2016

CAN-bus 接口函数库

使用手册 V1.0.1

----北京爱泰联合科技有限公司

目录

1. 接口函数库说明及其使用

1.1 接口卡设备类型定义

接口卡的类型定义如下：

设备名称	型号	类型号
USB 接口 CAN 卡	USBCAN1	3
ExpressCard 接口 CAN	iCANEC-I	
USB 接口 CAN 卡	USBCAN2	4
PCI 接口 CAN 卡	PCICAN-9810	2
PCI 接口光纤 CAN 卡	PCICAN-F1	
PCI 接口 CAN 卡	PCICAN-9820	5
PC104+接口 CAN 卡	PC104+9820	
CPCI 接口 CAN 卡	CPCI-9820-3U	
	CPCI-9820-6U	
PCI 接口 CAN 卡	PCICAN-9840	14
以太网接口 CAN 卡	CANET-UDP	12
	CANET-TCP	17
PCIe 接口 CAN 卡	PCIeCAN-9221	24

1.2 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

1.3 函数库中的数据结构定义

1.3.1. VCI_BOARD_INFO

描述：

VCI_BOARD_INFO 结构体包含 CAN 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。

1.3.2. 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。

1.3.3. 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。

1.3.4. 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。

1.3.5. 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

1.4 接口库函数说明

1.4.1. 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，当只有一个 USBCAN 时，索引号为 0，有两个时可以为 0 或 1。

Reserved

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 = 4;     //USBCAN2
int nDeviceInd = 0;      //第一个设备
int nReserved =0;              //无意义
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;
}
```

1.4.2. 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，当只有一个 USBCAN 时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;      // USBCAN2
int nDeviceInd = 0;       // 第一个设备
BOOL bRel;

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

1.4.3. 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 USBCAN 时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

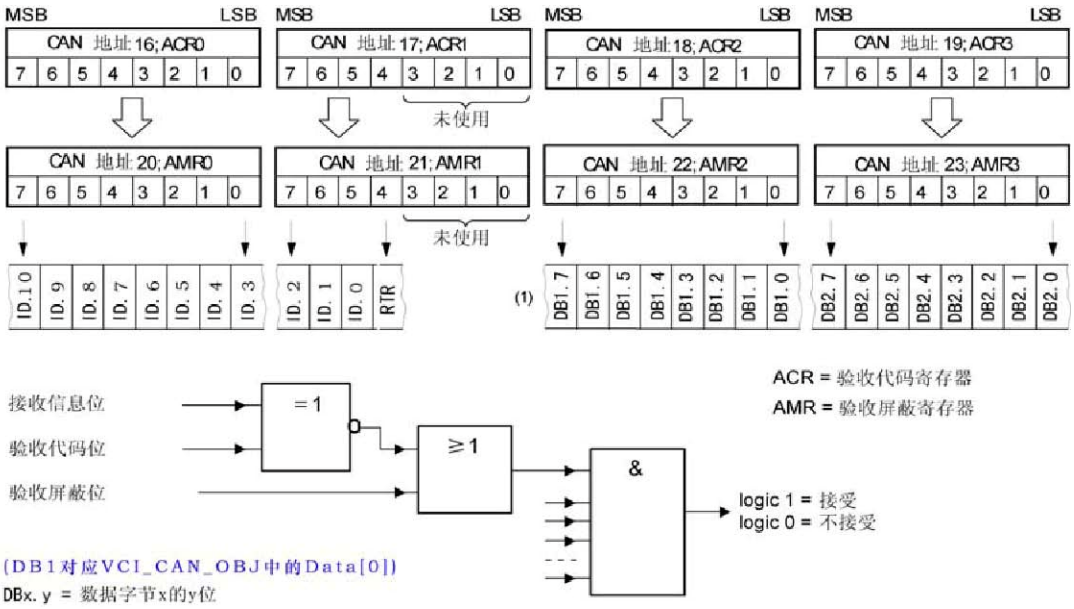
第几路 CAN。

pInitConfig

初始化参数结构

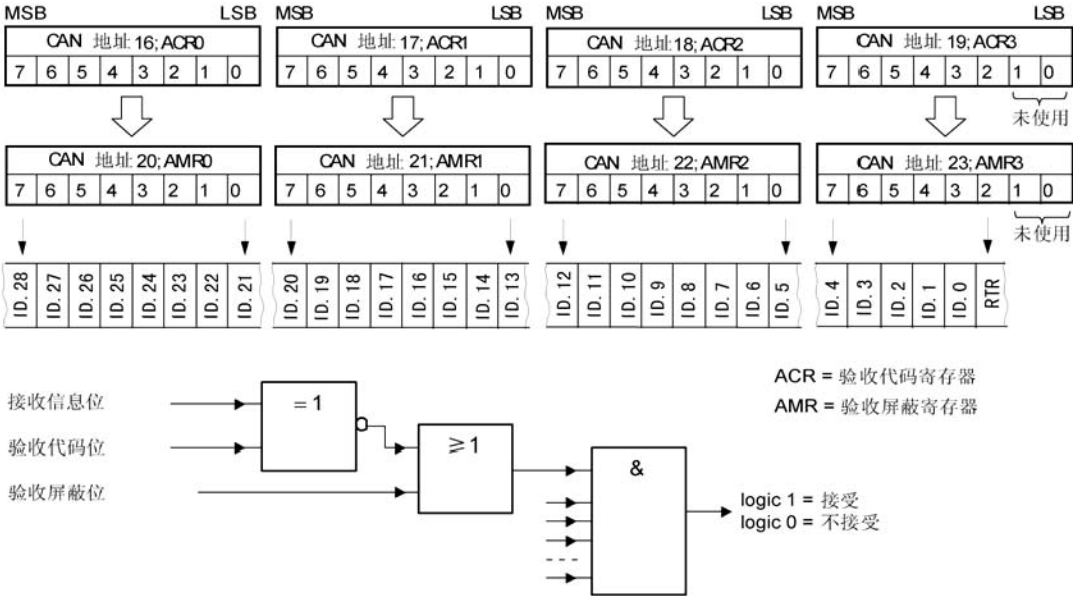
成员	功能描述
pInitConfig->AccCode	AccCode 对应 SJA1000 中的四个寄存器 ACR0，ACR1，ACR2，ACR3，其中高字节对应 ACR0，低字节对应 ACR3；AccMask 对应 SJA1000 中的四个寄存器 AMR0，AMR1，AMR2，AMR3，其中高字节对应 AMR0，低字节对应 AMR3。（请看表后说明）
pInitConfig->AccMask	
pInitConfig->Reserved	保留
pInitConfig->Filter	滤波方式，1 表示单滤波，0 表示双滤波
pInitConfig->Timing0	定时器 0
pInitConfig->Timing1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

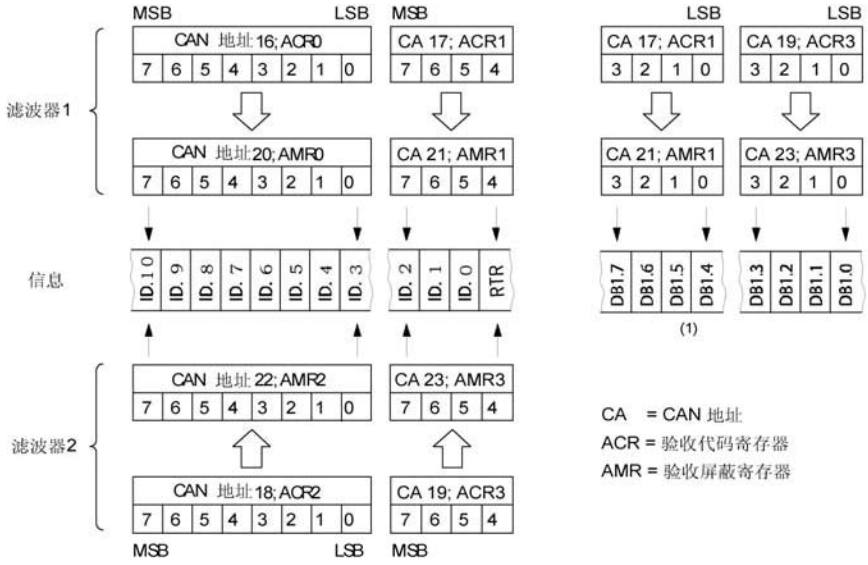


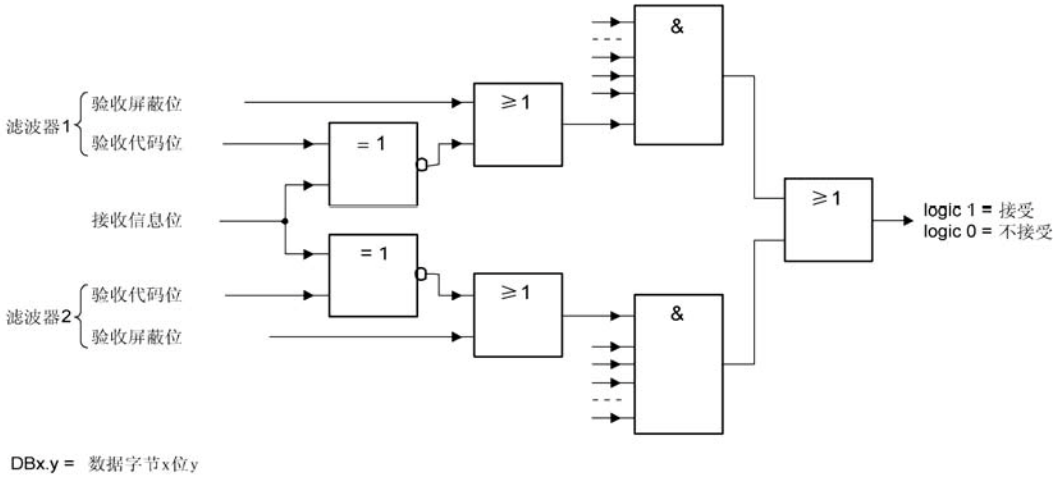
RTR 对应 VCI_CAN_OBJ 中的 RemoteFlag

当滤波方式为单滤波，接收帧为扩展帧时：

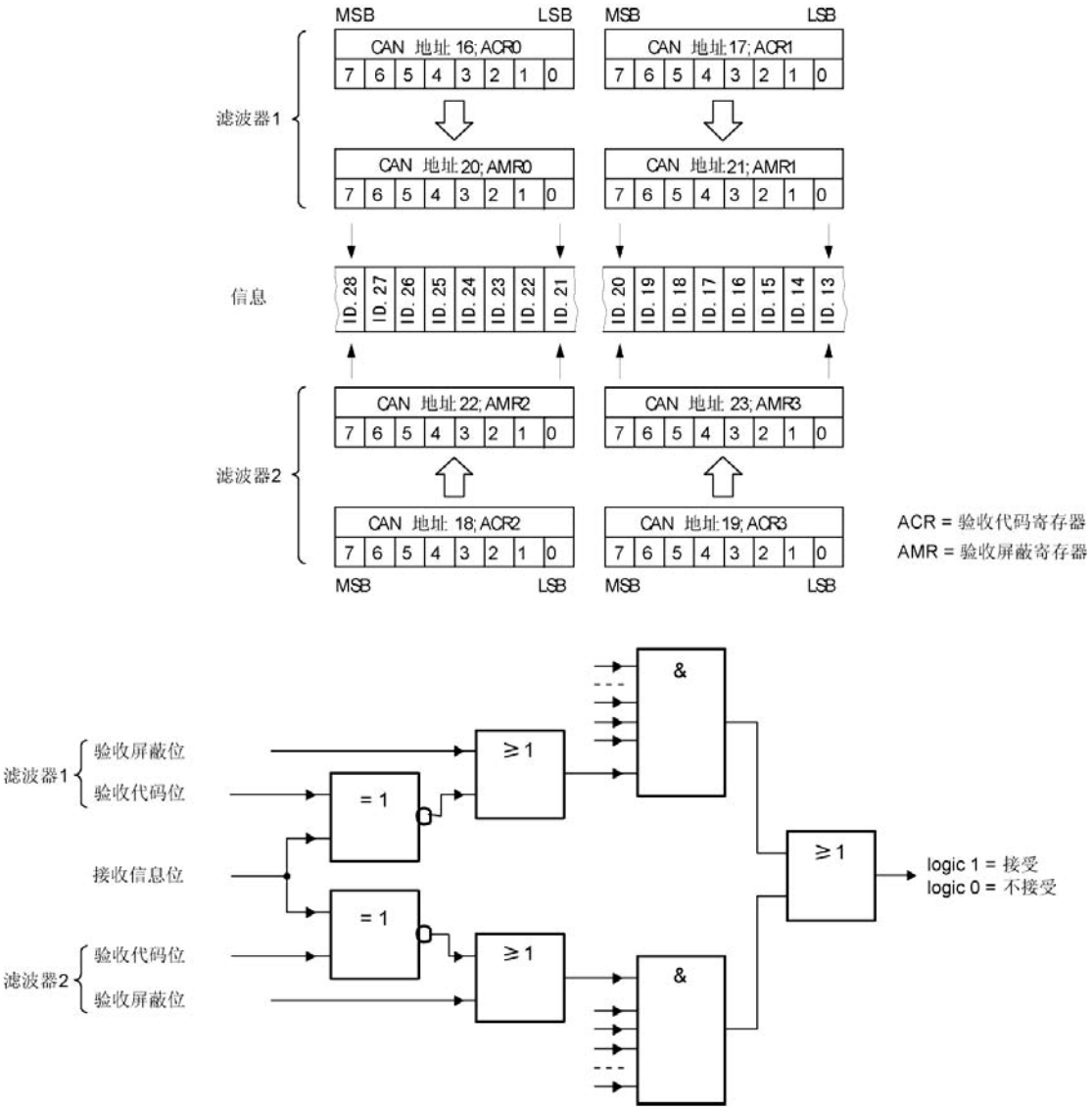


当滤波方式为双滤波，接收帧为标准帧时：





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。

示例：

```
#include "ControlCan.h"


int nDeviceType = 4;

int nDeviceInd = 0;

int nCANInd = 0;

int nReserved = 0;
```

```
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;
}
```

1.4.4. 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 USBCAN 时，索引号为 0，有两个时可以为 0 或 1。

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。

示例

```
#include "ControlCan.h"


int nDeviceType = 4;      // USBCAN2
int nDeviceInd = 0;
int nCANInd = 0;
VCI_INIT_CONFIG vic;
VCI_BOARD_INFO vbi;
DWORD dwRel;


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

1.4.5. 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 USBCAN 时，索引号为 0，有两个时可以为 0 或 1。

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。）

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 = 4;      // USBCAN
int nDeviceInd = 0;       // 第一个设备
int nCANInd = 0;
VCI_ERR_INFO vei;
DWORD dwRel;

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

1.4.6. 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。

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。

示例：

```
#include "ControlCan.h"

int nDeviceType = 4;
int nDeviceInd = 0;
int nCANInd = 0;
VCI_INIT_CONFIG vic;
VCI_CAN_STATUS vcs;
DWORD dwRel;

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

1.4.7. 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，只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 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。

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 = 4;
int nDeviceInd = 0;
int nCANInd = 0;
BYTE info[14];
DWORD dwRel;

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

1.4.8. 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，只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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。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。

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

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 = 4;
```

```
int nDeviceInd = 0;
int nCANInd = 0;
BYTE baud;
DWORD dwRel;

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

1.4.9. 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，当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;

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

1.4.10. 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，当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;

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

1.4.11. 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，只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;
int nDeviceInd = 0;
int nCANInd = 0;
int nReserved =0;
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;
}
```

1.4.12. 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，只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;

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

1.4.13. 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，只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;
int nDeviceInd = 0;
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);
```

1.4.14. 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，只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

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 = 4;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;
VCI_CAN_OBJ vco[100];

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

1.5 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。

1.5.1. 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

1.5.2. 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。

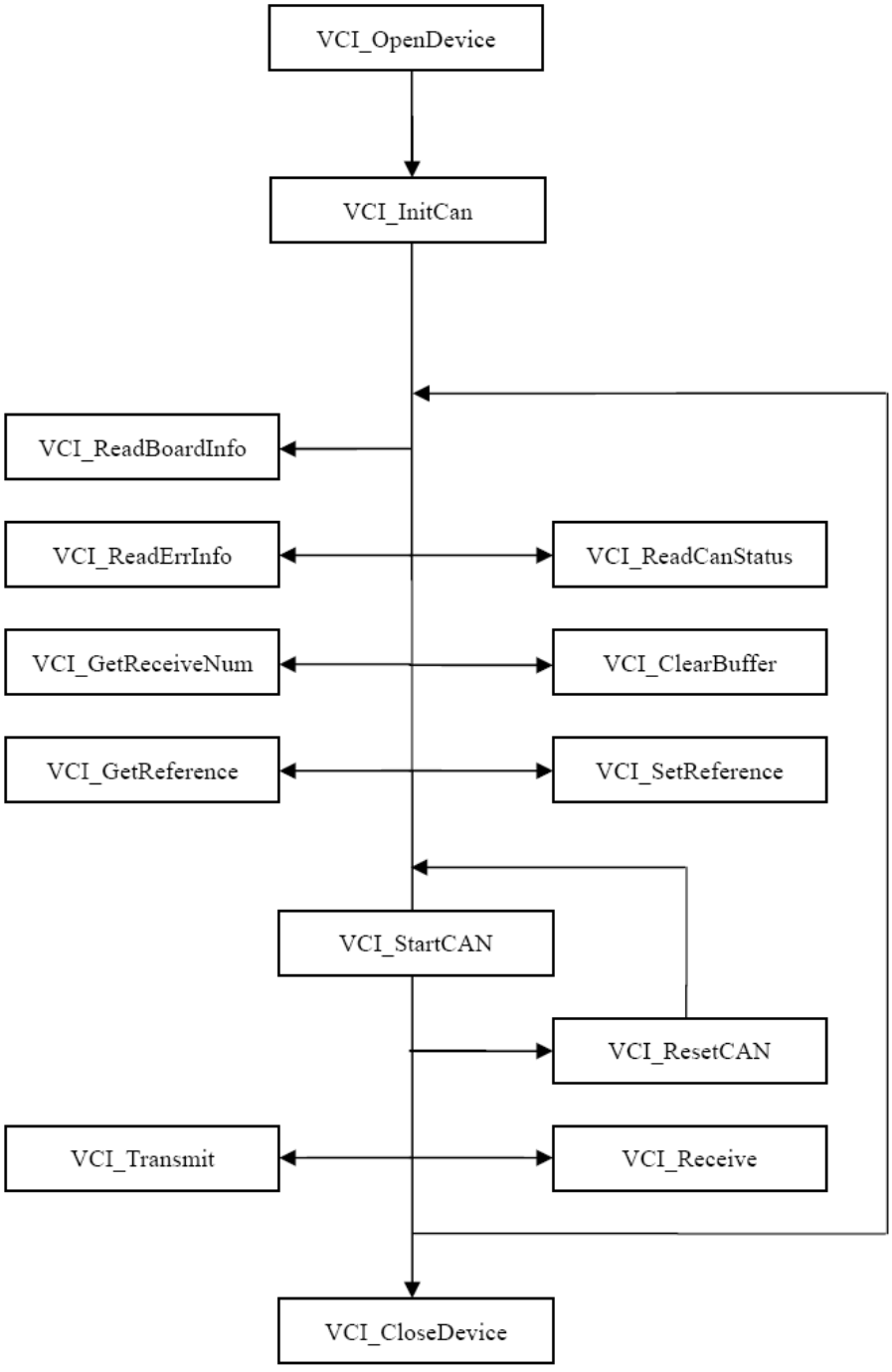
ByRef（可选）

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
```

1.6 接口库函数使用流程



2. Linux 下动态库的使用

2.1. 驱动程序的安装

所有驱动都在 Linux 2.6 下测试通过。

2.1.1. 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 2.6.20-8，则此目录的名称也为“2.6.20-8”，即跟 Linux 内核版本号相同）。

2.2. 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。

2.3. 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