CAN 接口卡系列产品
接口函数使用说明

	内容
关键词	CAN 接口库函数使用说明
摘要	CAN 接口库函数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	2020/06/01	创建文档
V1.01	2020/07/16	修改 VCI_SetReference 相关函数说明
V1.02	2020/7/23	修改了页脚，修复了部分表格格式
V1.03	2020/9/27	修复一些格式错误、错别字等

目 录

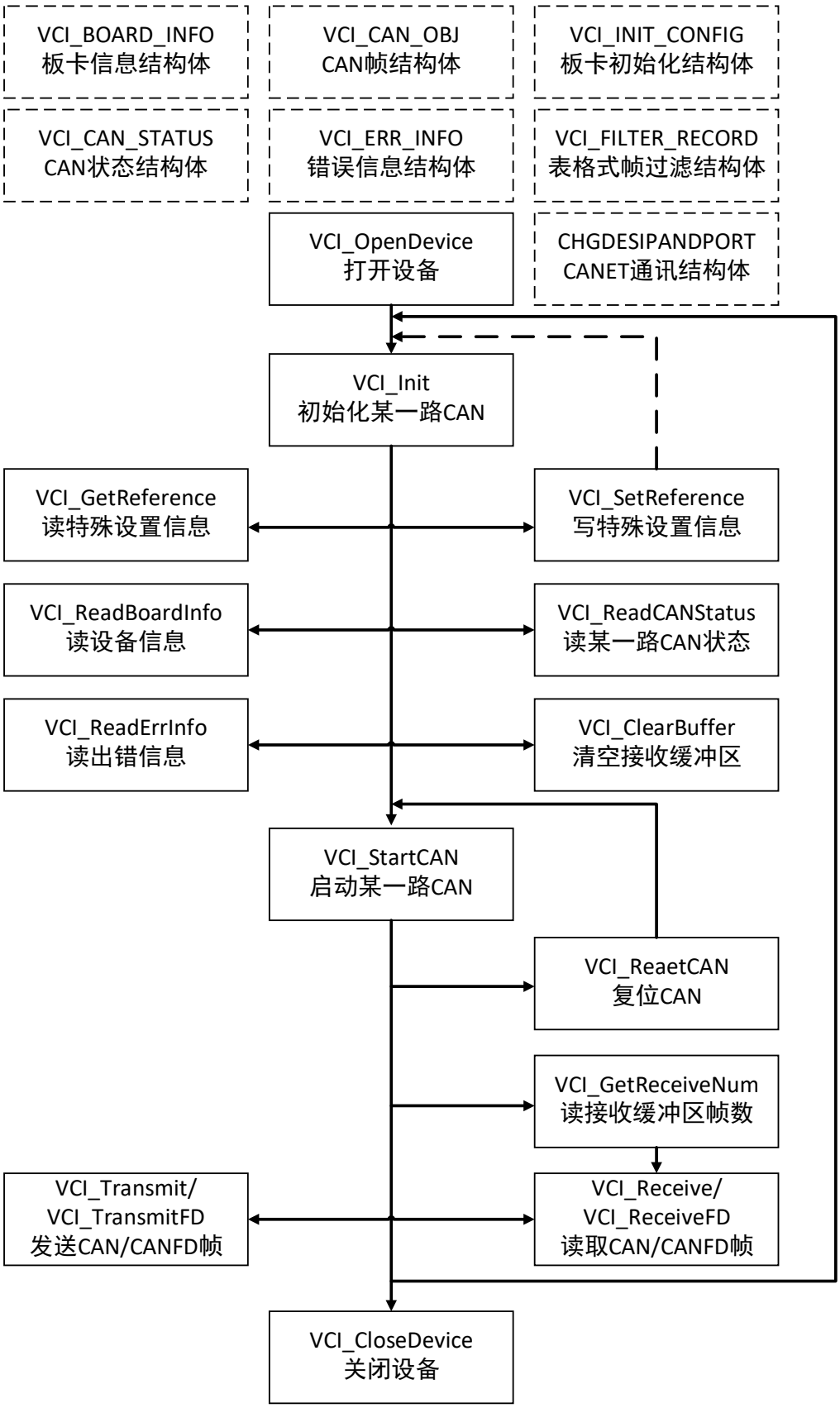
1. 接口函数库说明及其使用

1.1 接口卡设备类型定义

表 1.1 接口卡的类型定义

产品型号	动态库中的设备名称	类型号
USBCAN1	USBCAN1	3
USBCAN2	USBCAN2	4
USBCAN2A	USBCAN2A	4
CANETUDP	CANETUDP	12
CANETTCP	CANETTCP	17
USBCAN_E_U	USBCAN_E_U	20
USBCAN_2E_U	USBCAN_2E_U	21
USBCANFD_II	USBCANFD_II	41
USBCANFD_I	USBCANFD_I	42

1.2 接口库函数使用流程



1.3 函数库的特点与工作原理

本函数库涵盖了成石创新生产的所有 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）10 万帧接收缓冲区。即调用 VCI_OpenDevice 后，即在内存中开辟 10 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。所有的成石创新 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。

1.4 错误码定义

表 1.2 错误码定义

名称	错误码	描述
CAN 控制器错误码		
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（运行时库错误码）		
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_INDEXUS ED	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

1.5 函数库中的数据结构定义

1.5.1 VCI_BOARD_INFO

描述

VCI_BOARD_INFO 结构体包含接口卡的设备信息。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

① hw_Version
硬件版本号，用 16 进制表示。比如 0x0100 表示 V1.00。

② fw_Version
固件版本号，用 16 进制表示。

③ dr_Version
驱动程序版本号，用 16 进制表示。

④ in_Version
接口库版本号，用 16 进制表示。

⑤ irq_Num
板卡所使用的中断号。

⑥ can_Num
表示有几路 CAN 通道。

⑦ str_Serial_Num
此板卡的序列号。

⑧ str_hw_Type
硬件类型，比如“USBCANFD V1.00”（注意：包括字符串结束符'\0'）。

⑨ Reserved
系统保留。

1.5.2 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

① ID

32 位变量，数据格式为靠右对齐。

② TimeStamp

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

③ 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。

④ 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。

⑤ RemoteFlag

是否是远程帧：=0 时为为数据帧，=1 时为远程帧（数据段空）。

⑥ 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）。

⑦ 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。

⑧ Data[8]

CAN 帧的数据。由于 CAN2.0 帧数据部分最大是 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。

⑨ Reserved

系统保留。

1.5.3 VCI_FD_OBJ

描述

VCI_FD_OBJ 结构体是 CANFD 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_TransmitFD 和接收函数 VCI_ReceiveFD 中，被用来传送 CANFD 信息帧。

```
typedef  struct  _VCI_FD_OBJ {
    UINT    ID;
    UINT    TimeStamp;
    BYTE    TimeFlag;
    BYTE    SendType;
    BYTE    RemoteFlag;  //是否是远程帧
    BYTE    ExternFlag;  //是否是扩展帧
    BYTE    DataLen;
    BYTE    Data[64];
    BYTE    BRS;         // 是否使能 BRS(位速率切换)
    BYTE    FDF;         // CAN 类型定义, 0 为 CAN2.0 帧，1 为 CANFD 帧
    BYTE    Reserved[1];
}VCI_FD_OBJ, *PVCI_FD_OBJ;
```

成员

① ID

帧 ID。32 位变量，数据格式为靠右对齐。

② TimeStamp

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

③ 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。

④ 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。

⑤ RemoteFlag
是否是远程帧：=0 时为数据帧，=1 时为远程帧（数据段空）。

⑥ 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）。

⑦ DataLen

数据长度 DLC (<=64)，即 CANFD 帧 Data 有几个字节。需要注意的是 CAN FD 规定数据部分长度的取值为：0 ~8，12，16，20，24，32，48，64 字节。编程时请注意 DataLen 的取值范围。

⑧ Data[64]
CAN FD 帧的数据。由于 CAN FD 帧规定了数据部分最大是 64 个字节，所以这里预留了 64 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。

⑨ BRS
位速率开关，该标志只用于 CANFD 报文中，用于 CANFD 报文中数据域传输速率的切换。=1 时使能数据域位速率，=0 是禁用数据域位速率。

⑩ FDF
FD 格式标志：=0 是经典 CAN2.0 帧，=1 为 CANFD 帧。

⑪ Reserved
系统保留。

1.5.4 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(此函数只对 USBCAN1、USBCAN2、USBCAN2A 设备有效)。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

① ErrInterrupt
中断记录，读操作会清除中断。

② regMode
CAN 控制器模式寄存器值。

③ regStatus
CAN 控制器状态寄存器值。

④ regALCapture
CAN 控制器仲裁丢失寄存器值。

⑤ regECCapture
CAN 控制器错误寄存器值。

⑥ regEWLimit
CAN 控制器错误警告限制寄存器值。默认为 96。

⑦ 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。

⑧ 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。

⑨ Reserved
系统保留。

1.5.5 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 __VCI_ERR_INFO {
    UINT ErrCode;
    BYTE Passive_ErrData[3];
    BYTE ArLost_ErrDat
} VCI_ERR_INFO, *PVCI_ERR_INFO;
```

成员

① ErrCode
错误码。(对应着表 1.2 的错误码定义)

② 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。

③ 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。

备注

用于存储错误信息的 VCI_ERR_INFO 结构。ErrCode 可能为下列各个错误码的多中组合之一，见表 1.3.

表 1.3 错误码定义

ErrCode	Passive_ErrData			ArLost_ErrData	错误描述
	[0]	[1]	[2]		
CAN 控制器错误					
0x0001	/	/	/	/	CAN 控制器内部 FIFO 溢出
0x0002	/	/	/	/	CAN 控制器错误报警
0x0004	/	接收错误计数	发送错误计数	/	CAN 控制器消极错误
0x0008	/	/	/	/	CAN 控制器仲裁丢失
0x0010	/	/	/	/	CAN 控制器总线错误
通用错误码（运行时库错误）					
0x0100	/	/	/	/	设备已经打开
0x0200	/	/	/	/	打开设备错误
0x0400	/	/	/	/	设备没有打开
0x0800	/	/	/	/	缓冲区溢出
0x1000	/	/	/	/	此设备不存在
0x2000	/	/	/	/	装载动态库失败
0x4000	/	/	/	/	执行命令失败错误码
0x8000	/	/	/	/	内存不足

注：‘/’代表不需要关心此处的取值。

1.5.6 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 _VCI_INIT_CONFIG {
    DWORD  AccCode;
    DWORD  AccMask;
    DWORD  Reserved;
    UCHAR  Filter;
```

```
    UCHAR  Timing0;
    UCHAR  Timing1;
    UCHAR  Mode;

    //以下 USBCANFD-I/USBCANFD-II 设备中使用
    UCHAR    ChannelType;
    UCHAR    FD_Standard;
    UCHAR    EnBRS;
    UCHAR    ClockMHz;
    DWORD    nbtp;
    DWORD    dbtp;
    UCHAR    EnFilter;
    UCHAR    WorkMode;
} VCI_INIT_CONFIG, *PVCI_INIT_CONFIG;
```

成员

① AccCode

CAN 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。只对 USBCAN1、USBCAN2、USBCAN2A 设备有效。

② AccMask

CAN 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。只对 USBCAN1、USBCAN2、USBCAN2A 设备有效。

③ Reserved

保留。

③ Filter

滤波方式。=1 表示单滤波，=0 表示双滤波，只对 USBCAN1、USBCAN2、USBCAN2A 设备有效。

④ Timing0

波特率定时器 0(BTR0)。设置值见表 1.4。只对 USBCAN1、USBCAN2、USBCAN2A 设备有效。

⑤ Timing1

波特率定时器 1(BTR1)。设置值见表 1.4。只对 USBCAN1、USBCAN2、USBCAN2A 设备有效。

⑦ 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）。

⑧ ChannelType

CAN 通道的类型，=0 表示此通道只能收发经典 CAN2.0 帧，=1 表示此通道即可以收发 CAN2.0 帧也可以收发 CANFD 帧。只对 USBCANFD-II、USBCANFD-I 设备有效。

⑨ FD_Standard

CANFD 的标准，=0 为 ISO 标准，=1 为非 ISO 标准。只对 USBCANFD-II、USBCANFD-I 设备有效。

⑩ EnBRS

位速率开关，该标志只用于 CANFD 报文中，用于 CANFD 报文中数据域传输速率的切换。=1 时使能数据域位速率，=0 是禁用数据域位速率。只对 USBCANFD-II、USBCANFD-I 设备有效。

⑪ ClockMHz

CAN 总线波特率的基准时钟，单位位 MHz，取值范围 8、10、12、15、16、20、24、30、48、60。只对 USBCANFD-II、USBCANFD-I 设备有效。

⑫ nbtp

仲裁域位时间定义，设置值见表 1.5。只对 USBCANFD-II、USBCANFD-I 设备有效。

⑬ dbtp

数据域位时间定义，设置值见表 1.5。只对 USBCANFD-II、USBCANFD-I 设备有效。

⑭ EnFilter（暂未使用，USBCANFD-I/II 设备的滤波功能通过 VCI_SetReference 设置）

是否使能滤波。=0 禁用接收帧滤波，=1 使能接受帧过滤功能。

⑮ WorkMode

工作模式选择。=0 为正常模式，=1 为只听模式。只对 USBCANFD-II、USBCANFD-I 设备有效。

备注

1. 当设备类型为 USBCAN1、USBCAN2、USBCAN2A 时，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）设置如表 1.3：

表 1.4 USBCAN1、USBCAN2、USBCAN2A 常用波特率

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. 当设备类型为 USBCANFD-I、USBCANFD-II 时，ClockMHz 、nbtp 和 dbtp 用来设置 CANFD 的仲裁与波特率和数据域波特率，几种常见的波特率（采样点 80%）设置如表 1.5，此表的计算值都是基于 ClockMHz 设置为 60 得出的结果：

表 1.5 USBCANFD-I、USBCANFD-II 常用波特率

仲裁域波特率						数据域波特率					
波特率 (Kbps)	NBTP	TS1	TS2	SJW	BRP	波特率 (Kbps)	DBTP	TS1	TS2	SJW	BRP
5	0x31C18B2E	46	11	3	199	50	0x0740071E	30	7	0	29
10	0x18C18B2E	46	11	3	99	100	0x0741030E	14	3	2	29
20	0x0241BBEE	238	59	3	9	125	0x05C1830E	14	3	3	23
25	0x01C1BBEE	238	59	3	7	200	0x04C1020A	10	2	2	19
40	0x0101BBEE	238	59	3	4	250	0x02C1030E	14	3	2	11
50	0x00C1BBEE	238	59	3	3	400	0x0241020A	10	2	2	9
100	0x0041BBEE	238	59	3	1	500	0x0141030E	14	3	2	5
125	0x0041AFBE	190	47	3	1	800	0x0101020A	10	2	2	4
200	0x0001BBEE	238	59	3	0	1000	0x0081830E	14	3	3	2
250	0x0001AFBE	190	47	3	0	1500	0x0041830E	14	3	3	1
400	0x00418E3A	58	14	3	1	2000	0x0041020A	10	2	2	1
500	0x0001975E	94	23	3	0	2500	0x00410207	7	2	2	1
800	0x00018B3A	58	14	3	0	3000	0x0000030E	14	3	0	0
1000	0x00018B2E	46	11	3	0	4000	0x0001020A	10	2	2	0
						5000	0x00000009	9	0	0	0

注意：

1. 当设备类型为 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. 当设备类型为 USBCANFD-I、USBCANFD-II 时，真过滤不在这里设置，具体操作见 VCI_SetReference 说明。

1.5.7 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

① 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"。

② szdesip[20]

所要更改的目标 IP，比如为"192.168.0.111"。

③ desport

所要更改的目标端口，比如为 4000。

④ blisten

所要更改的工作模式，0 表示正常模式，1 表示只听模式。

1.5.8 REMOTE_CLIENT

描述

设备类型为 CANET-TCP，并且设备作为客户端时，用于删除一个客户端连接。

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

成员

① iIndex

连接的客户端的索引号。

② port

CANET-TCP 设备的工作的工作端口。

③ szip[32]

CANET-TCP 设备的 IP 地址。

1.5.9 VCI_FILTER_RECORD

描述

设备类型为 USBCAN-E-U、USBCAN-2E-U、USBCANFD-I、USBCANFD-II 时，定义了滤波器的滤波范围结构体。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

① 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。

② Start

滤波范围的起始帧 ID。

③ End

滤波范围的结束帧 ID。

1.5.10 VCI_AUTO_SEND_OBJ

描述

当设备类型为 USBCAN-E-U、USBCAN-2E-U 时，USBCANFD-I、USBCANFD-II 设备支持自定义一组硬件层自动定时发送帧列表（列表最大支持 100 帧），列表中的发送帧可以按照设置的周期定时发送 CAN 帧， 发送过程无需上位机软件干预， 该发送列表中的发送帧由 VCI_AUTO_SEND_OBJ 设置，该结构体在 VCI_SetReference 函数中被填充。

```
typedef struct _VCI_AUTO_SEND_OBJ{
    BYTE        Enable;
    BYTE        Index;
    DWORD       Interval;

    union {
        VCI_CAN_OBJ  ObjCAN;    // 经典 CAN 报文
        VCI_FD_OBJ   ObjFD;     // CANFD 报文
    } Obj;
} VCI_AUTO_SEND_OBJ,*PVCI_AUTO_SEND_OBJ;
```

成员

① Enable

是否使能本条报文。 0：禁能；1：使能。

② Index

报文编号。最大支持 32 条报文，即取值范围为 0 至 99。

③ Interval

定时发送时间。0.5ms 为单位。

④ Obj

定时发送帧结构。

备注

1. 当设备类型为 USBCAN-E-U、USBCAN-2E-U 时，使用 VCI_CAN_OBJ 作为帧发送结构。
2. 当设备类型为 USBCANFD-I、USBCANFD-II 时候，分两种情况：
   a. 当 VCI_INIT_CONFIG 结构中设置通道类型为 CAN (即 ChannelType 为 0) 时，使用 VCI_CAN_OBJ 作为帧发送结构。

b. 当 VCI_INIT_CONFIG 结构中设置通道类型为 CANFD（即 ChannelType 为 1）时，使用 VCI_FD_OBJ 作为帧发送结构。

1.6 接口库函数说明

1.6.1 VCI_OpenDevice

描述

此函数用于打开设备。注意一个设备只能打开一次。

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

参数

① DevType

设备类型号。对应不同的产品型号。见表 1.1。

② DevIndex

设备索引号。

③ 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 = 41; /* USBCANFD -II */
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;
}
```

1.6.2 VCI_CloseDevice

描述

此函数用于关闭设备。

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

参数

① DevType
设备类型号。

② 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 = 41;  // USBCANFD-II
int  nDeviceInd  = 0;   // 索 引 号 0
BOOL bRel;

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

1.6.3 VCI_InitCAN

描述

此函数用于初始化指定的 CAN 通道。一个设备有多个 CAN 通道时，需要多次调用。（当设备类型为 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

① DevType
设备类型号。

② DevIndex
设备索引号。

③ CANIndex
第几路 CAN 通道。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

④ pInitConfig

初始化参数结构，为一个 VCI_INIT_CONFIG 结构体变量。

特例

1. 当设备类型为 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，其他的成员可以忽略，具体设置见 VCI_SetReference 说明。
2. 当设备为 USBCANFD-I、USBCANFD-II 时，对滤波的设置应该放到 VCI_SetReference 里设置，具体设置见 VCI_SetReference 说明。

表 1.5 初始化参数结构

成员	功能描述	备注
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 SJA1000 中的四个寄存器 AMR0,ARM1,ARM2.ARM3，其中高字节对应 AMR0，低字节对应 AMR3。（请看表后说明）	只有设备类型为 USBCAN1、USBCAN2、USBCAN2A 时使用
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 结构体说明	
pInitConfig->Timing1	波特率定时器 1，详见 VCI_INIT_CONFIG 结构体说明	
pInitConfig->Mode	模式，0 表示正常模式，1 表示只听模式	
pInitConfig->ChannelType	CAN 通道的类型，0 为经典 CAN2.0 帧类型，1 为 CANFD 类型帧	当设备类型为 USBCANFD-I、USBCANFD-II 时使用
pInitConfig->FD_Standard	CANFD 的标准，0 为 ISO 标准，1 为非 ISO 标准。	
pInitConfig->EnBRS	是否使能数据域位速率，0 禁能数据域位速率，1 为使能数据域位速率。	
pInitConfig-> ClockMHz	CAN 总线波特率的基准时钟，单位为 MHz。	
pInitConfig->nbtp	仲裁域位时间定义。	
pInitConfig->dbtp	数据域位时间定义。	
pInitConfig->EnFilter	是否使能滤波（暂未使用，设置为 0）	
pInitConfig->WorkMode	工作模式选择。=0 为正常模式，=1 为只听模式。	

返回值

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

示例

```
#include "ControlCAN.h"

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;
VCI_INIT_CONFIG vic;
DWORD dwRel;

vic.ChannelType  = 1;   // CAN 通道类型为 CANFD 类型
vic.FD_Standard  = 0;   // ISO 标准 CANFD
vic.EnBRS        = 1;   // 使能数据域位速率
vic.ClockMHz     = 60;  // CAN 波特率基准时钟 60MHz
vic.nbtp      = 0x0001975E; // 500Kbps
vic.dbtp      = 0x0041020A; // 2000Kbps
vic.WorkMode  = 0; // 正常工作模式

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;
}
```

1.6.4 VCI_ReadBoardInfo

描述

此函数用于获取设备信息。

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

参数

① DevType

设备类型号。

② DevIndex

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

③ 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。

示例

```
#include "ControlCAN.h"

int nDeviceType  = 41;  // USBCANFD-II
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);
```

1.6.5 VCI_ReadErrInfo

描述

此函数用于获取 CAN 卡发生的最后一次错误信息。

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

参数

① DevType

设备类型号。

② DevIndex

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

③ 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 CANIndex 设为 - 1*）

④ pErrInfo

用来存储错误信息的 VCI_ERR_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 = 41;   // USBCANFD-II
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);
```

1.6.6 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

① DevType

设备类型号。

② DevIndex

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

③ CANIndex

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

④ 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  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;   // CAN0 通道
VCI_CAN_STATUS   vcs;
```

```
DWORD dwRel;
// 中间略去其他函数代码
dwRel = VCI_ReadCANStatus(nDeviceType, nDeviceInd, nCANInd, &vcs);
```

1.6.7 VCI_SetReference

描述

此函数用于设置 CANET、USBCAN-E-U、USBCAN-2E-U、USBCANFD-I、USBCANFD-II 等设备的相应参数，主要处理不同设备的特定操作。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。

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

参数

① DevType
设备类型号。

② DevIndex
设备索引号，比如当只有一个 USBCANFD-II 时，索引号为 0，这时再插入一个 USBCANFD-II， 那么后面插入的这个设备索引号就是 1，以此类推。

③ CANIndex
第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

④ RefType
参数类型。

⑤ 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 USBCAN-E-U、USBCAN-2E-U 时，参数定义如下表所示。

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： 0x00000104：1000Kbps 0x00000106：800Kbps 0x0000010C：500Kbps 0x0040010C：250Kbps 0x00C0010C：125Kbps 0x0100010C：100Kbps 0x0240010C：50Kbps 0x0600010C：20Kbps 0x0C40010C：10Kbps 0x0FC1070F：5Kbps	若用户设置位其他值，有可能工作不正常。如果需要例表中没有列出的波特率值，用户可咨询北京成石技术支持工程师 13718361868，或者发送邮件到：David.wang@itegva.com。 (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。
2	NULL,可忽略	按滤波表格中的设置启动滤波，滤波方式为白名单，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
5	指向 VCI_AUTO_SEND_OBJ 结构的指针	设置自动发送列表中的发送帧。设备可以设置无需软件干预的硬件层自动定时发送功能，最大支持 100 帧。
6	NULL,可忽略	清除自动发送帧列表。
7	指向 DWORD 类型的指针，该 DWORD 变量的值为发送 CAN 帧最小的时间间隔的值，例如 400，单位微妙。	该值最大值为 65535，超过会按 65535 处理。应该在调用 VCI_InitCAN 函数之后用 VCI_SetReference 函数进行设置。

2. 当设备类型 USBCANFD-I、USBCANFD-II 时，参数定义如下表所示。

RefType	pData	功能描述
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。
2	NULL,可忽略	按滤波表格中的设置启动滤波，滤波方式为白名单，调用即启用。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
5	指向 VCI_AUTO_SEND_OBJ 结构的指针	设置自动发送列表中的发送帧。设备可以设置无需软件干预的硬件层自动定时发送功能，最大支持 100 帧。
6	NULL,可忽略	清除自动发送帧列表。
7	指向 DWORD 类型的指针，该 DWORD 变量的值为发送 CAN 帧最小的时间间隔的值，例如 400，单位微妙。	该值最大值为 65535，超过会按 65535 处理。应该在调用 VCI_InitCAN 函数之后用 VCI_SetReference 函数进行设置。

3. 当设备类型 CANET-UDP 时，参数定义如下表所示。

RefType	pData	功能描述

0	字符串首指针，用来存储所指定操作的 CANET-UDP 设备的 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 设备的工作端口。	设置所要操作的 CANET-TCP 的工作端口。例如：
		`DWORD port;`
		`VCI_SetReference(VCI_CANETTCP,0,0,1,(PVOID)&port);`

4. 当设备类型 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 TegcanView 测试工具需要工作在服务器模式。如果设备工作在服务器模式，我们的 TegcanView 测试工具则需要工作在客户端模式，参数定义如下表所示。

RefType	pData	功能描述

0	字符串首指针，用来存储所指定操作的 IP 地址。（当 CANET-TCP 设备被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，存储所指定操作的工作端口。（当 CANET-TCP 设备工作在服务器模式时使用）	设置所要操作的 CANET-TCP 的工作端口。例如：
		`DWORD port;`
		`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。（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。
		1. 如果 CANET-TCP 设备工作在服务器模式则本机工作在客户端模式。
		2. 如果 CANET-TCP 设备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-TCP 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_CANETTCP,0,0,7,(PVOID)&cli);`

示例

```
#include "ControlCAN.h"

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;   // CAN0 通道
DWORD port = 4001;      // CANET-UDP 设备的工作端口
DWORD dwRel;

VCI_FILTER_RECORD vif;
vif.ExtFrame = 0;    // 过滤标准帧标准帧
vif.Start = 4;   // 帧起始 ID
vif.End   = 5;   // 帧结束 ID

// 中间略去其他函数代码

// 添加滤波表
dwRel = VCI_SetReference (nDeviceType, nDeviceInd, nCANInd,
                          1,(PVOID)&vif);
if (dwRel == STATUS_ERR) {
    MessageBox(_T("失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}

// 启动接收滤波, 白名单
dwRel = VCI_SetReference (nDeviceType, nDeviceInd, nCANInd,
                          2,NULL);
if (dwRel == STATUS_ERR) {
    MessageBox(_T("失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}

// 清除滤波表
dwRel = VCI_SetReference (nDeviceType, nDeviceInd, nCANInd,
                          3,NULL);
if (dwRel == STATUS_ERR) {
    MessageBox(_T("失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

1.6.8 VCI_GetReference

描述

此函数用于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

① DevType

设备类型号。

② DevIndex

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

③ CANIndex

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

④ RefType

参数类型。

⑤ 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，如下表所示。

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 设备的 IP 地址。	读取 CANET-UDP 设备的 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 设备的工作端口。	读取 CANET-UDP 设备的工作端口。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 TegcanView 测试工具需要工作在服务器模式。如果设备工作在服务器模式，我们的 TegcanView 测试工具则需要工作在客户端模式，参数定义如表下表所示。

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 设备的 IP 地址。（当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);`
		此函数调用成功，则在 szip 中返回 CANET_TCP 设备的 IP 地址。

1	长度为 4 个字节，存储读取出来的 CANET-TCP 设备的工作端口。（当设备工作在服务器模式时有效）	读取 CANET-TCP 设备的工作端口。例如：
		`DWORD port;`
		`VCI_GetReference(VCI_CANETTCP,0,0,1,(PVOID)&port);`
		此函数调用成功,则在 port 中返回 CANET-TCP 设备的工作端口。

2	长度为 4 个字节，存储本机上的 TCP 工作端口。（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);`
		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。
		1. 如果 CANET-TCP 设备工作在服务器模式则本机工作在客户端模式。
		2. 如果 CANET-TCP 设备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);`
		此函数调用成功，将读取本机工作模式。

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。（当设备工作在客户端模式下有效）	读取连接到本机上的 CANET-TCP 设备数量。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。（当设备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);`
		此函数调用成功，则在 cli 存储客户端的信息。

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

// 读取 CANET-UDP 设备的 IP 地址
dwRel = VCI_GetReference(nDeviceType, nDeviceInd,
                         nCANInd,0,(PVOID)szip);
```

1.6.9 VCI_StartCAN

描述

此函数用于启动 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

① DevType
设备类型号。

② DevIndex
设备索引号，比如当只有一个 USBCANFD-II 时，索引号为 0，这时再插入一个 USBCANFD-II，那么后面插入的这个设备索引号就是 1，以此类推。

③ 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  = 41;  // USBCANFD-II
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;
}
```

1.6.10 VCI_ResetCAN

描述

此函数用于复位 CAN 卡。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

① DevType

设备类型号。

② DevIndex

设备索引号，比如当只有一个 USBCANFD-II 时，索引号为 0，这时再插入一个

USBCANFD-II， 那么后面插入的这个设备索引号就是 1，以此类推。

③ 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。

备注

调用 VCI_ResetCAN 函数后，需要重新 VCI_StartCAN 才能使用。

示例

```
#include "ControlCAN.h"

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;   // CAN0 通道
DWORD dwRel;

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

1.6.11 VCI_GetReceiveNum

描述

此函数用于获取指定 CAN 通道接收到但尚未被读取的帧数量。主要用途是配合 VCI_Receive 或者 VCI_ReceiveFD 使用，即缓冲区有数据，再接收。用户无需一直调用 VCI_Receive 或者 VCI_ReceiveFD， 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

① DevType

设备类型号。

② DevIndex

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

③ 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  = 41;  // USBCANFD-II
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);
```

1.6.12 VCI_ClearBuffer

描述

此函数用于清空指定 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

① DevType

设备类型号。

② DevIndex

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

③ 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  = 41;  // USBCANFD-II
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);
```

1.6.13 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

① DevType

设备类型号。

② DevIndex

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

③ CANIndex

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

④ pSend

要发送的帧结构体 VCI_CAN_OBJ 数组的首指针。

⑤ 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。

备注

1. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指定相应通道的通道类型为 CAN 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_Transmit 函数进行数据发送。
2. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指定相应通道的通道类型为 CANFD 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_TransmitFD 进行数据发送。

示例

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

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
```

```
int nCANInd      = 0;   // CAN0 通道

DWORD dwRel;
VCI_INIT_CONFIG vic;
VCI_CAN_OBJ vco[2];     // 定义两帧的结构体数组

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;
}

vic.ChannelType  = 0;       // CAN 通道类型为 CAN 类型
vic.FD_Standard  = 0;       // ISO 标准 CANFD
vic.EnBRS        = 0;       // 禁止数据域位速率
vic.ClockMHz     = 60;      // CAN 波特率基准时钟 60MHz
vic.nbtp      = 0x0001975E; // 仲裁域波特率 500Kbps
vic.dbtp      = 0x0041020A; // 数据域波特率 2000Kbps
vic.WorkMode  = 0; // 正常工作模式

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;
}

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

// 发送两帧
dwRel = VCI_Transmit(nDeviceType, nDeviceInd, nCANInd, vco, 2);
```

1.6.14 VCI_Receive

描述

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 VCI_GetReceiveNum。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

① DevType

设备类型号。

② DevIndex

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

③ CANIndex

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

④ pReceive

用来接收的帧结构体 VCI_CAN_OBJ 数组的首指针。

⑤ 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）。

⑥ 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。

备注

1. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指

定相应通道的通道类型为 CAN 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_Recevie 函数进行数据接收。

2. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指定相应通道的通道类型为 CANFD 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_ReceiveFD 进行数据接收。

示例

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

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;   // CAN0 通道

DWORD dwRel;
VCI_INIT_CONFIG vic;
VCI_CAN_OBJ   vcoCAN[100];     // 定义 100 帧的结构体数组
VCI_FD_OBJ    vcoFD[100];      // 定义 100 帧的结构体数组

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;
}

vic.ChannelType  = 0;      // CAN 通道类型为 CAN 类型
vic.FD_Standard  = 0;      // ISO 标准 CANFD
vic.EnBRS        = 0;      // 禁止数据域位速率
vic.ClockMHz     = 60;     // CAN 波特率基准时钟 60MHz
vic.nbtp      = 0x0001975E; // 仲裁域波特率 500Kbps
vic.dbtp      = 0x0041020A; // 数据域波特率 2000Kbps
vic.WorkMode  = 0; // 正常工作模式

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;
}

// 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
if (vic.ChannelType == 0) { //如果通道类型为 CAN 使用 VCI_Receive 接收
    dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd,
                        vcoCAN, 100, 400);
} else {//如果通道类型为 CANFD 使用 VCI_ReceiveFD 接收
    dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd,
                        vcoFD, 100, 400);
}
```

1.6.15 VCI_TransmitFD

描述

发送函数。返回值为实际发送成功的帧数。

```
ULONG stdcall VCI_TransmitFD( DWORD DevType,
                              DWORD DevIndex,
                              DWORD CANIndex,
                              PVCI_FD_OBJ pSend,
                              ULONG Len);
```

参数

① DevType
设备类型号。

② DevIndex
设备索引号，比如当只有一个 USBCANFD-II 时，索引号为 0，这时再插入一个 USBCANFD-II， 那么后面插入的这个设备索引号就是 1，以此类推。

③ CANIndex
第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

① pSend
要发送的帧结构体 VCI_FD_OBJ 数组的首指针。

⑤ 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。

备注

1. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指定相应通道的通道类型为 CAN 时（见 VCI_INIT_CONFIG 结构体说明），调用

VCI_Transmit 函数进行数据发送。

2. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指定相应通道的通道类型为 CANFD 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_TransmitFD 进行数据发送。

示例

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

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;   // CAN0 通道

DWORD dwRel;
VCI_INIT_CONFIG vic;
VCI_FD_OBJ vco[2];      // 定义两帧的结构体数组

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;
}

vic.ChannelType  = 1;       // CAN 通道类型为 CANFD 类型
vic.FD_Standard  = 0;       // ISO 标准 CANFD
vic.EnBRS        = 1;       // 使能数据域位速率
vic.ClockMHz     = 60;      // CAN 波特率基准时钟 60MHz
vic.nbtp      = 0x0001975E; // 仲裁域波特率 500Kbps
vic.dbtp      = 0x0041020A; // 数据域波特率 2000Kbps
vic.WorkMode  = 0; // 正常工作模式

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;
}
```

```
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 = 64;       // 数据长度 1 个字节

// 填充数据
for (int i = 0; i  < 64; i++) {
    vco[0].Data[i] = i;
}

// 发送 1 帧数据
dwRel = VCI_TransmitFD(nDeviceType, nDeviceInd, nCANInd, vco, 1);
```

1.6.16 VCI_ReceiveFD

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 VCI_GetReceiveNum。函数获知缓冲区中有多少帧，然后对应地去接收。

```
ULONG stdcall VCI_ReceiveFD(  DWORD DevType,
                              DWORD DevIndex,
                              DWORD CANIndex,
                              PVCI_FD_OBJ pReceive,
                              ULONG Len,
                              INT WaitTime = －1);
```

参数

① DevType
设备类型号。

② DevIndex
设备索引号，比如当只有一个 USBCANFD-II 时，索引号为 0，这时再插入一个 USBCANFD-II， 那么后面插入的这个设备索引号就是 1，以此类推。

③ CANIndex
第几路 CAN。即对应卡的 CAN 通道号，CAN0 为 0，CAN1 为 1，以此类推。

② pReceive
用来接收的帧结构体 VCI_FD_OBJ 数组的首指针。

⑤ 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）。

⑥ 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。

备注

1. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指定相应通道的通道类型为 CAN 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_Recevie 函数进行数据接收。

2. 对于设备类型为 USBCANFD-I、USBCANFD-II 设备，当在调用 VCI_InitCAN 时指

定相应通道的通道类型为 CANFD 时（见 VCI_INIT_CONFIG 结构体说明），调用 VCI_ReceiveFD 进行数据接收。

示例

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

int nDeviceType  = 41;  // USBCANFD-II
int nDeviceInd   = 0;   // 索引号 0
int nCANInd      = 0;   // CAN0 通道

DWORD dwRel;
VCI_INIT_CONFIG vic;
VCI_CAN_OBJ   vcoCAN[100];     // 定义 100 帧的结构体数组
VCI_FD_OBJ    vcoFD[100];      // 定义 100 帧的结构体数组

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;
}

vic.ChannelType  = 1;       // CAN 通道类型为 CANFD 类型
vic.FD_Standard  = 0;       // ISO 标准 CANFD
vic.EnBRS        = 1;       // 使能数据域位速率
vic.ClockMHz     = 60;      // CAN 波特率基准时钟 60MHz
vic.nbtp      = 0x0001975E; // 仲裁域波特率 500Kbps
vic.dbtp      = 0x0041020A; // 数据域波特率 2000Kbps
vic.WorkMode  = 0; // 正常工作模式

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;
}

// 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
```

```
if (vic.ChannelType == 0) { //如果通道类型为 CAN 使用 VCI_Receive 接收
    dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd,
                        vcoCAN, 100, 400);
} else {//如果通道类型为 CANFD 使用 VCI_ReceiveFD 接收
    dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd,
                        vcoFD, 100, 400);
}
```

2. 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。

2.1 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. 免责声明

版权

本手册所陈述的产品文本及相关软件版权均属北京成石创新科技有限公司所有，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。

修改文档的权利

北京成石创新科技有限公司保留任何时候在不事先声明的情况下对本手册的修改的权力。