

Ginkgo USB-CAN Interface VTG203B API Library Instruction v1.2



1 D	evice Type Definition	- 1 -	
2 E	rror Code Definition	- 2 -	
3 F	unction Data Definition	- 3 -	
	3.1 VCI_BOARD_INFO	- 3 -	
	3.2 VCI_BOARD_INFO_EX	- 4 -	
	3.3 VCI_CAN_OBJ	- 4 -	
	3.4 VCI_CAN_STATUS	- 5 -	
	3.5 VCI_ERR_INFO	- 6 -	
	3.6 VCI_INIT_CONFIG	- 7 -	
	3.7 VCI_INIT_CONFIG_EX	- 8 -	
	3.8 VCI_FILTER_CONFIG	10 -	
4 In	terface Function Description	12 -	
	4.1 VCI_ScanDevice	12 -	
	4.2 VCI_OpenDevice	12 -	
	4.3 VCI_CloseDevice	13 -	
	4.4 VCI_InitCAN	14 -	
	4.5 VCI_InitCANEx	15 -	
	4.6 VCI_ReadBoardInfo	16 -	
	4.7 VCI_ReadBoardInfoEx	16 -	
	4.8 VCI_ReadErrInfo	17 -	
	4.9 VCI_ReadCANStatus	18 -	
	4.10 VCI_SetFilter	19 -	
	4.11 VCI_GetReceiveNum	20 -	
	4.12 VCI_ClearBuffer	20 -	
	4.13 VCI_StartCAN	21 -	
	4.14 VCI_ResetCAN	22 -	
	4.15 VCI_Transmit	23 -	
	4.16 VCI_Receive	24 -	
5 Fı	unction Usage	26 -	
	5.1 Call API from DLL in VC	26 -	
	5.2 Call API from DLL in VB	27 -	
6 Fı	unction Call Procedure	29 -	



1 Device Type Definition

Ginkgo Adapter type definition:

Device Name	Device Type
USB-I2C	1
USB-SPI	2
USB-CAN1	3
USB-CAN2	4



2 Error Code Definition

Ginkgo USB-CAN Adapter API Functionality, return code definition:

Name	Value	Description	
CAN Error Code			
ERR_CAN_OVERFLOW	0x0001	CAN Adapter Internal FIFO Overflow	
ERR_CAN_ERRALARM	0x0002	CAN Adapter Error Warring	
ERR_CAN_PASSIVE	0x0004	CAN Adapter Passive Error	
ERR_CAN_LOSE	0x0008	CAN Adapter lose Error	
ERR_CAN_BUSERR	0x0010	CAN Adapter Bus Error	
ERR_CAN_BUSOFF	_CAN_BUSOFF 0x0020 CAN Adapter Bus Off		
Common Error Code			
ERR_DEVICEOPENED	0x0100	Device Opened	
ERR_DEVICEOPEN	0x0200	Open Device Error	
ERR_DEVICENOTOPEN	0x0400	Device Not Open	
ERR_BUFFEROVERFLOW	R_BUFFEROVERFLOW 0x0800 Buffer Overflow		
ERR_DEVICENOTEXIST	0x1000	Device Not Exist	
ERR_LOADKERNELDLL	0x2000	DLL Loading Failed	
ERR_CMDFAILED	0x4000	Command Execution Failed	
ERR_BUFFERCREATE	0x8000	Memory Allocation Error	



3 Function Data Definition

3.1 VCI_BOARD_INFO

Description

VCI_BOARD_INFO structure include Ginkgo USB-CAN device information of adapters, structure should be filled out in function VCI_ReadBoardInfo.

```
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];
              str_hw_Type[40];
   CHAR
   USHORT
              Reserved[4];
} VCI_BOARD_INFO,*PVCI_BOARD_INFO;
```

Member

hw Version

Hardware version, in Hex format, eg: 0X0100 presents V1.00.

fw Version

Firmware version, in Hex format.

dr Version

Driver version, in Hex format.

in Version

API DLL version, in Hex format.

irq_Num

Interrupt number used by adapter.

can_Num

Adapter CAN channel number.

str_Serial_Num

Adapter Serial Number.

str_hw_Type

Hardware Type, for example "USBCAN V1.00" (Note: Include string terminator '\0').

Reserved

Reserved.



3.2 VCI_BOARD_INFO_EX

Description

VCI_BOARD_INFO_EX structure used for extension of VCI_BOARD_INFO, Adapter device information has been included, It will be filled out in function: VCI ReadBoardInfoEx.

```
typedef struct _VCI_BOARD_INFO_EX{
   USHORT
               dr Version;
              in Version;
   USHORT
   BYTE
              can_Num;
   CHAR
               str_hw_Name[100];
   CHAR
              str_fw_Version[100];
   CHAR
               str_hw_Version[100];
   CHAR
               str_Serial_Num[100];
   USHORT
               Reserved[4];
} VCI_BOARD_INFO_EX,*PVCI_BOARD_INFO_EX;
```

Member

dr Version

Driver version, in Hex format.

in Version

API DLL version, in Hex format.

can_Num

Adapter CAN channel number.

str_hw_Name

Hardware name, for example "Ginkgo-CAN-Adaptor" (Note: include string terminator '\0').

str fw Version

Firmware version,, for example: "2.0.0" (Note: include string terminator '\0').

str_hw_Version

Hardware version, for example: "2.1.0" ((Note: include string terminator '\0').

str_Serial_Num

Adapter serial number string.

Reserved

Reserved.

3.3 VCI CAN OBJ

Description

VCI_CAN_OBJ structure is used to transfer CAN frame in function VCI_Transmit and



VCI_Receive.

```
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

Text ID.

TimeStamp

Timestamp of receive data frame, started from initialization of CAN controller.

TimeFlag

If using TimeStamp, set to 1 effective TimeFlag and TimeStamp is available for receive frame only.

SendType

Frame type flag, 0: Normal send, 1: One time send, 2: Auto send, 3: One time auto send/receive, 5: Fast send, only send frame send/receive.

RemoteFlag

Remote frame flay, 1: remote frame, 0: data frame.

ExternFlag

Extension frame flag, 1: Extension frame, 0: Standard frame.

DataLen

Data length (<=8), the length of Data.

Data

Data.

Reserved

Reserved.

3.4 VCI CAN STATUS

Description

VCI_CAN_STATUS structure includes CAN controller status information. Structure will be filled out in function VCI ReadCanStatus.

```
typedef struct _VCI_CAN_STATUS{

UCHAR ErrInterrupt;

UCHAR regMode;
```



```
UCHAR regALCapture;
UCHAR regECCapture;
UCHAR regEWLimit;
UCHAR regRECounter;
UCHAR regTECounter;
UCHAR regTECounter;
UINT regESR;
UINT regTSR;
UINT BufferSize;
DWORD Reserved;

}VCI_CAN_STATUS,*PVCI_CAN_STATUS;
```

Member

ErrInterrupt

Reserved, undefined.

regMode

Reserved, undefined.

regStatus

Reserved, undefined.

regALCapture

Reserved, undefined.

regECCapture

Reserved, undefined.

regEWLimit

Reserved, undefined.

regRECounter

CAN adapter receive error counter.

regTECounter

CAN adapter send error counter.

regESR

CAN adapter error status register.

regTSR

CAN adapter send status register.

BufferSize

CAN adapter receive buffer free size.

Reserved

Reserved, undefined.

3.5 VCI_ERR_INFO

Description

VCI_ERR_INFO structure used to record error information, when VCI DLL running,



structure will be filled in function: VCI_ReadErrInfo.

Member

ErrCode

Error Code.

Passive_ErrData

Passive error when there is passive error, error, presents.

ArLost ErrData

Arbitration lost, Error data.

3.6 VCI_INIT_CONFIG

Description

VCI_INIT_CONFIG structure defined configuration of CAN initialization. Structure will be filled out in function: VCI_InitCan.

```
typedef struct _INIT_CONFIG{
    DWORDAccCode;
    DWORDAccMask;
    DWORDReserved;
    UCHAR Filter;
    UCHAR Timing0;
    UCHAR Timing1;
    UCHAR Mode;
}VCI_INIT_CONFIG,*PVCI_INIT_CONFIG;
```

Member

AccCode

Filter acknowledgment code.

AccMask

Filter mask code.

Reserved

Reserved.

Filter

Filter mode, 1: Single fitter, 0: Double fitter.

Timing0

Time 0 (BTR0)_o

Timing1

Time 1 (BTR1)_o

Mode



Mode, 0: normal, 1: Listen only.

Remark

Timing0 and Timing1 used for configure CAN band rate, several common used band rate set up see below:

CAN band rate	Timing0	Timing1
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

3.7 VCI_INIT_CONFIG_EX

Description

VCI_INIT_CONFIG_EX structure defined configuration of CAN initialization. Structure will be filled out in function: VCI_InitCanEx.

```
typedef struct _INIT_CONFIG_EX

{
    DWORDCAN_BRP;
    UCHAR CAN_SJW;
    UCHAR CAN_BS1;
    UCHAR CAN_BS2;
    UCHAR CAN_Mode;
    UCHAR CAN_Mode;
    UCHAR CAN_ABOM;
    UCHAR CAN_NART;
    UCHAR CAN_RFLM;
    UCHAR CAN_TXFP;
    DWORDReserved;
}VCI_INIT_CONFIG_EX;*
```

Member

CAN_BRP

Band rate, Divider. Range: 1~1024.



CAN_SJW

The time unit threshold of extending and shorten of data bits. Range: 1~4.

CAN BS1

The location of sampling. It includes PROP_SEG and PHASE_SEG1 in CAN standard. Range: 1~16.

CAN BS2

Define the location of sending. It includes PHASE_SEG2 in CAN standard. Range: 1~8.

CAN_Mode

CAN adapter working mode, 0: Normal, 1: Loop, 2: Silent, 3: Silent around.

CAN ABOM

Auto off management. 0: Auto off prohibition, 1: Enable auto off.

CAN NART

Packet retransmission management: '=0' make packet retransmission, '=1' no packet retransmission

CAN_RFLM

FIFO locked management of received new data. When FIFO is full. 0: overwrite old data, 1: ignore new data.

CAN_TXFP

Send priority management. 0: by identification, 1: by send request order.

Reserved

Reserved.

Remark

CAN_BRP, CAN_SJW, CAN_BS1, CAN_BS2 CAN band rate is affected by four. parameters: CAN_BRP = $36MHz/(CAN_BRP)/(CAN_SJW+CAN_BS1+CAN_BS2)$. For example CAN_BRP = 6, CAN_SJW = 1, CAN_BS1 = 3, CAN_BS2 = 2, CAN band rate = 36MHz/6/(1+3+2) = 1Mbps. Band rate set up table (four parameters).

Band rate (bps)	CAN_BRP	CAN_SJW	CAN_BS1	CAN_BS2
1M	6	1	3	2
900K	5	1	4	3
800K	5	1	5	3
600K	6	1	6	3
666K	9	1	5	3
500K	12	1	3	2
400K	10	1	5	3
300K	20	1	3	2
250K	24	1	3	2
200K	30	1	3	2
150K	40	1	3	2
125K	48	1	3	2
100K	60	1	3	2
90K	50	1	4	3



80K	75	1	3	2
60K	60	1	6	3
50K	120	1	3	2
40K	150	1	3	2
30K	120	1	6	3
20K	300	1	3	2
10K	600	1	3	2
5K	600	2	6	4
3K	1000	2	6	4
2K	1000	2	10	6

3.8 VCI_FILTER_CONFIG

Description

VCI_FILTER_CONFIG structure defined CAN adapter filter configuration. Structure will be filled in function: VCI_SetFilter.

```
typedef struct _VCI_FILTER_CONFIG{
    UCHAR Enable;
    UCHAR FilterIndex;
    UCHAR FilterMode;
    UCHAR ExtFrame;
    DWORDID_Std_Ext;
    DWORDID_IDE;
    DWORDID_RTR;
    DWORD MASK_Std_Ext;
    DWORD MASK_IDE;
    DWORD MASK_RTR;
    DWORD Reserved;
} VCI_FILTER_CONFIG,*PVCI_FILTER_CONFIG;
```

Member

Enable

Filter enable, 1: enable, 0: disable.

FilterIndex

Filter index, Range: 0~13.

FilterMode

Filter mode, 0: Mask bit mode, 1: Identification.

ExtFrame

Filter frame type flag, 1: the filter frame is list mode extension, 0: the filter frame is standard frame.

ID Std Ext

Verification ID.



ID_IDE

Verification IDE.

ID_RTR

Verification RTR.

MASK_Std_Ext

Mask ID, only used for filter mode is mask bit mode.

MASK_IDE

Mask IDE, only used for filter mode is mask bit mode.

MASK_RTR

Mask RTR, only used for filter mode is mask bit mode.

Reserved

Reserved.



4 Interface Function Description

4.1 VCI_ScanDevice

Description:

Scan all connected Ginkgo adapter. Return the connected device numbers.

Declaration:

```
DWORD __stdcall VCI_ScanDevice(UCHAR NeedInit=1);
```

Parameter:

NeedInit

If need to initiate device. Set to 1 when first time call this function, set to 0 or 1 after first time usage.

Return:

The connected device numbers.

Example:

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevNum;
DevNum = VCI_ScanDevice(1);
if(DevNum > 0){
    printf("Have %d device connected!\n",DevNum);
}else{
    printf("No device connected!\n");
    return 0;
}
```

4.2 VCI_OpenDevice

Description:

Open device.

Declaration:

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

Parameter:

DevType

Device type.

DevIndex

Device index, index number from 0.

Reserved



Reserved.

Return:

1: Success, 0: Failed.

Example:

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int Status;
Status = VCI_OpenDevice(DevType,DevIndex,0);
if(Status == 1){
    printf("Open device success!\n");
}else{
    printf("Open device failed!\n");
}
```

4.3 VCI_CloseDevice

Description:

Close device.

Declaration:

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

DevType

Device type.

DevIndex

Device index, index number from 0.

返回值:

1: Success, 0: Failed.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int Status;
Status = VCI_CloseDevice(DevType,DevIndex);
if(Status == 1){
    printf("Close device success!\n");
}else{
    printf("Close device failed!\n");
```



4.4 VCI InitCAN

```
Description:
    Initialize CANadapter.

Declaration:

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

DevType
    Device type.

DevIndex
    Device index.

CANIndex
    CANchannel index number.

pInitConfig
    Initialization parameter structure.

Return:

1: Success, 0: Failed.
```

```
#include <stdio.h>
#include "Ginkgo_Driver.h"
int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;
VCI_INIT_CONFIG InitConfig;
InitConfig.AccCode = 0x00000000;
InitConfig.AccMask = 0xFFFFFFF;
InitConfig.Filter = 1;
InitConfig.Timing0 = 0X00;
InitConfig.Timing1 = 0X14;
InitConfig.Mode = 0;
Status = VCI_InitCAN(DevType,DevIndex,CANIndex,&InitConfig);
if(Status == 1){
    printf("Initialize device success!\n");
}else{
    printf("Initialize device failed!\n");
```



4.5 VCI InitCANEX

Description:

Initialize CAN adapter, it is extension of VCI_InitCAN, used for more configuration of Ginkgo USB-CAN initialization.

Declaration:

```
DWORD __stdcall VCI_InitCANEx(DWORD DevType, DWORD DevIndex, DWORD CANIndex, PVCI_INIT_CONFIG_EX pInitConfig);
```

DevType

Device type.

DevIndex

Device index

CANIndex

CAN channel index number.

plnitConfig

Initialization parameter structure.

Return:

1: Success, 0: Failed.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"
int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;
VCI_INIT_CONFIG_EX InitConfig;
InitConfig.CAN BRP = 6;
InitConfig.CAN_SJW = 1;
InitConfig.CAN_BS1 = 3;
InitConfig.CAN_BS1 = 2;
InitConfig.CAN_Mode = 0;
InitConfig.CAN ABOM = 1;
InitConfig.CAN_NART = 0;
InitConfig.CAN RFLM = 1;
InitConfig.CAN_TXFP = 1;
Status = VCI_InitCANEx(DevType,DevIndex,CANIndex,&InitConfig);
if(Status == 1){
    printf("Initialize device success!\n");
}else{
```



```
printf("Initialize device failed!\n");
}
```

4.6 VCI_ReadBoardInfo

```
Description:
```

Acquire Ginkgo USB-CAN adapter information.

Declaration:

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

DevType

Device type.

DevIndex

Device index.

plnfo

The pointer of VCI BOARD INFO, used to record device information.

Return:

1: Success, 0: Failed.

Example:

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int Status;

VCI_BOARD_INFO Info;

Status = VCI_ReadBoardInfo(DevType,DevIndex,&Info);
if(Status == 1){
    printf("Read board information success!\n");
}else{
    printf("Read board information failed!\n");
}
```

4.7 VCI ReadBoardInfoEx

Description:

Acquire more information of Ginkgo USB-CAN.

Declaration:

```
DWORD __stdcall VCI_ReadBoardInfoEx(DWORD DevType, DWORD DevIndex, PVCI_BOARD_INFO_EX pInfo);
```

DevType



Device type.

DevIndex

Device index.

plnfo

The pointer of VCI_BOARD_INFO_EX, used for record device information.

Return:

1: Success, 0: Failed.

Example:

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int Status;

VCI_BOARD_INFO_EX Info;

Status = VCI_ReadBoardInfoEx(DevType,DevIndex,&Info);
if(Status == 1){
    printf("Read board information success!\n");
}else{
    printf("Read board information failed!\n");
}
```

4.8 VCI_ReadErrInfo

Description:

Get last error information of function call. For example, get one function error info, call this function just after it.

Declaration:

DWORD __stdcall VCI_ReadErrInfo(DWORD DevType,DWORD DevIndex,DWORD CANInd,PVCI_ERR_INFO pErrInfo);

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

pErrInfo

The pointer of VCI_ERR_INFO, used for record of error information.

Return:

1: Success, 0: Failed.



```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;
VCI_ERR_INFO ErrInfo;

Status = VCI_ReadErrInfo(DevType,DevIndex,CANIndex,&ErrInfo);
if(Status == 1){
        printf("Read error information success!\n");
}else{
        printf("Read error information failed!\n");
}
```

4.9 VCI_ReadCANStatus

Description:

Get CAN bus status.

Declaration:

DWORD __stdcall VCI_ReadCANStatus(DWORD DevType,DWORD DevIndex,DWORD CANIndex,PVCI CAN STATUS pCANStatus);

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

pCANStatus

The pointer of VCI_CAN_STATUS, used for record CAN bus status information.

Return:

1: Success, 0: Failed.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;
VCI_CAN_STATUS CANStatus;
```



```
Status = VCI_ReadCANStatus(DevType,DevIndex,CANIndex,&CANStatus);

if(Status == 1){
    printf("Read CAN status success!\n");
}else{
    printf("Read CAN status failed!\n");
}
```

4.10 VCI_SetFilter

Description:

Get CAN bus status.

Declaration:

```
DWORD __stdcall VCI_SetFilter(DWORD DevType,DWORD DevIndex,DWORD CANIndex,PVCI_FILTER_RECORD pFilter);
```

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

pFilter

The pointer of VCI_FILTER_CONFIG, used for record filter configuration parameter.

Return:

1: Success, 0: Failed.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;

VCI_FILTER_CONFIG Filter;

Filter.Enable = 1;
Filter.FilterIndex = 0;
Filter.FilterMode = 0;
Filter.ExtFrame = 0;
Filter.ID_Std_Ext = 0x000;
Filter.ID_IDE = 0;
Filter.ID_RTR = 0;
```



```
Filter.MASK_Std_Ext = 0x000;
Filter.MASK_IDE = 0;
Filter.MASK_RTR = 0;

Status = VCI_SetFilter(DevType,DevIndex,CANIndex,&Filter);
if(Status == 1){
    printf("Set CAN Filter success!\n");
}else{
    printf("Set CAN Filter failed!\n");
}
```

4.11 VCI GetReceiveNum

Description:

Get message number of CAN adapter received.

Declaration:

ULONG <u>__stdcall</u> VCI_GetReceiveNum(DWORD DevType,DWORD DevIndex,DWORD CANIndex);

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

Return:

Receive message number of CAN adapter received.

Example

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
long MsgNum;

MsgNum = VCI_GetReceiveNum(DevType,DevIndex,CANIndex);
printf("CAN Adapter get %d CAN frames!\n",MsgNum);
```

4.12 VCI ClearBuffer

Description:

Clear CAN adapter received message.



```
Declaration:
          __stdcall VCI_ClearBuffer(DWORD DevType,DWORD DevIndex,DWORD
DWORD
CANIndex);
  DevType
    Device type.
  DevIndex
    Device index.
  CANIndex
    CAN channel index.
 Return:
    1: Success, 0: Failed.
 Example:
#include <stdio.h>
#include "Ginkgo_Driver.h"
int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;
```

4.13 VCI_StartCAN

printf("Clear buffer success!\n");

printf("Clear buffer failed!\n");

MsgNum = VCI ClearBuffer(DevType, DevIndex, CANIndex);

Description:

if(Status == 1){

}else{

Start CAN adapter auto receive thread, call this function before stopping CAN adapter receiving or exit.

Declaration:

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

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

Return:



1: Success, 0: Failed.

Example:

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;

MsgNum = VCI_StartCAN(DevType,DevIndex,CANIndex);
if(Status == 1){
    printf("Start CAN success!\n");
}else{
    printf("Start CAN failed!\n");
}
```

4.14 VCI_ResetCAN

Description:

Stop CAN bus auto receive data. After stopping auto receive CAN adapter received data will be saved to CAN adapter internal data buffer.

Declaration:

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

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

Return:

1: Success, 0: Failed.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int Status;
```



```
MsgNum = VCI_ResetCAN(DevType,DevIndex,CANIndex);
if(Status == 1){
    printf("Reset CAN success!\n");
}else{
    printf("Reset CAN failed!\n");
}
```

4.15 VCI Transmit

Description:

Send CAN data.

Declaration:

```
ULONG __stdcall VCI_Transmit(DWORD DevType,DWORD DevIndex,DWORD CANIndex,PVCI_CAN_OBJ pSend,ULONG Len);
```

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

pSend

The pointer of data frame array to be sent.

Len

The array size of data frame array to be sent.

Return:

Actual sent data frame number.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
int i = 0,j = 0;
long MsgNum;

VCI_CAN_OBJ CAN_Data[10];

for(i=0;i<10;i++){
    CAN_Data[i].ID = i;
    CAN_Data[i].SendType = 0;
    CAN_Data[i].RemoteFlag = 0;
    CAN_Data[i].ExternFlag = 0;
```



```
CAN_Data[i].DataLen = 8;

for(j=0;j<8;j++){
        CAN_Data[i].Data[j] = j;
    }
}

MsgNum = VCI_Transmit(DevType,DevIndex,CANIndex,CAN_Data,10);
printf("Sended %d CAN frames!\n",MsgNum);
```

4.16 VCI Receive

Description:

Get received data from CAN adapter data buffer.

Declaration:

```
ULONG __stdcall VCI_Receive(DWORD DevType,DWORD DevIndex,DWORD CANIndex,PVCI_CAN_OBJ pReceive,ULONG Len,INT WaitTime=-1);
```

DevType

Device type.

DevIndex

Device index.

CANIndex

CAN channel index.

pReceive

The pointer of received data frame array.

Len

The size of array received data frame array.

WaitTime

Wait time, unit: ms, -1: no waiting required.

Return:

Actual received frame numbers, 0xFFFFFFF: failed to read data, there is error, call VCI_ReadErrInfo to get error code.

```
#include <stdio.h>
#include "Ginkgo_Driver.h"

int DevType = 4;//USB-CAN2
int DevIndex = 0;
int CANIndex = 0;
long MsgNum;
VCI_CAN_OBJ CAN_Data[10];

MsgNum = VCI_Receive(DevType,DevIndex,CANIndex,CAN_Data,10);
```



```
if(MsgNum != 0xFFFFFFF){
    printf("Get %d CAN frames!\n",MsgNum);
}else{
    printf("Get data error!\n");
}
```



5 Function Usage

5.1 Call API from DLL in VC

There are two ways to call API function from DLL. Static calling and dynamic calling.

- Procedure of static calling:
 - 1. Copy Ginkgo_Driver.dll to project Debug/Release dir;
 - 2. Copy Ginkgo_Driver.lib to project dir;
 - 3. Copy Ginkgo_Driver.h to project dir;
 - Setup link property to link Ginkgo_Driver.lib procedure of setup is:
 VC environment -> project ->property -> linker -> input -> additional dependency, click to open a editor dialog, type in Ginkgo_Driver.lib。
 - 5. In cpp file, include Ginkgo Driver.h.
 - Using static method to call API from DLL as DLL is generated in static method;
 - Set up: project -> property -> C/C++-> advanced -> call convention, select __stdcall (/Gz).
 - 7. May need to setup character property for some IDE, for example, ineed setup in VS2010 by: project -> property -> general ->charcter, select multi-byte, character set.
- Procedure of dynamic calling:
 - Copy Ginkgo_Driver.dll to project Debug/Release dir;
 - Define function pointer, accordingly, for example:

```
typedef int(*SPI Funtion)(char *in,char *out);
```

 Define function pointer variable, and get corresponding handler, function could be used by function pointer.

Below is a complete example showing how to use it.

```
#include <stdio.h>
#include <Windows.h>

//Define SPI Adapter API function pointer
typedef int(*SPI_Funtion)(char *in,char *out);
int main()
{
    int ErrorCode;
    char outString[10240]={0};
    SPI_Funtion EnumBoard,OpenDevice;
    HINSTANCE hDLL; //DLL handle
    hDLL = LoadLibrary("Ginkgo_Driver.dll");//Loading DLL
    //Acquire VTIF_USB_EnumBoard function pointer address
    EnumBoard = (SPI_Funtion)GetProcAddress(hDLL,"VTIF_USB_EnumBoard");
    //Acquire VTIF_OpenDevice function pointer address
```



```
OpenDevice = (SPI_Funtion)GetProcAddress(hDLL,"VTIF_OpenDevice");
//Scan device
ErrorCode = EnumBoard("100",outString);
if(ErrorCode>0){
    printf("Connected device number is:%d\n",ErrorCode);
}else{
    printf("No device connect!\n");
    return ErrorCode;
}
//Open device
ErrorCode = OpenDevice("0",outString);
if(ErrorCode!=0){
    printf("Open device error!\n");
    return ErrorCode:
}else{
    printf("Open device success!\n");
FreeLibrary(hDLL);
return 0;
```

5.2 Call API from DLL in VB

There are two methods to call API function from dll, static calling and dynastic calling, The most common way is static calling, so only static method has introduced hereby. The procedure of static calling is:

- 1. Copy Ginkgo_Driver.dll to project dir;
- Declare function inside dll in VB in windows level or module level or global module.
 Declaration format 1: Public/Private Declare Sub [Function name] Lib "DLL file name" [Alias] (Parameter variable table).

Declaration format 2: Public/Private Declare Function [Function name] Lib "DLL file name" [Alias] (Parameter variable table) [As data type of return value].

Firstly using keyword declare to declare function in DLL, In C, some of function type is void, there is no return value, then should using Sub. To declare it to procedure for the function has return value, using keyword function to declare it as function, and using keyword "AS" to specify function return type at the end of declaration statement .For example:

Private Declare Function VTIF_I2C_Init Lib "E:\myprogram\Ginkgo_Driver.dll" (ByVal a As Any, ByVal b As Any) As Long

Function declaration will be added to VB by this declaration, and could be used right way. When debugging VB function, declaration statement should specify dll's absolute directory

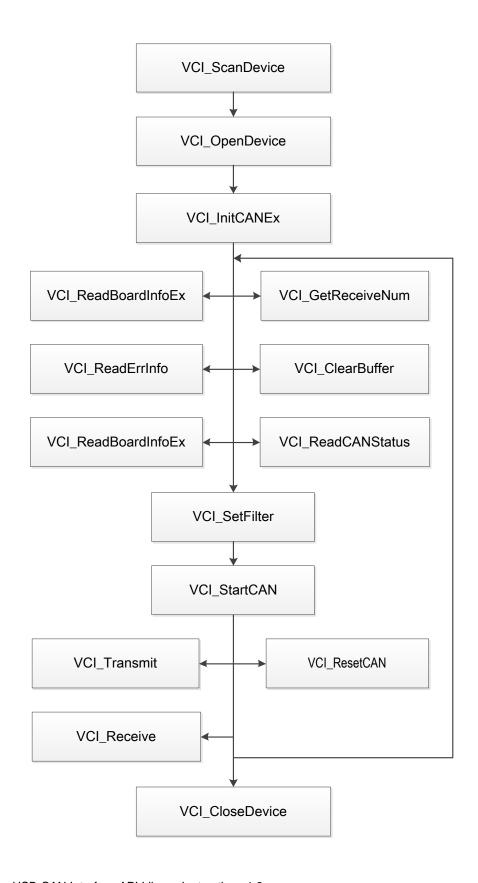


path. When generate executed program, it is not necessary to specify dll's directory in declaration statement.

Please refer viewtool website VIS application examples to get more information on the method usage.



6 Function Call Procedure





Technical Support and Service

- Lifetime technical support, free services and maintenance within one year.
- Down load diver, software, User's Manual and application routine source code at www.viewtool.com.
- Technical exchange forum: www.viewtool.com/bbs
- Technical support Email: <u>FAE@viewtool.com</u>
- Sales Email: <u>Sales@viewtool.com</u>