Technical Note

CAN 测试软件与接口函数使用手册

CAN 接口卡系列产品

TN01010101 V3.01 Date:2014/02/15 使用手册

类别	内容
关键词	CANTest 通用测试软件、CAN 接口函数库使用
摘　要	本软件可适用于广州致远电子股份有限公司出品的各种 CAN 接口卡。CANTest 测试软件可进行数据收发、查询等基本传输功能。是 CAN 总线测试的必备软件。CAN 接口函数库是提供给用户进行上位机二次开发，可以自行编程进行数据收发、处理等。

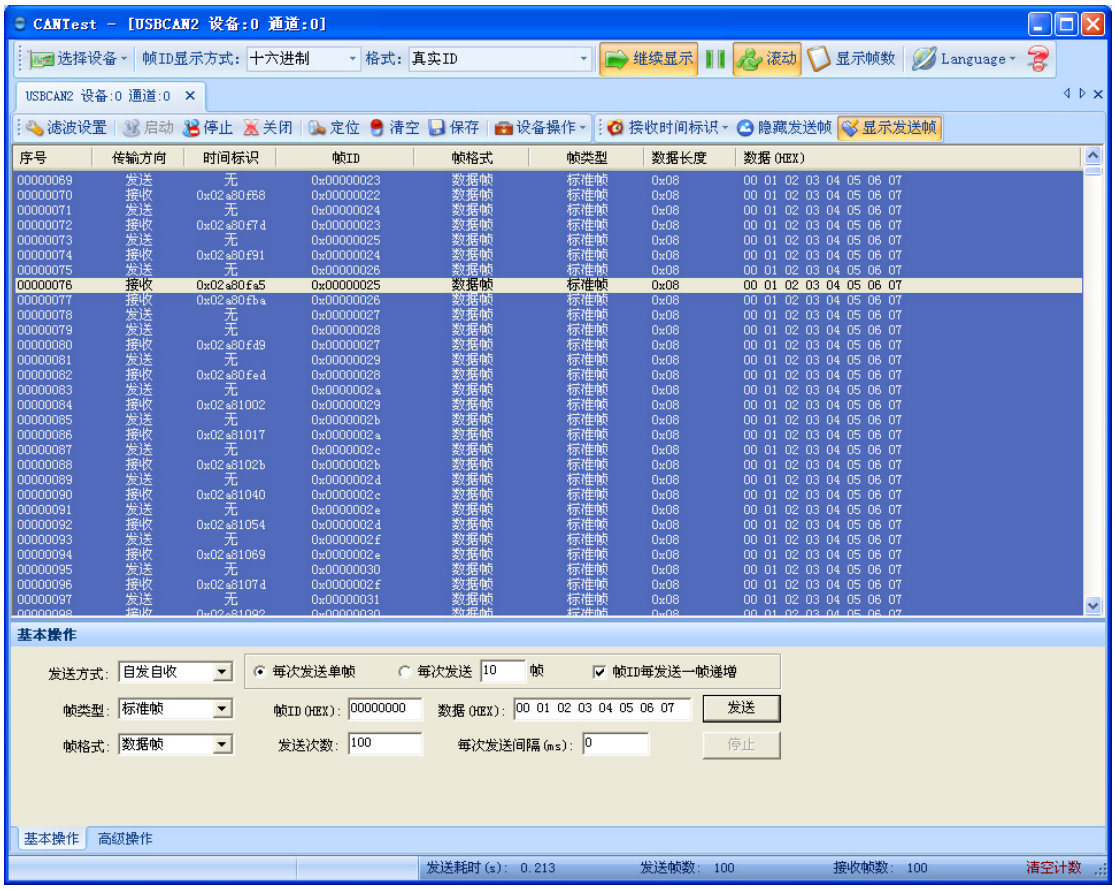
修订历史

版本	日期	原因
V1.00	2004/10/01	创建文档
V2.00	2008/09/01	修改架构
V3.00	2013/08/15	更换模版，去除已经停产的产品
V3.01	2014/02/15	修改错误

目 录

1. 测试软件使用说明

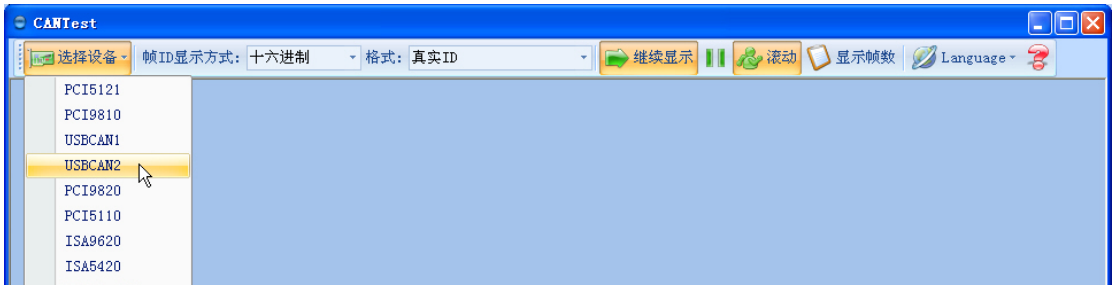
CAN-bus 通用测试软件是一个专门用来对所有的 ZLGCAN 系列板卡进行测试的软件工具，此软件操作简单，容易上手，通过运用此软件可以非常方便的对板卡进行测试，从而熟悉板卡的性能，其主界面如下：



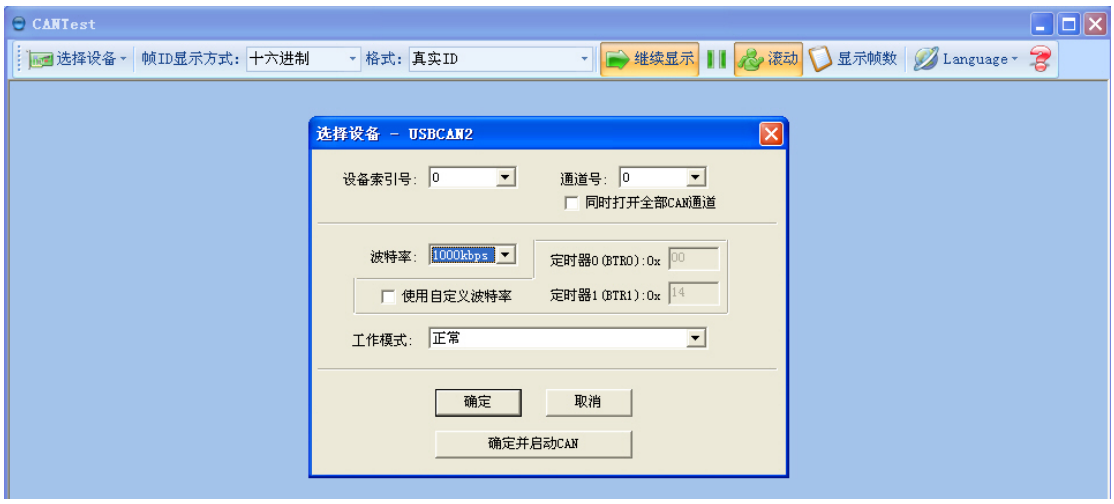
1.1. 设备操作

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：



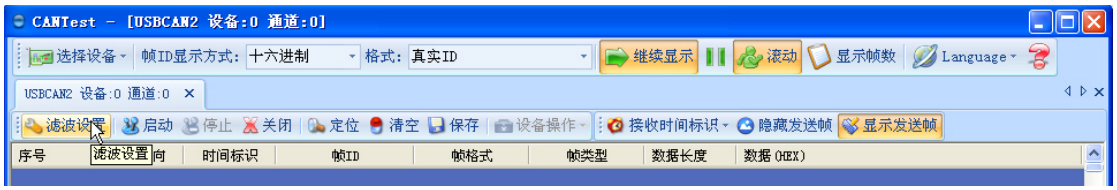
此时会弹出“选择设备”对话框：



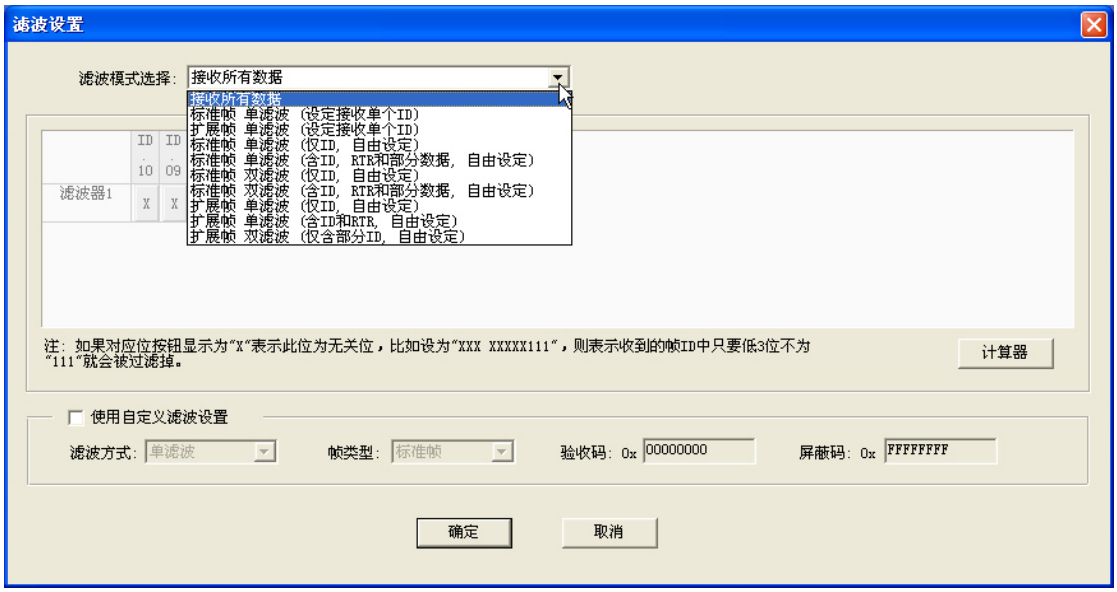
在这个对话框中您可以选择您要打开的设备索引号和 CAN 通道，以及设置 CAN 的初始化参数，然后点“确定”按钮来打开设备操作窗口（或者也可以点击“确定并启动 CAN”按钮打开设备操作窗口并自动打开设备和启动 CAN 通道）。

1.1.2. 滤波设置

接着，设备操作窗口中可以点击“滤波设置”按钮进行滤波设置（如果不需要设置滤波，可以略过此步骤）：



此时会弹出“滤波设置”对话框：



在其中先选择滤波模式，然后通过设定滤波器来设置需要过滤的 CAN 帧。

1.1.3. 启动CAN

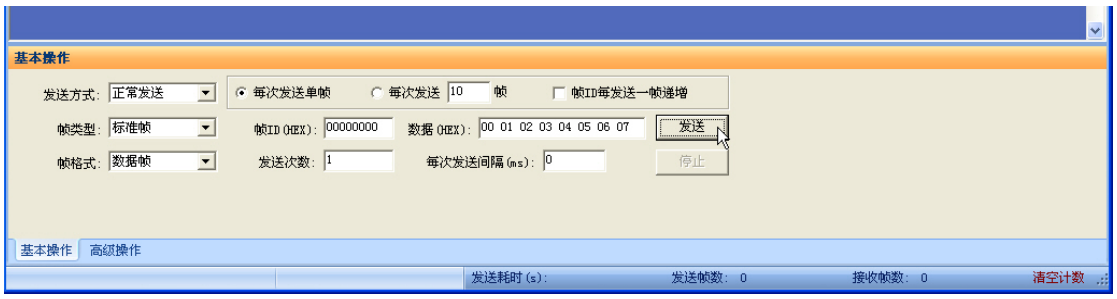
点击“启动”按钮启动 CAN 通道，此时接收到的 CAN 数据将会自动在数据列表中显示：

1.1.4. 获取设备信息

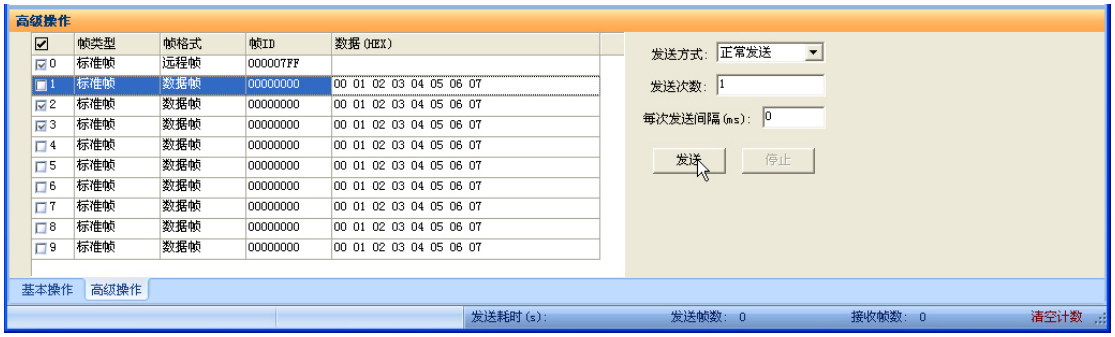
在启动 CAN 通道后，您可以选择“设备操作”菜单中的“设备信息”选项来获得当前设备的详细信息。

1.1.5. 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）：

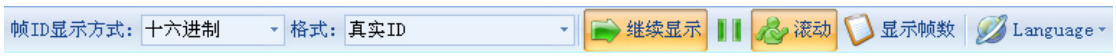


您还可以点击“高级操作”标签进入高级操作页面，在此页面您可以设置每次发送多个不同的 CAN 帧（最多可设置 10 帧）：



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：



1.2.1. 帧ID显示方式

帧 ID 有三种显示方式：二进制、十进制和十六进制，可根据需要自行设定。

1.2.2. 帧ID显示格式

帧 ID 显示格式有两种：真实 ID（靠右对齐）和兼容 SJA1000 格式（靠左对齐）。

1.2.3. 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。

1.2.4. 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。

1.2.5. 滚动

选择此选项后，当前数据列表中的最后一行总是可见。

1.2.6. 显示帧数

设定数据列表的显示帧数。

1.2.7. Language

选择语言。

2. 接口函数库说明及其使用

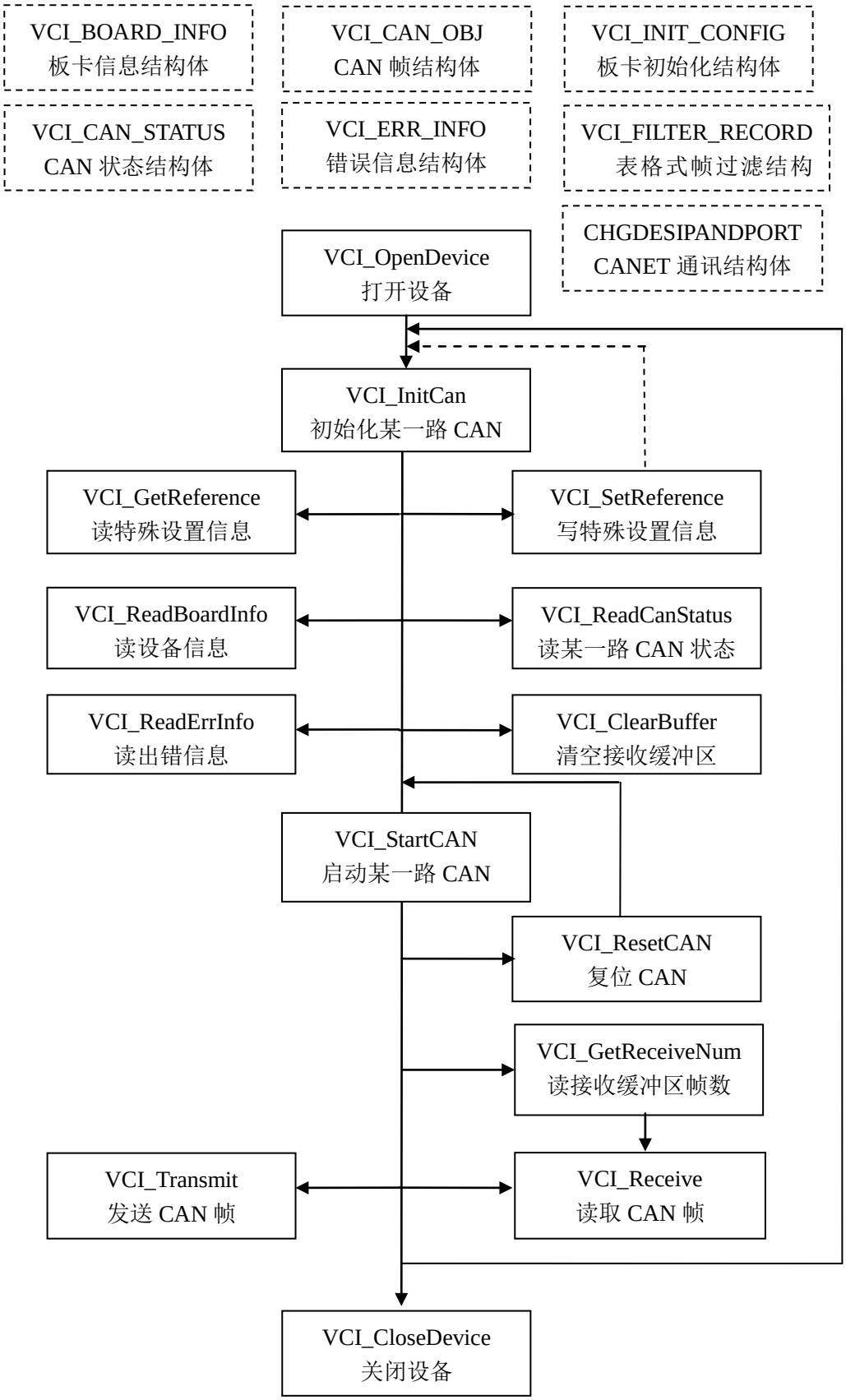
2.1 接口卡设备类型定义

各个接口卡的设备类型定义如表 2.1 所示，**注意本文档为 V3.00 版本，去除了已经停产型号的函数库说明，如果需要查看已经停产型号函数说明，请参见 V2.92 版本。**

表 2.1 接口卡的类型定义

产品型号	动态库中的设备名称	类型号
PCI-5121（已经停产）	PCI5121	1
PCI-9810I	PCI9810	2
USBCAN-I/I+	USBCAN1	3
USBCAN-II/II+	USBCAN2	4
PCI-9820	PCI9820	5
CAN232（已经停产）	CAN232	6
PCI-5110（已经停产）	PCI5110	7
CANmini（已经停产）	CANlite(CANmini)	8
ISA-9620（已经停产）	ISA9620	9
ISA-5420（已经停产）	ISA5420	10
PC104-CAN（已经停产）	PC104-CAN	11
CANET-100T/200T、CANET-E/2E+的 UDP 工作方式	CANET-UDP	12
DN-PCI9810（已经停产）	DNP9810	13
PCI-9840I	PCI9840	14
PC104-CAN2I	PC104-CAN2	15
PCI-9820I	PCI9820I	16
CANET-100T/200T 的 TCP 工作方式	CANET-TCP	17
PEC-9920（已经停产）	PEC-9920	18
PCIe-9220（已经停产）	PCIE-9220	18
PCI-5010-U	PCI-5010-U	19
USBCAN-E-U	USBCAN-E-U	20
USBCAN-2E-U	USBCAN-2E-U	21
PCI-5020-U	PCI-5020-U	22
EG20T-CAN（已经停产）	EG20T-CAN	23
PCIe-9221	PCIE-9221	24

2.2 接口库函数使用流程



2.3 驱动的特色与工作原理

本函数库涵盖了广州致远电子生产的所有 CAN 接口卡设备。为了保证高效稳定工作。工作原理如下：

（1）接收采用驱动库自动中断，压入缓冲区的方式，保证不丢帧。用户只需要调用 VCI_Receive 接收函数，从缓冲区中集中提取数据（可多帧提取），并且可以设置阻塞时间，避免无数据时线程死等。避免客户来操作中断，导致 CAN 卡丢帧、PC 死机或者蓝屏。

（2）13 万帧接收缓冲区。即调用 VCI_OpenDevice 后，即在内存中开辟 13 万帧缓冲区，即使客户不调用接收函数，也会自动接收并压入缓冲区，避免丢帧。

（3）发送可实现多帧发送。即调用一次函数，实现多帧发送，节约 PC 资源。发送返回实际成功的帧数。并且可以设置发送重试阻塞超时。默认是 1.5 秒-4 秒。

（4）丰富的错误代码。资深用户可以通过调用查看状态与错误寄存器，获得目前 CAN 卡和 CAN 总线的状态。分析后，制定正确的通讯策略

（5）灵活的接口移植性。所有的致远电子 CAN 接口卡均使用同一套动态库，客户可以在不改变主体程序的情况下，只需修改设备类型和特殊设置，即可实现程序的移植。比如只要将设备类型从 4 改成 16，即可很方便的将 USBCAN-II 的程序变成到 PCI-9820I 的程序。

（6）多平台兼容。强大的研发和维护团队，紧跟世界潮流，不但支持主流的 32bit 或者 64bit 的 WIN2000、WINXP、WIN7、WIN8 等，还定制支持各种内核的 LINUX 系统 VXWorks.

（7）丰富的例程。任何编程环境。只要您想得到的，我们就能提供。

2.4 错误码定义

表 2.2 错误码定义

名称	值	描述
CAN 错误码		
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_CAN_BUSOFF	0x00000020	CAN 控制器总线关闭
通用错误码		
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
CANET 错误码		
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
ERR_REF_TYPE_ID	0x00030001	SetReference 或 GetReference 是传递的 RefType 是不存在
ERR_CREATE_SOCKET	0x00030002	创建 Socket 时失败
ERR_OPEN_CONNECT	0x00030003	打开 socket 的连接时失败，可能设备连接已经存在
ERR_NO_STARTUP	0x00030004	设备没启动
ERR_NO_CONNECTED	0x00030005	设备无连接
ERR_SEND_PARTIAL	0x00030006	只发送了部分的 CAN 帧
ERR_SEND_TOO_FAST	0x00030007	数据发得太快，Socket 缓冲区满了

2.5 函数库中的数据结构定义

2.5.1 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。

2.5.2 VCI_CAN_OBJ

描述

VCI_CAN_OBJ 结构体是 CAN 帧结构体，即 1 个结构体表示一个帧的数据结构。在发送函数 VCI_Transmit 和接收函数 VCI_Receive 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

帧 ID。32 位变量，数据格式为靠右对齐。

TimeStamp

设备接收到某一帧的时间标识。只有智能卡才有时间标示，如 USBCAN 系列与 PCI-5010/20。时间标示从 CAN 卡上电开始计时，计时单位为 0.1ms。

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（发送失败会自动重发，重发最长时间为 1.5-3 秒）；

=1 时为单次发送（只发送一次，不自动重发）；

=2 时为自发自收（自测试模式，用于测试 CAN 卡是否损坏）；

=3 时为单次自发自收（单次自测试模式，只发送一次）。

只在此帧为发送帧时有意义。

RemoteFlag

是否是远程帧。=0 时为为数据帧，=1 时为远程帧（数据段空）。

ExternFlag

是否是扩展帧。=0 时为标准帧（11 位 ID），=1 时为扩展帧（29 位 ID）。

DataLen

数据长度 DLC (<=8)，即 CAN 帧 Data 有几个字节。约束了后面 Data[8]中的有效字节。

Data[8]

CAN 帧的数据。由于 CAN 规定了最大是 8 个字节，所以这里预留了 8 个字节的空间，受 DataLen 约束。如 DataLen 定义为 3，即 Data[0]、Data[1]、Data[2]是有效的。

Reserved

系统保留。

2.5.3 VCI_CAN_STATUS

描述

VCI_CAN_STATUS 结构体包含 CAN 设备中的 CAN 控制器状态信息。结构体将在 VCI_ReadCanStatus 函数中调用时，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，读操作会清除中断。

regMode

CAN 控制器模式寄存器值。

regStatus

CAN 控制器状态寄存器值。

regALCapture

CAN 控制器仲裁丢失寄存器值。

regECCapture

CAN 控制器错误寄存器值。

regEWLimit

CAN 控制器错误警告限制寄存器值。默认为 96。

regRECounter

CAN 控制器接收错误寄存器值。为 0-127 时，为错误主动状态，为 128-254 为错误被动状态，为 255 时为总线关闭状态。

regTECounter

CAN 控制器发送错误寄存器值。为 0-127 时，为错误主动状态，为 128-254 为错误被动状态，为 255 时为总线关闭状态。

Reserved

系统保留。

2.5.4 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。(对应着 2.2 的错误码定义)

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。

2.5.5 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，即初始化之前，要先填好这个结构体变量。

```
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。SJA1000 的帧过滤验收码。对经过屏蔽码过滤为“有关位”进行匹配，全部匹配成功后，此帧可以被接收。否则不接收。详见 VCI_InitCan。

AccMask

屏蔽码。SJA1000 的帧过滤屏蔽码。对接收的 CAN 帧 ID 进行过滤，对应位为 0 的是“有关位”，对应位为 1 的是“无关位”。屏蔽码推荐设置为 0xFFFFFFFF，即全部接收。

Reserved

保留。

Filter

滤波方式。=1 表示单滤波，=0 表示双滤波

Timing0

波特率定时器 0（BTR0）。设置值见下表。

Timing1

波特率定时器 1（BTR1）。设置值见下表。

Mode

模式。=0 表示正常模式（相当于正常节点），=1 表示只听模式（只接收，不影响总线）。

备注

该结构体的详细说明见表 2.3。

Timing0 和 Timing1 用来设置 CAN 波特率，几种常见的波特率（采样点 87.5%，SJW 为 0）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

注意：当设备类型为 PCI-5010-U、PCI-5020-U、USBCAN-E-U、 USBCAN-2E-U 时，波特率和帧过滤不在这里设置，具体操作见 VCI_SetReference 说明。

2.5.6 CHGDESIPANDPORT

描述

CHGDESIPANDPORT 结构体用于装载更改 CANET_UDP 与 CANET_TCP 的目标 IP 和端口的必要信息。此结构体在 CANETE_UDP 与 CANET_TCP 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。

2.5.7 VCI_FILTER_RECORD

描述

当设备类型为 PCI-5010-U、PCI-5020-U、USBCAN-E-U、USBCAN-2E-U 时，定义了滤波器的滤波范围 VCI_FILTER_RECORD 结构体。结构体在 VCI_SetReference 函数中被填充。

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

成员

ExtFrame

过滤的帧类型标志，为 1 代表要过滤的为扩展帧，为 0 代表要过滤的为标准帧。

Start

滤波范围的起始帧 ID

End

滤波范围的结束帧 ID

2.6 接口库函数说明

2.6.1 VCI_OpenDevice

描述

此函数用以打开设备。注意一个设备只能打开一次。

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

参数

DevType

设备类型号。对应不同的产品型号。见表 2.1。

DevIndex

设备索引号。比如当只有一个 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

Reserved

保留参数，通常为 0。（特例：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。）

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 = 24;     /* PCIe-9221 */
int nDeviceInd = 0;      /* 索引号0 */
int nReserved = 0;
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;
}
```

2.6.2 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。对应已经打开的设备。

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 = 24;    // PCIe-9221
int nDeviceInd = 0;      // 索引号0
BOOL bRel;

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

2.6.3 VCI_InitCan

描述

此函数用以初始化指定的 CAN 通道。有多个 CAN 通道时，需要多次调用。（当设备类型为 PCI-5010-U、PCI-5020-U、USBCAN-E-U、USBCAN-2E-U 时，必须在调用此函数之前调用 VCI_SetReference 对波特率进行设置）。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

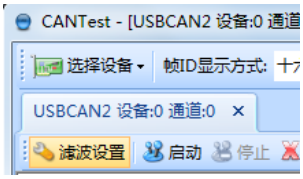
pInitConfig

初始化参数结构，为一个 VCI_INIT_CONFIG 结构体变量。（特例：当设备类型为 PCI-5010-U、PCI-5020-U、USBCAN-E-U、USBCAN-2E-U 时，对滤波和波特率的设置应该放到 VCI_SetReference 里设置，这里 pInitConfig 中的成员只有 Mode 需要设置，其他的 6 个成员可以忽略，具体设置见 VCI_SetReference 说明；）。

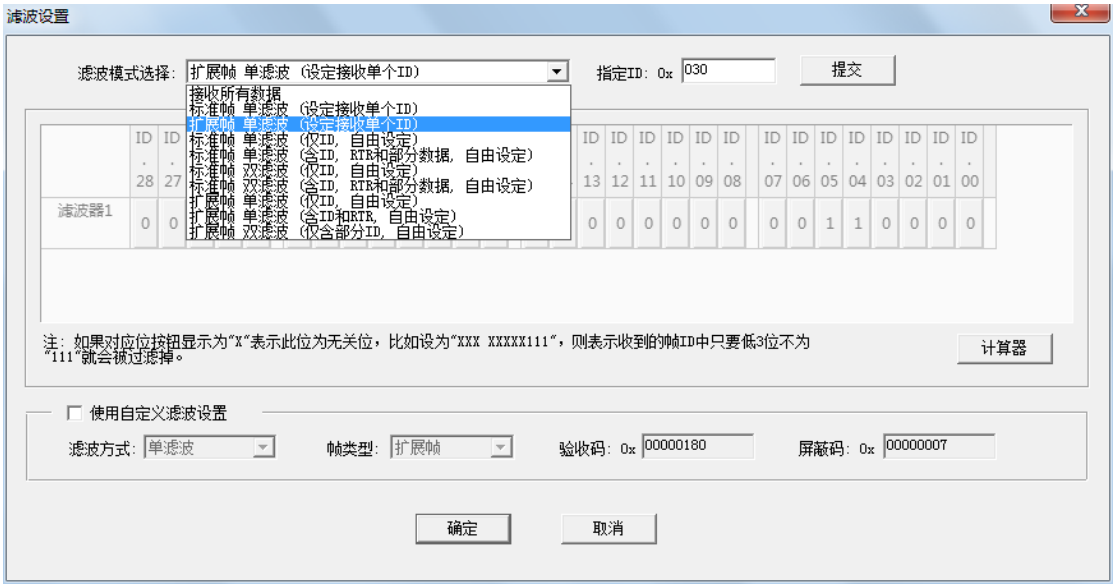
表 2.3 初始化参数结构

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	波特率定时器 0，详见 VCI_INIT_CONFIG
pInitConfig->Timing1	波特率定时器 1，详见 VCI_INIT_CONFIG
pInitConfig->Mode	模式，0 表示正常模式，1 表示只听模式

AccCode 与 AccMask 配置值请使用测试软件 CANtest 中“滤波设置”来计算：



比如：若只想接收 ID 为 0x030 的扩展帧，则如下图所示设置。点击提交，计算出来的验收码 0x00000180 即为 AccCode 值，屏蔽码 0x00000007 即为 AccMask 值。



VCI_InitCan 的返回值

为 1 表示操作成功，0 表示操作失败。（特例：在 CANET 中无需调用，调用会返回 1）

示例

```
#include "ControlCan.h"

int nDeviceType = 24;     // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;
VCI_INIT_CONFIG vic;
DWORD dwRel;
// 中间略去打开设备和填充vic结构体的代码
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;
}
```

2.6.4 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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

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。（特例：在 CANET 中无此函数，调用会返回 0，并且错误码填充为 ERR_CMDFAILED）

示例

```
#include "ControlCan.h"

int nDeviceType = 24;     // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
VCI_BOARD_INFO vbi;
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_ReadBoardInfo(nDeviceType, nDeviceInd, nCANInd, &vbi);
```

2.6.5 VCI_ReadErrInfo

描述

此函数用以获取 CAN 卡发生的最近一次错误信息。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。（特例：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：(CANET 相关错误代码，见 2.3 错误码定义)

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 = 24;    // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
VCI_ERR_INFO vei;
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_ReadErrInfo(nDeviceType, nDeviceInd, nCANInd, &vei);
```

2.6.6 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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

pCANStatus

用来存储 CAN 状态的 VCI_CAN_STATUS 结构体指针。

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，调用会返回 0，获取错误码 ERR_CMDFAILED）

示例

```
#include "ControlCan.h"

int nDeviceType = 24;    // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
VCI_INIT_CONFIG vic;
VCI_CAN_STATUS vcs;
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_ReadCANStatus(nDeviceType, nDeviceInd, nCANInd, &vcs);
```

2.6.7 VCI_GetReference

描述

此函数用以获取设备的相应参数（主要是 CANET 的相关参数）。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

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_GetReference 这个函数是用来针对各个不同设备的一些特定操作的。

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

(2) 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 CANtest 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 CANtest 测试工具则需要工作在客户端模式。

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 CANET 工作在服务器模式时使用)	读取已经连接上的 CANET-TCP 的 IP 地址。例如： char szip[20]; VCI_GetReference(VCI_CANETTCP,0,0,0,(PVOID)szip); 如果此函数调用成功,则在 szip 中返回 CANET_TCP 的地址。
1	长度为 4 个字节，存储读取出来的 CANET-TCP 的工作端口。(当 CANET 工作在服务器模式时有效)	读取 CANET-TCP 的工作端口。例如： DWORD port; VCI_GetReference(VCI_CANETTCP,0,0,1,(PVOID)&port); 如果此函数调用成功,则在 port 中返回 CANET-TCP 的工作端口。
2	长度为 4 个字节，存储本机上的 TCP 工作端口。(CANET 在服务器和客户端模式时同时有效)	读取本机端口。例如： DWORD port; VCI_GetReference(VCI_CANETTCP,0,0,2,(PVOID)&port); 如果此函数调用成功，则在 port 中返回本机上的工作端口。
4	长度为 4 个字节，存储本机的 TCP 工作模式。	读取本机的工作模式，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。例如： DWORD iType ; VCI_GetReference(VCI_CANETTCP,0,0,4,(PVOID)&iType); 如果此函数调用成功，读取本机工作模式。
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。(当 CANET 工作在客户端模式下有效)	读取连接到本机上客户端 CANET-TCP 数量。例如： DWORD iCount; VCI_GetReference(VCI_CANETTCP,0,0,5,(PVOID)&iCount);

6	使用 REMOTE_CLIENT 结构，获取一个连接的信息。(当 CANET 工作在客户端模式下有效)	当有客户端连接到本机时，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_CANETTCP,0,0,6,(PVOID)&cli);
		如果此函数调用成功，则在 cli 存储客户端的信息。

REMOTE_CLIENT 结构

```
typedef struct tagRemoteClient{
    int iIndex; // 连接的客户端索引号
    DWORD port; // 连接的客户端工作端口
```

```
        HANDLE hClient;
        char szip[32]; // 连接的客户端IP地址
    }REMOTE_CLIENT;
```

示例

```
#include "ControlCan.h"

int nDeviceType = 12;     // CANET-UDP
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
char szip[20];
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_GetReference(nDeviceType, nDeviceInd, nCANInd,0,(PVOID)szip);
```

2.6.8 VCI_SetReference

描述

此函数用以设置 CANET 与 PCI-5010-U/PCI-5020-U/USBCAN-E-U/ USBCAN-2E-U 等设备的相应参数，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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

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 这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。

(1) 当设备类型为 PCI-5010-U、PCI-5020-U、USBCAN-E-U、USBCAN-2E-U 时：

RefType	pData	功能描述
0	指向 DWORD 类型的指针，该 DWORD 变量的值为写入波特率寄存器 BTR 的值。 一些标准波特率设置如下： 0x060003 ： 1000Kbps 0x060004 ： 800Kbps 0x060007 ： 500Kbps 0x1C0008 ： 250Kbps 0x1C0011 ： 125Kbps 0x160023 ： 100Kbps 0x1C002C ： 50Kbps 0x1600B3 ： 20Kbps 0x1C00E0 ： 10Kbps 0x1C01C1 ： 5Kbps	**若用户设置为其他值，有可能工作不正常。如果需要列表中没有列出的波特率值，用户应咨询致远电子 CAN-bus 技术支持工程师 020-22644381 或者发送邮件到：CANscope@zlg.cn），计算出合适的波特率值。** （注意：CAN 网络最大通讯波特率不应该超过 1000Kbps，所以对波特率的设置不能超过此值，否则设置失败） **必须在调用 VCI_InitCan 之前调用本函数在这里设置通讯的波特率。**

1	指向 VCI_FILTER_RECORD 结构的指针	填充 CAN 滤波器的滤波表格（每添加一条记录调用本函数一次）用于这些类型设备的帧接收过滤。应该**在调用 VCI_InitCan 函数之后用 VCI_SetReference 函数进行设置**详操作见光盘例程。
2	NULL,可忽略	按滤波表格中的设置启动滤波，调用即启用。应该**在调用 VCI_InitCan 函数之后用 VCI_SetReference 函数进行设置**
3	NULL,可忽略	清除滤波
4	指向 DWORD 类型的指针，该 DWORD 变量的值为发送的超时时间，单位为 ms	设置发送重发的超时时间，单位为 ms，如果没有调用本函数进行设置，默认超时为 4000ms。建议用户在这里设置超时时间不应小于 1500ms，否则可能导致 VCI_Transmit 总是返回失败。

(2) 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 地址例如： char szip[20]; VCI_SetReference(VCI_CANETUDP,0,0,0,(PVOID)szip);
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(VCI_CANETUDP,0,0,1,(PVOID)&port);

(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 CANtest 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 CANtest 测试工具则需要工作在客户端模式。

RefType	pData	功能描述
0	字符串首指针，用来存储所指定操作的 CANET-TCP 的 IP 地址。（当 CANET 工作在服务器模式时使用）	设置所要操作的 CANET-TCP 的 IP 地址。例如： char szip[20]; VCI_SetReference(VCI_CANETTCP,0,0,0,(PVOID)szip);
1	长度为 4 个字节，存储所指定操作的 CANET-TCP	设置所要操作的 CANET-TCP 的工作端口。例如： DWORD port;

	的工作端口。(当 CANET 工作在服务器模式时使用)	VCI_SetReference(VCI_CANETTCP,0,0,1,(PVOID)&port);
2	长度为 4 个字节，存储本机上的 TCP 工作端口。(CANET 在服务器和客户端模式时同时有效)	设置本机 TCP 端口。例如： DWORD port; VCI_SetReference(VCI_CANETTCP,0,0,2,(PVOID)&port);
4	长度为 4 个字节，存储本机的 TCP 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。例如： DWORD iType = 0; VCI_SetReference(VCI_CANETTCP,0,0,4,(PVOID)&iType);

7	使用 REMOTE_CLIENT 结构，删除一个连接。(当 CANET 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_CANETTCP,0,0,7,(PVOID)&cli);

REMOTE_CLIENT 结构

```
typedef struct tagRemoteClient{
    int iIndex; // 连接的客户端索引号
    DWORD port; // 连接的客户端工作端口
    HANDLE hClient;
    char szip[32]; // 连接的客户端IP地址
}REMOTE_CLIENT;
```

示例

```
#include "ControlCan.h"

int nDeviceType = 12;    // CANET-UDP
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
DWORD port=4001;         // CANET-UDP的工作端口
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_SetReference(nDeviceType, nDeviceInd, nCANInd,1,(PVOID)&port);
```

2.6.9 VCI_StartCAN

描述

此函数用以启动 CAN 卡的某一个 CAN 通道。有多个 CAN 通道时，需要多次调用。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

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 = 24;    // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
DWORD dwRel;
VCI_INIT_CONFIG vic;
// 中间略去其他函数代码
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;
}
```

2.6.10 VCI_ResetCAN

描述

此函数用以复位 CAN。主要用与 VCI_StartCAN 配合使用，无需再初始化，即可恢复 CAN 卡的正常状态。比如当 CAN 卡进入总线关闭状态时，可以调用这个函数。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

返回值

为 1 表示操作成功，0 表示操作失败。（注：在 CANET-TCP 中将会断开网络，需要重新 VCI_StartCAN 才能使用）

示例

```
#include "ControlCan.h"

int nDeviceType = 24;     // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
DWORD dwRel;

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

2.6.11 VCI_GetReceiveNum

描述

此函数用以获取指定 CAN 通道的接收缓冲区中，接收到但尚未被读取的帧数量。主要用途是配合 VCI_Receive 使用，即缓冲区有数据，再接收。用户无需一直调用 VCI_Receive，可以节约 PC 系统资源，提高程序效率。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

返回值

返回尚未被读取的帧数。

示例

```
#include "ControlCan.h"

int nDeviceType = 24;     // PCIe-9221
int nDeviceInd = 0;       // 索引号0
int nCANInd = 0;          // CAN0通道
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_GetReceiveNum(nDeviceType, nDeviceInd, nCANInd);
```

2.6.12 VCI_ClearBuffer

描述

此函数用以清空指定 CAN 通道的缓冲区。主要用于需要清除接收缓冲区数据的情况。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

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 = 24;     // PCIe-9221
int nDeviceInd = 0;       // 索引号0
int nCANInd = 0;          // CAN0通道
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_ClearBuffer(nDeviceType, nDeviceInd, nCANInd);
```

2.6.13 VCI_Transmit

描述

发送函数。返回值为实际发送成功的帧数。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

pSend

要发送的帧结构体 VCI_CAN_OBJ 数组的首指针。

Len

要发送的帧结构体数组的长度（发送的帧数量）。

返回值

返回实际发送成功的帧数。

示例

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

int nDeviceType = 24;     // PCIe-9221
int nDeviceInd = 0;       // 索引号0
int nCANInd = 0;          // CAN0通道
DWORD dwRel;
VCI_CAN_OBJ vco[2];       // 定义两帧的结构体数组
// 中间略去其他函数代码
vco[0].ID = 0x00000001;   // 填写第一帧的ID
vco[0].SendType = 0;      // 正常发送
vco[0].RemoteFlag = 0;    // 数据帧
vco[0].ExternFlag = 0;    // 标准帧
vco[0].DataLen = 1;       // 数据长度1个字节
vco[0].Data[0] = 0x66;    // 数据0为0x66
vco[1].ID = 0x00000002;   // 填写第二帧的ID
vco[1].SendType = 0;      // 正常发送
vco[1].RemoteFlag = 0;    // 数据帧
vco[1].ExternFlag = 0;    // 标准帧
vco[1].DataLen = 1;       // 数据长度1个字节
vco[1].Data[0] = 0x55;    // 数据0为0x55
dwRel = VCI_Transmit(nDeviceType, nDeviceInd, nCANInd, vco, 2); // 发送两帧
```

2.6.14 VCI_Receive

描述

接收函数。此函数从指定的设备 CAN 通道的接收缓冲区中读取数据。建议在调用之前，先调用**错误!未找到引用源。**函数获知缓冲区中有多少帧，然后对应地去接收。

```
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 PCIe-9221 时，索引号为 0，这时再插入一个 PCIe-9221，那么后面插入的这个设备索引号就是 1，以此类推。

CANIndex

第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

pReceive

用来接收的帧结构体 VCI_CAN_OBJ 数组的首指针。

Len

用来接收的帧结构体数组的长度（本次接收的最大帧数，实际返回值小于等于这个值）。

WaitTime

缓冲区无数据，函数阻塞等待时间，以毫秒为单位。若为-1 则表示无超时，一直等待。

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 = 24;    // PCIe-9221
int nDeviceInd = 0;      // 索引号0
int nCANInd = 0;         // CAN0通道
DWORD dwRel;
VCI_CAN_OBJ vco[100];
// 中间略去其他函数代码
dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd, vco, 100, 400);
// 一次最多能接收100帧，如果缓冲区无数据，接收函数等待400毫秒后退出，返回0
```

3. 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。

3.1 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

3.2 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。

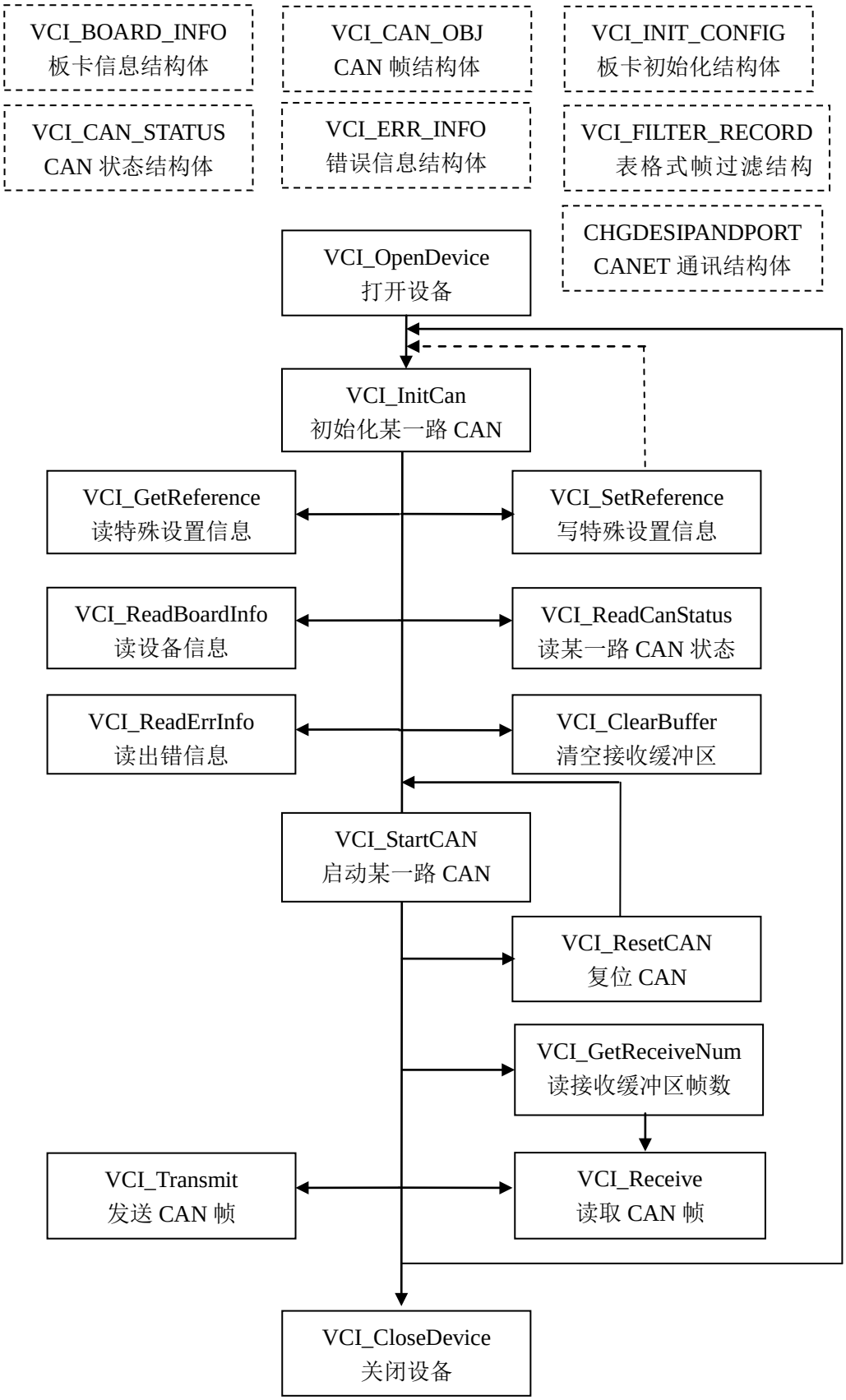
ByRef（可选）

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
```

3.3 接口库函数使用流程



3.4 Linux下动态库的使用

3.4.1 驱动程序的安装

所有驱动都在 Linux 2.4.20-8 下测试通过。

3.4.2 USBCAN-I/II/I+/II+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）。

3.4.3 PCI-9820 驱动的安装

把 driver 目录下的 pci9820b.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）。

3.5 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。

3.6 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

4. 参考资料

[1]周立功著，项目驱动——CAN-bus 现场总线基础教程，第 1 版，北京：北京航空航天出版社，2012，ISBN:7512408218, 9787512408210
[2]蔡豪格（德）著，周立功译，现场总线 CANopen 设计与应用，第 1 版，北京：北京航空航天出版社，2011，ISBN: 9787512404861, 7512404867

5. 免责声明

应用信息

本应用信息中的案例或意图均为假设，仅方便用户熟悉产品的特性以及使用方法。客户在开发产品前必须根据其产品特性给予修改并验证。

修改文档的权利

本手册所陈述的产品文本及相关软件版权均属广州致远电子股份有限公司所有，其产权受国家法律绝对保护，未经本公司授权，其它公司、单位、代理商及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。广州致远电子股份有限公司保留在任何时候修订本用户手册且不需通知的权利。

您若需要我公司产品及相关信息，请及时与我们联系，我们将热情接待。

销售与服务网络

广州致远电子股份有限公司

地址：广州市天河区车陂路黄洲工业区 7 栋 2 楼
邮编：510660
传真：(020)28267891
技术支持邮箱：CANScope@zlg.cn
网址：www.zlg.cn

全国服务电话：400-888-4005

全国销售与服务电话：400-888-4005

销售与服务网络：

广州总公司
广州市天河区车陂路黄洲工业区 7 栋 2 楼
电话：(020)28872342　22644261

北京分公司
北京市海淀区知春路 108 号豪景大厦 A 座 19 层
电话：(010)62536178　62635573

深圳分公司
深圳市福田区深南中路 2072 号电子大厦 12 楼
电话：(0755)83640169　83783155

武汉分公司
武汉市洪山区广埠屯珞瑜路 158 号 12128 室(华中电脑数码市场)
电话：(027) 87168497　87168397

成都分公司
成都市一环路南二段 1 号数码科技大厦 403 室
电话：(028) 85439836　85432683

上海分公司—上海
上海市北京东路 668 号科技京城东楼 12E 室
电话：(021) 53865521　53083451

上海分公司—南京
南京市珠江路 280 号珠江大厦 1501 室
电话：(025) 68123923　68123920

上海分公司—杭州
杭州市天目山路 217 号江南电子大厦 502 室
电话：(0571)89719491　89719493

重庆分公司
重庆市九龙坡区石桥铺科园一路二号大西洋国际大厦（赛格电子市场）2705 室
电话：(023)68796438　68797619

西安办事处
西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881295　87881296

请您用以上方式联系我们，我们会为您安排样机现场演示，感谢您对我公司产品的关注！