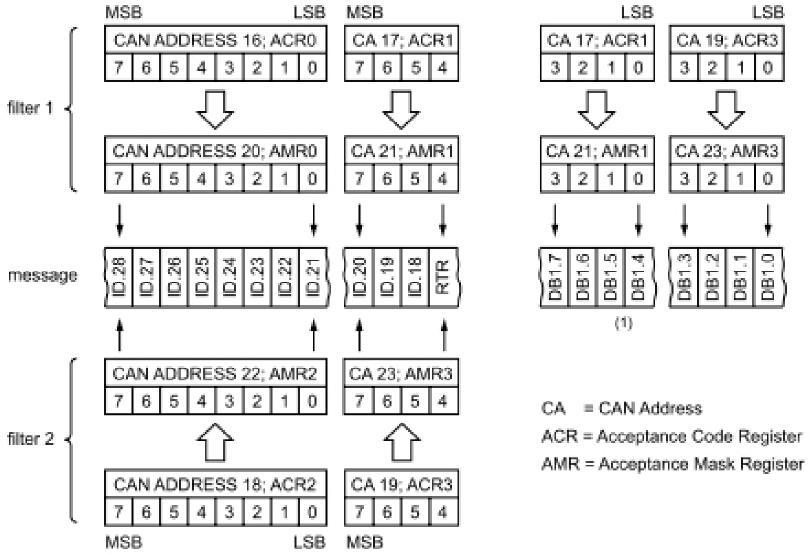
DBX.Y means data byte X, bit Y.

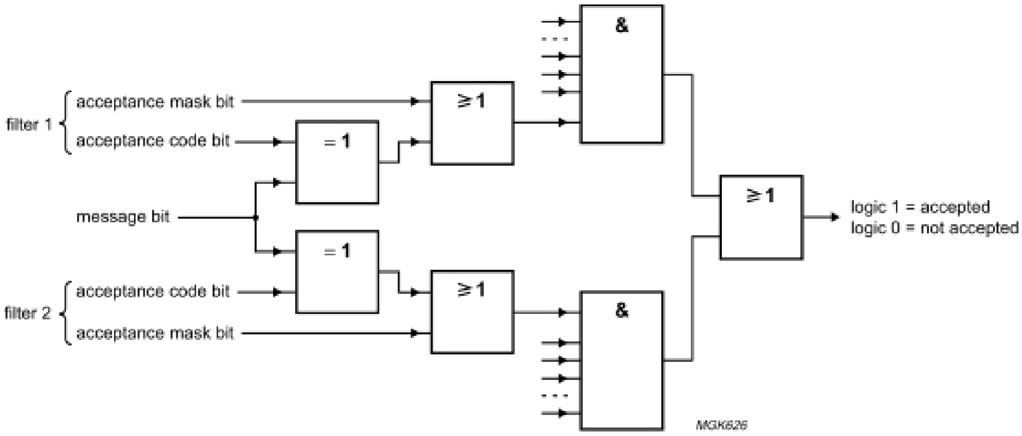
RTR is corresponding to the RemoteFlag in VCI_CAN_OBJ:

For single filtering, when the received frame is set to be a extend frame:



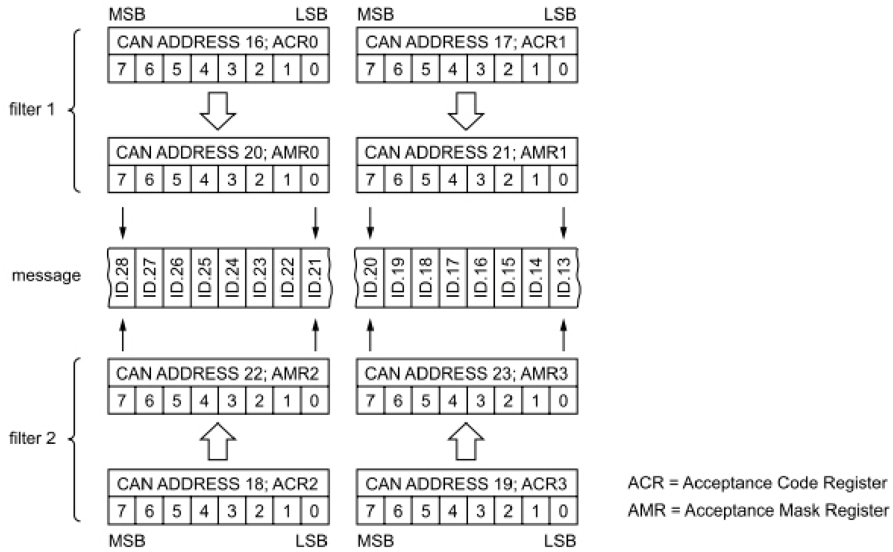
For double filtering mode, when the received frame is set to be a standard frame:

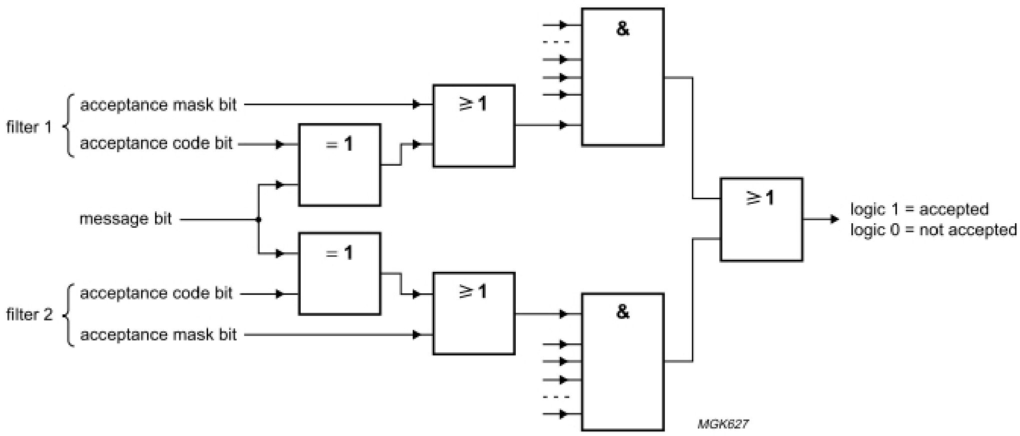




DBX.Y = data byte X, bit Y.

For double filtering mode, when the received frame is set to be a extend frame:





Return value

When return 1, operation is successful; when return 0, operation is failure. (Notes: This function is not exist in CANET, so it will return 1 when calling this function)

2.4.3.3 Example

```
#include "ControlCan.h"

int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
int nReserved = 9600;       // Baudrate
VCI_INIT_CONFIG vic;
DWORD dwRel;

dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("Open device fails!"), _T("Warning"), MB_OK|MB_ICONQUESTION);
```

```
    return FALSE;
}
dwRel = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("Initialize device fails!"), _T("Warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

2.4.4 VCI_ReadBoardInfo

2.4.4.1 Description

This function is used to obtain device information. (For EG20T-CAN, this function is not supported)

```
DWORD __stdcall VCI_ReadBoardInfo(DWORD DevType, DWORD DevIndex, PVCI_BOARD_INFO pInfo);
```

2.4.4.2 Parameter

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

pInfo

This is a VCI_BOARD_INFO structure pointer for device information storage.

Return value

When return 1, operation is successful; when return 0, operation is failure. (Notes: This function is not exist in CANET, so it will return 0 when calling this function, and error code ERR_CMDFAILED will be return as well）

2.4.4.3 Example

```
#include "ControlCan.h"

int nDeviceType = 6;       // CAN232
int nDeviceInd = 0;        // COM1
int nCANInd = 0;
VCI_INIT_CONFIG vic;
VCI_BOARD_INFO vbi;
```

```
DWORD dwRel;

bRel = VCI_ReadBoardInfo(nDeviceType, nDeviceInd, nCANInd, &vbi);
```

2.4.5 VCI_ReadErrInfo

2.4.5.1 Description

This function is used to obtain the last error information (For EG20T-CAN, this function is not supported)

```
DWORD __stdcall VCI_ReadErrInfo(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_ERR_INFO pErrInfo);
```

2.4.5.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel. (Notes: When reading device error, this parameter should be set to -1. For example, if VCI_OpenDevice, VCI_CloseDevice or VCI_ReadBoardInfo function call fails, when calling this function to obtain error code, user should set CANIndex to -1 at first)

pErrInfo

This is the VCI_ERR_INFO structure pointer for error information storage. pErrInfo->ErrCode may be any combination of the following error codes. (For other errors, please refer to error code definition 2.2)

ErrCode	Passive_ErrData	ArLost_ErrData	Description
0x0100	No	No	Device is open
0x0200	No	No	Open device fails
0x0400	No	No	Device is not open
0x0800	No	No	Buffer overflow
0x1000	No	No	This device is not exist
0x2000	No	No	Load dynamic library fails
0x4000	No	No	Execute command fails
0x8000		No	Insufficient memory
0x0001	No	No	CAN controller internal FIFO overflow

ErrCode	Passive_ErrData	ArLost_ErrData	Description
0x0002	No	No	CAN controller error warning
0x0004	Yes, for more information please refer to the table below	No	CAN controller passive error
0x0008	No	Yes, for more information please refer to the table below	CAN controller arbitration lost
0x0010	No	No	CAN controller bus error

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.5.3 Remark

When (PErrInfo->ErrCode&0x0004)==0x0004, CAN controller passive error occurs.

The table below lists the bit interpretation of PErrInfo->Passive_ErrData[0] error code capture bit.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Error code type		Error attribute	Error segment indication				

Bit interpretation of bits ECC.7 and ECC.6

Bit ECC.7	Bit ECC.6	Function
0	0	Bit error
0	1	Format error
1	0	Stuff error
1	1	Other error

2.4.5.4 Error attribute

bit5 = 0: Error occurs on data sending.

= 1: Error occurs on data receiving.

The table below lists the functions of different error segment values.

bit4	bit 3	bit 2	bit 1	bit 0	Function
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 to ID.21
0	0	1	1	0	ID.20 to ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 to ID.13

bit4	bit 3	bit 2	bit 1	bit 0	Function
0	1	1	1	1	ID.12 to ID.5
0	1	1	1	0	ID.4 to ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	acknowledge slot
1	1	0	1	1	acknowledge delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

PErrInfo->Passive_ErrData[1] is for receive error counter

PErrInfo->Passive_ErrData[2] is for send error counter

When (PErrInfo->ErrCode&0x0008)==0x0008, CAN controller arbitration lost error occurs.

For PErrInfo->ArLost_ErrData, bit interpretation of the error code capture register

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
———	———	———	Error segment function value				

Bit interpretation of the arbitration lost capture register is shown in the table below:

Bit					Decimal value	Function
ALC.4	ALC.3	ALC.2	ALC.1	ALC.0		
0	0	0	0	0	0	arbitration lost in bit 1 of identifier
0	0	0	0	1	1	arbitration lost in bit 2 of identifier
0	0	0	1	0	2	arbitration lost in bit 3 of identifier
0	0	0	1	1	3	arbitration lost in bit 4 of identifier
0	0	1	0	0	4	arbitration lost in bit 5 of identifier
0	0	1	0	1	5	arbitration lost in bit 6 of identifier
0	0	1	1	0	6	arbitration lost in bit 7 of identifier
0	0	1	1	1	7	arbitration lost in bit 8 of identifier
0	1	0	0	0	8	arbitration lost in bit 9 of identifier
0	1	0	0	1	9	arbitration lost in bit 10 of identifier
0	1	0	1	0	10	arbitration lost in bit 11 of identifier

Bit					Decimal value	Function
0	1	0	1	1	11	arbitration lost in bit SRTR
0	1	1	0	0	12	arbitration lost in bit IDE
0	1	1	0	1	13	arbitration lost in bit 12 of identifier
0	1	1	1	0	14	arbitration lost in bit 13 of identifier
0	1	1	1	1	15	arbitration lost in bit 14 of identifier
1	0	0	0	0	16	arbitration lost in bit 15 of identifier
1	0	0	0	1	17	arbitration lost in bit 16 of identifier
1	0	0	1	0	18	arbitration lost in bit 17 of identifier
1	0	0	1	1	19	arbitration lost in bit 18 of identifier
1	0	1	0	0	20	arbitration lost in bit 19 of identifier
1	0	1	0	1	21	arbitration lost in bit 20 of identifier
1	0	1	1	0	22	arbitration lost in bit 21 of identifier
1	0	1	1	1	23	arbitration lost in bit 22 of identifier
1	1	0	0	0	24	arbitration lost in bit 23 of identifier
1	1	0	0	1	25	arbitration lost in bit 24 of identifier
1	1	0	1	0	26	arbitration lost in bit 25 of identifier
1	1	0	1	1	27	arbitration lost in bit 26 of identifier
1	1	1	0	0	28	arbitration lost in bit 27 of identifier
1	1	1	0	1	29	arbitration lost in bit 28 of identifier
1	1	1	1	0	30	arbitration lost in bit 29 of identifier
1	1	1	1	1	31	arbitration lost in bit ERTR

2.4.5.5 Example

```
#include "ControlCan.h"

int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
VCI_ERR_INFO vei;
DWORD dwRel;

bRel = VCI_ReadErrInfo(nDeviceType, nDeviceInd, nCANInd, &vei);
```

2.4.6 VCI_ReadCanStatus

2.4.6.1 Description

This function is used to obtain CAN channel state.

```
DWORD __stdcall VCI_ReadCanStatus(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_CAN_STATUS pCANStatus);
```

2.4.6.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

pCANStatus

This is the VCI_CAN_STATUS structure pointer for CAN state storage.

Return value

When return 1, operation is successful; when return 0, operation is failure. (Notes: This function is not exist in CANET, so it will return 0 when calling this function, and error code ERR_CMDFAILED will be return as well)

2.4.6.3 Example

```
#include "ControlCan.h"

int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
VCI_INIT_CONFIG vic;
VCI_CAN_STATUS vcs;
DWORD dwRel;

bRel = VCI_ReadCANStatus(nDeviceType, nDeviceInd, nCANInd, &vcs);
```

2.4.7 VCI_GetReference

2.4.7.1 Description

This function is used to obtain the parameters of device.

```
DWORD __stdcall VCI_GetReference(DWORD DevType, DWORD DevIndex, DWORD CANIndex, DWORD RefType, PVOID pData);
```

2.4.7.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

RefType

This is the type of the reference parameter.

pData

This is the buffer head pointer for parameter storage.

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.7.3 Remark

(1) For PCI5121, CI5110 or SA5420:

RefType	pData	Description
1	Length: 2 bytes pData[0] is the control register address of CAN controller pData[1] is the control register value to be read in the CAN controller	Read the value from the register in CAN chip For example, reading a value from the offset address 09H in a register UCHAR pData[2] = {9,0}; VCI_GetReference(DeviceType,DeviceInd,CANInd,1,pData); For successful calling, the read out value will be stored in pData[1]

(2) For USBCAN1 or USBCAN2:

RefType	pData	Description
1	Length: 1 byte When it is used as an input parameter, pData[0] is the address of the control register of CAN controller to be read; When it is used as an output parameter, pData[0] is the value of the control register of CAN controller	Read the value from a specific control register of CAN controller Take USBCAN1 for example: BYTE val=0; VCI_GetReference(VCI_USBCAN1,0,0,1,(PVOID)&val); If this function is called successfully, register value will be returned in the parameter val

(3) For WITCAN-I:

RefType	pData	Description
1	Length: 1 byte When it is used as an input parameter, pData[0] is the address of the control register of CAN controller to be read; When it is used as an output parameter, pData[0] is the value of the control register of CAN controller	Read the value from a specific control register of CAN controller Take USBCAN1 for example: BYTE val=0; VCI_GetReference(VCI_USBCAN1,0,0,1,(PVOID)&val); If this function is called successfully, register value will be returned in val

20	Length: the data length + 4 bytes;
	When it is used as an input parameter, byte 0 and byte 1 are the address of the data, and byte 2 and byte 3 are the length of the data.
	When it is used as an output parameter, it stores the data to be read
		Read the data from EEPROM
		For example:
		BYTE buf[12];
		WORD addr=0,readlen=8;
		memcpy(buf,&addr,2);// set the address
		memcpy(buf+2,&readlen,2);// set the length to be read
		VCI_GetReference(VCI_USBCAN1,0,0,20,(PVOID)&b uf);
		If this function is called successfully, the data being read will be stored in the byte 0~byte 7 of the parameter buf

(4) For CANET-UDP:

RefType	pData	Description
0	Character string head pointer; it is used to store the IP address read out from the CANETE-E	Read IP address from CANET-E For example: char szip[20]; VCI_GetReference(VCI_CANETE,0,0,0,(PVOID)szip); If this function is called successfully, CANET-E address will be returned in szip
1	Length: 4 bytes; it is used to the CANET-E operation port to be read out	Read CANET-E operation port For example: int port; VCI_GetReference(VCI_CANETE,0,0,1,(PVOID)&port); If this function is called successfully, the CANET-E operation port will be returned in the parameter port

(5) For CANET-TCP:

This device has two operation modes; they are Client mode and Server mode. When the device is working at Client mode, the test tool should be set to Server mode, and vice versa.

RefType	pData	Description
0	Character string head pointer; it is used to store the IP address read out from the CANETE-E	Read IP address from CANET-E For example: char szip[20]; VCI_GetReference(VCI_CANETE,0,0,0,(PVOID)szip); If this function is called successfully, CANET-E address will be returned in szip
1	Length: 4 bytes; it is used to the CANET-E operation port to be read out	Read CANET-E operation port For example: int port; VCI_GetReference(VCI_CANETE,0,0,1,(PVOID)&port); If this function is called successfully, the CANET-E operation port will be returned in the parameter port
2	Length: 4 bytes; it is used to store the TCP server port to be read out (server mode and client mode are both available)	Read or set TCP server port For example: int port; VCI_SetReference(VCI_CANET_TCP,0,0,2,(PVOID)&port); If this function is called successfully, the operation port in the device will be set
4	Length: 4 bytes; it is the operation mode of TCP device	0 is for Client mode, and 1 is for Server mode For example: int iType = 1; VCI_SetReference(VCI_CANET_TCP,0,0,4,(PVOID)&iType); If this function is called successfully, device will be set to work at Server mode
5	Length: 4 bytes; it is used to obtain the number of the connected Client (it is available only when device is under Client mode）	Read or set TCP server port For example: int iCount; VCI_GetReference(VCI_CANET_TCP,0,0,5,(PVOID)&iCount); If this function is called successfully, the CANET-E operation port will be returned in the parameter port

6	It should be used with REMOTE_CLIENT structure to obtain the connection information (it is available only when device is under	When server (local device) is connected with a client, this command can be used to obtain client port information.
		For example:
		REMOTE_CLIENT cli;
		cli.iIndex = 0; //Get the client port 0 in the server
		VCI_GetReference(VCI_CANET_TCP ,0, 0,6,(PVOID)&cli);

RefType	pData	Description
	Client mode）	If this function is called successfully, the Client information will be return in the parameter cli

REMOTE_CLIENT structure

```
typedef struct tagRemoteClient{
        int iIndex;
        DWORD port;
        HANDLE hClient;
        char szip[32];
}REMOTE_CLIENT;
```

(6) For CAN232:

RefType	pData	Description
1	Length: 14 bytes; When it is used as an input parameter, only the 1st byte is valid, it indicates the serial number of the filter to be read, and the valid value of it is 1, 2, 3 or 4 When it is used as an output parameter, please refer to the tables below for the meanings of each byte	Get the specific filter parameter. For example, to read the parameter of the first filter, the setting is as following: BYTE info[14]; info[0]=1; VCI_GetReference(VCI_CAN232,0,0,1,(PVOID)info); If this function is called successfully, the parameter of the first filter in 14 bytes will be returned in info.
2	Length: 1 byte When it is used as an input parameter, pData[0] is the address of the control register of CAN controller to be read; When it is used as an output parameter, pData[0] is the value of the control register of CAN controller	Read the value from a specific control register of CAN controller For example: BYTE val=0; VCI_GetReference(VCI_CAN232,0,0,2,(PVOID)&val); If this function is called successfully, register value will be returned in val

When RefType=1, the meanings of each byte in the returned pData are as following:

pData[0] is for reserved

pData[1] is for the value of CAN controller BTR0;

pData[2] is for the value of CAN controller BTR1;

pData[3] is for reading the operation mode of acceptance filter; its bit interpretation is listed as below:

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
——						MFORMATB	AMODEB

MFORMATB =1; acceptance filter is only available for extend frame information; the standard frame information will be ignored.

=0; acceptance filter is only available for standard frame information; the extend frame information will be ignored.

AMODEB =1; single acceptance filter option is enabled.

=0; double acceptance filter option is enabled.

pData[4] is for reading the state of acceptance filter (enabled or not); its bit interpretation is listed as below:

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
——						BF2EN	BF1EN

BF2EN =1; filter 2 is enabled, and writing shield and code registers are not allowed;

=0; filter 2 is disabled, and shield and code registers can be written.

BF1EN =1; filter 1 is enabled, and writing shield and code registers are not allowed;
=0; filter 1 is disabled, and shield and code registers can be written.

Notes: *For single filter mode, the single filter is related to the filter 1 enable bit. And filter 2 enable bit is ineffective under this mode.*

pData[5] is for reading the priory level of acceptance filter (enabled or not); its bit interpretation is listed as below:

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
——						BF2PRIO	BF1PRIO

BF2PRIO =1; Filter 2 has higher priory level. If there is message come into filer 2, receive interrupt will be generated immediately;

=0； Filter 2 has lower priory level. When FIFO level has passed the receive interrupt level, receive interrupt will be generated;

BF1PRIO =1; Filter 1 has higher priory level. If there is message come into filer 2, receive interrupt will be generated immediately;

=0； Filter 1 has lower priory level. When FIFO level has passed the receive interrupt level, receive interrupt will be generated.

pData[6—9] is for the ACR value of this filter;

pData[a—d] is for the AMR value of this filter.

2.4.7.4 Example

```
#include "ControlCan.h"


int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
BYTE info[14];
```

```
DWORD dwRel;

info[0] = 1;
bRel = VCI_GetReference(nDeviceType, nDeviceInd, nCANInd, 1, (PVOID)info);
```

2.4.8 VCI_SetReference

2.4.8.1 Description

This function is used to set the parameters of the device, to handle different specific operations of the device.

```
DWORD __stdcall VCI_SetReference(DWORD DevType, DWORD DevIndex, DWORD CANIndex, DWORD RefType, PVOID pData);
```

2.4.8.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

RefType

This is the type of the reference parameter.

pData

This is the buffer head pointer for parameter storage.

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.8.3 Remark

VCI_SetReference and VCI_GetReference functions are used for specific operations, such as modifying the baud rate of CAN232 or setting message filter. The meaning of the PVOID type parameter pData is depended on operation.

(1) For PCI5121, PCI5110 or ISA5420:

RefType	pData	Description
1	Length: 2 bytes pData[0] is the control register address of CAN controller pData[1] is the control register value to be written	Write the value into the specific register in CAN chip

(2) For USBCAN1 or USBCAN2:

RefType	pData	Description
1	Length: 2 bytes pData[0] is the control register address of CAN controller pData[1] is the control register value to be written	Write the value into the specific register in CAN chip

(3) For WITCAN-I:

RefType	pData	Description
1	Length: 2 bytes pData[0] is the control register address of CAN controller pData[1] is the control register value to be written	Write the value into the specific register in CAN chip

20	Length: the data length + 4 bytes;
	When it is used as an input parameter, byte 0 and byte 1 are the address of the data, and byte 2 and byte 3 are the length of the data.
	Byte 4 and the remain bytes are used to store the data to be wrritten
		Write data into EEPROM
		For example:
		BYTE buf[12];
		WORD addr=0,writelen=8;
		memcpy(buf,&addr,2);//set the address
		memcpy(buf+2,& writelen,2);//set the length to be written
		memset(buf+4,0,8);//set the data to be written
		VCI_SetReference(VCI_USBCAN1,0,0,20,(PVOID)&buf);

(4) For CANET-UDP:

RefType	pData	Description
0	Character string head pointer; it is used to store the IP address read out from the CANETE-UDP	Set the IP address of CANET-UDP to be operated
1	Length: 4 bytes; it is used	Set the working port of CANET-UDP to be operated

RefType	pData	Description
	to the CANET-UDP operation port to be read out	DWORD port=5000; VCI_SetReference(12,0,0,1,(PVOID)&port);

(5) For CANET-TCP:

RefType	pData	Description
0	Character string head pointer; it is used to store the IP address of the CANETE-TCP	Set the IP address of CANET-TCP to be operated
1	Length: 4 bytes; it is used to store the CANET- TCP operation port (Target)	Set the working port of CANET-TCP to be operated
2	Length: 4 bytes; it is used to store the operation port on local device	Set the operation port of local device
4	Length: 4 bytes; it is used to store the operation mode of TCP device	Set the operation mode of local device, if CANET－TCP is working at Server mode, the local device should be work at Client mode; if CANET-TCP is working at Client mode, the local device should be work at Server mode. 0 is for Client mode, and 1 is for Server mode
5	Length: 4 bytes; it is used to store the number of the Client connected to the local Server	It is read only.
6	Length: REMOTE_CLIENT, it is used to store the connection information	It is read only.

7	It should be used with REMOTE_CLIENT structure to delete a connection (it is available only when device is under Client mode）	For example:
		REMOTE_CLIENT cli;
		cli.iIndex = 0; // delete the connection of Client 0
		VCI_SetReference(VCI_CANET_TCP,0,0,7,(PVOID)&cli);

REMOTE_CLIENT structure

```
    typedef struct tagRemoteClient{
        int iIndex;
        DWORD port;
        HANDLE hClient;
        char szip[32];
}REMOTE_CLIENT;
```

(6) For CAN232:

RefType	pData	Description
1	Length: 1 byte ＝0； 10Kbps ＝1； 20Kbps ＝2； 50Kbps ＝3； 125Kbps ＝4； 250Kbps ＝5； 500Kbps ＝6； 800Kbps ＝7； 1000Kbps	Modify the CAN baud rate, for example, set the CAN baud rate to 10Kbps: BYTE baud=0; VCI_SetReference(VCI_CAN232,0,0,1,(PVOID)&baud);
2	Length: 12 bytes, for more information please refer the table below	Set the filter parameter
3	Length: 1 byte ＝1； 2.4Kbps ＝2； 4.8Kbps ＝3； 9.6Kbps ＝4； 14.4Kbps ＝5； 19.2Kbps ＝6； 28.8Kbps ＝7； 57.6Kbps	Modify serial port baud rate
4	Length: 2 bytes pData[0] is the control register address of CAN controller pData[1] is the control register value to be written	Write the value into the specific register in CAN chip
5	Length: 1 byte, ＝ 0xAA; use timestamp ＝ others; do not use timestamp	Set timestamp

When RefType=2, the meanings of each byte in the returned pData are as following:

pData[0] is used to set the which group of acceptant filter to be used, there are 4 group of filters:

=1: Set the 1st group

=2: Set the 2nd group

=3: Set the 3rd group

=4: Set the 4th group

pData[1] is used to set the operation mode of the acceptant filter; its bit interpretation is listed as below:

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
———						MFORMATB	AMODEB

MFORMATB =1; acceptance filter is only available for extend frame information; the standard frame information will be ignored.

=0; acceptance filter is only available for standard frame information; the extend frame information will be ignored.

AMODEB =1; single acceptance filter option is enabled.

=0; double acceptance filter option is enabled.

pData[2] is for reading the state of acceptance filter (enabled or not); its bit interpretation is listed as below:

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
———						BF2EN	BF1EN

BF2EN =1; filter 2 is enabled, and writing shield and code registers are not allowed;

=0; filter 2 is disabled, and shield and code registers can be written.

BF1EN =1; filter 1 is enabled, and writing shield and code registers are not allowed;

=0; filter 1 is disabled, and shield and code registers can be written.

Notes: *For single filter mode, the single filter is related to the filter 1 enable bit. And filter 2 enable bit is ineffective under this mode.*

pData[3] is for reading the priory level of acceptance filter (enabled or not); its bit interpretation is listed as below:

STATUS.7	STATUS.6	STATUS.5	STATUS.4	STATUS.3	STATUS.2	STATUS.1	STATUS.0
———						BF2PRIO	BF1PRIO

BF2PRIO =1; Filter 2 has higher priory level. If there is message come into filer 2, receive interrupt will be generated immediately;

=0; Filter 2 has lower priory level. When FIFO level has passed the receive interrupt level, receive interrupt will be generated;

BF1PRIO =1; Filter 1 has higher priory level. If there is message come into filer 2, receive interrupt will be generated immediately;

=0; Filter 1 has lower priory level. When FIFO level has passed the receive interrupt level, receive interrupt will be generated.

pData[4---7] are corresponding to ACR0---ACR3 of SJA1000 to be set;

pData[8---b] are corresponding to AMR0---AMR3 of SJA1000 to be set.

(7) For PCI-5010-U/PCI-5020-U/USBCAN-E-U/ USBCAN-2E-U:

RefType	pData	Description
0	Pointer to DWORD type data , this DWORD variable value is the value written into the baud rate register BTR	Set baud rate. The calculation formulate is as following: BPS= Peripheral bus clock /(BRP+1)*(TESG1+TESG2+3)

RefType	pData	Description
	The relationship between some standard baud rate and BTR setting: 0x060003 ： 1000Kbps 0x060004 ： 800Kbps 0x060007 ： 500Kbps 0x1C0008 ： 250Kbps 0x1C0011 ： 125Kbps 0x160023 ： 100Kbps 0x1C002C ： 50Kbps 0x1600B3 ： 20Kbps 0x1C00E0 ： 10Kbps 0x1C01C1 ： 5Kbps	Where: Peripheral bus clock: 36000Kbps BRP: 0~9bit of BTR TESG1: 16~19bit of BTR TESG2: 20~22bit of BTR The condition below is recommended to follow when setting TESG1 or TESG2: 80% <= $\frac{TESG1+2}{TESG1+TESG2+3}$ <= 90% For other values, SBCAN-E-U/USBCAN-2E-U may work improperly. It is recommended to set the remained bits of 32-bit register BTR to 0. (Notice: The maximum baud rate of CAN network should not exceed 1000Kbps, so the baud rate setting should be lower than this value, otherwise settings may fail.) For USBCAN-E-U/ USBCAN-2E-U, it is necessary to call this function to set the baud rate before calling VCI_InitCan
1	Pointer to VCI_FILTER_RECORD structure	Stuff the filter table of CAN filer (This function is called for each time adding a new record)
2	NULL	Start filer according to the settings in the filter table
3	Pointer to DWORD type data , this DWORD variable value is the transmission overtime, unit: ms	Set the transmission overtime, unit: ms. If the configuration is set without calling this function, then the transmission overtime is 4000ms by default. It is recommended the transmission over should be less than 1500ms, or else CAM bus communication may occur error.

2.4.8.4 Example

```
#include "ControlCan.h"

int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
BYTE baud;
DWORD dwRel;

baud = 0;
bRel = VCI_SetReference(nDeviceType, nDeviceInd, nCANInd, 1, (PVOID)baud);
```

2.4.9 VCI_GetReceiveNum

2.4.9.1 Description

This function is used to obtain the quantity of frames that has been received but not read out.

```
ULONG __stdcall VCI_GetReceiveNum(DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

Return value

Return the frame number that is not read.

2.4.9.2 Example

```
#include "ControlCan.h"

int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
DWORD dwRel;

bRel = VCI_GetReceiveNum(nDeviceType, nDeviceInd, nCANInd);
```

2.4.10 VCI_ClearBuffer

2.4.10.1 Description

This function is used to clear the specific buffer.

```
DWORD __stdcall VCI_ClearBuffer(DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.10.2 Example

```
#include "ControlCan.h"

int nDeviceType = 6;       // CAN232
int nDeviceInd = 0;        // COM1
int nCANInd = 0;
DWORD dwRel;

bRel = VCI_ClearBuffer(nDeviceType, nDeviceInd, nCANInd);
```

2.4.11 VCI_StartCAN

2.4.11.1 Description

This function is used to start CAN.

```
DWORD __stdcall VCI_StartCAN(DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

Return value

When return 1, operation is successful; when return 0, operation is failure.

2.4.11.2 Example

```
#include "ControlCan.h"

int nDeviceType = 6;       // CAN232
int nDeviceInd = 0;        // COM1
int nCANInd = 0;
int nReserved = 9600;      // Baudrate
```

```
VCI_INIT_CONFIG vic;
DWORD dwRel;

dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
	MessageBox(_T("Open device fails!"), _T("Warning"), MB_OK|MB_ICONQUESTION);
	return FALSE;
}
dwRel = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
	VCI_CloseDevice(nDeviceType, nDeviceInd);
	MessageBox(_T("Initialize device fail!"), _T("Warning"), MB_OK|MB_ICONQUESTION);
	return FALSE;
}
dwRel = VCI_StartCAN(nDeviceType, nDeviceInd, nCANInd);
if (dwRel == STATUS_ERR)
{
	VCI_CloseDevice(nDeviceType, nDeviceInd);
	MessageBox(_T("Start device fail!"), _T("Warning"), MB_OK|MB_ICONQUESTION);
	return FALSE;
}
```

2.4.12 VCI_ResetCAN

2.4.12.1 Description

This function is used to reset CAN.

```
DWORD __stdcall VCI_ResetCAN(DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

Return value

When return 1, operation is successful; when return 0, operation is failure.

(Notes: For CANET-TCP, in the case of network disconnection, user should recall VCI_StartCAN function to use CANET-TCP connect to the network.)

2.4.12.2 Example

```
#include "ControlCan.h"

int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
DWORD dwRel;

bRel = VCI_ResetCAN(nDeviceType, nDeviceInd, nCANInd);
```

2.4.13 VCI_Transmit

2.4.13.1 Description

Return the actual transmitted frame number.

```
ULONG __stdcall VCI_Transmit(DWORD DevType, DWORD DevIndex, DWORD CANIndex,
PVCI_CAN_OBJ pSend, ULONG Len);
```

2.4.13.2 Parameters

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

pSend

This is the head pointer to the data frame array to be sent.

Len

This is the length of the data frame array to be sent.

Return value

Return the actual transmitted frame number.

2.4.13.3 Example

```
#include "ControlCan.h"
#include <string.h>
```

```
int nDeviceType = 6;        // CAN232
int nDeviceInd = 0;         // COM1
int nCANInd = 0;
DWORD dwRel;
VCI_CAN_OBJ vco;

ZeroMemory(&vco, sizeof (VCI_CAN_OBJ));
vco.ID = 0x00000000;
vco.SendType = 0;
vco.RemoteFlag = 0;
vco.ExternFlag = 0;
vco.DataLen = 8;
lRet = VCI_Transmit(nDeviceType, nDeviceInd, nCANInd, &vco, i);
```

2.4.14 VCI_Receive

2.4.14.1 Description

This function is used to read data from the specific device.

```
ULONG __stdcall VCI_Receive(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_CAN_OBJ
pReceive, ULONG Len, INT WaitTime=－1);
```

DevType

This is the type of device.

DevIndex

This is the index number of device. For example, when there is only one PCI5121, the index number is 0. For two devices, the index number can be 0 or 1. (Notice: For CAN232, 0 is for opening COM1, and 1 is for opening COM2.)

CANIndex

This is the index number of CAN channel.

pReceive

This is used to receive the head pointer of the data frame array.

Len

This is the length of the data frame array to be received.

WaitTime

This is the wait overtime (unit: ms);

Return value

Return the frame number read actually. If the return value is 0xFFFFFFFF, it means reading data fails and error occurs. In this case, you should call VCI_ReadErrInfo function to obtain the error code.

2.4.14.2 Example

```
#include "ControlCan.h"
#include <string.h>

int nDeviceType = 6;       // CAN232
int nDeviceInd = 0;        // COM1
int nCANInd = 0;
DWORD dwRel;
VCI_CAN_OBJ vco[100];

lRet = VCI_Receive(nDeviceType, nDeviceInd, nCANInd, vco, 100, 400);
```

2.5 Interface library function usage

Firstly, put the library function file into the relevant directory. There are three files: ControlCAN.h, ControlCAN.lib and ControlCAN.dll and one folder kerneldlls.

2.5.1 Calling dynamic library with VC

Contain ControlCAN.h head file in the file with the extension .CPP.

For example: #include “ControlCAN.h”

Connect with the ControlCAN.lib file by setting the connector in the project.

For example: under VC7, enter the configuration property→Connector→Input→Additional item in project property page, to add ControlCAN.lib.

2.5.2 Calling dynamic library with VB

The library can be called by declaring with the method below:

Syntax:

[Public | Private] Declare Function name Lib "libname" [Alias "aliasname"] [([arglist])] [As type]

The syntax of the Declare statement is as following:

Public (optional)

It is used to declare the function that can be use in all the procedures for all modules.

Private (optional)

It is used to declare the function that can only be use in module under this declaration.

Name (optional)

It can be any legal function name. The dynamic link library entry (entry points) is case sensitive.

Libname (optional)

It contains the function link library name or code resource name declared.

Alias (optional)

It means the called function has another name in the dynamic link library (DLL). When the external function name is the same with certain function name, this parameter can be used. When the function in the dynamic link library has the same name with the name of public variable, constant or any procedure, you can use Alias too. If a character in this DLL function has confliction with the naming convention of the dynamic link library, Alias will be used.

Aliasname (optional)

It is dynamic link library. If the first character is not "#", then aliasname will be the name of the entry point in this function. If the first character is a "#", the subsequent character should specific the sequence number of the function entry.

Arglist (optional)

It is the variable table for passing parameter when calling this function.

Type (optional)

It is the data type of the Function return value. It can be Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (not supported in current), Date, String (only support varchar) or Variant, user defined type or object type.

The syntax of the parameter arglist is as following:

[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]

Partial description:

Optional (optional)

It means the parameter is not necessary. When using this option, the subsequent parameters in arglist are optional too, and all of them should use the key work Optional for declaration. However, if ParamArray is used, no parameter can be Optional.

ByVal (optional)

It means the parameter is passed by value.

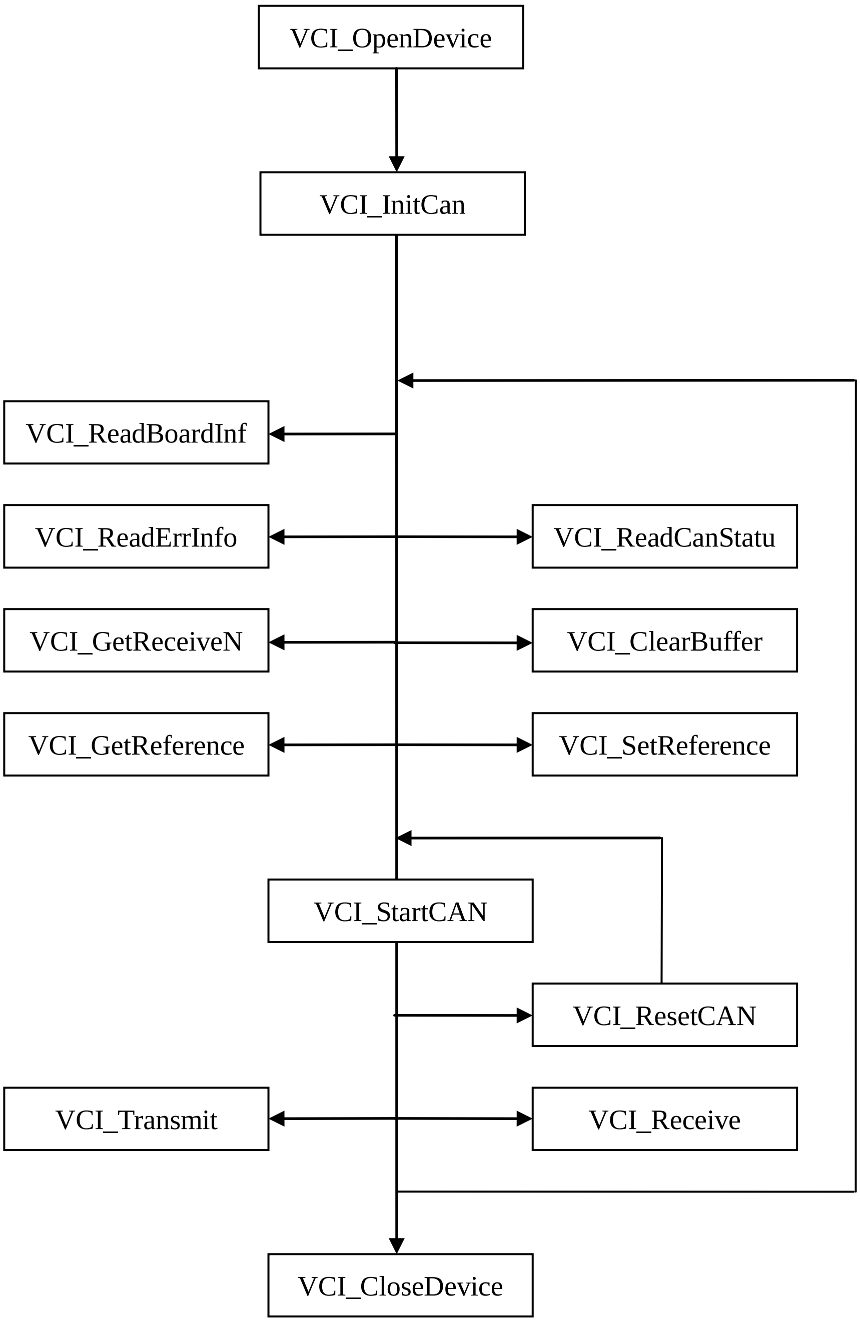
ByRef (optional)

It means the parameter is passed by address.

For example:

Public Declare Function VCI_OpenDevice Lib "ControlCAN" (ByVal devicetype As Long, ByVal deviceind As Long, ByVal reserved As Long) As Long

2.6 Interface library function usage flow



Chapter 3: Dynamic Library Usage in Linux

3.1 Driver installation

All the drivers have been tested under Linux 2.4.20-8.

3.1.1 USBCAN driver installation

Copy the usbcan.o file from the directory driver to /lib/modules/(*)/kernel/drivers/usb, then the driver installation is completed, in which (*) is vary with the Linux version. For example, if Linux version is 2.4.20-8, the name of this directory will be "2.4.20-8" too, that is the same with the Linux core version number).

3.1.2 PCI5121 driver installation

Copy the pci51xx.o file from the directory driver to /lib/modules/(*)/kernel/drivers/char, then the driver installation is completed, in which (*) is vary with the Linux version. For example, if Linux version is 2.4.20-8, the name of this directory will be "2.4.20-8" too, that is the same with the Linux core version number).

3.2 Dynamic library installation

Copy libcontrolcan.so file in the dll folder and kerneldlls folder to directory /lib, and then run the command ldconfig /lib to finish the dynamic library installation.

3.3 Call and compile dynamic library

It is very easy to call dynamic library. Copy controlcan.h file in the dll folder to the current project directory, and then contain controlcan.h file to your source code by using #include "controlcan.h". Now the functions in the dynamic library can be used.

For GCC compiling, you just need to add the option –lcontrolcan.

For example:

gcc –lcontrolcan –g –o test test.c

Chapter 4: Rights & Statements

This document only provides the information about the product of Guangzhou ZHIYUAN Electronics Stock Co., Ltd. (ZHIYUAN Electronics) This document will not (explicitly, implicitly or otherwise) license any intellectual property rights. ZHIYUAN Electronics shall not be liable for any responsibilities other than those specified explicitly in its sales terms and conditions. To the maximum extent permitted by law, ZHIYUAN Electronics shall in no event be liable for or guarantee (explicitly, implicitly or otherwise) the correctness, completeness, accuracy, durability, assurance of features, reliability, workability, merchantability, quality, fitness for a particular purpose, achievement of results, non-infringement of proprietary rights or the absence of any deficiencies of any products sale by ZHIYUAN. The products of ZHIYUAN Electronics are not designed for medical usage, life saving or life maintenance purpose. ZHIYUAN Electronics reserves the right to change the product standard and specifications without prior notice.

Company name:	Guangzhou ZLGMCU Technology Co., Ltd.
Address:	Floor 2, No.7 Building, Huangzhou Industrial Estate Guangzhou, CHINA
Post code:	510660
Website:	www.zlgmcu.com
Sales:	+86-20-2264-4249
Tech. Support:	+86-20-2264-4361
Facsimile:	+86-20-3860-1859
Sales Email:	80c51mcu@zlgmcu.com
Tech. Sup. Email:	printer@zlgmcu.com