接口函数库（二次开发库）

使用说明书

说明书版本：V2.05

更新日期：2019.06.01

目　　录

第一部分 概述

用户如果只是利用USB-CAN总线接口适配器进行CAN总线调试，可以直接利用随机提供的USB-CAN Tool工具软件，进行收发数据的测试。

如果用户打算编写自己产品的软件程序。请认真阅读以下说明，并参考我们提供的：

① C++Builder ②C# ③VC ④VB ⑤VB.NET ⑥Delphi ⑦LabVIEW ⑧ LabWindows/CVI ⑨Matlab等示例代码。

开发用库文件：ControlCAN.lib, ControlCAN.DLL

VC平台函数声明文件：ControlCAN.h

VB平台函数声明文件：ControlCAN.bas

LabVIEW平台库函数封装模块：ControlCAN.llb

Delphi平台函数声明文件：ControlCAN.pas

注意：ControlCAN.lib, ControlCAN.DLL依赖于VC2008运行库，该运行库一般的系统都会包含，只有在极个别的精减系统中没有，需要安装一下。

第二部分 兼容 ZLG 函数库及数据结构

2.1 类型定义

2.1.1 Device Type

类型定义	类型值	描述
VCI_USBCAN2	4	USBCAN-2A USBCAN-2C CANalyst-II

2.1.2 VCI_BOARD_INFO

VCI_BOARD_INFO结构体包含USB-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。比如0x0100表示V1.00。

dr_Version

驱动程序版本号，用16进制表示。比如0x0100表示V1.00。

in_Version

接口库版本号，用16进制表示。比如0x0100表示V1.00。

irq_Num

保留参数。

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。

2.1.3 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。详情请参照：《*8.附件1：ID对齐方式.pdf*》说明文档。

TimeStamp

设备接收到某一帧的时间标识。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，重发时间为4秒，4秒内没有发出则取消）；

=1时为单次发送（只发送一次，发送失败不会自动重发，总线只产生一帧数据）；

其它值无效。

（二次开发，建议**SendType**=1，提高发送的响应速度）

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。

2.1.4 VCI_INIT_CONFIG

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，允许设置为0-3，详细请参照2.2.3节的滤波模式对照表。

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），

=2表示自发自收模式（环回模式）。

备注：

关于滤波器的设置，请参照：《*9.附件2：CAN参数设置.pdf*》说明文档。

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	Timing0(BTR0)	Timing1(BTR1)
10 Kbps	0x31	0x1C
20 Kbps	0x18	0x1C
40 Kbps	0x87	0xFF
50 Kbps	0x09	0x1C
80 Kbps	0x83	0xFF
100 Kbps	0x04	0x1C
125 Kbps	0x03	0x1C
200 Kbps	0x81	0xFA
250 Kbps	0x01	0x1C
400 Kbps	0x80	0xFA
500 Kbps	0x00	0x1C
666 Kbps	0x80	0xB6
800 Kbps	0x00	0x16
1000 Kbps	0x00	0x14
33.33 Kbps	0x09	0x6F
66.66 Kbps	0x04	0x6F
83.33 Kbps	0x03	0x6F

注：

1.配置波特率时，用户只需要按照SJA1000（16MHz）给的波特率参数进行设置即可。

2.常规波特率直接按照上表的值配置即可。其它非常规波特率，可以使用附带的波特率侦测工具进行侦测，并到得相应的波特率参数。或是使用USB_CAN TOOL安装目录下的波特率计算工具计算。（参考《6.插件2：波特率侦测工具使用说

明书.pdf》）

3.本适配器使用最新高速CAN收发器，不再支持10K以下波特率。

2.2 函数描述

2.2.1 VCI_OpenDevice

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。对应不同的产品型号*详见：适配器设备类型定义*。

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器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。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"
int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */
int   nDeviceInd = 0;         /*  第1个设备  */
DWORD dwRel;
dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, 0);
if(dwRel != 1)
{
   MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
   return FALSE;
}
```

2.2.2 VCI_CloseDevice

此函数用以关闭设备。

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

参数：

DevType

设备类型。对应不同的产品型号*详见：适配器设备类型定义*。

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。对应已经打开的设备。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"
int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */
int   nDeviceInd = 0;         /*  第1个设备  */
DWORD dwRel;
dwRel = VCI_CloseDevice(nDeviceType, nDeviceInd);
if(dwRel != 1)
{
   MessageBox(_T("关闭设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
   return FALSE;
}
```

2.2.3 VCI_InitCan

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。

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

参数：

DevType

设备类型。对应不同的产品型号*详见：适配器设备类型定义*。

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

pInitConfig

初始化参数结构。

参数成员列表：

成员	功能描述
pInitConfig->AccCode	由AccCode和AccMask可以共同决定哪些报文能够被接受，这两个寄存器均采用ID的左对齐方式设置，即AccCode与AccMask的最高位（Bit31）与ID值的最高位对齐。 关于ID的对齐方式，请参照：《*附件1：ID对齐方式详细说明*》说明文档。 例如： 若把AccCode的值设为0x24600000（即0x123左移21位的结果）， AccMask的值设为0x00000000，则只有CAN信息帧ID为0x123的报文能够被接受(AccMask的值0x00000000表示所有位均为相关位)。若把AccCode的值设为0x24600000，AccMask的值设为0x600000（0x03左移21位的结果），则只有CAN信息帧ID为0x120～0x123的报文都能够被接受(AccMask的值0x600000表示除了bit0~bit1其他位(bit2~bit10)均为相关位)。 注：本滤波器设置示例以标准帧为例，高11位有效；若为扩展帧，则ID为29位，AccCode和
pInitConfig->AccMask	

成员	功能描述
	AccMask设置时高29位对扩展帧有效！
pInitConfig->Reserved	保留
pInitConfig->Filter	滤波方式的设置请参照本节的*滤波模式对照表*。
pInitConfig->Timing0	波特率 T0 设置
pInitConfig->Timing1	波特率 T1 设置
pInitConfig->Mode	工作模式： *0*-正常工作 *1*-仅监听模式 *2*-自发自收测试模式

滤波模式对照表：

值	名称	说明
0/1	接收所有类型	滤波器同时对标准帧与扩展帧过滤！
2	只接收标准帧	滤波器只对标准帧过滤，扩展帧将直接被滤除。
3	只接收扩展帧	滤波器只对扩展帧过滤，标准帧将直接被滤除。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"

int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */

int   nDeviceInd = 0;         /*  第1个设备  */

int   nCANInd = 0;            /*  第1个通道  */

DWORD dwRel;

VCI_INIT_CONFIG vic;

dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, 0);

if(dwRel != 1)

{

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

```
    return FALSE;
}
Vic.AccCode=0x80000008;
vic.AccMask=0xFFFFFFFF;
vic.Filter=1;
vic.Timing0=0x00;
vic.Timing1=0x14;
vic.Mode=0;
dwRel = VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic);
if(dwRel !=1)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

2.2.4 VCI_ReadBoardInfo

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。对应不同的产品型号*详见：适配器设备类型定义。*

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器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：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"

int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */

int   nDeviceInd = 0;         /*  第1个设备  */

VCI_BOARD_INFO vbi;

DWORD dwRel;

dwRel = VCI_ReadBoardInfo(nDeviceType, nDeviceInd, &vbi);

if(dwRel != 1)

{

   MessageBox(_T("获取设备信息失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);

   return FALSE;

}
```

2.2.5 VCI_GetReceiveNum

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。

实际应用中，用户可以忽略该函数，直接循环调用VCI_Receive，可以节约PC系统资源，提高程序效率。

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

参数：

DevType

设备类型。对应不同的产品型号*详见：适配器设备类型定义。*

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

返回值：

返回尚未被读取的帧数，=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"

int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */

int   nDeviceInd = 0;         /*  第1个设备  */

int   nCANInd = 0;            /*  第1个通道  */

DWORD dwRel;

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

2.2.6 VCI_ClearBuffer

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，同时发送缓冲区数据也会一并清除。

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

参数：

DevType

设备类型。对应不同的产品型号*详见：适配器设备类型定义。*

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"

int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */

int   nDeviceInd = 0;         /*  第1个设备  */

int   nCANInd = 0;            /*  第1个通道  */

DWORD dwRel;
```

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

2.2.7 VCI_StartCAN

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。对应不同的产品型号*详见：适配器设备类型定义*。

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"
int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */
int   nDeviceInd = 0;         /*  第1个设备  */
int   nCANInd = 0;            /*  第1个通道  */
DWORD dwRel;
VCI_INIT_CONFIG vic;
if(VCI_OpenDevice(nDeviceType, nDeviceInd, 0) != 1)
{
   MessageBox(_T("打开设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
   return FALSE;
}
if(VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic) != 1)
{
```

```
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("初始化设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
if(VCI_StartCAN(nDeviceType, nDeviceInd, nCANInd) !=1)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("启动设备失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

2.2.8 VCI_ResetCAN

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。对应不同的产品型号*详见：适配器设备类型定义*。

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"
int    nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */
int    nDeviceInd = 0;         /*  第1个设备  */
```

```
int    nCANInd = 0;                /* 第1个通道 */
DWORD dwRel;
dwRel = VCI_ResetCAN(nDeviceType, nDeviceInd, nCANInd);
if(dwRel != 1)
{
   MessageBox(_T("复位失败!"), _T("警告"), MB_OK|MB_ICONQUESTION);
   return FALSE;
}
```

2.2.9 VCI_Transmit

发送函数。返回值为实际发送成功的帧数。

```
DWORD __stdcall VCI_Transmit(DWORD DeviceType,DWORD DeviceInd,DWORD CANInd,PVCI_CAN_OBJ pSend,DWORD Length);
```

参数：

DevType

设备类型。对应不同的产品型号*详见：适配器设备类型定义。*

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

pSend

要发送的帧结构体 VCI_CAN_OBJ数组的首指针。

Length

要发送的帧结构体数组的长度（发送的帧数量）。最大为1000，建议设为1，每次发送单帧，以提高发送效率。

返回值：

返回实际发送的帧数，=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"
int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */
int   nDeviceInd = 0;         /*  第1个设备  */
int   nCANInd = 0;            /*  第1个通道  */
DWORD dwRel;
VCI_CAN_OBJ vco[48];
for(int i=0;i<48;i++)
{
   vco[i].ID = i;
   vco[i].RemoteFlag = 0;
   vco[i].ExternFlag = 0;
   vco[i].DataLen = 8;
   for(int j = 0;j<8;j++)
      vco.Data[j] = j;
}
dwRel = VCI_Transmit(nDeviceType, nDeviceInd, nCANInd, vco,48);
```

2.2.10 VCI_Receive

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。

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

参数：

DevType

设备类型。对应不同的产品型号*详见：适配器设备类型定义*。

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，这时再插入一个USB-CAN适配器那么后面插入的这个设备索引号就是1，以此类推。

CANIndex

CAN通道索引。第几路 CAN。即对应卡的CAN通道号，CAN1为0，CAN2为1。

pReceive

用来接收的帧结构体VCI_CAN_OBJ数组的首指针。注意：数组的大小一定要比下面的len参数大，否则会出现内存读写错误。

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)。该值为所提供的存储空间大小，适配器中为每个通道设置了2000帧左右的接收缓存区，用户根据自身系统和工作环境需求，在1到2000之间选取适当的接收数组长度。一般pReceive数组大小与Len都设置大于2000，如：2500为宜，可有效防止数据溢出导致地址冲突。同时每隔30ms调用一次VCI_Receive为宜。(在满足应用的时效性情况下,尽量降低调用VCI_Receive的频率，只要保证内部缓存不溢出，每次读取并处理更多帧，可以提高运行效率。)

WaitTime

保留参数。

返回值：

返回实际读取的帧数，=-1表示USB-CAN设备不存在或USB掉线。

示例：

```
#include "ControlCan.h"
int   nDeviceType = 4;        /* USBCAN-2A或USBCAN-2C或CANalyst-II */
int   nDeviceInd = 0;         /* 第1个设备 */
int   nCANInd = 0;            /* 第1个通道 */
DWORD dwRel;
VCI_CAN_OBJ vco[2500];
dwRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd, vco,2500,0);
if(lRel > 0)
{
…                             /* 数据处理 */
}
else if(lRel == -1)
{
…                             /* USB-CAN设备不存在或USB掉线，可以调用VCI_CloseDevice并重新
```

```
                                VCI_OpenDevice。如此可以达到USB-CAN设备热插拔的效果。 */
}
```

第三部分 其他函数及数据结构描述

本章描述了USB-CAN适配器的接口函数库ControlCAN.dll中为用户提供的非兼容周立功（ZLG）接口库的其他数据类型及函数说明，目的是扩展USB-CAN适配器功能，使其发挥最佳性能。若用的是兼容ZLG的模式进行二次开发，请勿调用这些函数，以免影响兼容性。

3.1 类型定义

3.2 函数描述

3.2.1 VCI_UsbDeviceReset

复位USB-CAN适配器，复位后需要重新使用VCI_OpenDevice打开设备。等同于插拔一次USB设备。

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

参数：

DevType

设备类型。*详见：适配器设备类型定义。*

DevIndex

设备索引，比如当只有一个USB-CAN适配器时，索引号为0，当有多个时，索引号从0开始依次递增。

Reserved

保留。

返回值：

返回值=1，表示操作成功；=0表示操作失败；=-1表示设备不存在。

示例：

```
#include "ControlCan.h"

int   nDeviceType = 4;        /*USB-CAN2.0 */

int   nDeviceInd = 0;         /*  第0个设备  */

int   nCANInd = 0;

DWORD dwRel;

bRel = VCI_UsbDeviceReset(nDeviceType, nDeviceInd, 0);
```

3.2.2 VCI_FindUsbDevice2

当同一台PC上使用多个USB-CAN的时候，可用此函数查找当前的设备，并获取所有设备的序列号。最多支持50个设备。

```
DWORD __stdcall VCI_FindUsbDevice2(PVCI_BOARD_INFO pInfo);
```

参数：

pInfo

结构体数组首地址，用来存储设备序列号等信息的结构体数组。数组长度建议定义为50，如：VCI_BOARD_INFO pInfo[50]。

返回值：

返回计算机中已插入的USB-CAN适配器的数量。

示例：

```
#include "ControlCan.h"
CString ProductSn[50];
VCI_BOARD_INFO pInfo [50];
int num=VCI_FindUsbDevice2(pInfo);
CString strtemp,str;
for(int i=0;i<num;i++)
{
    str="";
    for(int j=0;j<20;j++)
    {
      strtemp.Format("%c", pInfo [i]. str_Serial_Num [j]);
      str+=strtemp;
    }
    ProductSn[i]="USBCAN-"+str;
}
```

主要作用：

1、软件与硬件绑定：用户二次开发软件时，可以实现软件与硬件唯一绑定。通过调用该函数，可以获取到设备的序列号，与用户应用程序预先设定的序列号对比。实现软件与硬件的

唯一绑定与加密。单台设备使用时，也可以直接用VCI_ReadBoardInfo函数填充的VCI_BOARD_INFO结构体中的str_Serial_Num参数用于加密。

2、多卡同机时，设备索引号与序列号一一匹配：多台设备同时插在一台电脑时，要操作对应设备，就需要知道此时对应设备的设备索引号。通过调用该函数，结构体数组pInfo中对应的结构体中元素str_Serial_Num中即可保存所有设备的序列号，pInfo[0].str_Serial_Num、pInfo[1].str_Serial_Num、pInfo[2].str_Serial_Num设备索引号依次为0、1、2、3。

第四部分 接口库函数使用流程

为丰富设备功能，我们额外的提供了一些函数（下图中绿色背景表示的函数），这些函数包括：VCI_FindUsbDevice2、VCI_UsbDeviceReset。在进行二次开发时，这些函数不是必须调用的，在忽略这些函数的情况下就可以实现USB-CAN适配器的所有功能。