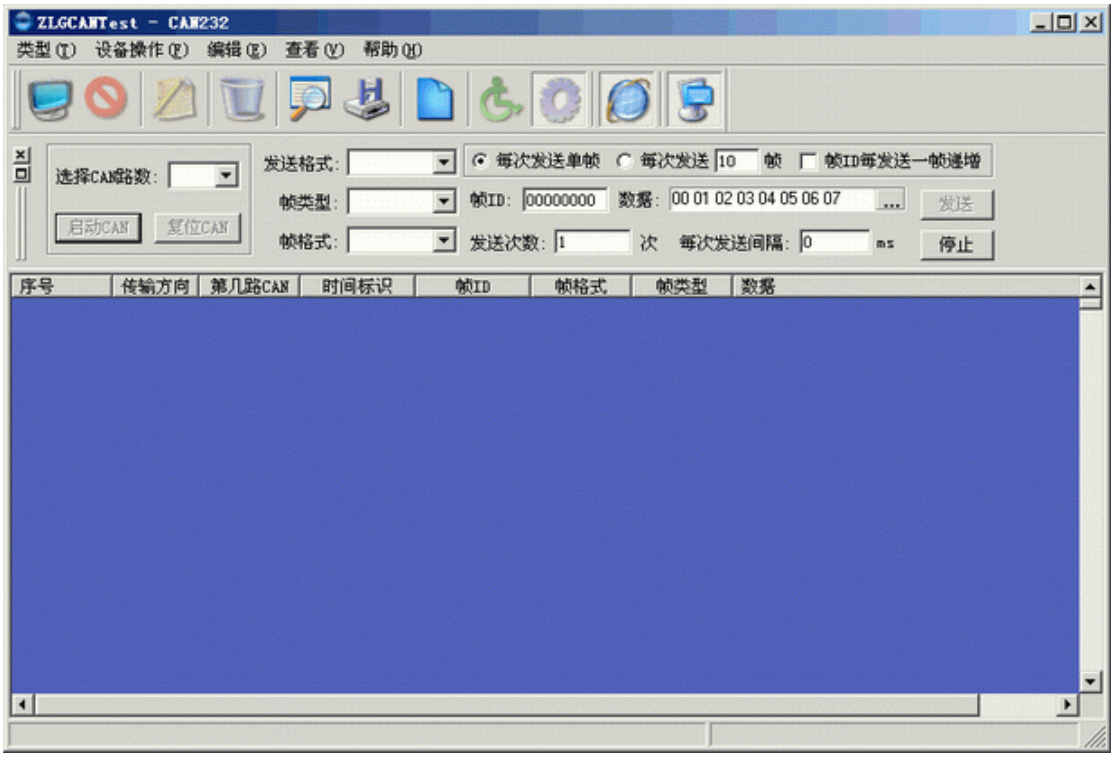
CAN-bus 通用测试软件及接口函数库使用手册

V2.0

目 录

1. 测试软件使用说明

CAN-bus 通用测试软件是一个专门用来对所有的 ZLGCAN 系列板卡进行测试的软件工具，此软件操作简单，容易上手，通过运用此软件可以非常方便的对板卡进行测试，从而熟悉板卡的性能，其主界面如下：



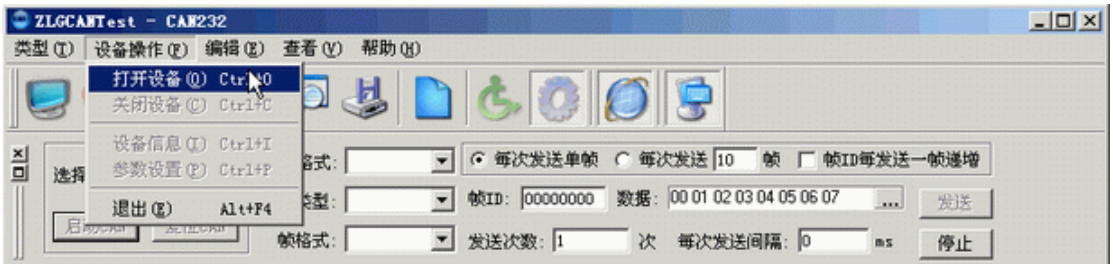
1.1. 设备操作

1.1.1. 设备类型选择

在进行操作之前,首先得从“类型”菜单中选择您想要操作的设备类型，如下图所示：

1.1.2. 打开设备

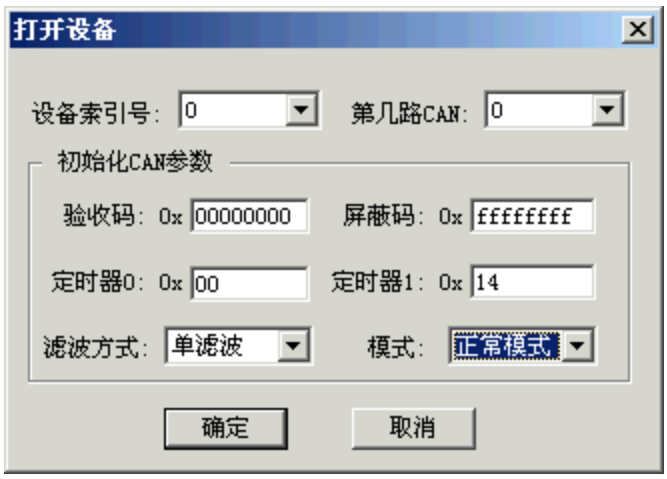
接着，从“设备操作”菜单中选择“打开设备”选项来打开设备：



(1) 这时，如果您选择要操作的设备是 CAN232 的话，就会弹出下面这个对话框：

在对话框中选择您想要打开的端口和用以打开端口的波特率，然后点“确定”按钮来打开设备。

(2) 而如果您选择的是其它设备的话，则会弹出下面这个对话框：

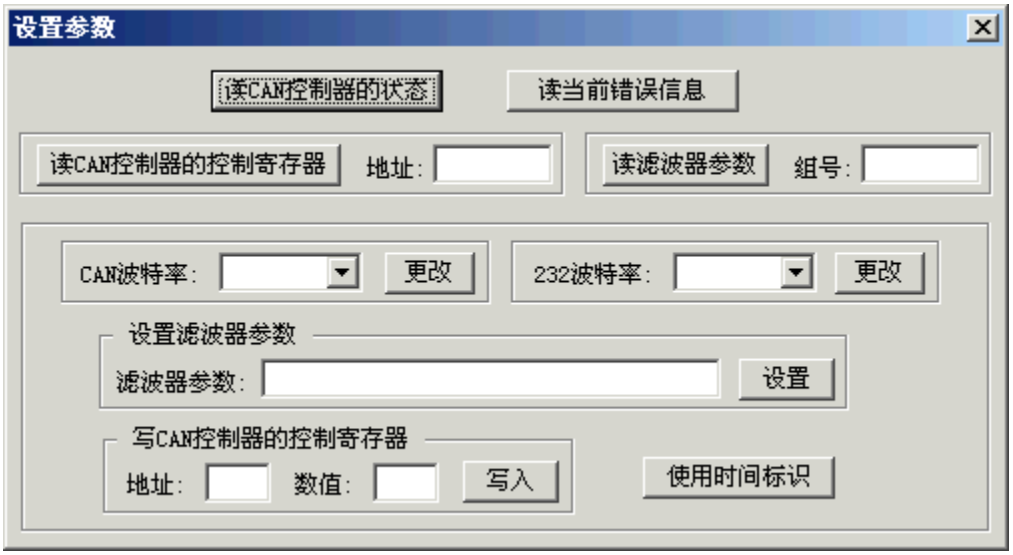


在这个对话框中您可以选择您要打开的设备索引号和第几路 CAN，以及设置 CAN 的初始化参数，然后点“确定”按钮来打开设备。

1.1.3. 设置参数

当您打开设备成功后，您可以从“设备操作”菜单中选择“参数设置”选项来进行一些参数设置（如果您想采用默认设置的话，可以略过此步骤）：

(1) 如果您操作的设备是 CAN232 的话，将会弹出以下对话框：



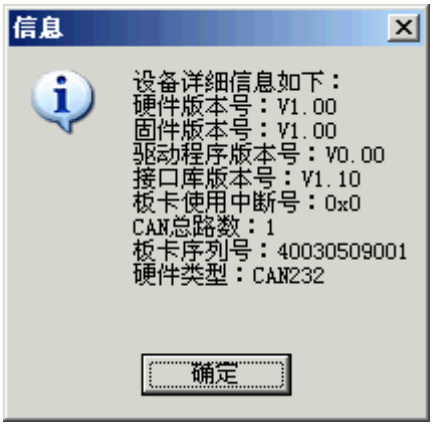
在此对话框中可以读取和更改 CAN 控制器的一些参数设置。在设置滤波器时，需要输入的参数长度为 12 个字节，输入格式类似于“FF FF FF FF FF FF FF FF FF FF FF FF ”（为 16 进制），具体参数怎么设置请参阅后面的接口函数库说明中的 VCI_SetReference 函数中的参数说明。

(2) 而如果您操作的是其它设备的话，则会弹出以下对话框：

此时你可以读取 CAN 控制器的状态和当前的错误信息。由于在初始化的时候已经把 CAN 控制器的参数都设置好了，所以在这里就没有设置参数的选项。

1.1.4. 获取设备信息

在打开设备后，您可以选择“设备操作”菜单中的“设备信息”选项来获得当前设备的详细信息：



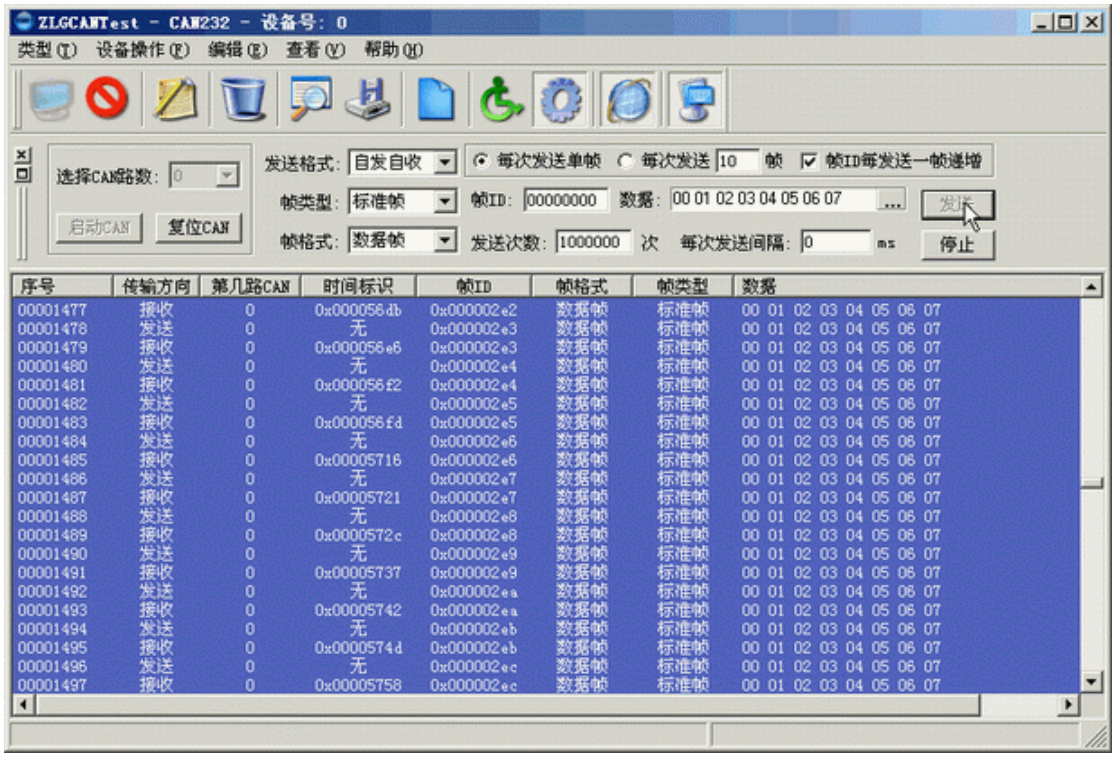
1.1.5. 启动CAN和复位CAN

在打开设备后，要想能够发送和接收数据，则必须点击下图中的“启动 CAN”按钮（注：在 CANET-E 中无需启动 CAN 就可以收发数据，只有这个设备比较特殊）：

1.1.6. 发送数据

当您启动 CAN 成功后，在下图中设置好您要发送的 CAN 帧的各项参数，然后点击“发送”按钮就可以发送数据了（其中发送格式下拉框中的自发自收选项表示发送出去的 CAN 帧自己也能收到，这个选项在测试的时候才需用到，在实际的应用中请选用正常发送）：

下图为正在发送数据中的画面：



1.2. 辅助操作

本软件中还设置了一些辅助操作，以方便您能够更好的观察和分析 CAN 数据，在“编辑”菜单中的各项操作即为辅助操作：

1.2.1. 设置数据列表缓冲帧数

通过设置缓冲区大小来改变当前屏幕所能缓冲显示的最大帧数，如下图：

1.2.2. 定位到指定帧

在下面这个对话框中输入您想要观察的数据行，就可以跳转到指定行：

1.2.3. 保存数据到文件

通过选择此选项来把当前屏幕列表中的数据存储到文件中。

1.2.4. 刷新数据列表

选择此选项清空屏幕列表中的数据。

1.2.5. 暂停显示发送和接收的数据

选择此选项后，接收和发送都在后台进行，其数据不在屏幕上显示出来。

1.2.6. 继续显示发送和接收的数据

选择此选项后，接收和发送都在前台进行，其数据在屏幕上显示出来。

1.2.7. 总是显示最后一行数据

选择此选项后，当前数据列表中的最后一行总是可见。

2. 接口函数库说明及其使用

2.1. 接口卡设备类型定义

各个接口卡的类型定义如下：

设备名称	设备类型号
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

2.2. 错误码定义

名称	值	描述
ERR_CAN_OVERFLOW	0x00000001	CAN 控制器内部 FIFO 溢出
ERR_CAN_ERRALARM	0x00000002	CAN 控制器错误报警
ERR_CAN_PASSIVE	0x00000004	CAN 控制器消极错误
ERR_CAN_LOSE	0x00000008	CAN 控制器仲裁丢失
ERR_CAN_BUSERR	0x00000010	CAN 控制器总线错误
ERR_DEVICEOPENED	0x00000100	设备已经打开
ERR_DEVICEOPEN	0x00000200	打开设备错误
ERR_DEVICENOTOPEN	0x00000400	设备没有打开
ERR_BUFFEROVERFLOW	0x00000800	缓冲区溢出
ERR_DEVICENOTEXIST	0x00001000	此设备不存在
ERR_LOADKERNELDLL	0x00002000	装载动态库失败
ERR_CMDFAILED	0x00004000	执行命令失败错误码
ERR_BUFFERCREATE	0x00008000	内存不足
ERR_CANETE_PORTOPENED	0x00010000	端口已经被打开
ERR_CANETE_INDEXUSED	0x00020000	设备索引号已经被占用

2.3. 函数库中的数据结构定义

2.3.1. VCI_BOARD_INFO

描述

VCI_BOARD_INFO 结构体包含 ZLGCAN 系列接口卡的设备信息。结构体将在 VCI_ReadBoardInfo 函数中被填充。

```
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;
```

成员

hw_Version

硬件版本号，用 16 进制表示。比如 0x0100 表示 V1.00。

fw_Version

固件版本号，用 16 进制表示。

dr_Version

驱动程序版本号，用 16 进制表示。

in_Version

接口库版本号，用 16 进制表示。

irq_Num

板卡所使用的中断号。

can_Num

表示有几路 CAN 通道。

str_Serial_Num

此板卡的序列号。

str_hw_Type

硬件类型，比如“USBCAN V1.00”（注意：包括字符串结束符’\0’）。

Reserved

系统保留。

2.3.2. VCI_CAN_OBJ

描述

VCI_CAN_OBJ 结构体在 VCI_Transmit 和 VCI_Receive 函数中被用来传送 CAN 信息帧。

```
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;
```

成员

ID

报文 ID。

TimeStamp

接收到信息帧时的时间标识，从 CAN 控制器初始化开始计时。

TimeFlag

是否使用时间标识，为 1 时 TimeStamp 有效，TimeFlag 和 TimeStamp 只在此帧为接收帧时有意义。

SendType

发送帧类型，=0 时为正常发送，=1 时为单次发送，=2 时为自发自收，=3 时为单次自发自收，只在此帧为发送帧时有意义。

RemoteFlag

是否是远程帧。

ExternFlag

是否是扩展帧。

DataLen

数据长度(<=8)，即 Data 的长度。

Data

报文的数据。

Reserved

系统保留。

2.3.3. VCI_CAN_STATUS

描述

VCI_CAN_STATUS 结构体包含 CAN 控制器状态信息。结构体将在 VCI_ReadCanStatus 函数中被填充。

```
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;
```

成员

ErrInterrupt

中断记录，读操作会清除。

regMode

CAN 控制器模式寄存器。

regStatus

CAN 控制器状态寄存器。

regALCapture

CAN 控制器仲裁丢失寄存器。

regECCapture

CAN 控制器错误寄存器。

regEWLimit

CAN 控制器错误警告限制寄存器。

regRECounter

CAN 控制器接收错误寄存器。

regTECounter

CAN 控制器发送错误寄存器。

Reserved

系统保留。

2.3.4. VCI_ERR_INFO

描述

VCI_ERR_INFO 结构体用于装载 VCI 库运行时产生的错误信息。结构体将在 VCI_ReadErrInfo 函数中被填充。

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

成员

ErrCode

错误码。

Passive_ErrData

当产生的错误中有消极错误时表示为消极错误的错误标识数据。

ArLost_ErrData

当产生的错误中有仲裁丢失错误时表示为仲裁丢失错误的错误标识数据。

2.3.5. VCI_INIT_CONFIG

描述

VCI_INIT_CONFIG 结构体定义了初始化 CAN 的配置。结构体将在 VCI_InitCan 函数中被填充。

```
typedef struct _INIT_CONFIG {
    DWORD   AccCode;
    DWORD   AccMask;
    DWORD   Reserved;
    UCHAR   Filter;
    UCHAR   Timing0;
    UCHAR   Timing1;
```

```
    UCHAR   Mode;
} VCI_INIT_CONFIG, *PVCI_INIT_CONFIG;
```

成员

AccCode

验收码。

AccMask

屏蔽码。

Reserved

保留。

Filter

滤波方式。

Timing0

定时器 0（BTR0）。

Timing1

定时器 1（BTR1）。

Mode

模式。

备注

Timing0 和 Timing1 用来设置 CAN 波特率，几种常见的波特率设置如下：

CAN 波特率	定时器 0	定时器 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

2.3.6. CHGDESIPANDPORT

描述

CHGDESIPANDPORT 结构体用于装载更改 CANET-E 的目标 IP 和端口的必要信息。此结构体在 CANETE-E 中使用。

```
typedef struct _tagChgDesIPAndPort {
    char szpwd[10];
    char szdesip[20];
    int desport;
```

```
    BYTE blisten;
} CHGDESIPANDPORT;
```

成员

szpwd[10]

更改目标 IP 和端口所需要的密码，长度小于 10，比如为“11223344”。

szdesip[20]

所要更改的目标 IP，比如为“192.168.0.111”。

desport

所要更改的目标端口，比如为 4000。

blisten

所要更改的工作模式，0 表示正常模式，1 表示只听模式。

2.4. 接口库函数说明

2.4.1. VCI_OpenDevice

描述

此函数用以打开设备。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

Reserved

当设备为 CAN232 时，此参数表示为用以打开串口的波特率，可以为 2400，4800，9600，14400，19200，28800，57600。当设备为 CANET-UDP 时，此参数表示要打开的本地端口号，建议在 5000 到 40000 范围内取值。当设备为 CANET-TCP 时，此参数固定为 0。当为其他设备时此参数无意义。

返回值

为 1 表示操作成功，0 表示操作失败。

示例

```
#include "ControlCan.h"

int nDeviceType = 6;    /* CAN232 */
int nDeviceInd = 0;     /* COM1 */
int nReserved = 9600;   /* Baudrate */
DWORD dwRel;

dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

2.4.2. VCI_CloseDevice

描述

此函数用以关闭设备。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

返回值

为 1 表示操作成功，0 表示操作失败。

示例

```
#include "ControlCan.h"

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

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

2.4.3. VCI_InitCan

描述

此函数用以初始化指定的 CAN。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

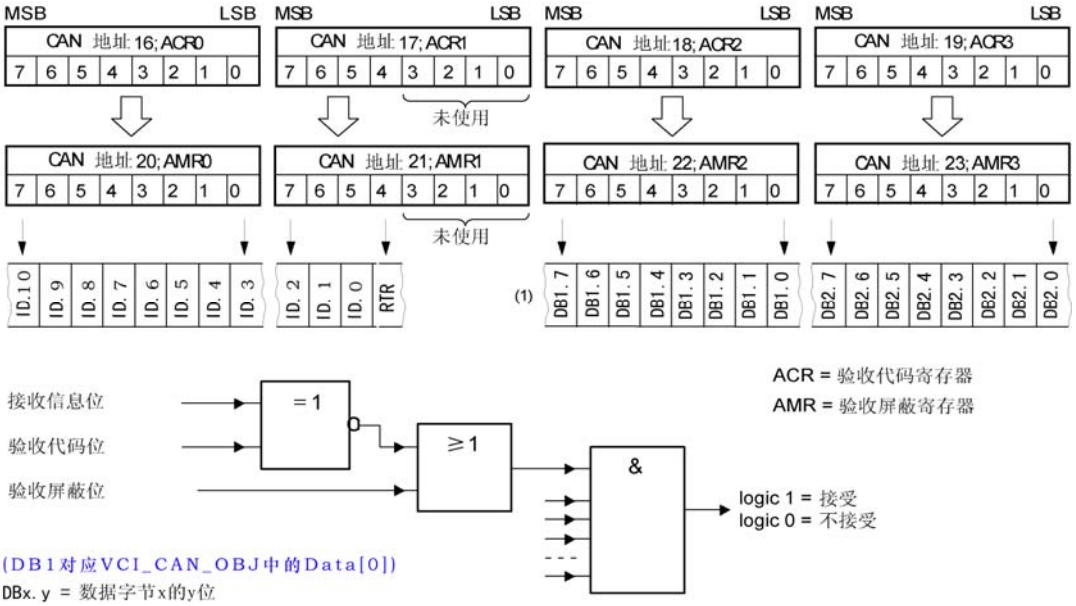
第几路 CAN。

pInitConfig

初始化参数结构（注：当为 CAN232 时，忽略此参数，其值设为 NULL）。

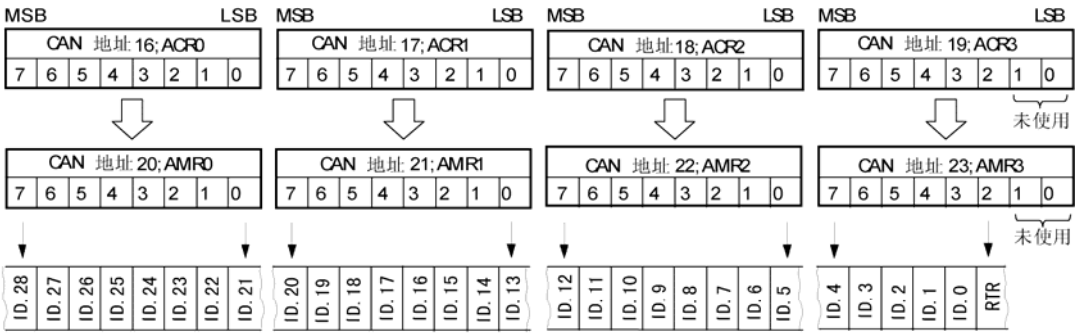
成员	功能描述
pInitConfig->AccCode	AccCode 对应 SJA1000 中的四个寄存器 ACR0，ACR1，ACR2，ACR3，其中高字节对应 ACR0，低字节对应 ACR3；AccMask 对应 SJA1000 中的四个寄存器 AMR0，AMR1，AMR2，AMR3，其中高字节对应 AMR0，低字节对应 AMR3。（请看表后说明）
pInitConfig->AccMask	
pInitConfig->Reserved	保留
pInitConfig->Filter	滤波方式，1 表示单滤波，0 表示双滤波
pInitConfig->Timing0	定时器 0
pInitConfig->Timing1	定时器 1
pInitConfig->Mode	模式，0 表示正常模式，1 表示只听模式

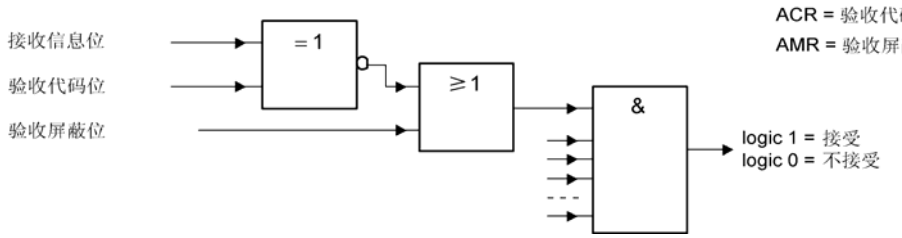
当滤波方式为单滤波，接收帧为标准帧时：



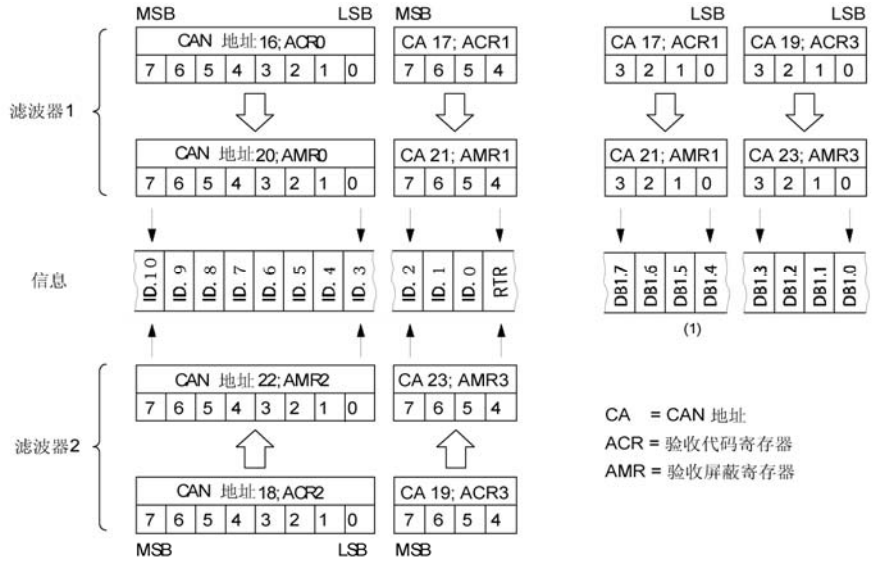
RTR 对应 VCI_CAN_OBJ 中的 RemoteFlag

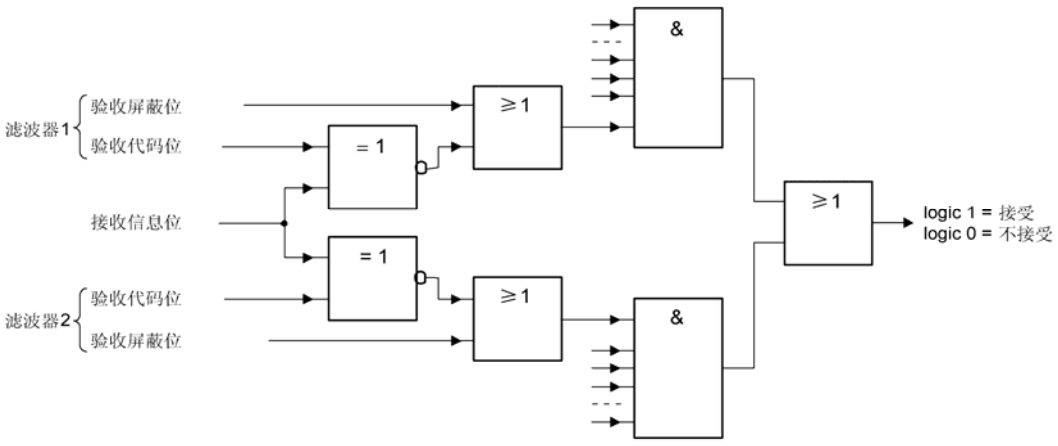
当滤波方式为单滤波，接收帧为扩展帧时：





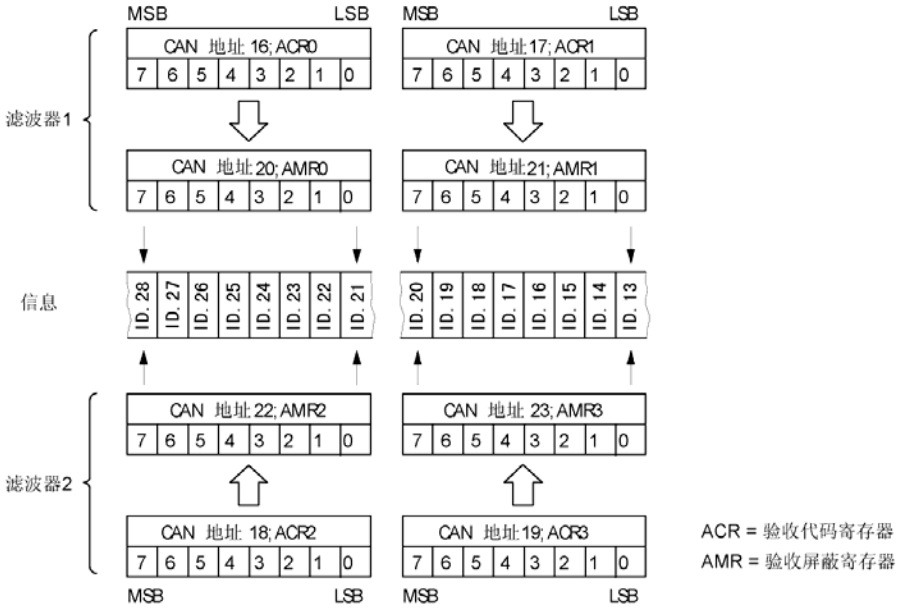
当滤波方式为双滤波，接收帧为标准帧时：

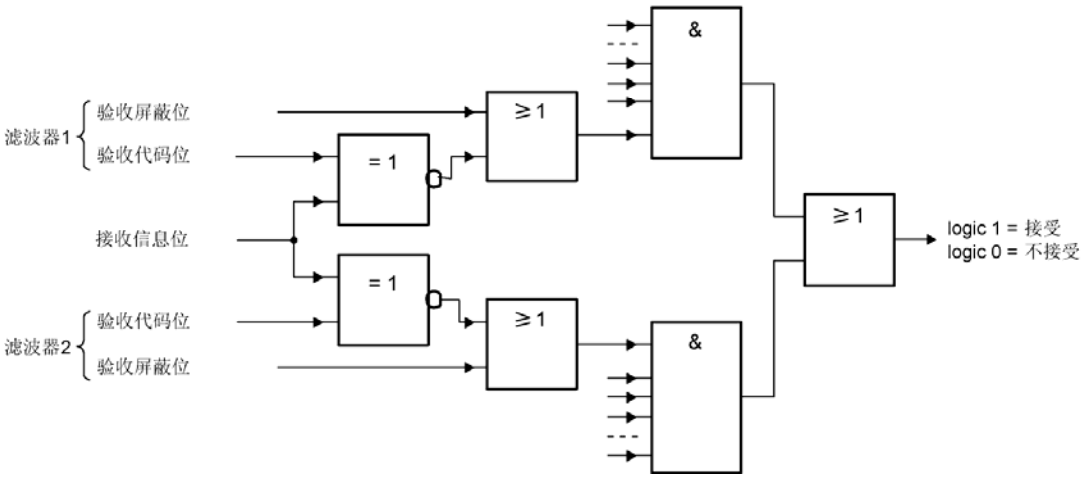




DBx.y = 数据字节x位y

当滤波方式为双滤波，接收帧为扩展帧时：





返回值

为 1 表示操作成功，0 表示操作失败。（注：在 CANET-E 中无此函数）

示例

```
#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("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
```

```
    return FALSE;
}
dwRel = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

2.4.4. VCI_ReadBoardInfo

描述

此函数用以获取设备信息。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

pInfo

用来存储设备信息的 VCI_BOARD_INFO 结构指针。

返回值

为 1 表示操作成功，0 表示操作失败。（注：在 CANET 中无此函数）

示例

```
#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

描述

此函数用以获取最后一次错误信息。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。（注：当要读取设备错误的时候，此参数应该设为－1。比如当调用 VCI_OpenDevice，VCI_CloseDevice 和 VCI_ReadBoardInfo 这些与特定的第几路 CAN 操作无关的操作函数失败后，调用此函数来获取失败错误码的时候应该把 CANIndex 设为－1。）

pErrInfo

用来存储错误信息的 VCI_ERR_INFO 结构指针。pErrInfo->ErrCode 可能为下列各个错误码的多种组合之一：

ErrCode	Passive_ErrData	ArLost_ErrData	错误描述
0x0100	无	无	设备已经打开
0x0200	无	无	打开设备错误
0x0400	无	无	设备没有打开
0x0800	无	无	缓冲区溢出
0x1000	无	无	此设备不存在
0x2000	无	无	装载动态库失败
0x4000	无	无	表示为执行命令失败错误
0x8000		无	内存不足
0x0001	无	无	CAN 控制器内部 FIFO 溢出
0x0002	无	无	CAN 控制器错误报警
0x0004	有，具体值见表后	无	CAN 控制器消极错误
0x0008	无	有，具体值见表后	CAN 控制器仲裁丢失
0x0010	无	无	CAN 控制器总线错误

返回值

为 1 表示操作成功，0 表示操作失败。

备注

当(PErrInfo->ErrCode&0x0004)==0x0004 时，存在 CAN 控制器消极错误。

PErrInfo->Passive_ErrData[0]错误代码捕捉位功能表示

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
错误代码类型		错误属性	错误段表示				

错误代码类型功能说明

位 ECC.7	位 ECC.6	功能

0	0	位错
0	1	格式错
1	0	填充错
1	1	其它错误

错误属性

bit5 =0； 表示发送时发生的错误。

=1； 表示接收时发生的错误。

错误段表示功能说明

bit4	bit 3	bit 2	bit 1	bit 0	功能
0	0	0	1	1	帧开始
0	0	0	1	0	ID.28-ID.21
0	0	1	1	0	ID.20-ID.18
0	0	1	0	0	SRTR 位
0	0	1	0	1	IDE 位
0	0	1	1	1	ID.17-ID.13
0	1	1	1	1	ID.12-ID.5
0	1	1	1	0	ID.4-ID.0
0	1	1	0	0	RTR 位
0	1	1	0	1	保留位 1
0	1	0	0	1	保留位 0
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据区
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 定义符
1	1	0	0	1	应答通道
1	1	0	1	1	应答定义符
1	1	0	1	0	帧结束
1	0	0	1	0	中止
1	0	0	0	1	活动错误标志
1	0	1	1	0	消极错误标志
1	0	0	1	1	支配（控制）位误差
1	0	1	1	1	错误定义符
1	1	1	0	0	溢出标志

PErrInfo->Passive_ErrData[1] 表示接收错误计数器

PErrInfo->Passive_ErrData[2] 表示发送错误计数器

当(PErrInfo->ErrCode&0x0008)==0x0008 时，存在 CAN 控制器仲裁丢失错误。

PErrInfo->ArLost_ErrData 仲裁丢失代码捕捉位功能表示

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
———	———	———	错误段表示				

错误段表示功能表示

位					十进制值	功能
ALC.4	ALC.3	ALC.2	ALC.1	ALC.0		
0	0	0	0	0	0	仲裁丢失在识别码的 bit1
0	0	0	0	1	1	仲裁丢失在识别码的 bit2
0	0	0	1	0	2	仲裁丢失在识别码的 bit3
0	0	0	1	1	3	仲裁丢失在识别码的 bit4
0	0	1	0	0	4	仲裁丢失在识别码的 bit5
0	0	1	0	1	5	仲裁丢失在识别码的 bit6
0	0	1	1	0	6	仲裁丢失在识别码的 bit7
0	0	1	1	1	7	仲裁丢失在识别码的 bit8
0	1	0	0	0	8	仲裁丢失在识别码的 bit9
0	1	0	0	1	9	仲裁丢失在识别码的 bit10
0	1	0	1	0	10	仲裁丢失在识别码的 bit11
0	1	0	1	1	11	仲裁丢失在 SRTR 位
0	1	1	0	0	12	仲裁丢失在 IDE 位
0	1	1	0	1	13	仲裁丢失在识别码的 bit12
0	1	1	1	0	14	仲裁丢失在识别码的 bit13
0	1	1	1	1	15	仲裁丢失在识别码的 bit14
1	0	0	0	0	16	仲裁丢失在识别码的 bit15
1	0	0	0	1	17	仲裁丢失在识别码的 bit16
1	0	0	1	0	18	仲裁丢失在识别码的 bit17
1	0	0	1	1	19	仲裁丢失在识别码的 bit18
1	0	1	0	0	20	仲裁丢失在识别码的 bit19
1	0	1	0	1	21	仲裁丢失在识别码的 bit20
1	0	1	1	0	22	仲裁丢失在识别码的 bit21
1	0	1	1	1	23	仲裁丢失在识别码的 bit22
1	1	0	0	0	24	仲裁丢失在识别码的 bit23
1	1	0	0	1	25	仲裁丢失在识别码的 bit24
1	1	0	1	0	26	仲裁丢失在识别码的 bit25
1	1	0	1	1	27	仲裁丢失在识别码的 bit26
1	1	1	0	0	28	仲裁丢失在识别码的 bit27
1	1	1	0	1	29	仲裁丢失在识别码的 bit28
1	1	1	1	0	30	仲裁丢失在识别码的 bit29
1	1	1	1	1	31	仲裁丢失在 ERTR 位

示例

```
#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

描述

此函数用以获取 CAN 状态。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

pCANStatus

用来存储 CAN 状态的 VCI_CAN_STATUS 结构指针。

返回值

为 1 表示操作成功，0 表示操作失败。（注：在 CANET-E 中无此函数）

示例

```
#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

描述

此函数用以获取设备的相应参数。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

RefType

参数类型。

pData

用来存储参数有关数据缓冲区地址首指针。

返回值

为 1 表示操作成功，0 表示操作失败。

备注

(1) 当设备类型为 PCI5121，PCI5110 或 ISA5420 时：

RefType	pData	功能描述
1	总长度为 2 个字节，pData[0] 表示 CAN 控制器的控制寄存器的地址，pData[1]表示要读出 CAN 控制器的控制寄存器的值。	读取 CAN 芯片某个寄存器的值。 例如要读取地址为 9 的寄存器值： UCHAR pData[2] = {9,0}; VCI_GetReference(DeviceType,DeviceInd,CANInd,1,pData); 如果调用成功，pData[1]将存放读出的值。

(2) 当设备类型为 USBCAN1，USBCAN2 时：

RefType	pData	功能描述
1	总长度 1 个字节，当作为输入参数时，表示为所要读取的 CAN 控制器的控制寄存器的地址。 当作为输出参数时，表示为 CAN 控制器的控制寄存器的值。	读 CAN 控制器的指定控制寄存器的值。 例如对 USBCAN1： BYTE val=0; VCI_GetReference(VCI_USBCAN1,0,0,1,(PVOID)&val); 如果此函数调用成功,则在 val 中返回寄存器的值。

(3) 当设备类型为 CANET-UDP 时：

RefType	pData	功能描述
0	字符串首指针，用来存储所读取出来的CANETE-E的IP	读取 CANET-E 的 IP 地址。 例如：

	地址。	char szip[20]; VCI_GetReference(VCI_CANETE,0,0,0,(PVOID)szip); 如果此函数调用成功,则在 szip 中返回 CANET-E 的地址。
1	长度为 4 个字节，存储读取出来的 CANET-E 的工作端口。	读取 CANET-E 的工作端口。 例如： int port; VCI_GetReference(VCI_CANETE,0,0,1,(PVOID)&port); 如果此函数调用成功,则在 port 中返回 CANET-E 的工作端口。

(3) 当设备类型为 CANET-TCP 时：

此设备有两种工作模式，分别为客户端和服务器模式，如果设备工作在客户端模式，我们的测试工具需要工作在服务器模式，而设备工作在服务器模式，我们的测试工作则工作在客户端模式。

RefType	pData	功能描述
0	字符串首指针，用来存储所读取出来的 CANET-TCP 的 IP 地址。（当设备工作在服务器模式时使用）	读取 CANET-TCP 的 IP 地址。 例如： char szip[20]; VCI_GetReference(VCI_CANETE,0,0,0,(PVOID)szip); 如果此函数调用成功,则在 szip 中返回 CANET-E 的地址。
1	长度为 4 个字节，存储读取出来的 CANET-E 的工作端口。(当设备工作在服务器模式时有效）	读取 CANET-TCP 的工作端口。 例如： int port; VCI_GetReference(VCI_CANET_TCP,0,0,1,(PVOID)&port); 如果此函数调用成功,则在 port 中返回 CANET-E 的工作端口。
2	长度为 4 个字节，存储读取出来的 TCP 服务器的端口。(设备在服务器和客户端模式时同时有效）	读取或设置 TCP 服务器的端口。 例如： int port; VCI_SetReference(VCI_CANET_TCP,0,0,2,(PVOID)&port); 如果此函数调用成功,设置本机器上的工作端口。
4	长度为 4 个字节，TCP 设备的工作模式。	0 为客户端方式，1 为服务器方式。 例如： int iType = 1; VCI_SetReference(VCI_CANET_TCP,0,0,4,(PVOID)&iType); 如果此函数调用成功，设置本机器工作在服务器模式。
5	长度为 4 个字节，获取连接的客户端的数目。（当设备工作在客户端模式下有效）	读取或设置 TCP 服务器的端口。 例如： int iCount; VCI_GetReference(VCI_CANET_TCP,0,0,5,(PVOID)&iCount); 如果此函数调用成功,则在 port 中返回 CANET-E 的工作端口。
6	使用 REMOTE_CLIENT 结构，获取一个连接的信息。(当设备工作在客户端模式下有效）	当有客户端连接到次服务器（本机）时，使用此命令获取客户端信息。 例如： REMOTE_CLIENT cli; cli.iIndex = 0; //获取第 0 个连接到服务器的客户端

		VCI_GetReference(VCI_CANET_TCP ,0, 0,6,(PVOID)&cli); 如果此函数调用成功，则在 cli 里面返回客户端的信息。

REMOTE_CLIENT 结构

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

(4) 当设备类型为 CAN232 时：

RefType	pData	功能描述
1	总长度 14 个字节，当作为输入参数时，只有第一个字节有效，值为所读取的滤波器的序号，以为 1，2，3，4。 当作为输出参数时，具体各个字节所代表的意义见表后。	取得指定的滤波器参数，例如要读取第一个滤波器的参数，可以这样： BYTE info[14]; info[0]=1; VCI_GetReference(VCI_CAN232,0,0,1,(PVOID)info); 如果此函数调用成功，则在 info 中返回 14 个字节的第一个滤波器的参数。
2	总长度 1 个字节，当作为输入参数时，表示为所要读取的 CAN 控制器的控制寄存器的地址。 当作为输出参数时，表示为 CAN 控制器的控制寄存器的值。	读 CAN 控制器的指定控制寄存器的值，例如： BYTE val=0; VCI_GetReference(VCI_CAN232,0,0,2,(PVOID)&val); 如果此函数调用成功,则在 val 中返回寄存器的值。

当 RefType=1 时,此时返回的 pData 各个字节所代表的意义如下：

pData[0]信息保留

pData[1]表示 CAN 控制器 BTR0 的值；

pData[2]表示 CAN 控制器 BTR1 的值；

pData[3]读取该组验收滤波器模式,位功能

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

MFORMATB =1; 验收滤波器该组仅用于扩展帧信息。标准帧信息被忽略。

=0; 验收滤波器该组仅用于标准帧信息。扩展帧信息被忽略。

AMODEB =1; 单验收滤波器选项使能－长滤波器有效。

=0; 双验收滤波器选项使能－短滤波器有效。

pData[4]读取该组验收滤波器的使能,位功能

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

BF2EN =1; 该组滤波器 2 使能，不能对相应的屏蔽和代码寄存器进行写操作。

=0; 该组滤波器 2 禁止，可以改变相应的屏蔽和代码寄存器。

BF1EN =1; 该组滤波器 1 使能，不能对相应的屏蔽和代码寄存器进行写操作。

=0；该组滤波器 1 禁止，可以改变相应的屏蔽和代码寄存器。

注：如果选择单滤波器模式，该单滤波器与对应的滤波器 1 使能位相关。滤波器 2 使能位在单滤波器模式中不起作用。

pData[5]读取该组验收滤波器的优先级,位功能

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

BF2PRIO =1；该组滤波器 2 优先级高，如果信息通过该组滤波器 2，立即产生接收中断。

=0；该组滤波器 2 优先级低，如果 FIFO 级超过接收中断级滤波器，产生接收中断。

BF1PRIO =1；该组滤波器 1 优先级高，如果信息通过该组滤波器 1，立即产生接收中断。

=0；该组滤波器 1 优先级低，如果 FIFO 级超过接收中断级滤波器，产生接收中断。

pData[6—9]表示该组滤波器 ACR 的值。

pData[a—d]表示该组滤波器 AMR 的值。

示例

```
#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

描述

此函数用以设置设备的相应参数，主要处理不同设备的特定操作。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

RefType

参数类型。

pData

用来存储参数有关数据缓冲区地址首指针。

返回值

为 1 表示操作成功，0 表示操作失败。

备注

VCI_SetReference 和 VCI_GetReference 这两个函数是用来针对各个不同设备的一些特定操作的。比如 CAN232 的更改波特率，设置报文滤波等等。函数中的 PVOID 型参数 pData 随不同设备的不同操作而具有不同的意义。

(1) 当设备类型为 PCI5121，PCI5110 或 ISA5420 时：

RefType	pData	功能描述
1	总长度为 2 个字节，pData[0] 表示 CAN 控制器的控制寄存器的地址，pData[1] 表示要写入的数值。	写 CAN 控制器的指定控制寄存器

(2) 当设备类型为 USBCAN1，USBCAN2 时：

RefType	pData	功能描述
1	总长度为 2 个字节，pData[0] 表示 CAN 控制器的控制寄存器的地址，pData[1] 表示要写入的数值。	写 CAN 控制器的指定控制寄存器

(3) 当设备类型为 CANET-UDP 时：

RefType	pData	功能描述
0	字符串首指针，用来存储所指定操作的 CANETE-UDP 的 IP 地址。	设置所要操作的 CANET-UDP 的 IP 地址

1	长度为 4 个字节，存储所指定操作的 CANET-UDP 的工作端口。	设置所要操作的 CANET-UDP 的工作端口 DWORD port=5000; VCI_SetReference(12,0,0,1,(PVOID)&port);

(4) 当设备类型为 CANET-TCP 时：

RefType	pData	功能描述
0	字符串首指针，用来存储所指定操作的 CANET-TCP 的 IP 地址。	设置所要操作的 CANET-TCP 的 IP 地址
1	长度为 4 个字节，存储所指定操作的 CANET-TCP 的工作端口。(目标)	设置所要操作的 CANET-TCP 的工作端口
2	长度为 4 个字节，存储本机器地上工作端口。	设置本机端口
4	长度为 4 个字节，存储本机器 TCP 的工作模式。	设置本机的工作模式，如果 CANET－TCP 工作在服务器模式则本机工作在客户端模式，如果 CANET-TCP 工作在客户端模式则本机工作在服务器模式。0 为客户端方式，1 为服务器方式。
5	长度为 4 个字节，存储连接到本机服务器上的客户端个数。	只能获取。
6	长度为 REMOTE_CLIENT，存储所一个连接的信息。	只能获取。
7	使用 REMOTE_CLIENT 结构，删除一个连接。（当设备工作在客户端模式下有效）	例如： REMOTE_CLIENT cli; cli.iIndex = 0;　　//删除第 0 个连接的客户端 VCI_SetReference(VCI_CANET_TCP,0,0,7,(PVOID)&cli);

REMOTE_CLIENT 结构

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

(5) 当设备类型为 CAN232 时：

RefType	pData	功能描述
1	总长度为 1 个字节 ＝0；　10Kbps ＝1；　20Kbps ＝2；　50Kbps ＝3；　125Kbps ＝4；　250Kbps ＝5；　500Kbps	更改 CAN 波特率，例如要设置 CAN 的波特率为 10Kbps： BYTE baud=0; VCI_SetReference(VCI_CAN232,0,0,1,(PVOID)&baud);

	＝6； 800Kbps ＝7； 1000Kbps	
2	总长度为 12 个字节，其各个字节所代表的意义见表后。	设置滤波器参数
3	总长度为 1 个字节 ＝1； 2.4Kbps ＝2； 4.8Kbps ＝3； 9.6Kbps ＝4； 14.4Kbps ＝5； 19.2Kbps ＝6； 28.8Kbps ＝7； 57.6Kbps	更改 232 波特率
4	总长度为 2 个字节，pData[0] 表示 CAN 控制器的控制寄存器的地址，pData[1] 表示要写入的数值。	写 CAN 控制器的指定控制寄存器
5	总长度为 1 个字节， ＝0xAA；使用时间标识 ＝其他；不使用时间标识	设置时间标识

当 RefType=2 时，此时的 pData 各个字节所代表的意义如下：

pData[0]设置哪一组验收滤波器；共有 4 组：

=1 设置第 1 组

=2 设置第 2 组

=3 设置第 3 组

=4 设置第 4 组

pData[1]设置该组验收滤波器模式,位功能

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

MFORMATB =1 验收滤波器该组仅用于扩展帧信息。标准帧信息被忽略。

=0 验收滤波器该组仅用于标准帧信息。扩展帧信息被忽略。

AMODEB =1 单验收滤波器选项使能－长滤波器有效。

=0 双验收滤波器选项使能－短滤波器有效。

pData[2]设置该组验收滤波器的使能,位功能

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

BF2EN =1 该组滤波器 2 使能，不能对相应的屏蔽和代码寄存器进行写操作。

=0 该组滤波器 2 禁止，可以改变相应的屏蔽和代码寄存器。

BF1EN =1 该组滤波器 1 使能，不能对相应的屏蔽和代码寄存器进行写操作。

=0 该组滤波器 1 禁止，可以改变相应的屏蔽和代码寄存器。

注：如果选择单滤波器模式，该单滤波器与对应的滤波器 1 使能位相关。滤波器 2 使能位在单滤波器模式中不起作用。

pData[3]设置该组验收滤波器的优先级，位功能

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

BF2PRIO =1 该组滤波器 2 优先级高，如果信息通过该组滤波器 2 ，立即产生接收中断。

=0 该组滤波器 2 优先级低，如果 FIFO 级超过接收中断级滤波器，产生接收中断。

BF1PRIO =1 该组滤波器 1 优先级高，如果信息通过该组滤波器 1 ，立即产生接收中断。

=0 该组滤波器 1 优先级低，如果 FIFO 级超过接收中断级滤波器，产生接收中断。

pData[4---7] 分别对应要设置的 SJA1000 的 ACR0---ACR3 的值；

pData[8---b] 分别对应要设置的 SJA1000 的 AMR0---AMR3 的值；

示例

```
#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

描述

此函数用以获取指定接收缓冲区中接收到但尚未被读取的帧数。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

返回值

返回尚未被读取的帧数。

示例

```
#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

描述

此函数用以清空指定缓冲区。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

返回值

为 1 表示操作成功，0 表示操作失败。

示例

```
#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

描述

此函数用以启动 CAN。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

返回值

为 1 表示操作成功，0 表示操作失败。（注：在 CANET 中无此函数）

示例

```
#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("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
dwRel = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if (dwRel == STATUS_ERR)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
dwRel = VCI_StartCAN(nDeviceType, nDeviceInd, nCANInd);
if (dwRel == STATUS_ERR)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("启动设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
```

```
}
```

2.4.12. VCI_ResetCAN

描述

此函数用以复位 CAN。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

返回值

为 1 表示操作成功，0 表示操作失败。（注：在 CANET 中无此函数）

示例

```
#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

描述

返回实际发送的帧数。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

pSend

要发送的数据帧数组的首指针。

Len

要发送的数据帧数组的长度。

返回值

返回实际发送的帧数。

示例

```
#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

描述

此函数从指定的设备读取数据。

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

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个 PCI5121 时，索引号为 0，有两个时可以为 0 或 1。（注：当为 CAN232 时，0 表示要打开的是 COM1，1 表示要打开的是 COM2。）

CANIndex

第几路 CAN。

pReceive

用来接收的数据帧数组的首指针。

Len

用来接收的数据帧数组的长度。

WaitTime

等待超时时间，以毫秒为单位。

返回值

返回实际读取到的帧数。如果返回值为 0xFFFFFFFF，则表示读取数据失败，有错误发生，请调用 VCI_ReadErrInfo 函数来获取错误码。

示例

```
#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. 接口库函数使用方法

首先，把库函数文件都放在工作目录下。库函数文件总共有三个文件：ControlCAN.h、ControlCAN.lib、ControlCAN.dll 和一个文件夹 kerneldlls。

2.5.1. VC调用动态库的方法

(1) 在扩展名为.CPP 的文件中包含 ControlCAN.h 头文件。

如：#include “ControlCAN.h”

(2) 在工程的连接器设置中连接到 ControlCAN.lib 文件。

如：在 VC7 环境下，在项目属性页里的配置属性→连接器→输入→附加依赖项中添加 ControlCAN.lib

2.5.2. VB调用动态库的方法

通过以下方法进行声明后就可以调用了。

语法:

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

Declare 语句的语法包含下面部分：

Public（可选）

用于声明在所有模块中的所有过程都可以使用的函数。

Private（可选）

用于声明只能在包含该声明的模块中使用的函数。

Name（必选）

任何合法的函数名。动态链接库的入口处（entry points）区分大小写。

Libname（必选）

包含所声明的函数动态链接库名或代码资源名。

Alias（可选）

表示将被调用的函数在动态链接库 (DLL) 中还有另外的名称。当外部函数名与某个函数重名时，就可以使用这个参数。当动态链接库的函数与同一范围内的公用变量、常数或任何其它过程的名称相同时，也可以使用 Alias。如果该动态链接库函数中的某个字符不符合动态链接库的命名约定时，也可以使用 Alias。

Aliasname（可选）

动态链接库。如果首字符不是数字符号 (#)，则 aliasname 是动态链接库中该函数入口处的名称。如果首字符是 (#)，则随后的字符必须指定该函数入口处的顺序号。

Arglist（可选）

代表调用该函数时需要传递参数的变量表。

Type（可选）

Function 返回值的数据类型；可以是 Byte、Boolean、Integer、Long、Currency、Single、Double、Decimal（目前尚不支持）、Date、String（只支持变长）或 Variant，用户定义类型，或对象类型。

arglist 参数的语法如下：

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

部分描述：

Optional（可选）

表示参数不是必需的。如果使用该选项，则 arglist 中的后续参数都必需是可选的，而且必须都使用 Optional 关键字声明。如果使用了 ParamArray，则任何参数都不能使用 Optional。

ByVal（可选）

表示该参数按值传递。

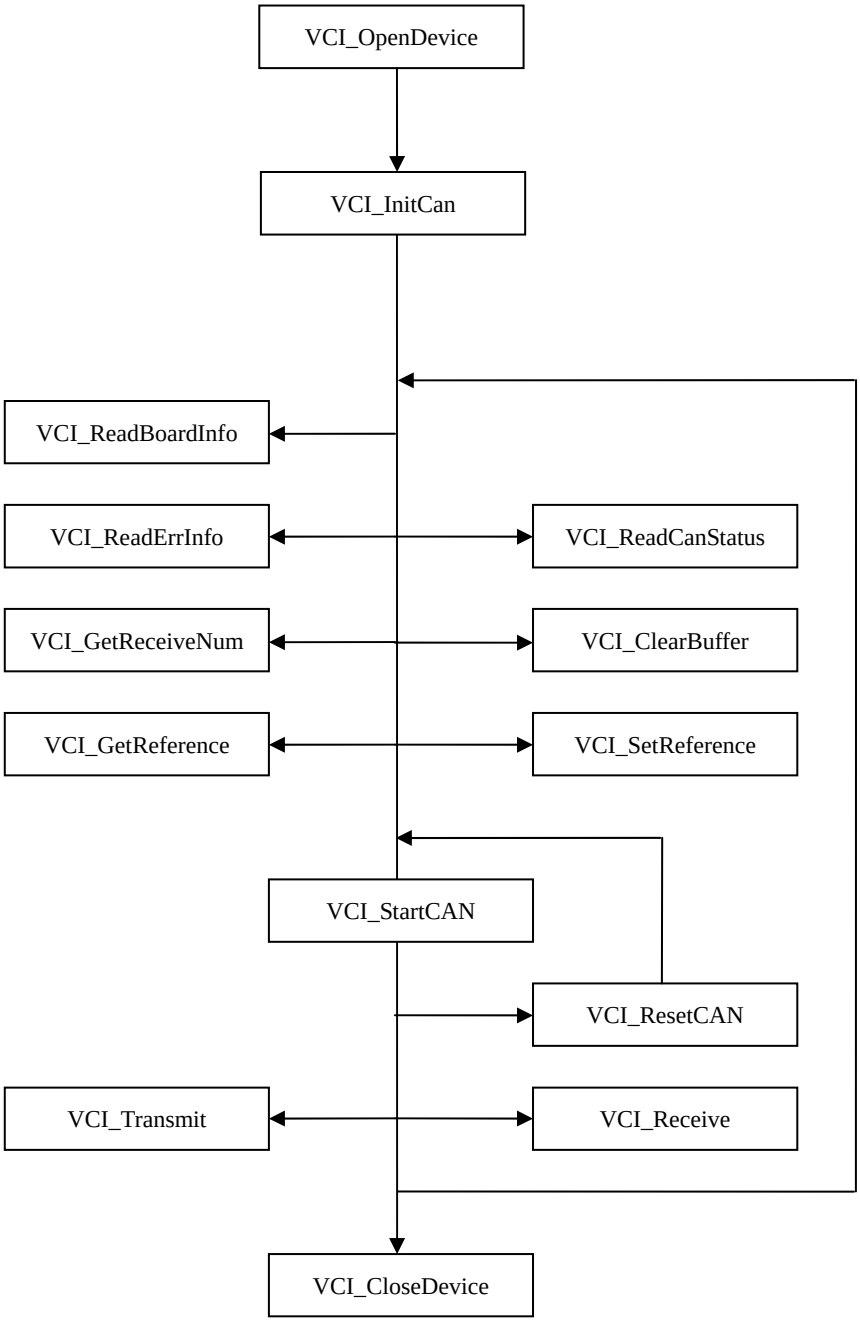
ByRef（可选）

表示该参数按地址传递。

例如：

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

2.6. 接口库函数使用流程



3. Linux下动态库的使用

3.1. 驱动程序的安装

所有驱动都在 Linux 2.4.20-8 下测试通过。

3.1.1. USBCAN驱动的安装

把 driver 目录下的 usbcan.o 文件拷贝到/lib/modules/(*)/kernel/drivers/usb 目录下，就完成了驱动的安装（其中(*)根据 Linux 版本的不同而不同，比如 Linux 版本为 2.4.20-8，则此目录的名称也为“2.4.20-8”，即跟 Linux 内核版本号相同)。

3.1.2. PCI5121 驱动的安装

把 driver 目录下的 pci51xx.o 文件拷贝到/lib/modules/(*)/kernel/drivers/char 目录下，就完成了驱动的安装（其中(*)根据 Linux 版本的不同而不同，比如 Linux 版本为 2.4.20-8，则此目录的名称也为“2.4.20-8”，即跟 Linux 内核版本号相同)。

3.2. 动态库的安装

把 dll 文件夹中的 libcontrolcan.so 文件和 kerneldlls 文件夹一起拷贝到/lib 目录，然后运行 ldconfig /lib 命令，就可以完成动态库的安装。

3.3. 动态库的调用及编译

动态库的调用是非常简单的，只需要把 dll 文件夹中的 controlcan.h 文件拷贝到你的当前工程目录下，然后用#include “controlcan.h”把 controlcan.h 文件包含到你的源代码文件中，就可以使用动态库中的函数了。

在用 GCC 编译的时候只需要添加 -lcontrolcan 选项就可以了，比如：

gcc –lcontrolcan –g –o test test.c