NuCOMâ

PCI-7841/cPCI-7841/PM-7841

Dual-Port Isolated

CAN Interface Card

User's Guide

Manual first edition: June 1, 1998
Manual Rev. 2.00: July 15, 1998
Manual Rev. 2.11: Oct. 8, 1999
Manual Rev. 3.00: May. 28 2000

The information in this document is subject to change without prior notice in order to improve reliability, design and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

Trademarks

PCI-7841, cPCI-7841, and PM-7841 are registered trademarks of ADLink Technology Inc. Other product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

CONTENTS

1

Introduction

The PCI/cPCI/PM-7841 is a Controller Area Network (CAN) interface card used for industrial PC with PCI, Compact-PCI, and PC104 bus. It supports dual ports CAN's interface that can run independently or bridged at the same time. The built-in CAN controller provides bus arbitration and error detection with auto correction and re-transmission function. The PCI cards are plug and play therefore it is not necessary to set any jumper for matching the PC environment.

The CAN (Controller Area Network) is a serial bus system originally developed by Bosch for use in automobiles, is increasing being used in industry automation. It multi-master protocol, real-time capability, error correction and high noise immunity make it especially suited for intelligent I/O devices control network.

The PCI/cPCI/PM-7841 is programmed by using the ADLink's software library. The programming of this PCI card is as easy as AT bus add-on cards.

1.1 PCI/cPCI/PM-7841 Features

The PCI-7841 is a Dual-Port Isolated CAN Interface Card with the following features:

- Two independent CAN network operation
- Bridge function supports
- Compatible with CAN specification 2.0 parts A and B
- Optically isolated CAN interface up to 2500 Vrms isolation protection
- Direct memory mapping to the CAN controllers
- Powerful master interface for CANopen, DeviceNet and SDS application layer protocol
- Up to 1Mbps programmable transfer rate
- Supports standard DeviceNet data rates 125, 250 and 500 Kbps
- PCI bus plug and play
- DOS library and examples included

The cPCI-7841 is a Dual-Port Isolated CAN Interface Card with the following features:

- Two independent CAN network operation
- Bridge function supports
- Compatible with CAN specification 2.0 parts A and B
- Optically isolated CAN interface up to 2500 Vrms isolation protection
- Direct memory mapping to the CAN controllers
- Powerful master interface for CANopen, DeviceNet and SDS application layer protocol
- Up to 1Mbps programmable transfer rate
- Supports standard DeviceNet data rates 125, 250 and 500 Kbps
- PCI bus plug and play
- compact-PCI industry bus
- DOS library and examples included

The PM-7841 is a Dual-Port Isolated CAN Interface Card with the following features:

- Two independent CAN network operation
- Bridge function supports
- Compatible with CAN specification 2.0 parts A and B
- Optically isolated CAN interface up to 2500 Vrms isolation protection
- Direct memory mapping to the CAN controllers
- Powerful master interface for CANopen, DeviceNet and SDS application layer protocol
- Up to 1Mbps programmable transfer rate

- Supports standard DeviceNet data rates 125, 250 and 500 Kbps
- DIP-Switch for base address configuration
- Software Programmable Memory-Mapped Address
- PC-104 industry form factor
- DOS library and examples included

1.2 Applications

- Industry automation
- Industry process monitoring and control
- Manufacture automation
- Product testing

1.3 Specifications

PCI-7841 Specification Table

Ports	2 CAN channels (V2.0 A,B)
CAN Controller	SJA1000
CAN Transceiver	82c250
Signal Support	CAN_H, CAN_L
Isolation Voltage	2500 Vrms
Connectors	Dual DB-9 male connectors
Operation Temperature	0 ~ 60° C
Storage Temperature	-20° ~ 80° C
Humidity	5% ~ 95% non-condensing
IRQ Level	Set by Plug and Play BIOS
I/O port address	Set by Plug and Play BIOS
Power Consumption (without external devices)	400mA @5VDC (Typical) 900mA @5VDC (Maximum)
Size	132(L)mm x 98(H)mm

cPCI-7841 Specification Table

Ports	2 CAN channels (V2.0 A,B)
CAN Controller	SJA1000
CAN Transceiver	82c250
Signal Support	CAN_H, CAN_L
Isolation Voltage	2500 Vrms
Connectors	Dual ?? male connectors
Operation Temperature	0 ~ 60° C
Storage Temperature	-20° ~ 80° C
Humidity	5% ~ 95% non-condensing
IRQ Level	Set by Plug and Play BIOS
I/O port address	Set by Plug and Play BIOS
Power Consumption (without external devices)	400mA @5VDC (Typical) 900mA @5VDC (Maximum)
Size	132(L)mm x 98(H)mm

PM-7841 Specification Table

Ports	2 CAN channels (V2.0 A,B)
CAN Controller	SJA1000
CAN Transceiver	82c250/82c251
Signal Support	CAN_H, CAN_L
Isolation Voltage	1000 Vrms
Connectors	Dual 5 male connectors
Operation Temperature	0 ~ 60° C
Storage Temperature	-20° ~ 80° C
Humidity	5% ~ 95% non-condensing
IRQ Level	Set by Jumper
I/O port address	Set by DIP Switch
Memory Mapped Space	128 Bytes by Software
Power Consumption (without external devices)	400mA @5VDC (Typical) 900mA @5VDC (Maximum)
Size	90.17(L)mm x 95.89(H)mm

2

Installation

This chapter describes how to install the PCI/cPCI/PM-7841. At first, the contents in the package and unpacking information that you should be careful are described.

2.1 Before Installation PCI/cPCI/PM-7841

Your PCI/cPCI/PM-7841 card contains sensitive electronic components that can be easily damaged by static electricity.

The card should be done on a grounded anti-static mat. The operator should be wearing an anti-static wristband, grounded at the same point as the anti-static mat.

Inspect the card module carton for obvious damage. Shipping and handling may cause damage to your module. Be sure there are no shipping and handing damages on the module before processing.

After opening the card module carton, exact the system module and place it only on a grounded anti-static surface component side up.

Note: DO NOT APPLY POWER TO THE CARD IF IT HAS BEEN DAMAGED.

You are now ready to install your PCI/cPCI/PM-7841.

2.2 Installing PCI-7841

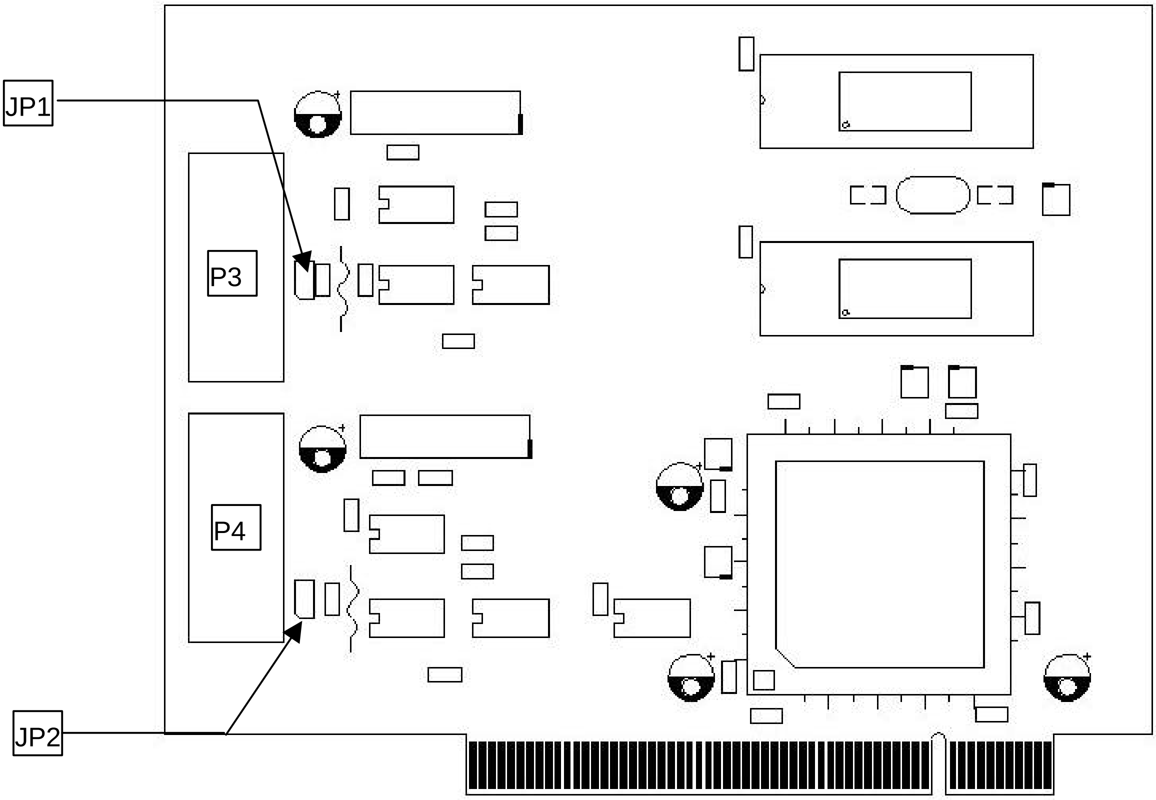
What do you have

In addition to this *User's Manual*, the package includes the following items:

- PCI-7841 Dual Port PCI Isolated CAN Interface Card
- ADLink All-xxxxx CD-ROM

If any of these items is missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton in case you want to ship or store the product in the future.

PCI-7841 Layout

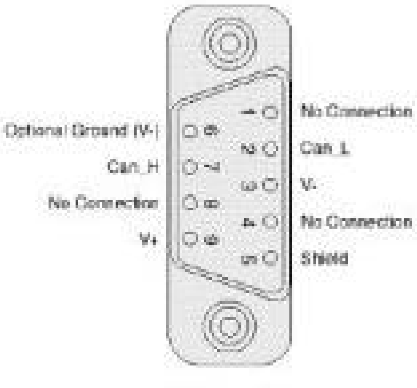


Terminator Configuration

A 120 Ω terminal resistor is installed for each port, while JP1 enables the terminal resistor for port0 and JP2 enables the terminal resistor for port 1

Connector Pin Define

The P3 and P4 are CAN connector, the below picture is their pin define



DB-9 Connector

2.3 Installing cPCI-7841

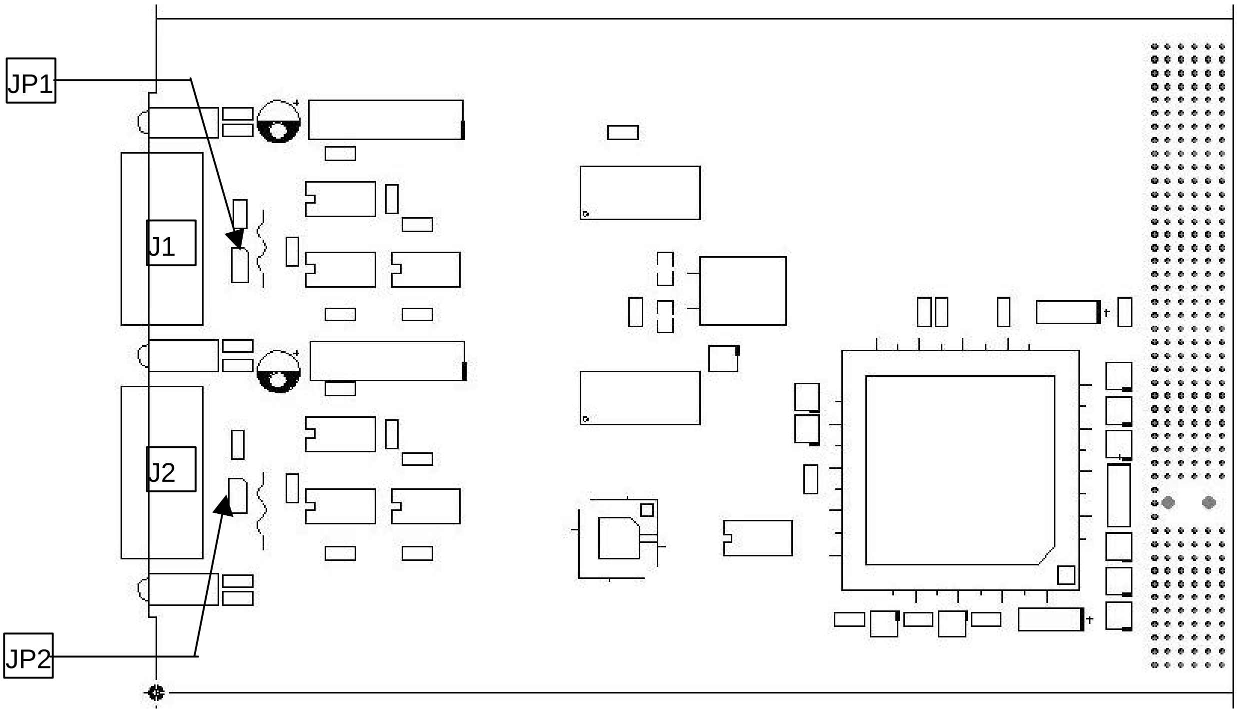
What do you have

In addition to this *User's Manual*, the package includes the following items:

- cPCI-7841 Dual Port Compact-PCI Isolated CAN Interface Card
- ADLink All-xxxxx CD-ROM

If any of these items is missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton in case you want to ship or store the product in the future.

cPCI-7841 Layout

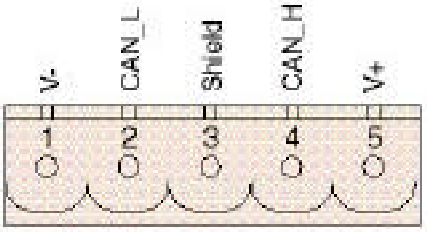


Terminator Configuration

A 120 Ω terminal resistor is installed for each port, while JP1 enables the terminal resistor for port0 and JP2 enables the terminal resistor for port 1

Connector Pin Define

The J1 and J2 are CAN Connector, the below picture is their pin define



Combicon-Style Connector

2.4 Installing PM-7841

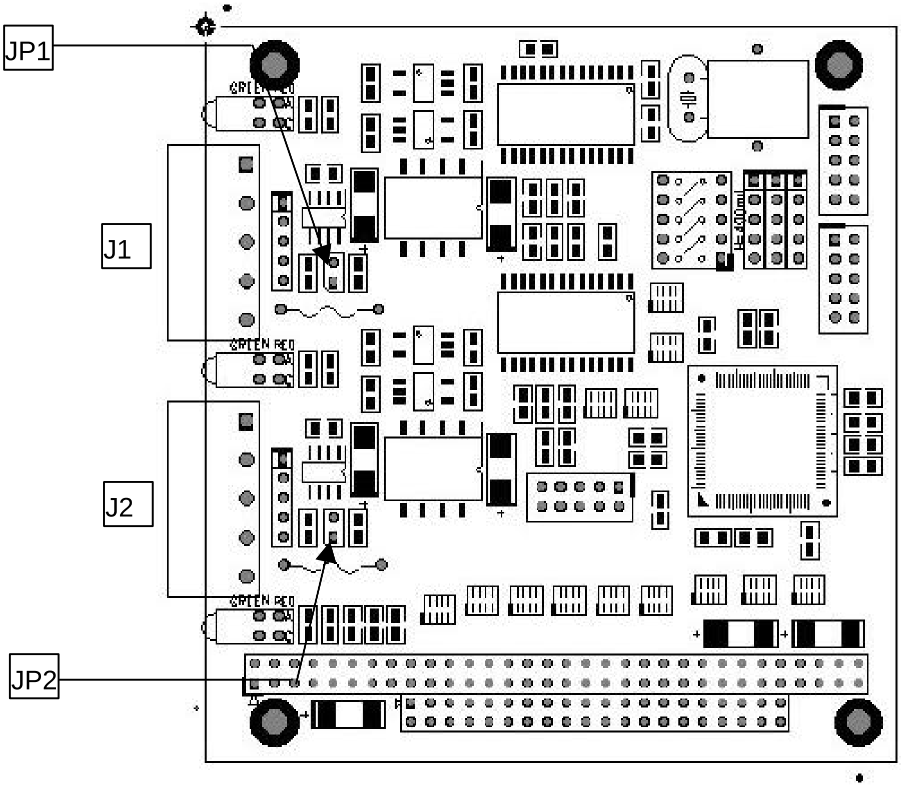
What do you have

In addition to this *User's Manual*, the package includes the following items:

- PM-7841 Dual Port PC-104 Isolated CAN Interface Card
- ADLink All-xxxxx CD-ROM

If any of these items is missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton in case you want to ship or store the product in the future.

PM-7841 Layout

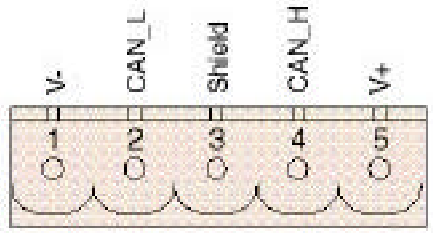


Terminator Configuration

A 120 Ω terminal resistor is installed for each port, while JP1 enables the terminal resistor for port0 and JP2 enables the terminal resistor for port 1

Connector Pin Define

The J1 and J2 are CAN Connector, the below picture is their pin define



Combicon-Style Connector

2.4 Jumper and DIP Switch Description

You can configure the output of each channel and base address by setting jumpers and DIP switches on the PM-7841. The card's jumpers and switches are preset at the factory. Under normal circumstances, you should not need to change the jumper settings.

A jumper switch is closed (sometimes referred to as "shorted") with the plastic cap inserted over two pins of the jumper. A jumper is open with the plastic cap inserted over one or no pin(s) of the jumper.

2.5 Base Address Setting

The PM-7841 requires 16 consecutive address locations in I/O address space. The base address of the PM-7841 is restricted by the following conditions.

1. The base address must be within the range 200hex to 3F0hex.

2. The base address should not conflict with any PC reserved I/O address. .

The PM-7841's I/O port base address is selectable by an 5 position DIP switch SW1 (refer to Table 2.1). The address settings for I/O port from Hex 200 to Hex 3F0 is described in Table 2.2 below. The default base address of your PM-7841 is set to **hex 200** in the factory(see Figure below).

SW1 : Base Address = 0x200

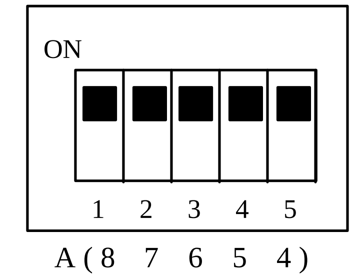


Figure Default Base Address Configuration

I/O port address(hex)	fixed A9	1 A8	2 A7	3 A6	4 A5	5 A4
200-20F	OFF (1)	ON (0)	ON (0)	ON (0)	ON (0)	ON (0)
210-21F	OFF (1)	ON (0)	ON (0)	ON (0)	ON (0)	OFF (1)
:						
(*) 2C0-2CF	OFF (1)	ON (0)	OFF (1)	OFF (1)	ON (0)	ON (0)
:						
300-30F	OFF (1)	OFF (1)	ON (0)	ON (0)	ON (0)	ON (0)
:						
3F0-3FF	OFF (1)	OFF (1)	OFF (1)	OFF (1)	OFF (1)	OFF (1)

(*) : default setting ON : 0

X : don't care OFF : 1

Note: A4, ..., A9 correspond to PC-104(ISA) bus address lines.

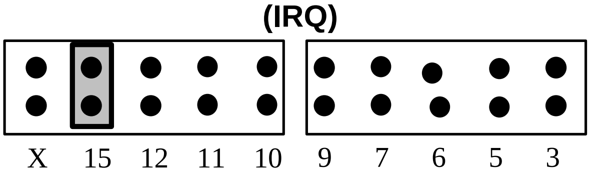
2.6 IRQ Level Setting

A hardware interrupt can be triggered by the external Interrupt signal which is from JP3 ad JP4.

The jumper setting is specified as below:

Note : Be aware that there is no other add-on cards sharing the same interrupt level in the system.

Interrupt Default Setting = IRQ15



IRQ Setting

3

Function Reference

The cPCI/PCI-7841 functions are organize into the following sections:

- CAN layer functions
  - Card Initialization and configuration functions
  - CAN layer I/O functions
  - CAN layer status functions
  - CAN layer Error and Event Handling functions
- DeviceNet layer functions
  - Send and Receive packet functions
  - Connection establish and release functions
  - DeviceNet object class functions

The particular functions associated with each function are presented in next page.

3.1 Functions Table

CAN layer functions		
Function Type	**Function Name**	**Page**
PM-7841 Initial	*PM7841_Install()*	22
	GetDriverVersion()	22
	CanOpenDriver()	24
	CanCloseDriver()	25
	CanConfigPort()	26
	CanDetectBaudrate()	27
	_7841_Read()	28
	_7841_Write()	28
	CanEnableReceive()	29
	CanDisableReceive()	29
	CanSendMsg()	30
	CanRcvMsg()	31
	CanGetRcvCnt()	*41*
	CanClearOverrun()	32
	CanClearRxBuffer()	33
	CanClearTxBuffer()	34
	CanGetErrorCode()	35
	CanGetErrorWarningLimit()	35
	CanSetErrorWarningLimit()	36
	CanGetRxErrorCount()	37
	CanGetTxErrorCount()	37
	CanSetTxErrorCount()	*38*
	CanGetPortStatus()	39
	CanGetLedStatus()[1]	39
	CanSetLedStatus()[1]	40

Error and Event handling functions		
Operation System	**Function Name**	**Page**
DOS	*CanInstallCallBack()*	42
	CanRemoveCallBack()	43
Windows 95/98/NT	*CanInstallEvent()*	45

DeviceNet layer functions		
Function Type	**Function Name**	**Page**
Send and Receive packet functions	*SendDeviceNetPacket()*	48
	RcvDeviceNetPacket()	49
	SendGroup2Message()	50
	RcvGroup2Message()	52
	DNetSendIO()	54
	DNetRcvIO()	55
Connection establish and release functions	*DNetOpenExp()*	56
	DNetCloseExp()	56
	DNetOpenIO()	57
	DNetCloseIO()	57
	DNetScan()	58
DeviceNet Object class functions	*DNetIdentity()*	60
	DNetDNet()	60
	DNetSetID()	61
	DNetSetBaud()	62

Note 1: only for compact PCI and PC-104 version

3.1.1 PORT_STRUCT structure define

The **PORT_STRUCT** structure defines the mode of id-mode, acceptance code, acceptance mask and baud rate of a physical CAN port. It is used by the ***CanPortConfig()***, and ***CanGetPortStatus()*** functions.

```
typedef struct _tagPORT_STRUCT
{
    int mode;                   //   0    for 11-bit;      1 for 29-bit
    DWORD accCode, accMask;
    int baudrate;
    BYTE brp, tseg1, tseg2;     //  Used only if baudrate = 4
    BYTE sjw, sam;              //   Used only if baudrate = 4
}PORT_STRUCT;
```

Members

mode : 0 means using 11-bit in CAN-ID field

1 means using 29-bit in CAN-ID field.

accCode : Acceptance Code for CAN controller.

accMask : Acceptance Mask for CAN controller.

baudrate : Baud rate setting for the CAN controller.

Value	Baudrate
0	125 Kbps
1	250 Kbps
2	500 Kbps
3	1M Kbps
4	User-Defined

brp, tseg1, tseg2, sjw, sam : Use for User-Defined Baudrate

See Also

CanPortConfig(), CanGetPortStatus(), and ***PORT_STATUS*** structure

3.1.2 PORT_STATUS structure define

The **PORT_STATUS** structure defines the status register and **PORT_STRUCT** of CAN port. It is used by the ***CanGetPortStatus()*** functions.

```
typedef struct _tagPORT_STATUS
{
    PORT_STRUCT port;
    PORT_REG status;
}PORT_STATUS;
```

Members

port : PORT_STRUCT data

status : status is the status register mapping of CAN controller.

```
typedef union _tagPORT_REG
{
    struct PORTREG_BIT bit;
    unsigned short reg;
}PORT_REG;
struct PORTREG_BIT
{
    unsigned short RxBuffer       : 1;
    unsigned short DataOverrun    : 1;
    unsigned short TxBuffer       : 1;
    unsigned short TxEnd          : 1;
    unsigned short RxStatus       : 1;
    unsigned short TxStatus       : 1;
    unsigned short ErrorStatus    : 1;
    unsigned short BusStatus      : 1;
    unsigned short reserved       : 8;
};
```

See Also

CanGetPortStatus(), and ***PORT_STATUS*** structure

3.1.3 CAN_PACKET structure define

The **CAN_PACKET** structure defines the packet format of CAN packet. It is used by the ***CanSendMsg()***, and ***CanRcvMsg()*** functions.

```
typedef struct _tagCAN_PACKET
{
    DWORD CAN_ID;
    BYTE rtr;
    BYTE len;
    BYTE data[8]
    DWORD time;
    BYTE reserved
}CAN_PACKET;
```

Members

CAN_ID	:	CAN ID field (32-bit unsigned integer)
rtr	:	CAN RTR bit.
len	:	Length of data field.
data	:	Data (8 bytes maximum)
time	:	Reserved for future use
reserved	:	Reserved byte

See Also

CanSendMsg(), and ***CanRcvMsg()***

3.1.4 DEVICENET_PACKET structure define

The **DEVICENET_PACKET** structure defines the packet format of DeviceNet packet. It is widely used by the DeviceNet layer functions.

```
typedef struct _tagDEVICENET_PACKET
{
    BYTE Group;
    BYTE MAC_ID;
    BYTE HostMAC_ID;
    BYTE MESSAGE_ID;
    BYTE len;
    BYTE data[8];
    DWORD time;
    BYTE reserved;
```

}DEVICENET_PACKET;

Members

Group	:	Group of DeviceNet packet
MAC_ID	:	Address of destination.
HostMAC_ID	:	Address of source.
MESSAGE_ID	:	Message ID of DeviceNet packet
len	:	Length of data field.
data	:	Data (8 bytes maximum)

See Also

SendDeviceNetPacket(), and ***RcvDeviceNetPacket()***

3.2 CAN LAYER Functions

CAN-layer Card Initialization Functions

PM7841_Install(base, irq_chn, 0xd000)

Purpose Get the version of driver

Prototype **C/C++**

int PM7841_Install(int baseAddr, int irq_chn, int memorySpace)

Visual Basic(Windows 95/98)

Parameters

baseAddr	:	Base Address of PM-7841(DIP Switch)
Irq_chn	:	IRQ channel (Jumpper)
MemorySpace:		Memory Mapping Range

Return Value A 16-bit unsigned integer

High byte is the major version

Low byte is the major version

Remarks Call this function to retrieve the version of current using driver.

This function is for your program to get the version of library and dynamic-linked library.

See Also none

Usage **C/C++**

```
#include “pm7841.h”

WORD version = GetDriverVersion();
majorVersion = version >> 8;
minorVersion = version & 0x00FF;
```

Visual Basic(Windows 95/98)

GetDriverVersion()

Purpose Get the version of driver

Prototype **C/C++**

WORD GetDriverVersion(void)

Visual Basic(Windows 95/98)

Parameters none

Return Value A 16-bit unsigned integer

High byte is the major version

Low byte is the major version

Remarks Call this function to retrieve the version of current using driver. This function is for your program to get the version of library and dynamic-linked library.

See Also none

Usage **C/C++**

```
#include “pci7841.h”

WORD version = GetDriverVersion();
majorVersion = version >> 8;
minorVersion = version & 0x00FF;
```

Visual Basic(Windows 95/98)

CanOpenDriver()

Purpose Open a specific port, and initialize driver.

Prototype **C/C++**

int CanOpenDriver(int card, int port)

Visual Basic(Windows 95/98)

Parameters	card	:	index of card
	port	:	index of port

Return Value Return a handle for open port

-1 if error occurs

Remarks Call this function to open a port.

Under DOS operation system, you will receive –1 if there is not enough memory. If writing program for the Windows system. It will return -1, if you want to open a port had been opened. And you must use *CanCloseDriver()* to close the port after using.

See Also *CanCloseDriver()*

Usage **C/C++**

```
#include "pci7841.h"
int handle = CanOpenDriver();
CanSendMsg(handle, &msg);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanCloseDriver()

Purpose Close an opened port, and release driver.

Prototype **C/C++**

int CanCloseDriver(int handle)

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
port	:	index of port

Return Value Return 0 if successful

-1 if error occurs

Remarks Call this function to close a port.

See Also ***CanOpenDriver****()*

Usage See usage of *CanOpenDriver()*.

CanConfigPort()

Purpose Configure properties of a port.

Prototype **C/C++**

*int CanConfigPort(int handle, PORT_STRUCT *ptrStruct)*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
ptrStruct	:	a pointer of *PORT_STRUCT* type

Return Value Return 0 is successful

-1 if error occurs

Remarks Configure a port that had been opened.

The properties of a CAN port such as baud rate, acceptance code, acceptance mask, operate mode. After configuration is over, the port is ready to send and receive data.

See Also ***Can****ConfigPort()*

Usage **C/C++**

```
#include “pci7841.h
PORT_STRUCT port_struct;
int handle = CanOpenDriver(0, 0); //      Open port 0 of card 0

port_struct.mode = 0;                //      CAN2.0A (11-bit CAN id)
port_struct.accCode = 0;             //      This setting of acceptance code and
port_struct.accMask = 0x7FF;         //      mask enable all MAC_IDs input
port_struct.baudrate = 0;            //      125K bps
CanConfigPort(handle, &port_struct);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanDetectBaudrate()

Purpose Perform auto-detect baud rate algorithm.

Prototype **C/C++**

int CanDetectBaudrate(int handle, int miliSecs)

Visual Basic(Windows 95/98)

Parameters

handle : handle retrieve from *CanOpenDriver()*

miliSecs : timeout time(ms)

Return Value Return –1 if error occurs

Others is the baudrate

Value	Baudrate
0	125 Kbps
1	250 Kbps
2	500 Kbps
3	1M Kbps

Remarks Call this function to detect the baud rate of a port.

The function performs an algorithm to detect your baud rate. It needs that there are activities on the network. And it will return a –1 when detecting no activity on the network or time was exceeded.

See Also none

Usage **C/C++**

```
#include "pci7841.h
PORT_STRUCT port_struct;"
int handle = CanOpenDriver();
…
port_struct.mode = 0;            //   CAN2.0A (11-bit CAN id)
port_struct.accCode = 0;         //   This setting of acceptance code and
port_struct.accMask = 0x7FF;     //   mask enable all MAC_IDs input
port_struct.baudrate = CanDetectBaudrate(handle, 1000):
CanConfigPort(handle, &port_struct);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanRead()

Purpose Direct read the register of PCI-7841.

Prototype **C/C++**

BYTE CanRead(int handle, int offset)

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
offset	:	offset of register

Return Value Return data read from port**.**

Remarks Direct read the register of PCI-7841.

See Also ***Can****Write()*

Usage none

CanWrite()

Purpose Direct write the register of PCI-7841.

Prototype **C/C++**

void CanWrite(int handle, int offset, BYTE data)

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
offset	:	offset of register
data	:	data write to the port

Return Value none

Remarks Call this function to directly write a register of PCI-7841.

See Also ***Can****Read()*

Usage none

CAN-layer I/O Functions

CanEnableReceive()

Purpose Enable receive of a CAN port.

Prototype **C/C++**

void CanEnableReceive(int handle);

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Call this function to enable receive.

Any packet on the network that can induce a interrupt on your computer. If that packet can pass your acceptance code and acceptance mask setting. So if your program doesn' t want to be disturbed. You can call *CanDisableReceive()* to disable receive and *CanEnableReceive() to enable receives.*

See Also *CanDisableReceive()*

Usage none

CanDisableReceive()

Purpose Enable receive of a CAN port.

Prototype **C/C++**

void CanEnableReceive(int handle);

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Please refer the *CanEnableReceive()*.

See Also *CanEnableReceive()*

Usage none

CanSendMsg()

Purpose Send can packet to a port

Prototype **C/C++**

*int CanSendMsg(int handle, CAN_PACKET *packet);*

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

packet : *CAN_PACKET* data

Return Value Return 0 is successful

-1 if error occurs

Remarks Send a message to an opened CAN port.

Actually, this function copies the data to the sending queue. Error occurs when the port has not been opened yet or the packet is a NULL pointer. You can use the Error and Event handling functions to handle the exceptions.

See Also CanRcvMsg()

Usage **C/C++**

```
#include "pci7841.h
PORT_STRUCT port_struct;
CAN_PACKET sndPacket, rcvPacket;
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
CanConfigPort(handle, &port_struct);
CanSendMsg(handle, &sndPacket);
if(CanRcvMsg(handle, &rcvPacket) == 0)
{
…
}
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanRcvMsg()

Purpose Receive a can packet from a port

Prototype **C/C++**

*int CanSendMsg(int handle, CAN_PACKET *packet);*

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

packet : *CAN_PACKET* data

Return Value Return 0 is successful

-1 if error occurs

Remarks Receive a message from an opened CAN port.

There are only 64-bytes FIFO under hardware. It can store from 3 to 21 packets. So there are memory buffer under driver. When data comes, the driver would move it from card to memory. It starts after your port configuration is done. This function copies the buffer to your application. So if your program has the critical section to process the data on the network. We suggest that you can call the *CanClearBuffer()* to clear the buffer first. Error would be happened most under the following conditions:

1. You want to access a port that has not be opened
2. Your packet is a NULL pointer.
3. The receive buffer is empty.

You can use the Status handling functions to handle the exceptions.

See Also *CanSendMsg()*

Usage See the *CanSendMsg()*

CAN-layer Status Functions

CanClearOverrun()

Purpose Clear data overrun status

Prototype **C/C++**

void CanClearOverrun(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Clear the data overrun status

Sometimes if your system has heavy load, and the bus is busy. The data overrun would be signalled. A Data Overrun signals, that data are lost, possibly causing inconsistencies in the system.

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
CanClearOverrun(handle);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanClearRxBuffer()

Purpose Clear data in the receive buffer

Prototype **C/C++**

void CanClearRxBuffer(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Clear the data in the receive buffer

There are 2-type of buffer defined in the driver. First one is the FIFO in the card, the second one is the memory space inside the driver. Both of them would be cleared after using this function.

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //      open the port 0 of card 0
…
CanClearRxBuffer(handle);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanClearTxBuffer()

Purpose Clear Transmit Buffer

Prototype **C/C++**

void CanClearTxBuffer(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Clear the data in the transmit buffer.

Under a busy DeviceNet Network, your transmit request may not be done due to the busy in the network. The hardware will send it automatically when bus is free. The un-send message would be stored in the memory of the driver. The sequence of outgoing message is the FIRST-IN-FIRST-OUT. According this algorithm, if your program need to send an emergency data, you can clear the transmit buffer and send it again.

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
CanClearTxBuffer(handle);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanGetErrorCode()

Purpose Get the Error Code

Prototype **C/C++**

BYTE CanGetErrorCode(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value error code

Return error code is an 8-bit data

Bit	Symbol	Name	Value	Function
7	ERRC1	Error Code 1		
6	ERRC0	Error Code 0		
5	DIR	Direction	1	Rx error occurred during reception
			0	Tx error occurred during transmission
4	SEG4	Segment 4		
3	SEG3	Segment 3		
2	SEG2	Segment 2		
1	SEG1	Segment 1		
0	SEG0	Segment 0		

Bit interpretation of ERRC1 and ERRC2

Bit ERRC1	Bit ERRC2	Function
0	0	bit error
0	1	form error
1	0	stuff error
1	1	other type of error

Bit interpretation of SEG4 to SEG 0

SEG4	SEG3	SEG2	SEG1	SEG0	Function
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 to ID.21
0	0	1	1	0	ID.20 to ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 to ID.13
0	1	1	1	1	ID.12 to ID.5
0	1	1	1	0	ID.4 to ID.0
0	1	1	0	0	RTR bit
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	Data length code
0	1	0	1	0	Data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	acknowledge slot

1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Remarks Get the information about the type and location of errors on the bus. When bus error occurs, if your program installed the call-back function or error-handling event. The error-bit position would be captured into the card. The value would be fixed in the card until your program read it back.

See Also *CanGetErrorWarningLimit(),CanSetErrorWarningLimit()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
BYTE data = CanGetErrorCode();
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanSetErrorWarningLimit()

Purpose Set the Error Warning Limit

Prototype **C/C++**

void CanSetErrorWarningLimit(int handle, BYTE value)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

value : Error Warning Limit

Return Value none

Remarks Set the error warning limit,

If your program has installed the error warning event or call-back function. The error warning will be signaled after the value of error counter passing the limit you set.

See Also *CanGetErrorWarningLimit()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
```

CanSetErrorWarning(handle, 96);

CanCloseDriver(handle);

Visual Basic(Windows 95/98)

CanGetErrorWarningLimit()

Purpose Get the Error Warning Limit

Prototype **C/C++**

BYTE CanGetErrorWarningLimit(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Get the error warning limit

See Also *CanSetErrorWarningLimit()*

Usage **C/C++**

#include "pci7841.h

int handle = CanOpenDriver(0, 0); // open the port 0 of card 0

…

BYTE limit = CanClearOverrun(handle);

CanCloseDriver(handle);

Visual Basic(Windows 95/98)

CanGetRxErrorCount()

Purpose Get the current value of the receive error counter

Prototype **C/C++**

BYTE CanGetRxErrorCount(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value value

Remarks This function reflects the current of the receive error counter. After hardware reset happened, the value returned would be initialized to 0. If a bus-off event occurs, the returned value would be 0.

See Also *CanRcvMsg()*

Usage **C/C++**

#include "pci7841.h

int handle = CanOpenDriver(0, 0); // open the port 0 of card 0

…

BYTE error_count = CanGetTxErrorCount();

CanCloseDriver(handle);

Visual Basic(Windows 95/98)

CanGetTxErrorCount()

Purpose Get the current value of the transmit error counter

Prototype **C/C++**

BYTE CanGetTxErrorCount(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value value

Remarks This function reflects the current of the transmit error counter. After hardware reset happened, the value would set to 127. A bus-off event occurs when the value reaches 255. You can call the *CanSetTxErrorCount()* to set the value from 0 to 254 to clear the bus-off event.

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
BYTE error_count = CanGetRxErrorCount(handle);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanSetTxErrorCount()

Purpose Set the current value of the transmit error counter

Prototype **C/C++**

void CanSetTxErrorCount(int handle, BYTE value)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

value : a byte value

Return Value value

Remarks This function set the current of the transmit error counter.

Please see the remark of *CanGetTxErrorCount()*.

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include "pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
CanSetRxErrorCount(handle, 0);
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanGetPortStatus()

Purpose Clear data overrun status

Prototype **C/C++**

void CanClearOverrun(int handle)

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

Return Value none

Remarks Clear the data overrun status and clean the buffer

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include "pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
CanClearOverrun();
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanGetLedStatus()

Purpose Get the LED status of cPCI-7841 and PM-7841

Prototype **C/C++**

BYTE CanGetLedStatus (int card, int index);

Visual Basic(Windows 95/98)

Parameters card : card number

index : index of LED

Return Value status of Led

Value	Function
0	Led Off
1	Led On

Remarks Get the status of Led

This function supports the cPCI-7841 and PM-7841.

See Also *CanSetLEDStatus()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
BYTE flag = CanGetLedStatus(0, 0);;
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanSetLedStatus()

Purpose Set the Led Status of cPCI-7841

Prototype **C/C++**

```
void   CanSetLedStatus(int card, int index, int flashMode);
```

Visual Basic(Windows 95/98)

Parameters

card	:	card number
index	:	index of Led
flashMode :		

Value	Function
0	Led Off
1	Led On

Return Value none

Remarks Set Led status of cPCI-7841 and PM-7841

This function supports the cPCI-7841 and PM-7841

See Also *CanRcvMsg()*

Usage **C/C++**

```
#include “pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
…
CanSetLedStatus(0, 0, 2);             //     Set Led to flash
CanCloseDriver(handle);
```

Visual Basic(Windows 95/98)

CanGetRcvCnt()

Purpose Get the how many message in the FIFO

Prototype **C/C++**

int _stdcall CanGetRcvCnt(int handle)

Visual Basic(Windows 95/98)

Parameters card : card number

Return Value How many messages …

Remarks Get the unread message count in the FIFO

See Also *CanGetReceiveEvent()*

Usage **C/C++**

```
#include "pci7841.h
int handle = CanOpenDriver(0, 0); //     open the port 0 of card 0
….
int count = CanGetRcvCnt(handle);.
```

Visual Basic(Windows 95/98)

Error and Event Handling Functions

When the exception occurs, your program may need to take some algorithm to recover the problem. The following functions are operation-system depended functions. You should care about the restriction in the operation-system.

DOS Environment

CanInstallCallBack()

Purpose Install callback function of event under DOS environment

Prototype **C/C++**

*void far*CanInstallCallBack(int handle, int index, void (far* proc)());*

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

index : event type

Index	Type
2	Error Warning
3	Data Overrun
4	Wake Up
5	Error Passive
6	Arbitration Lost
7	Bus Error

void (far *proc)(): Call-back function

The suggest prototype of the call-back function is like void (far ErrorWarning)();

Return Value Previous call back function (NULL when there is no Call back installed)

Remarks Install the call-back function for event handling

In normal state, all hardware interrupt of cPCI/PCI-7841 wouldn' t be set except receive and transmit interrupt. After calling the *CanInstallCallBack()*, the corresponding interrupt would be activated. The interrupt occurs when the event happened. It will not be disabled until using *CanRemoveCallBack()* or a hardware reset.

Actually, the call-back function is a part of ISR. You need to care about the DOS reentrance problem, and returns as soon as possible to preventing the lost of data.

See Also *CanRemoveCallBack()*

Usage **C/C++**

#include "pci7841.h

```
void (far ErrorWarning)();
int handle = CanOpenDriver(0, 0); //      open the port 0 of card 0
…
//      Installs the ErrorWarning handling event and stores the previous one.
void (far *backup) = CanInstallCallBack(0, 2, ErrorWarning);
CanRemoveCallBack(0, 2, NULL);  //      Remove the call-back function
CanCloseDriver(handle);
```

CanRemoveCallBack()

Purpose Remove the callback function of event under DOS environment

Prototype **C/C++**

int CanRemoveCallBack(int handle, int index, void (far proc)());*

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

index : event type

Index	Type
2	Error Warning
3	Data Overrun
4	Wake Up
5	Error Passive
6	Arbitration Lost
7	Bus Error

void (far *proc)(): Previous call-back function

Return Value Return 0 is successful

-1 if error occurs

Remarks Install the call-back function for event handling

In normal state, all hardware interrupt of cPCI/PCI-7841 wouldn’t be set except receive and transmit interrupt. After calling the *CanInstallCallBack()*, the corresponding interrupt would be activated. The interrupt occurs when the event happened. It will not be disabled until using *CanRemoveCallBack()* or a hardware reset.

Actually, the call-back function is a part of ISR. You need to care about the DOS reentrance problem, and returns as soon as possible to preventing the lost of data.

See Also *CanRemoveCallBack()*

Usage **C/C++ (DOS)**

```
#include “pci7841.h
void (far ErrorWarning)();
int handle = CanOpenDriver(0, 0); //      open the port 0 of card 0
```

```
…
//	Installs the ErrorWarning handling event and stores the previous one.
void (far *backup) = CanInstallCallBack(0, 2, ErrorWarning);
CanRemoveCallBack(0, 2, NULL);  //	Remove the call-back function
CanCloseDriver(handle);
```

Windows 95/98 Environment

CanGetReceiveEvent()

Purpose Install the event under Windows 95/98/NT system

Prototype **C/C++ (Windows 95/98/NT)**

*void CanGetReceiveEvent(int handle, HANDLE *hevent);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
hevent	:	HANDLE point for receive event

Return Value none

Remarks Retrieve receive notify event

Under Windows 95/98/NT environment, your program can wait the input message by waiting an event. You can refer to following program to use this function. But the CAN system is a heavy-load system. Under the full speed(of course, it depends on your system), the hardware receives the message faster than the event occurs. Under this condition, the event could be combined by OS. So the total count of event may be less than actually receive. You can call the *CanGetRcvCnt()* to retrieve the unread message in the driver' s FIFO.

See Also *CanGetRcvCnt()*

Usage **C/C++ (Windows 95/98)**

```
#include "pci7841.h
HANDLE recvEvent0;
int handle = CanOpenDriver(0, 0); //	open the port 0 of card 0
int count1;
…
if(WaitForSingleObject(rcvEvent0, INFINITE) == WAIT_OBJECT_0)
{
	//	You need not to call ResetEvent() .….
	err = CanRcvMsg(handle, &rcvMsg[0][rcvPatterns[0]]);
	rcvPatterns[0]++;
}
cout1 = CanGetRcvCnt(handle[0]);        //    To retrieve number of unread
                                      //    in the FIFO
```

CanInstallEvent()

Purpose Install the event under Windows 95/98/NT system

Prototype **C/C++ (Windows 95/98/NT)**

int CanInstallEvent(int handle, int index, HANDLE hEvent);

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

index : event type

Index	**Type**
2	Error Warning
3	Data Overrun
4	Wake Up
5	Error Passive
6	Arbitration Lost
7	Bus Error

hEvent : HANDLE created from *CreateEvent()(Win32 SDK)*

Return Value Return 0 is successful

-1 if error occurs

Remarks Install the notify event

Unlike the Dos environment, there is only one error handling function under Windows 95/98/NT environment. First you need to create an event object, and send it to the DLL. The DLL would make a registry in the kernel and pass it to the VxD(SYS in NT system). You can' t release the event object you created, because it was attached to the VxD. The VxD would release the event object when you installed another event. One way to disable the event handling is that you install another event which handle is NULL (*ex: CanInstallEvent(handle, index, NULL)*). And you can create a thread to handle the error event.

See Also *CanRemoveCallBack(),CanInstallCallBack()*

Usage **C/C++ (Windows 95/98)**

```
#include "pci7841.h
int handle = CanOpenDriver(0, 0); //    open the port 0 of card 0
…
//    Installs the ErrorWarning handling event and stores the previous one.
HANDLE hEvent = CreateEvent(NULL, FALSE, TRUE, "ErrorWarning");
CanInstallEvent(0, 2, hEvent);
```

..create a thread .…

Thread function

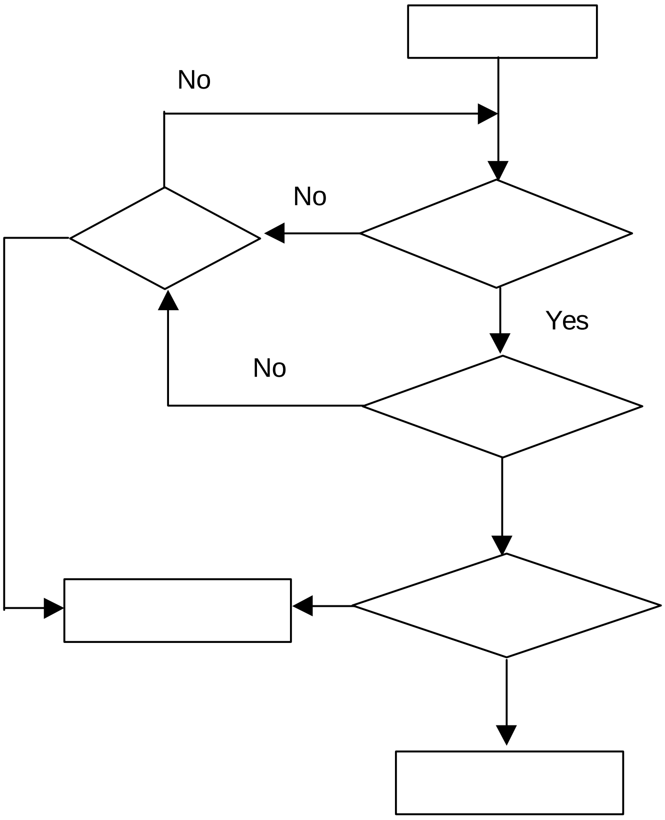
WaitForSingleObject(hEvent, INFINITE);

ResetEvent(hEvent);

…   //   Event handling

3.3 DeviceNet Layer Functions

DeviceNet Layer functions are a set building under CAN-layer functions. DeviceNet is a protocol constructed under physical CAN bus. Most the following functions work with the follow chart.



The set of DeviceNet functions are divided into the following groups:

1. DeviceNet send and receive packet functions

   Provide general functions to send and receive and DeviceNet packet from a CAN port.

2. DeviceNet connection establish and release functions

   Functions under this group are used for connection establishing. It supports explicit and Poll-IO connections.

3. DeviceNet object class functions

   The object access functions are provided here.

DeviceNet Send and Receive packet functions

SendDeviceNetPacket()

Purpose Send a DeviceNet packet to a CAN port.

Prototype **C/C++(DOS and Windows 95/98)**

int SendDeviceNetPacket(int handle, DEVICENET_PACKET packet);*

Visual Basic(Windows 95/98)

Parameters handle : handle retrieve from *CanOpenDriver()*

packet : *DEVICENET_PACKET* format pointer

Return Value Return 0 is successful

-1 if error occurs

Remarks Send a DeviceNet Packet to a opened port

This is the basic function of DeviceNet. It sends packed data to the hardware and returns immediately. It won't wait any response. This function can send from group 1 to group 4 packet.

See Also *RcvDeviceNetPacket()*

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
DEVICENET_PACKET msg;
msg.MAC_ID = 63;
msg.MESSAGE_ID = 7;
msg.Group = 2;
msg.len = 8;
msg.data[0] = 0x00; …..
SendDeviceNetPacket(handle, &msg);
```

Visual Basic(Windows 95/98).

RcvDeviceNetPacket()

Purpose Receive a DeviceNet packet from a CAN port.

Prototype **C/C++(DOS and Windows 95/98)**

int RcvDeviceNetPacket(int handle, DEVICENET_PACKET packet);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
packet	:	*DEVICENET_PACKET* format pointer

Return Value Return 0 is successful

-1 if error occurs

Remarks Receive a DeviceNet Packet from a opened port

This is the basic function of DeviceNet. It receives a packet from a CAN port. This function will wait 50 ms if there is nothing to read. It can recognize group 1 to group 4 messages. For an unknown packet, the group field in the message would be 255.

See Also *RcvDeviceNetPacket()*

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
DEVICENET_PACKET msg;

RcvDeviceNetPacket(handle, &msg);
```

Visual Basic(Windows 95/98).

SendGroup2Message()

Purpose Send a DeviceNet group 2 packet to a CAN port.

Prototype **C/C++(DOS and Windows 95/98)**

*int SendGroup2Message(int handle, BYTE *argu);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
argu	:	a 7 bytes BYTE array
		argu[0] = Host MAC_ID
		argu[1] = Destination MAC_ID
		argu[2] = Message ID
		argu[3] = Service Code
		argu[4] = Class ID
		argu[5] = Instance ID
		argu[6] = Attribute

Return Value Return 0 is successful

-1 if error occurs

Remarks Send a Group 2 DeviceNet Packet to a opened port

Most operation under DeviceNet is passed by Group 2 message. You can use this function to demand most explicit request. But under some special case, the Group 2 message needs more argument. You should call the *SendDeviceNetPacket()* directly. By reference the document of the module, if you find that there are more argument after Attribute field. You need to call the *SendDeviceNetPacket()* rather than *SendGroup2Message()*.

See Also *RcvDeviceNetPacket()*

Usage **C/C++(DOS and Windows 95/98)**

```
#include “pci7841.h”
int handle = CanOpenDriver(0, 0);
BYTE argu[8];
argu[0] = 0;      //    My MAC_ID is 0
argu[1] = 63;     //    Destination MAC_ID is 63
argu[2] = 6;      //    Message ID is 6
argu[3] = 0x4c;   //    Service Code is 0x4C;   Close
argu[4] = 3;      //    Class ID = 3
argu[5] = 1;      //    Instance = 1
argu[6] = 1;      //    Release choice = 0x01;   Close Explicit connection
SendGroup2DeviceNetPacket(handle, argu);
```

Visual Basic(Windows 95/98).

RcvGroup2Message()

Purpose Receive a Group 2 DeviceNet packet from a CAN port.

Prototype **C/C++(DOS and Windows 95/98)**

*int RcvGroup2Message(int handle, BYTE *data);*

Visual Basic(Windows 95/98)

Parameters

handle : handle retrieve from *CanOpenDriver()*

data : a 11-byte BYTE array

After the function returns, the data field stores

argu[0]	:	receiver MAC_ID
argu[1]	:	Message ID
argu[2]	:	the returned data length
argu[3-10]	:	the returned data field

Return Value 0 if successful

1 if timeout (50 ms)

2 if received message was not Group 2 message

-1 if null array

Remarks Receive a DeviceNet Packet from a opened port

Unlike the SendGroup2Message(), this function could be used to receive all kinds of packets you want if it is the Group 2 messages.

And the argument data is a **11 bytes** array, it is different.

See Also *RcvDeviceNetPacket()*

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
BYTE data[11];
int ret;
ret = RcvGroup2Message(handle, data);
switch(ret)
{
case 0:     //    success
case 1:     //    timeout and received nothing
case 2:     //    received message is not Group 2 (data is useless, try to read
            //    again)
case –1:    //    error
default :   //    unknown (it should not be happened!!)
}
```

Visual Basic(Windows 95/98).

DNetSendIO()

Purpose Send an IO command to a CAN port.

Prototype **C/C++(DOS and Windows 95/98)**

*int SendIO(int handle, int len, BYTE *argu);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
len	:	*the length in the argument*
argu	*:*	*a 8-byte length storage*

Return Value Return 0 is successful

-1 if error occurs

Remarks Send a Poll-IO packet to a opened port

Poll-IO command in DeviceNet is different from module to module. This function will pack your request to DeviceNet Poll-IO master request and send it to network. Note, you should open an IO connection first.

See Also *RcvDeviceNetPacket()*

Usage **C/C++(DOS and Windows 95/98)**

```
#include “pci7841.h”
int handle = CanOpenDriver(0, 0);
BYTE data[8];
data[0] = 0x00;
SendIO(handle, 0, data[0]);
```

Visual Basic(Windows 95/98).

DNetRcvIO()

Purpose Receive a DeviceNet packet from a CAN port.

Prototype **C/C++(DOS and Windows 95/98)**

*int DNetRcvIO(int handle, BYTE DestMAC_ID, BYTE *argu);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
DestMAC_ID	:	Destination MAC_ID
argu	:	An 8-byte array

Return Value Return -1 if error occurs

others is the returned data length

Remarks Receive a Poll-IO from a opened port

The Slave I/O response message is Group 1, message id = 15.

DnetRcvIO() will drop other group id until timeout or receive right packet.

See Also *RcvDeviceNetPacket()*

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
int len;
BYTE data[8];

len = DNetRcvIO(handle, data);
```

Visual Basic(Windows 95/98).

DeviceNet-layer Connection establishing and release functions

DNetOpenExp()

Purpose Open an explicit connection

Prototype **C/C++(DOS and Windows 95/98)**

int DNetOpenExp(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID);

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID

Return Value Return 0 is successful

-1 if error occurs

Remarks Open an explicit connection with a device.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
int len;
BYTE data[8];

DNetOpenExp(handle, 0, DestMAC_ID);
```

Visual Basic(Windows 95/98).

DNetCloseExp()

Purpose Close an explicit connection

Prototype **C/C++(DOS and Windows 95/98)**

int DNetCloseExp(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID);

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID

Return Value Return 0 is successful

-1 if error occurs

Remarks Close an explicit connection with a device.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
int len;
BYTE data[8];

DNetCloseExp(handle, 0, DestMAC_ID);
```

Visual Basic(Windows 95/98).

DNetOpenIO()

Purpose Open an Poll-IO connection

Prototype **C/C++(DOS and Windows 95/98)**

int DNetOpenIO(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID);

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID

Return Value Return 0 is successful

-1 if error occurs

Remarks Open an Poll-IO connection with a device.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
int len;
BYTE data[8];

DNetOpenIO(handle, 0, DestMAC_ID);
```

Visual Basic(Windows 95/98).

DNetCloseIO()

Purpose Close an Poll-IO connection

Prototype **C/C++(DOS and Windows 95/98)**

int DNetCloseIO(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID);

Visual Basic(Windows 95/98)

Parameters	handle	:	handle retrieve from *CanOpenDriver()*
	HostMAC_ID	:	the MAC_ID of your port
	DestMAC_ID	:	Destination MAC_ID

Return Value Return 0 is successful
-1 if error occurs

Remarks Close a Poll-IO connection with a device.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
int len;
BYTE data[8];

DNetCloseIO(handle, 0, DestMAC_ID);
```

Visual Basic(Windows 95/98).

DNetScan()

Purpose Scan the DeviceNet network

Prototype **C/C++(DOS and Windows 95/98)**

```
int    DNetScan(int handle, BYTE HostMAC_ID, BYTE* list);
```

Visual Basic(Windows 95/98)

Parameters	handle	:	handle retrieve from *CanOpenDriver()*
	HostMAC_ID	:	the MAC_ID of your port
	list	:	a 64-byte array

Return Value Return -1 if error occurs
others is the number of found

Remarks Scan a DeviceNet network.
It will perform an algorithm to search the entire network except the MAC_ID of your port.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
#include "pci7841.h"
int handle = CanOpenDriver(0, 0);
int modules;
BYTE list[64];
```

```
modules = DNetScan(handle, 0, list);        //It will search the entire network
```

Visual Basic(Windows 95/98).

DeviceNet-layer DeviceNet Object Class Functions

DNetIdentity()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetDNet()

Purpose Get the Information of DeviceNet DeviceNet object.

Prototype **C/C++(DOS and Windows 95/98)**

```
int    DNetDNet(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int
Instance, int Attribute, BYTE* array);
```

Visual Basic(Windows 95/98)

Parameters

handle	handle retrieve from *CanOpenDriver()*
HostMAC_ID	the MAC_ID of your port
DestMAC_ID	Destination MAC_ID
Instance	Instance of DeviceNet Identity Object
Attribute	Attribute of DeviceNet Identity Object
array	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get DeviceNet Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
BYTE argu[8];
int baudrate;
//      Get the baudrate of the device
DNetDNet(handle, HostMAC_ID, DestMAC_ID, 1, 2, argu);
baudrate = argu[0];
```

Visual Basic(Windows 95/98).

DNetSetID()

Purpose Change the MAC ID of the device.

Prototype **C/C++(DOS and Windows 95/98)**

```
int    DNetSetID(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, BYTE
newMAC_ID);
```

Visual Basic(Windows 95/98)

Parameters

handle	handle retrieve from *CanOpenDriver()*
HostMAC_ID	the MAC_ID of your port
DestMAC_ID	Destination MAC_ID

	newMAC_ID	:	revised MAC ID.

Return Value Return 0 is successful

-1 if error occurs

Remarks Set new MAC ID of one device.

After success calling, the device would send a duplicate check message immediately. And previous connection will be lost. You need to make a connection again.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
//	Change MAC ID to address 63
if(DNetSetID(handle, HostMAC_ID, DestMAC_ID, 63) != 0)
{
…Set failed!!
}
```

Visual Basic(Windows 95/98).

DNetSetBaud()

Purpose Set the baud rate of device.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetSetBaud(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, BYTE NewBaudrate);

Visual Basic(Windows 95/98)

Parameters	handle	:	handle retrieve from *CanOpenDriver()*
	HostMAC_ID	:	the MAC_ID of your port
	DestMAC_ID	:	Destination MAC_ID
	NewBaudrate	:	new baudrate

Return Value Return 0 is successful

-1 if error occurs

Remarks Set the new baud rate.

The newer configuration of baud rate would be activated after a software Reset command or re-power on.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
//	Set baud rate as 500 Kbps
if(DNetSetBaud(handle, HostMAC_ID, DestMAC_ID, 2) != 0)
```

{

…Set failed!!

}

Visual Basic(Windows 95/98).

DNetAsmIn()

Purpose Get the Assembly Input object of DeviceNet object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetAsmIn(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Assembly Input Object Information.

Get the Assembly Input data. The format of returned data is different from device to device. You need to take care of format. The returned data is only the data field(other fields such as service code, instance, and attribute are cleared).

See Also

Usage **C/C++(DOS and Windows 95/98)**

none

Visual Basic(Windows 95/98).

none

DNetAsmOut()

Purpose Set the Assembly Output object of DeviceNet object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetAsmOut(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, int len, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
len	:	len of data (from 0 to 3)
array	:	data to send (4-bytes data)

Return Value Return 0 is successful

-1 if error occurs

Remarks Set DeviceNet Assembly Output.

Set the Assembly Output data. The format of returned data is different from device to device. You need to take care of format. The maximum data length is 3 bytes.

See Also

Usage **C/C++(DOS and Windows 95/98)**

none

Visual Basic(Windows 95/98).

none

DNetGetSafeOut()

Purpose Get the safety output of the device

Prototype **C/C++(DOS and Windows 95/98)**

int DNetGetSafeOut(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID);

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID

Return Value Return 0-255 is the safety output

-1 if error occurs

Remarks Get safety output of a device

Most output device has its safety output, and the value is stored inside the hardware. You can use this function to set the safety output value of device. This function supports only one byte safety returns. For other format, you can call the *DNetAsmOut()* to get.

See Also

Usage **C/C++(DOS and Windows 95/98)**

none

Visual Basic(Windows 95/98).

DNetSetSafeOut()

Purpose Set the safety output of a device

Prototype **C/C++(DOS and Windows 95/98)**

int DNetSetSafeOut(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, BYTE safeValue);

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID

Return Value Return 0 is successful

-1 if error occurs

Remarks Set the safety output value of a device.

It supports only a BYTE configuration. For other type of assembly out(more bytes), you can use the *DNetAsmOut()*.

See Also

Usage **C/C++(DOS and Windows 95/98)**

none

DNetConnExp()

Purpose Get the Information of DeviceNet Explicit Connection Object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetConnExp(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Attribute	:	Attribute of DeviceNet Explicit Connection Object

	array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get the information of the DeviceNet Connection Object(Explicit part). There are several connection defined in the DeviceNet. This version (version : 0.3) supports only explicit connection and Poll-IO connections. Use *DNetConnExp()* to access explicit connection object and use *DNetConnIO()* to access Poll-IO one.

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the explicit connection state
if(DNetConnExp(handle, HostMAC_ID, DestMAC_ID, 1, argu) != 0)
{
…Get failed!!
}
```

Visual Basic(Windows 95/98).

DNetSetWDTime()

Purpose Set watchdog timeout time.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetSetWDTime(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, WORD time);

Visual Basic(Windows 95/98)

Parameters	handle	:	handle retrieve from *CanOpenDriver()*
	HostMAC_ID	:	the MAC_ID of your port
	DestMAC_ID	:	Destination MAC_ID
	time	:	active time(really time is 10*time ms)

Return Value Return 0 is successful

-1 if error occurs

Remarks Set the timeout time for the Watchdog function enable.

If the watchdog function be enable, the watchdog event would be signaled after the timeout time with receiving anything from the master. The activity is configured by *DNetSetWD();*

See Also

Usage **C/C++(DOS and Windows 95/98)**

Visual Basic(Windows 95/98).

DNetSetWD()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetConnIO()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetDIP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetDOP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetSetDOP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetAIP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

```
int   DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int
Instance, int Attribute, BYTE* array);
```

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port

DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//	Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetSetAIP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetAOP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

int DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int Instance, int Attribute, BYTE array);*

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).

DNetSetAOP()

Purpose Get the Information of DeviceNet Identity object.

Prototype **C/C++(DOS and Windows 95/98)**

```
int    DNetIdentity(int handle, BYTE HostMAC_ID, BYTE DestMAC_ID, int
Instance, int Attribute, BYTE* array);
```

Visual Basic(Windows 95/98)

Parameters

handle	:	handle retrieve from *CanOpenDriver()*
HostMAC_ID	:	the MAC_ID of your port
DestMAC_ID	:	Destination MAC_ID
Instance	:	Instance of DeviceNet Identity Object
Attribute	:	Attribute of DeviceNet Identity Object
array	:	returned data

Return Value Return 0 is successful

-1 if error occurs

Remarks Get Identity Object Information.

The Table is a part of DeviceNet Information

See Also

Usage **C/C++(DOS and Windows 95/98)**

```
WORD vendor;
```

```
BYTE argu[8];
//      Get the vendor of the device
if(DNetIdentity(handle, HostMAC_ID, DestMAC_ID, 1, 1, argu) != 0)
{
…Get failed!!
}
vendor = argu[1];
vendor <<= 8;
vendor |= argu[0];
return vendor;
```

Visual Basic(Windows 95/98).