GY8507 USB-CAN Adapter

Version：2.0

Contents

Chapter 1 Overview

1.1 General introduction

GY8507 USB-CAN Adapter is a kind of converter for bi-directional communication between with CAN bus interface and USB2.0 interface. By using this adapter, Computer can connect to a standard CAN bus net from USB port.

As a standard CAN nodes, GY8507 USB-CAN Adapter is a powerful tool for CAN Bus product develop, CAN Bus equipment testing, CAN Bus net monitoring and data analysis. Beside that, GY8507 is small in size and easy to be installed. And it can be wildly used for constructing a field bus laboratories, industrial control, intelligent buildings, cars and electronics, etc.

GY8507 USB-CAN Adapter can use CANTools software provided by Glinker to directly finish CAN Bus configuration, message sending, and receiving. The users can also use DLL dynamic library provided by Glinker company to develop application software of CAN system. And Glinker have provided the VC/VB demo source code for user's reference. When you use GY8507 CAN Adapter to make second software development, you don't need to understand complex USB interface communication protocol.

In order to achieve optoelectronic Isolation, GY8507 uses independent power for CAN bus circuit. So the device has a strong anti-jamming capability, and greatly reliability in the harsh environment.

1.2 List of Performance and Specifications

- Protocol conversion of USB and CAN bus;
- USB interface supports USB2.0, compatible with USB1.1;
- Support agreement of CAN2.0A, CAN2.0B, standard frame and extended frame;
- Support bi-directional transmission, sending and receiving of CAN data;
- Support data frame and remote frame format;
- Support Self-Reception test;
- The CAN bus baud rate is between 5Kbps and 1Mbps, can be configured by CANTools.
- Photoelectric isolation for CAN bus interface and DC-DC power–supplied isolation;
- The maximum flow is 3000 Frames/s;
- The Receive buffer is 200 CAN Messages(2600 bytes);
- Directly powered by USB port, no need for external power supply;
- DC-DC Insulation voltage isolation module: 1000 Vrms;
- Power consumption: 400 mw.
- Operating temperature: -20 to +85 ℃;
- Shell Size: 110*70*23 mm.

1.3 Typical Applications

- Communication to CAN bus device with PC or Notebook;

- CAN Bus - USB gateway;
- USB interface to the CAN bus;
- Extending CAN Bus network communications distance;
- CAN network monitor for industrial field.

1.4 List of Products

1) GY8507 USB-CAN adapter;
2) USB cable;
3) CD-ROM.(datasheet, CANTools software, develop files, VC/VB demo source code)

1.5 Glinker PC-CAN adapter

Part Name	Interface	CAN Channel	MAX Rx Frames Per Second	Rx-Buffer	CAN baud rate
GY8502	RS232	1	300	100 Messages	5-1000kbps
GY8503	RS485	1	300	100 Messages	5-1000kbps
GY8505	Ethernet UDP	1	1000	200 Messages	5-1000kbps
GY8506	Ethernet UDP	2	1000	120 Messages	5-1000kbps
GY8507	**USB**	**1**	**3000**	**200 Messages**	**5-1000kbps**
GY8508	USB	2	3000	240 **Messages**	5-1000kbps
GY7841	PCI	1	3000	3000 Messages	5-1000kbps
GY7842	PCI	2	3000	3000 Messages	5-1000kbps

Technical Support Mail: yyd315@163.com
Web site: www.glinker.cn

Chapter 2 Appearance and Interface

2.1 Hardware Interface

USB port is for Computer connecting.

10pin terminal is CAN bus interface

Red LED shows power status;

Green LED for Transmission status. When the CAN Bus is active (Successful sending or receiving), the LED flashes.

The layout of interface is as follows：

Picture 1 appearance of GY8507

2.2 Pin description

Name	Discription
RES+	Terminal resistor R+. If you need 120 ohm, please connect R+ and R- with wire.
RES-	Terminal resistor R-
CANL	CAN bus Signal L
CANH	CAN bus Signal H

GY8507 work way：

1) Send:

When GY8507 receives a data package form USB port, it will construct the data to be a CAN message frame immediately, then send it to CAN bus port.

2) Receive:

When GY8507 receives CAN data frame for CAN bus port, it will save the message to

it's R-buffer region. And when the computer asks for the R-buffer, it will send back all messages of the R-buffer to USB port.

2.3 Default Value

CAN bus baud rate：1Mbps;

Acceptance Filter: No filter. It can receive CAN message regardless of ID;

Terminal resistor：If you want use 120Ω terminal resistor, you just use a wire to connect R- with R+.

Chapter 3 Introduction of CANTools

3.1 Driver Installation

When you connect device with USB port of pc or notebook first, the Windows 98/2000/xp/vista/win7/win10 will tell you to install the USB driver according to the wizard. You can find the driver in CD Rom of Glinker.

You can also setup it in "My computer->Device Manager" by yourself.

Special attention: Select the directory Driver V3.3, do not choose x86 or x64, otherwise the installation may not be successful. When the installation is complete, the device manager will indicate that there is a "USB-CAN Device".

Note: In win7/win8/win10 operating system, the system may automatically help you install the driver, which is displayed as USB EXPRESS DEVICE in Device Manager. Please uninstall the driver first. When uninstalling, please check "Uninstall the driver file at the same time". Then reinstall the driver in the CD.

In win10 systems, sometimes the system will prompt for a hash error. Then you need to set the stop-disable digital signature driver in win8-win10. Method Reference:

https://www.maketecheasier.com/install-unsigned-drivers-windows10/

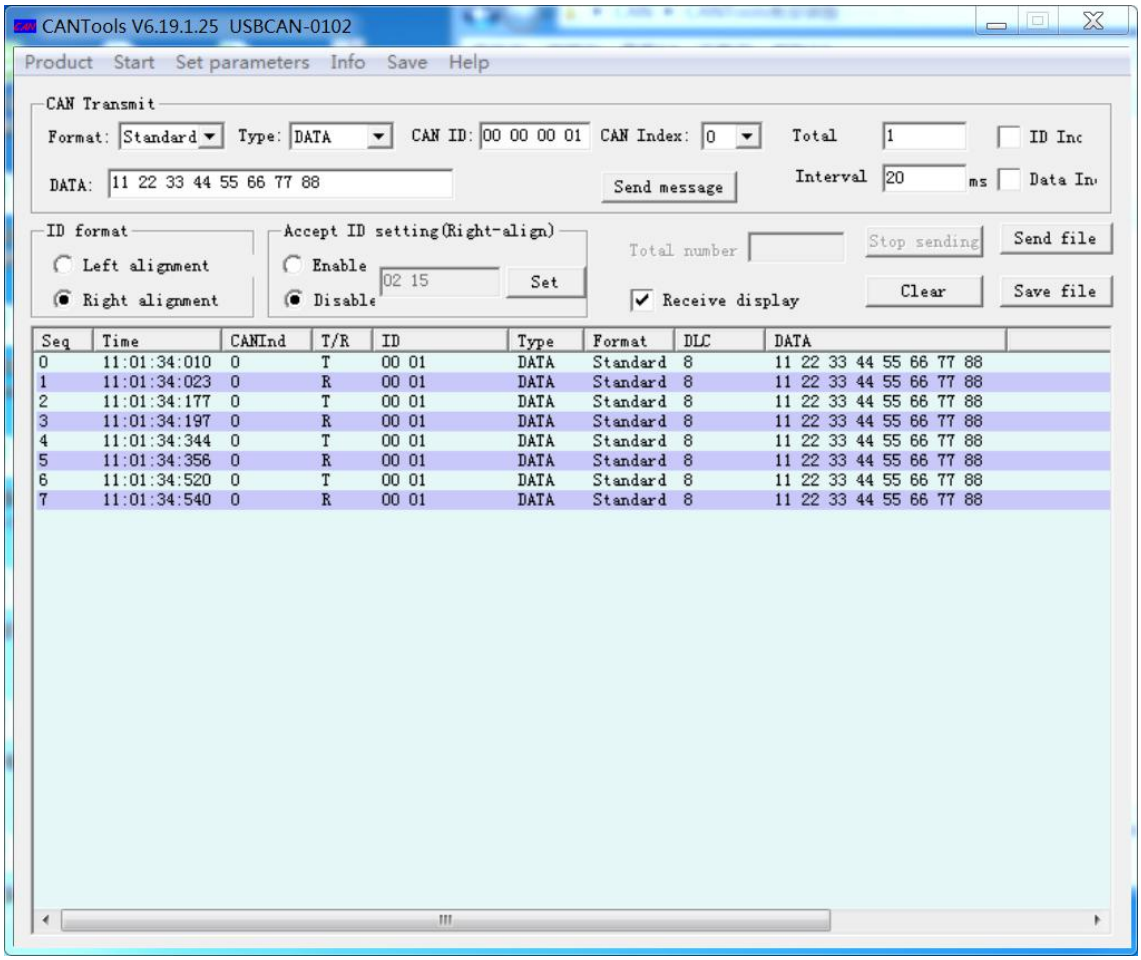
3.2 CANTools software

Run CANTools_setup.exe, then you will setup the software for CAN Bus test.

Connect the device to USB port of your computer.

Run CANTools.exe

Below is the introduction for CANTools.



Picture 2 CANTools

1) Menu “Product”

Select the device type, “ GY8507 USB-CAN” .

2) Menu “Start”, for Open and start the device.

“Start->Open device” will connect USB port to the device, and start CAN function.

“Start->Close device” will close USB connection.

3) Menu “Set parameters->CAN parameters”, you can configure the CAN interface parameters for the device.

“Baud Rate”: You can set the baud rate for CAN communication. It’s from 5Kbps to 1000Kbps.

“Work Mode” is for Normal mode and Self-Reception mode selection.

“Filter” is for receive message control.

For “Acc code” and “Acc mask”, it include 32 bit (4 bytes). The MSB of ID filter is at MSB of the first bytes.

For standard CAN filter, ID10 is at the MSB of first bytes.

Example: When ACR= 00 20 00 00, AMR= 00 1F FF FF, the device will only receive the message for ID=0x0001(direct ID of right alignment).

For extended CAN filer, ID29 is at the MSB of first bytes.

Example: When ACR= 00 00 00 F8, AMR= 00 00 00 07, the device will only receive the message for ID=0x0000001F(direct ID of right alignment).

Note: when Mask code value is “FF FF FF FF”, it means no filter. The device can receive all CAN message in CAN bus network regardless ID.

The setting format is according SJA1000 register. You can refer the datasheet of Philip SJA1000 CAN controller.

4) Menu “Set parameters->Recovery of factory parameters”

When you click it, the device will be resumed to default mode.

The above configuration will be saved in the EEPROM of the device, and will be active when it works next time. (Note: the work mode will be recovery to normal mode every time when you power the device. Only the setting will not be saved.)

5) Menu “Info->Current parameters”

When you click it, you can get the current configuration of CAN interface.

6) Menu “Info->Device information”

When you click it, you can get the GY8507 device information.

7) Main View ->Send message

You can select the message frame type, fill ID value and fill the Data values. When clicks the “Send”, a CAN message will be sent to CAN interface.

8) Main View->CAN Receive

If you want receive message, the receive function will be enable.

9) Main View->ID format

SJA1000 format: It means left alignment. The MSB of ID is at MSB of first byte in ID value, when you send or receive display. Example: when you want to send a standard CAN message with ID=2, the ID area will be filled with 00 40 00 00. And for extended message, you should fill 00 00 00 10.

Direct ID format: It means right alignment. The LSB of ID is at LSB of last byte in ID value, when you send or receive display. Example: when you want to send a standard or extended CAN message with ID=2, the ID area will be filled with 00 00 00 02.

10) Main View->Accetp ID setting

It is an intelligent and easy way for ID filter setting. If you only want receive the messages with ID 0x0215(direct ID value), you can select “Filter on” and fill the edit 02 15.

And if you select “Filter off”, it will be no filter, can receive all messages regardless ID.

11) Main View->Save File

It will help you save the messages in the list view to an Excel file.

12) Main View->Send File

It can send the all messages in a file which format is same with saved file.You can use pre-saved files.

3.3 Self-Reception Test

After getting GY8507 USB-CAN adapter, users can test the product by themselves：

1、 Connect R+ with R-.

Note: In self-reception mode, you need the terminal resistor.

2、 Connect the USB port of Computer to GY8507, the Red-LED and CAN-LED

will be light on.

3、 Then install driver and CANTools software, according to the manual. It can be found in CD Rom.

4、 Run CANTools.

5、 Choose device type in software menu: “Product->USB-CAN”.

Start the device. “Start->Open device”.

Change the patterns of work mode to “Self-Reception”:“Setting->CAN Setting”

Turn back to the main view, use “√”to mark “receive display”.

Send message, and see whether you can receive back the same message you just sent out. At the same time, you can also observe that green LED will flicker.

If you can’t operate successfully, please check procedures carefully. And if it still does not work, please contact us to get help.

Chapter 4 Software development

If users intend to make a program for their own application, they need to read following descriptions very carefully, and refer the demo source code.

Develop files include VCI_CAN.lib, VCI_CAN.DLL, SiUsbxp.DLL

If you use VC, you can use ControlCAN.h

If you use VB, you can use VCI_CAN.bas.

Also you can use PB, Delphi, C++Builder, C#, Labview to call the interface function.

4.1 Data Structure of Library

4.4.1 Device type value.

DEV_CAN232B 1
DEV_USBCAN 2
DEV_USBCAN200 3
DEV_NETCAN100 4
DEV_NETCAN200 5
DEV_PCICAN2 6

4.4.2 Address of Configuration parameters

REFTYPE_MODE 0
REFTYPE_FILTER 1
REFTYPE_ACR0 2
REFTYPE_ACR1 3

REFTYPE_ACR2 4
REFTYPE_ACR3 5
REFTYPE_AMR0 6
REFTYPE_AMR1 7
REFTYPE_AMR2 8
REFTYPE_AMR3 9
REFTYPE_kCANBAUD 10
REFTYPE_TIMING0 11
REFTYPE_TIMING1 12
REFTYPE_CANRX_EN 13
REFTYPE_UARTBAUD 14
REFTYPE_ALL 15
REFTYPE_DEVICE_IP0 16
REFTYPE_DEVICE_IP1 17
REFTYPE_DEVICE_IP2 18
REFTYPE_DEVICE_IP3 19
REFTYPE_HOST_IP0 20
REFTYPE_HOST_IP1 21
REFTYPE_HOST_IP2 22
REFTYPE_HOST_IP3 23

4.1.3 VCI_BOARD_INFO

The structure contains the information of CY850X series interface adapter, and there are 32 bytes totally. The structure will be 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;
BYTE    reserved;
CHAR    str_Serial_Num[8];
CHAR    str_hw_Type[16];
CHAR str_GYUsb_Serial[4][4];
} VCI_BOARD_INFO, *PVCI_BOARD_INFO;
```

Members:

hw_Version	hardware version code，16 hexadecimal。Eg:0x0100 means V1.00 。
fw_Version	firmware version code，16 hexadecimal。
dr_Version	driver software version code，16 hexadecimal。
in_Version	interface library version code，16 hexadecimal。
irq_Num	not used, reserved
can_Num	the number of CAN channels.
str_Serial_Num	the board code

str_hw_Type	hardware type information
str_GYUsb_Serial	USB-CAN number，it can support 4 USB device in one computer

4.1.4 VCI_CAN_OBJ

it is used to tansmit CAN information frame in the VCI_Transmit and VCI_Receive functions .

```
typedef struct _VCI_CAN_OBJ {
BYTE ID[4];
UINT TimeStamp;
BYTE TimeFlag;
BYTE SendType;
BYTE RemoteFlag;
BYTE ExternFlag;
BYTE DataLen;
BYTE Data[8];
BYTE Reserved[3];} VCI_CAN_OBJ, *PVCI_CAN_OBJ;
```

Menbers:

ID	packet ID, 4 bytes
TimeStamp	not used, reserved
TimeFlag	not used, reserved
SendType	not used, reserved
RemoteFlag	remote frame or not
ExternFlag	extended frame or not
DataLen	data length(<=8)，it is the length of data
Data	data of packet
Reserved	system reservation

4.1.5 VCI_CAN_STATUS

It’s defined the state information of CAN controller. The structure will be filled in 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;
```

Members

ErrInterrupt	interruption records, removing read operation
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 error receiver register
regTECounter	CAN controller sending error register
Reserved	

4.1.6 VCI_INIT_CONFIG

It's used to initiate CAN configuration. The structure will be filled in VCI_InitCan function.

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

Menbers:

AccCode	acceptance code for filter
AccMask	mask code for filter
Reserved	not used
Filter	filter mode ,single or double
Timing0	timer0 (BTR0)
Timing1	timer1 (BTR1）
Mode	workmode 0:normal work, 1:self reception

Timing0 and remark Timing1 is uesd for setting CAN baud rate. The following table is about setting of 15 kinds of common baud rates.

Index No. kCanBaud	CAN Baud Rate	Timing0	Timing1
0			
1	5Kbps	0xBF	0xFF
2	10Kbps	0x31	0x1C
3	20Kbps	0x18	0x1C
4	40Kbps	0x87	0xFF
5	50Kbps	0x09	0x1C
6	80Kbps	0x83	0Xff
7	100Kbps	0x04	0x1C
8	125Kbps	0x03	0x1C
9	200Kbps	0x81	0xFA
10	250Kbps	0x01	0x1C

11	400Kbps	0x80	0xFA
12	500Kbps	0x00	0x1C
13	666Kbps	0x80	0xB6
14	800Kbps	0x00	0x16
15	1000Kbps	0x00	0x14

4.2 Function description

Parameters:

DevType: device type value

DevIndex: device index, when it is CAN232,0 means COM1 is to be opened，1 means COM2 is to be opened.

When it is NET-CAN, DevIndex represents IP OF NET-CAN device. Please pay attention to order, and low numbers are in the MSB. example: 0x0A00A8C0 represents 192.168.0.10

When equipment is USB-CAN, DevIndex represents USB-CAN device channel. Usually users fill it with 0.

CANIndex: CAN channel. If the device has only one CAN channel, it will be 0.

Return value:

0: fail

1: successful

-1: device not open or error.

4.2.1 VCI_OpenDevice

The function is used to open the device.

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

Reserved: when equipment is CAN232,this parameter represents baud rate of RS232. 9600,19200,38400,57600 are valid parameters. When the device is NET-CAN,USB-CAN ,you can fill it with 0

Example:

```
#include "ControlCan.h"
if(VCI_OpenDevice(DEV_USBCAN, 0,0)!=1)
{
MessageBox("open fail");
return;
}
```

4.2.2 VCI_CloseDevice

The function is uesd for closing equipment

DWORD __stdcall VCI_CloseDevice(DWORD DevType, DWORD DevIndex);

Example:

```
#include "ControlCan.h"
if(VCI_CloseDevice(DEV_USBCAN,0)!=1)
{
    MessageBox("close fail");
    return;
}
```

4.2.3 VCI_InitCan

The function is uesd for initiate designated CAN.

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

CANIndex CAN channel

pInitConfig Init parameters structure.

AccCode	Function
pInitConfig->AccCode	AccCode corresponds to four registers in SJA1000 mode: the MSB of ID value is at the MSB of ACR0
pInitConfig->AccMask	
pInitConfig->Reserved	reserved
pInitConfig->Filter	Filter mode， 0- single filter， 1-dual filter
pInitConfig->kCanBaud	CAN baud rate index
pInitConfig->Timing0	Baud rate timer 0
pInitConfig->Timing1	Baud rate timer 1
pInitConfig->Mode	0 normal mode, 1 self-reception

Example:

```
VCI_INIT_CONFIG InitInfo[1];
InitInfo->kCanBaud=15;
InitInfo->Timing0=0x00;
InitInfo->Timing1=0x14;
InitInfo->Filter=0;
InitInfo->AccCode=0x80000008;
InitInfo->AccMask=0xFFFFFFFF;
InitInfo->Mode=0;
InitInfo->CanRx_IER=1;
if(VCI_InitCAN(m_DevType,m_DevIndex, 0,InitInfo)!=1) //can-0
{
    MessageBox("Init-CAN failed!");
    return;
}
```

4.2.4 VCI_ReadBoardInfo

The function is used for getting device information.

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

pInfo: VCI_BOARD_INFO structure for device information

Example

```
VCI_BOARD_INFO pData[1];
if(VCI_ReadBoardInfo(m_DevIndex,m_DevIndex,pData)!=1)
{
    MessageBox("reading failure");
    return;
}
```

4.2.5 VCI_ReadCanStatus

The function is uesd for getting CAN state.

DWORD __stdcallVCI_ReadCanStatus(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_CAN_STATUS pCANStatus);

Example

```
#include "ControlCan.h"
VCI_CAN_STATUS vcs;
VCI_ReadCANStatus(nDeviceType, nDeviceInd, nCANInd, &vcs);
```

4.2.6 VCI_GetReference

The function is used for getting all configuration, and the index is Address of Configuration parameters table.

DWORD __stdcall VCI_GetReference(DWORD DeviceType, DWORD DeviceInd, DWORD CANInd, DWORD Reserved, BYTE *pData);

Example:

```
BYTE pData[32];
if(VCI_GetReference(m_DevType,m_DevIndex,0,REFTYPE_ALL,pData)!=1)
{
    MessageBox("fail！ ");
    return;
}
```

4.2.7 VCI_SetReference

The function is used for setting relevant parameters of device.

DWORD __stdcall VCI_SetReference(DWORD DeviceType, DWORD DeviceInd, DWORD CANInd, DWORD RefType, BYTE *pData);

RefType: parameters type list here, and it also is the address of configuration parameters table.

pData is the data buffer address first pointer of parameters.

Parameters that can be set and REFTYPE code:

REFTYPE_kCANBAUD	buffer length 3
REFTYPE_MODE	buffer length 1
REFTYPE_FILTER	buffer length 1
REFTYPE_ACR0	buffer length 4
REFTYPE_AMR0	buffer length 4

REFTYPE_CANRX_EN buffer length 1
REFTYPE_UARTBAUD buffer length 1 //for CAN232B
REFTYPE_DEVICE_IP0 buffer length 4
REFTYPE_HOST_IP0 buffer length 4
REFTYPE_ALL buffer length >=15

Example:

```
BYTE pData[15];
pData[0]=15;
pData[1]=0x00;
pData[2]=0x14;
if(VCI_SetReference(DEV_USBCAN,0,0,REFTYPE_kCANBAUD,pData)!=1)
{
MessageBox("fail");
return;
}
```

4.2.8 VCI_ResumeConfig

The function will make the device resume the parameters to factory default value.

DWORD __stdcall VCI_ResumeConfig(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd);

Example:

```
if(VCI_ResumeConfig(DEV_CAN232B, 0, 0)!=1)
{
    MessageBox("fail to restore to factory settings");
    return;
}
```

4.2.9 VCI_StartCAN

The function is will start the CAN controller, and CAN interruption receiving is enabled.

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

Example:

```
if(VCI_OpenDevice(DEV_CAN232B,0,57600)!=1)
{
    MessageBox("fail");
    return;
}
if(VCI_StartCAN(DEV_CAN232B,0, 0)!=1)
{
    MessageBox("starting CAN failed");
    return;
}
```

4.2.10 VCI_ResetCAN

The function will make the CAN controller reset with the current parameters.

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

4.2.11 VCI_Transmit

It't for CAN message send.

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

Example:

```
VCI_CAN_OBJ sendbuf[1];
sendbuf->ExternFlag=0;
sendbuf->DataLen=8;
sendbuf->RemoteFlag=0;
sendbuf->ID[0]=0x00;// SJA1000 mode
sendbuf->ID[1]=0x60;//    ID=3
sendbuf->ID[2]=0x00;
sendbuf->ID[3]=0x00;
sendbuf->Data[0]=0x00;
sendbuf->Data[1]=0x11;
sendbuf->Data[2]=0x22;
sendbuf->Data[3]=0x33;
sendbuf->Data[4]=0x44;
sendbuf->Data[5]=0x55;
sendbuf->Data[6]=0x66;
sendbuf->Data[7]=0x77;
flag=VCI_Transmit(DEV_CAN232B,0,0,sendbuf);
if(flag!=1)
{
    MessageBox("send fail");
    return;
}
```

4.2.12 VCI_Receive

The function read data from specified equipment.

DWORD __stdcall VCI_Receive(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_CAN_OBJ pReceive);

Return value: if value >=1,it means have received CAN messages. Value is the Frame number.

Example:

```
#include "ControlCan.h"
VCI_CAN_OBJ databuf[300];
Value=VCI_Receive(DEV_CAN232B,0,0, databuf);
If(Value>0)
```

```
{
    //data processing
}
```

Note: PC need to request the receive message in time, avoiding the overflow of the device R-buffer. And databuf need to be larger than the R-buffer size of the device. Suggest you set buffer to 300.

1）You can use PC's Timer interrupt, and call the function every 5 -50ms.

2）You can make another multi-thread to call the function.

4.2.13 VCI_ FindGyUsbDevice

Note: you could not use the function, if you have only one adapter.

The function is used for finding Glinker USB-CAN in the computer.

DWORD __stdcall VCI_FindGyUsbDevice(PVCI_BOARD_INFO pInfo);

Example:

```
CString ProductSn[5];
VCI_BOARD_INFO pData[1];
int num=VCI_FindGyUsbDevice(pData);
CString strtemp,str;
for(int i=0;i<num;i++)
{
 str="";
 for(int j=0;j<4;j++)
 {
  strtemp.Format("%c",pData->str_GYUsb_Serial[i][j]);
  str+=strtemp;
 }
 ProductSn[i]="USBCAN-"+str;
}
```

Chapter 5 Appendix

For more information about BTR, ACR, AMR, etc, please refer the SJA1000 data manual.

Appendix 1: CAN2.0B agreement frame format (refer to SJA1000 CAN Controller)

5.1 CAN2.0B Standard frame.

There are 11 bytes in CAN standard frame, dividing it into two parts: information and data. The first three bytes are part of information.

	7	6	5	4	3	2	1	0
Byte 1	FF	RTR	X	X	DLC(data length)			
Byte2	(message ID)					ID.10-ID.3		
Byte3	ID.2-ID.0			X	X	X	X	X
Byte4	Data1							
Byte5	Data2							
Byte6	Data3							
Byte7	Data4							
Byte8	Data5							
Byte9	Data6							
Byte10	Data7							
Byte11	Data8							

Byte 1 is frame information. The bit 7 represents frame format, and in standard frame, FF=0; The Bit6 represents type of frame, and RTR=0 means data frame, RTR=1 means remote frame. DLC represents the actual data length in data frame.

Bytes 2-3 are message ID, and 11bits is effect.

Bytes 4-11 are actual data area, and it is invalid for remote frame.

5.2 CAN2.0B Extended frame

CAN extended frame information are 13 bytes, dividing it into two parts: information and data. The first five bytes are part of information.

	7	6	5	4	3	2	1	0
Byte 1	FF	RTR	X	X	DLC(data length)			
Byte2	(message ID)					ID.10-ID.3		
Byte3	ID.20-ID.13							

Byte4	ID.12-ID.5			
Byte5	ID.4-ID.0	X	X	X
Byte6	Data1			
Byte7	Data2			
Byte8	Data3			
Byte9	Data4			
Byte10	Data5			
Byte11	Data6			
Byte12	Data7			
Byte13	Data8			

Byte 1 is frame information. The bit7 is frame format, and in extended frame FF=1; The bit6 is type of frame, and RTR=0 represents data frame, RTR=1 is remote frame; DLC represents the actual data length in data frame.

Bytes 2-5 are message ID, its high 29 are effect.

Bytes 6-13 are the actual data area, and it is invalid for remote frame.