USB-CAN Bus Interface Adapter Interface Function Library User Instruction
## CONTENT

1. Part One Overview ........................................................................................................... 3

2. Part Two Compatible Function Library and Data Structure ........................................ 3
   
   2.1. Type Definition ........................................................................................................ 3
       2.1.1. Device Type .................................................................................................. 3
       2.1.2. VCI_BOARD_INFO ...................................................................................... 3
       2.1.3. VCI_CAN_OBJ ........................................................................................................ 5
       2.1.4. VCI_INIT_CONFIG ........................................................................................ 6

   2.2. Function description ............................................................................................. 8
       2.2.1. VCI_OpenDevice .......................................................................................... 8
       2.2.2. VCI_CloseDevice ........................................................................................ 9
       2.2.3. VCI_InitCan .................................................................................................. 9
       2.2.4. VCI_ReadBoardInfo ................................................................................... 13
       2.2.5. VCI_GetReceiveNum ...................................................................................... 14
       2.2.6. VCI_ClearBuffer .......................................................................................... 15
       2.2.7. VCI_StartCAN .............................................................................................. 16
       2.2.8. VCI_ResetCAN ............................................................................................ 17
       2.2.9. VCI_Transmit .............................................................................................. 18
       2.2.10. VCI_Receive ............................................................................................. 20

3. Part Three Other Functions and Data Structure Description .................................... 22
   
   3.1 Function description ............................................................................................. 22
       3.1.1. VCI_UsbDeviceReset ................................................................................. 22
       3.1.2. VCI_FindUsbDevice2 ............................................................................... 23
1. **PART ONE OVERVIEW**

If the user just use USB-CAN bus interface adapter to go on CAN bus communication test, and then he can directly use the supplied USB-CAN Tool software for sending and receiving data of the test.

If the user intends to write software program for his own products. Please carefully read the following instructions and take reference from the sample code we provide:

1. C++Builder
2. C#
3. VC
4. VB
5. VB.NET
6. Delphi
7. LabVIEW
8. LabWindows/CVI
9. Matlab
10. QT
11. Python/Python-can.

Develop library file : ControlCAN.lib, ControlCAN.DLL
VC version function declaration file : ControlCAN.h
VB version function declaration file: ControlCAN.bas
LabVIEW version library function package module : ControlCAN.llb
Delphi version function declaration file: ControlCAN.pas

2. **PART TWO COMPATIBLE FUNCTION LIBRARY AND DATA STRUCTURE**

2.1. **TYPE DEFINITION**

2.1.1. Device Type

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Type value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV_USBCAN2</td>
<td>4</td>
<td>USBCAN-2A/USBCAN-2C/CANalyst-II MiniPCIe-CAN</td>
</tr>
</tbody>
</table>

2.1.2. VCI_BOARD_INFO

VCI_BOARD_INFO structure contains USB-CAN Series interface card device information. The structure will be filled in VCI_ReadBoardInfo function.
```c
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;
```

**Member:**

**hw_Version**

Hardware version number, hexadecimal notation. E.g. 0x0100 represents V1.00.

**fw_Version**

Hardware version number, hexadecimal notation. E.g. 0x0100 represents V1.00.

**dr_Version**

Driver version number, hexadecimal notation. E.g. 0x0100 represents V1.00.

**in_Version**

Interface library version number, hexadecimal notation. E.g. 0x0100 represents V1.00.

**irq_Num**

System reserved.

**can_Num**

Represents the total number of CAN channel.

**str_Serial_Num**

This board card's serial number.

**str.hw_Type**

Hardware type, such as “USBCAN V1.00” (Note: Includes string terminator ‘\0’).

**Reserved**

System reserved.
2.1.3. VCI_CAN_OBJ

In the functions VCI_Transmit and VCI_Receive, VCI_CAN_OBJ structure is used to transmit CAN message frame.

```c
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;
```

Member:

**ID**

Message identifier. Direct ID format, right-aligned, please refer to: Annex One: ID Alignment Details.

**TimeStamp**

Receiving the stamp information of the time frame, start timing when the CAN controller is initialized, the unit is 0.1ms.

**TimeFlag**

In terms of whether to use the time stamp, 1 is the effective TimeStamp. TimeFlag and TimeStamp are only meaningful when the frame is received.

**SendType**

Sending type. = 0 indicates Normal type, = 1 indicates Single Send.

**RemoteFlag**

Whether it is a remote flag. = 1 indicates remote flag, = 0 indicates data flag.

**ExternFlag**

Whether it is an extern flag. = 1 indicates extern flag, = 0 indicates standard flag.

**DataLen**
Data length(<=8), that is, the length of data.

Data
Packet data.

Reserved
System reserved.

2.1.4. VCI_INIT_CONFIG

VCI_INIT_CONFIG structure defines the initialization configuration of the CAN. The structure will be filled in VCI_InitCan function.

```c
typedef struct _INIT_CONFIG {
    DWORD AccCode;
    DWORD AccMask;
    DWORD Filter;    //0,1 receives all frames, 2 standard frame filtering, 3 is an extended frame filtering.
    UCHAR Timing0;   //SJA1000 Baud rate parameter, Timing0 (BTR0)
    UCHAR Timing1;   //SJA1000 Baud rate parameter, Timing1 (BTR1)
    UCHAR Mode;      //mode, 0 represents normal mode, 1represents listening-only mode, 2
                     represents self-test mode.
} VCI_INIT_CONFIG, *PVCI_INIT_CONFIG;
```

Member:

AccCode
Receive filtered acceptance code.

AccMask
Receive filter mask.

Reserved
Reserved.

Filter
Filtering method, allowing setting range 0-3, refer to section 2.2.3 of the filter mode table for details.
Timing0

SJA1000 Baud rate parameter, Timing0 (BTR0).

Timing1

SJA1000 Baud rate parameter, Timing1 (BTR1).

Mode

Operating mode, 0 = normal operation, 1 = Listen-only mode, 2 = spontaneous admission and sending test mode.

Remarks:

About the filter settings please refer to: Annex II: CAN parameter setup instructions.

CAN Timing0 and Timing1 are used to set baud rate, these two parameters are only used at the initialization stage.

Conventional Baud reference table:

<table>
<thead>
<tr>
<th>CAN Baud rate</th>
<th>Timing0(BTR0)</th>
<th>Timing1(BTR1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k bps</td>
<td>0x31</td>
<td>0x1C</td>
</tr>
<tr>
<td>20k bps</td>
<td>0x18</td>
<td>0x1C</td>
</tr>
<tr>
<td>40k bps</td>
<td>0x87</td>
<td>0xFF</td>
</tr>
<tr>
<td>50k bps</td>
<td>0x09</td>
<td>0x1C</td>
</tr>
<tr>
<td>80k bps</td>
<td>0x83</td>
<td>0xFF</td>
</tr>
<tr>
<td>100k bps</td>
<td>0x04</td>
<td>0x1C</td>
</tr>
<tr>
<td>125k bps</td>
<td>0x03</td>
<td>0x1C</td>
</tr>
<tr>
<td>200k bps</td>
<td>0x81</td>
<td>0xFA</td>
</tr>
<tr>
<td>250k bps</td>
<td>0x01</td>
<td>0x1C</td>
</tr>
<tr>
<td>400k bps</td>
<td>0x80</td>
<td>0xFA</td>
</tr>
<tr>
<td>500k bps</td>
<td>0x00</td>
<td>0x1C</td>
</tr>
<tr>
<td>666k bps</td>
<td>0x80</td>
<td>0xB6</td>
</tr>
<tr>
<td>800k bps</td>
<td>0x00</td>
<td>0x16</td>
</tr>
<tr>
<td>1000k bps</td>
<td>0x00</td>
<td>0x14</td>
</tr>
<tr>
<td>33.33 Kbps</td>
<td>0x09</td>
<td>0x6F</td>
</tr>
<tr>
<td>66.66 Kbps</td>
<td>0x04</td>
<td>0x6F</td>
</tr>
<tr>
<td>83.33 Kbps</td>
<td>0x03</td>
<td>0x6F</td>
</tr>
</tbody>
</table>

Note:
1. Users only need to follow SJA1000 (16MHz) to set the Baud rate parameter.
2. The adapter does not support temporarily Baud rate below 10K.

## 2.2. FUNCTION DESCRIPTION

### 2.2.1. VCI_OpenDevice

This function is used to connect devices.

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

**Parameters:**

- `DevType`
  - Device type. See: Adapter device type definition.

- `DevIndex`
  - Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

- `Reserved`
  - Retention parameters, fill in 0.

**Returns:**

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

**E.g.:**

```c
#include "ControlCan.h"

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

DWORD dwRel;

dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, 0);

if(dwRel != 1)
{
    MessageBox(_T("Fail to open the device!")), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```
2.2.2. VCI_CloseDevice

This function is used to close the connection.

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

**Parameters:**

*DevType*

Device type. *See: Adapter device type definition.*

*DevIndex*

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

**Returns:**

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

**E.g.:**

```
#include "ControlCan.h"
int nDeviceType = 4;  /*USB-CAN2.0 */
int nDeviceInd = 0;   /* zeroth device */
DWORD dwRel;
dwRel = VCI_CloseDevice(nDeviceType, nDeviceInd);
if(dwRel != 1)
{
    MessageBox(_T("Fail to close the device!")},_T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

2.2.3. VCI_InitCan

This function is used to initialize the specified CAN.

```
DWORD __stdcall VCI_InitCan(DWORD DevType, DWORD DevIndex, DWORD CANIndex,
```

www.waveshare.com

www.waveshare.com/wiki
Parameters:

DevType

Device type. See: Adapter device type definition.

DevIndex

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

CANIndex

CAN channel index, such as when there is only one CAN channel, the index number is 0, if there are two, the index number can be 0 or 1.

pInitConfig

Initialization parameter structure.

Parameter list of members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Functional Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pInitConfig-&gt;AccCode</td>
<td>AccCode and AccMask can work together to determine which packets can be accepted. These two registers are used to set the ID left-aligned, that is, the highest bit (Bit31) of the AccCode and AccMask is aligned with the highest bit of the ID value. About ID alignment refer annexes: Annex I: ID alignment details. E.g.: If you set the value of the AccCode as 0x24600000 (i.e. 0x123 is shifted to the left by 21 bits), AccMask value is set to 0x00000000, and then only the packets with CAN message frame ID is 0x123 can be accepted (AccMask value of 0x00000000 indicates that all bits are relevant)</td>
</tr>
<tr>
<td>Member</td>
<td>Functional Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pInitConfig-&gt;Reserved</td>
<td>reserved</td>
</tr>
<tr>
<td>pInitConfig-&gt;Filter</td>
<td>Filtering mode settings please refer to the section of the filter mode table.</td>
</tr>
<tr>
<td>pInitConfig-&gt;Timing0</td>
<td>Baud rateT0 setting</td>
</tr>
<tr>
<td>pInitConfig-&gt;Timing1</td>
<td>Baud rateT1 setting</td>
</tr>
<tr>
<td>pInitConfig-&gt;Mode</td>
<td>Operating mode: 0-normal operation 1-Listen-only mode 2-spontaneous admission and sending test mode (this value is excluded from the ZLG function library)</td>
</tr>
</tbody>
</table>

**Filter mode table:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

Note: This filter setting examples to the standard frame, for example, high 11-bit is the valid bit; in the case of the extended frame, and then the valid ID is 29-bit. AccCode and AccMask set high 29-bit as the valid bit!
<table>
<thead>
<tr>
<th>Value</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Receive all types</td>
<td>Suitable to both standard and extended frame!</td>
</tr>
<tr>
<td>2</td>
<td>Only receive standard frame</td>
<td>Suitable to standard frame, and extended</td>
</tr>
<tr>
<td>3</td>
<td>Only receive extended frame</td>
<td>Suitable to extended frame, and standard frame will be removed by filtration directly!</td>
</tr>
</tbody>
</table>

**Returns:**

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

**E.g.:**
#include "ControlCan.h"

int nDeviceType = 4;  /*USB-CAN 2.0 */
int nDeviceInd = 0;   /* zeroth device */

DWORD dwRel;
VCI_INIT_CONFIG vic;
dwRel = VCI_OpenDevice(nDeviceType, nDeviceInd, 0);
if (dwRel != 1)
{
    MessageBox(_T("Fail to open the device!")), _T("warning"), 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("Fail to initialize the device!")), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}

2.2.4. VCI_ReadBoardInfo

This function is used to read the adapter hardware information. Generally speaking, it
can be ignored.
DWORD __stdcall VCI_ReadBoardInfo(DWORD DevType, DWORD DevIndex, PVCI_BOARD_INFO pInfo);

**Parameters:**

DevType

Device type. See: *Adapter device type definition*.

DevIndex

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

*pInfo*

VCI_BOARD_INFO is used to store device information structure pointer.

**Returns:**

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

**E.g.**:

```c
#include "ControlCan.h"

int nDeviceType = 4;       /*USB-CAN2.0 */
int nDeviceInd = 0;        /* zeroth device */
int nCANInd = 0;
VCI_BOARD_INFO vbi;
DWORD dwRel;

bRel = VCI_ReadBoardInfo(nDeviceType, nDeviceInd, nCANInd, &vbi);
if (dwRel != 1)
{
    MessageBox(_T("Fail to obtain device information!"), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

### 2.2.5. VCI_GetReceiveNum
This function is used to specify the received but has not been read frames in the designated receiving buffer.

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

**Parameters:**

- **DevType**: Device type. See: Adapter device type definition.

- **DevIndex**: Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

- **CANIndex**: CAN channel index.

**Returns:**

Return frames that have not been read yet.

**E.g.:**

```c
#include "ControlCan.h"

int ret=VCI_GetReceiveNum(2,0,0);
```

---

### 2.2.6. VCI_ClearBuffer

This function is used to clear the receive and send buffer of the designated channel specified by USB-CAN adapter.

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

**Parameters:**

- **DevType**: Device type. See: Adapter device type definition.

- **DevIndex**
Device Index, for example, when there is only one USB-CAN adapter, the index number is 0; when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

CANIndex

CAN channel index.

Returns:

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

E.g.:

```c
#include "ControlCan.h"

int nDeviceType = 4;  /*USB-CAN2.0*/
int nDeviceInd = 0;  /* zeroth device */
int nCANInd = 0;  /* zeroth channel */
DWORD dwRel;
bRel = VCI_ClearBuffer(nDeviceType, nDeviceInd, nCANInd);
```

2.2.7. VCI_StartCAN

This function is used to start the CAN controller and the internal interrupt reception function of the adapter.

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

Parameters:

DevType

Device type. See: Adapter device type definition.

DevIndex

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

CANIndex

CAN channel index.

Returns:
Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

E.g.:

```c
#include "ControlCan.h"

int nDeviceType = /*USB-CAN2.0 */
4; int nDeviceInd = /* zeroth device */
0; /* zeroth channel */

int nCANInd = 0;
DWORD dwRel;
VCI_INIT_CONFIG vic;

if(VCI_OpenDevice(nDeviceType, nDeviceInd, 0) != 1)
{
    MessageBox(_T("Fail to open the device"), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}

if(VCI_InitCAN(nDeviceType, nDeviceInd, nCANInd, &vic) != 1)
{
    VCI_CloseDevice(nDeviceType, nDeviceInd);
    MessageBox(_T("Fail to initialize the device"), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}

if(VCI_StartCAN(nDeviceType, nDeviceInd, nCANInd) != 1)
{
    MessageBox(_T("Fail to start the device"), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

### 2.2.8. VCI_ResetCAN

This function is used to reset the CAN controller.

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

DeviceType

Device type. See: Adapter device type definition.

DevIndex

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

CANIndex

CAN channel index.

Returns:

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

E.g.:

```c
#include "ControlCan.h"

int nDeviceType = 4; /*USB-CAN2.0*/
int nDeviceInd = 0; /* zeroth device */
int nCANInd = 0;
DWORD dwRel;
bRel = VCI_ResetCAN(nDeviceType, nDeviceInd, nCANInd);
if(dwRel != 1)
{
    MessageBox(_T("Fail to reset!")), _T("warning"), MB_OK|MB_ICONQUESTION);
    return FALSE;
}
```

### 2.2.9. VCI_Transmit

This function is used to send CAN message frame.

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

**DevType**

Device type. See: *Adapter device type definition*.

**DevIndex**

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

**CANIndex**

CAN channel index.

**pSend**

The first address of the data frame arrays that have to be sent.

**Length**

The number of the data frames that have to be sent, the maximum number is 1000, the recommended value is 48 under high speed.

**Returns:**

Return the actual number of frames already sent, the return value = -1 indicates a device error.

**E.g.:**

```c
#include "ControlCan.h"
int nDeviceType = 4; /*USB-CAN2.0 */
int nDeviceInd = 0; /* zeroth device */
```
```c
int nCANInd = 0;
DWORD dwRel;
VCI_CAN_OBJ vco[48];
ZeroMemory(&vco, sizeof(VCI_CAN_OBJ)*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_TxReceive

This function is used to request reception.

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

**Parameters:**

- **DevType**
  
  Device type. See: *Adapter device type definition*.

- **DevIndex**
  
  Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

- **CANIndex**
  
  CAN channel index.

- **pReceive**
  
  To receive the first set pointer of the data frames.
The array length of the data frame must be more than 2500 to return normal message.

Otherwise, the return length will be zero whether the message is received or not. The adapter set a 2000-frame buffer for every channel. Based on his own system and working environment, the user can choose an appropriate array length from 2500.

*WaitTime*

Reserved.

**Returns:**

Return the number of frames that actually have been read, -1 indicates device errors.

**E.g.**

```c
#include "ControlCan.h"
int nDeviceType = 4; /*USB-CAN2.0*/
int nDeviceInd = 0; /* zeroth device */
int nCANInd = 0;
long lRel;
VCI_CAN_OBJ vco[2500];
lRel = VCI_Receive(nDeviceType, nDeviceInd, nCANInd, &vco,2500,0);
if(lRel > 0)
{
...
/*data processing */
}
else if(lRel == -1)
{
...
/* receive error treatment */
}
3. PART THREE OTHER FUNCTIONS AND DATA STRUCTURE DESCRIPTION

This chapter describes other data types and functions of the incompatible ZLG interface library contained in USB-CAN adapter interface library ControlCAN.dll. Please do not call these functions if use a compatible ZLG model for secondary development so as not to affect compatibility.

3.1 FUNCTION DESCRIPTION

3.1.1. VCI_UsbDeviceReset

Reset USB-CAN adapter, need to re-open the device after reset by using VCI_OpenDevice.

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

Parameters:

DevType

Device type. See: Adapter device type definition.

DevIndex

Device Index, for example, when there is only one USB-CAN adapter, the index number is 0, when there are multiple USB-CAN adapters, the index numbers in an ascending order starting from 0.

Reserved

Reserved.

Returns:

Return value = 1, which means that the operation is successful; = 0 indicates that the operation failed; = -1 indicates that the device does not exist.

E.g.:

```
#include "ControlCan.h"

int nDeviceType = 4;    /*USB-CAN2.0 */
int nDeviceInd = 0;     /* zeroth device */
int nCANInd = 0;
DWORD dwRel;
```
bRel = VCI_UsbDeviceReset(nDeviceType, nDeviceInd, 0);

### 3.1.2. VCI_FindUsbDevice2

When the same PC using multiple USB-CAN, user can use this function to find the current device.

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

**Parameters:**

`pInfo`

`pInfo` is used to store the parameters of the first data buffer address pointer.

**Returns:**

Return the number of the USB-CAN adapter plugged into the computer.

```c
#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;
}
```
Part Four Interface Library Functions Using Process

In order to multiply device function, we provided additional functions (functions presented with a green background), these functions include: `VCI_FindUsbDevice2` and `VCI_UsbDeviceReset`. During the second development, these functions are not necessarily to be invoked. Even these functions are ignored, all USB-CAN adapter functions can be achieved.