PISO-CAN Series
CANopen Master Library

User's Manual

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assume no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Trademark

The names used for identification only maybe

registered trademarks of their respective companies.

Tables of Content

1 General Information

1.1 CANopen Introduction

The CAN (Controller Area Network) is a serial communication protocol, which efficiently supports distributed real-time control with a very high level of security. It is an especially suited for networking "intelligent" devices as well as sensors and actuators within a system or sub-system. In CAN networks, there is no addressing of subscribers or stations in the conventional sense, but instead, prioritized messages are transmitted. CANopen is one kind of the network protocols based on the CAN bus and it is applied in a low level network that provides connections between simple industrial devices (sensors, actuators) and higher-level devices (controllers), as shown in Figure 1.1.

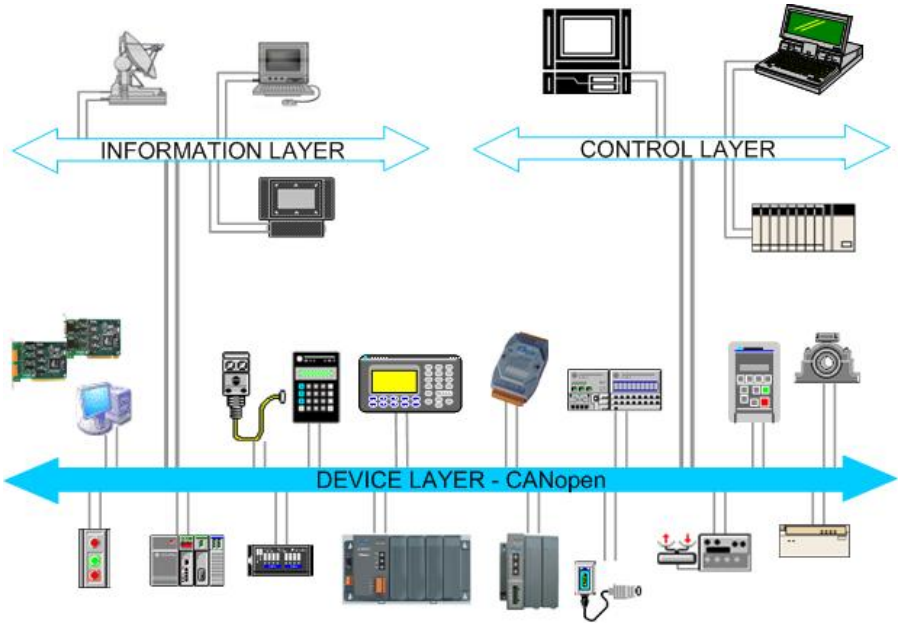


Figure 1.1 Example of the CANopen network

CANopen was developed as a standardized embedded network with highly flexible configuration capabilities. It provides standardized communication objects for real-time data (Process Data Objects, PDO), configuration data (Service Data Objects, SDO), network management data (NMT message, and Error Control), and special functions (Time Stamp, Sync message, and Emergency message). Nowadays, CANopen is used in many various application fields, such as medical equipment, off-road vehicles, maritime electronics, public transportation, building automation and so on.

1.2 CANopen Applications

CANopen is the standardized network application layer optimized for embedded networks. Its specifications cover the standardized application layer, frameworks for the various applications (e.g. general I/O, motion control system, maritime electronics and so forth) as well as device, interface, and application profiles.

The main CANopen protocol and products are generally applied in the low-volume and mid-volume embedded systems. The following examples show some parts of the CANopen application fields. (For more information, please refer to the web site, http://www.can-cia.org)**:**

- Truck-based superstructure control systems
- Off-highway and off-road vehicles
- Passenger and cargo trains
- Maritime electronics
- Factory automation
- Industrial machine control
- Lifts and escalators
- Building automation
- Medical equipment and devices
- Non-industrial control
- Non-industrial equipment

1.3 CANopen Master Library Characteristics

ICP DAS CANopen Master DLL Library provides users to establish CANopen communication network rapidly. It is special for ICPDAS PISO-CAN200/400/200U/400U/200E and PCM-CAN200 PCI interface card. Using the library, most of the CANopen communication protocols will be handled by the library function automatically. Therefore, it can help users reduce the complexity of developing a CANopen master interface, and let users to ignore the CANopen protocol detail technology information. The library mainly supports the predefined master-slave connection set, which include some useful functions to control the CANopen slave device in the CANopen network. The general application architecture is demonstrated as the Figure 1.2.

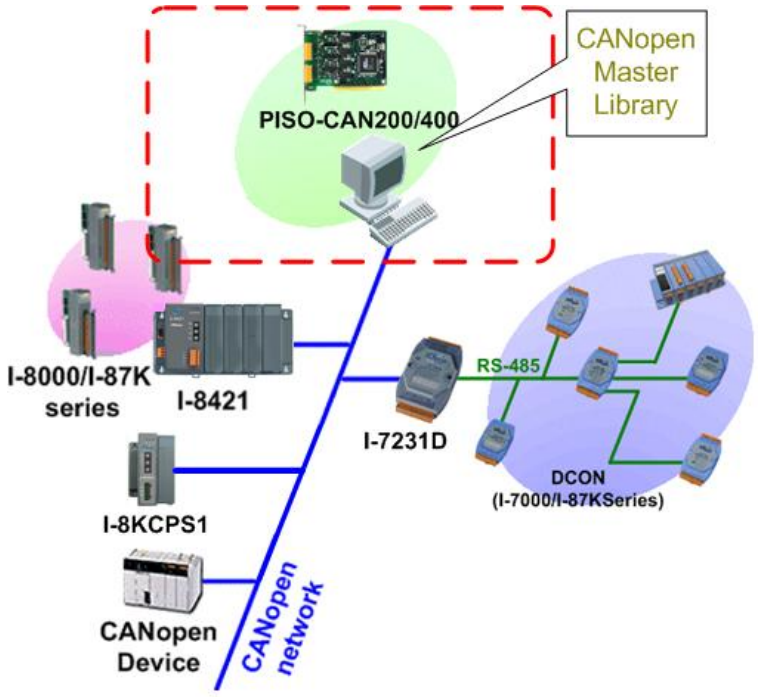


Figure 1.2 Application architecture

The CANopen Master Library follows the CiA CANopen specification DS-301 V4.01, and supports the several CANopen features. The CANopen communication general concept is shown as Figure 1.3.

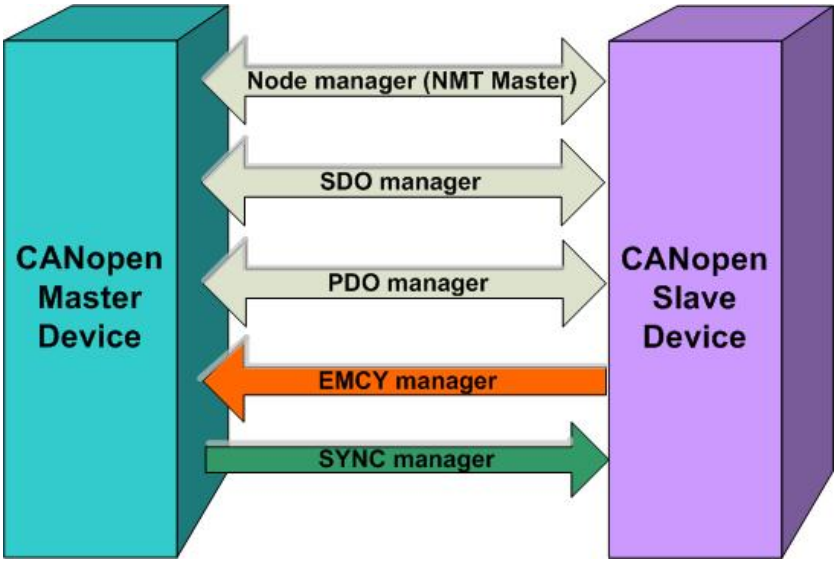


Figure 1.3 CANopen communication general concept

- **Node Manager (NMT Master)**
  - Provide function to change the slave device state
  - Node Guarding Protocol for error control
  - Support Emergency (EMCY) message
- **SDO Manager**
  - Expedited, segmented and block methods for SDO download and upload
- **PDO Manager**
  - Support transmission type and event timer
- **SYNC Manager**
  - SYNC messages production
  - SYNC cycles of 1ms resolution
- **EMCY Manager**
  - EMCY messages consumer

For more information about the CANopen functions described above, please refer to the demo programs and the description in the section 3.

1.4 Features

- Driver supported for Windows 98/ME/NT4/2000/XP
- Programmable transfer-rate 10K, 20K, 50K, 125K, 250K, 500K, 800K, 1M
- Each Port support maximum nodes up to 127
- Follow CiA DS-301 V4.01
- Support upload and download SDO Segment and Block protocol
- Extended mode of COB-ID is supported
- Support Node Guarding protocol

2 Software Installation

The CANopen DLL driver is the CANopen specification function collections for the PISO-CAN200/400 cards used in Windows 98/Me/NT/2000/XP systems. The application structure is presented in the following figure. The users' CANopen master application programs can be developed by the following program development tools: Delphi, Borland C++ Builder, VB, and visual C++. When users use these program development tools to develop the CANopen master interface, the PISOCANCPM.DLL must be used. Because the PISOCANCPM.DLL calls the function of PISO-CAN200/400 driver, the PISO-CAN200/400 driver must be installed before using the PISOCANCPM.DLL driver. The driver architecture is shown in the following Figure.

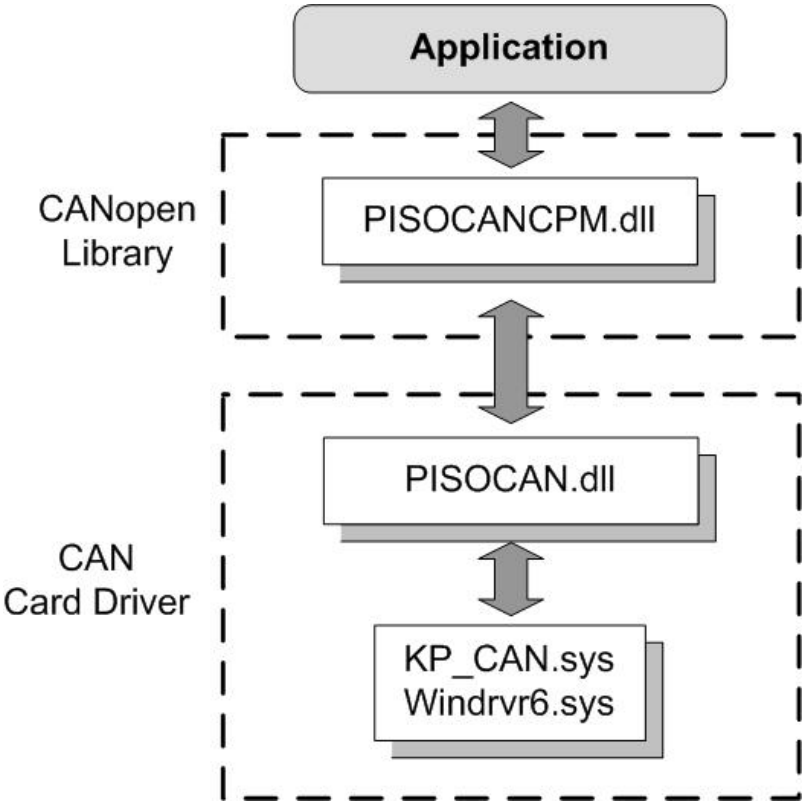


Figure 2.1 Driver concept of CANopen library

2.1 Installation Driver Step by Step

When users want to use the CANopen specification DLL driver, the PISO-CAN series CAN card driver must be installed firstly. Afterwards, users should install the CANopen Master Library. After finishing the installation process, the demo programs may be a good reference for users to build their CANopen master interface by using VC++, BCB and VB. The demo programs also give a simple interface to show the basic functions of master/slave connection and CANopen master program architectures. It is very helpful for users to understand how to use these functions and how to develop their CANopen master application. The following description displays the step-by-step procedures about how to install the PISO-CAN driver and PISOCANCPM.DLL driver.

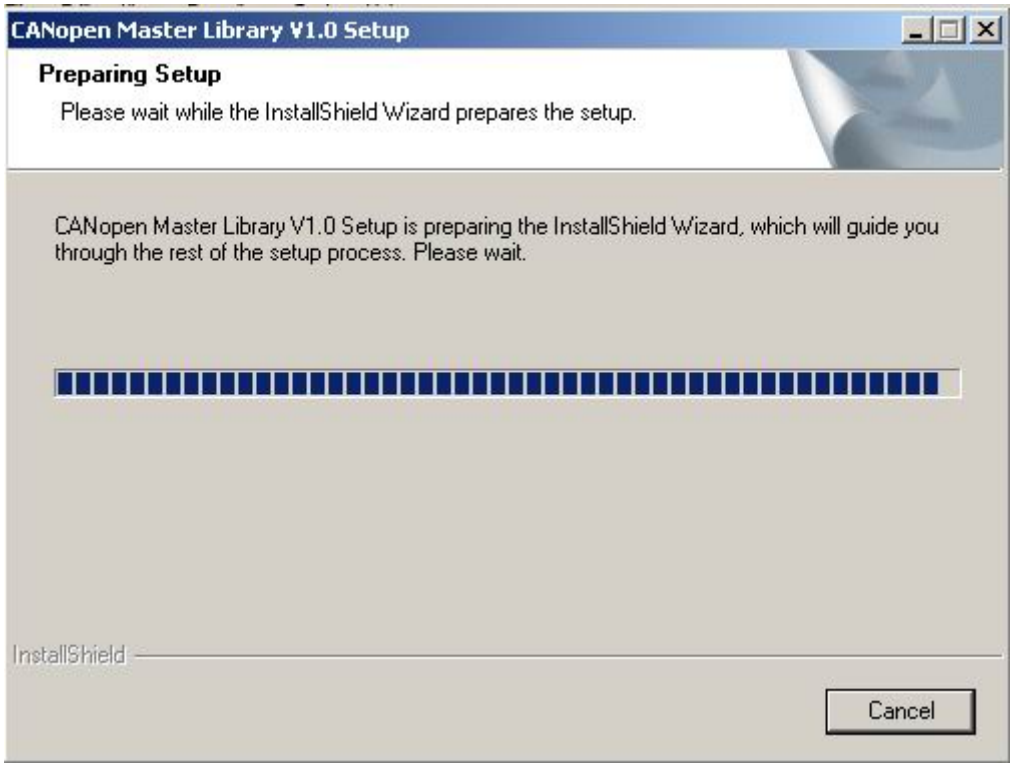
Install and Remove the PISO-CAN series CAN card driver

Please refer to the manual PISO-CAN.pdf. It can be found in the web path http://ftp.icpdas.com/pub/cd/fieldbus_cd/can/pci/piso-can200_400/manual/ and the CD path \CAN\PCI\PISO-CAN200_400\Manual.

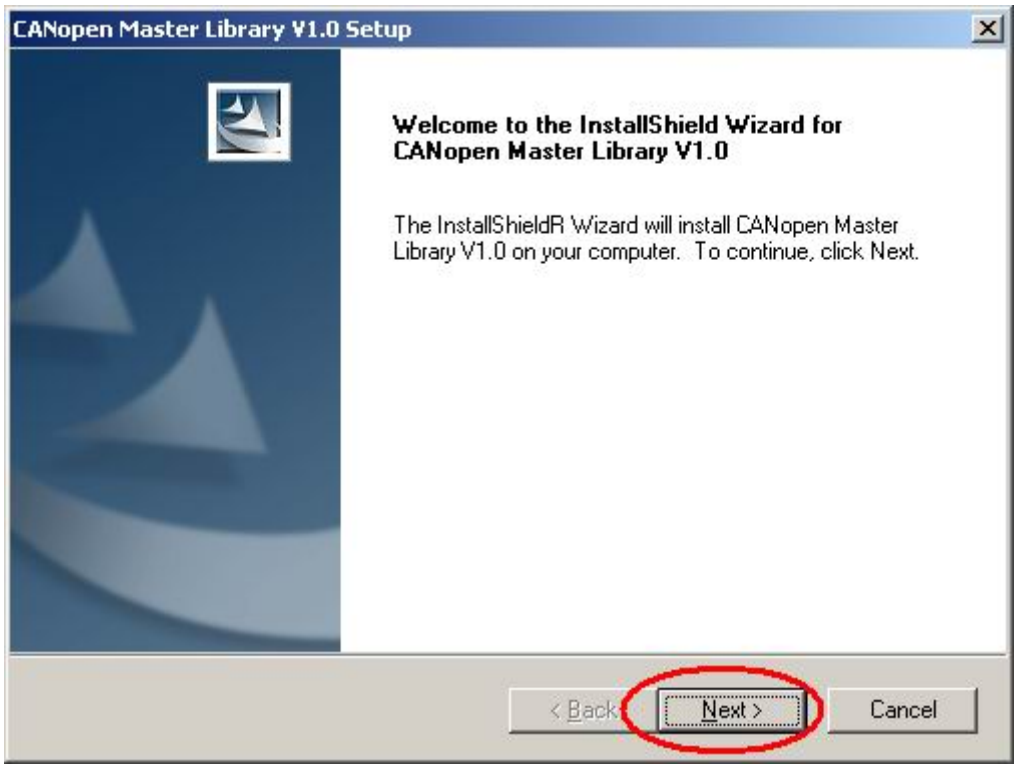
Install the CANopen Master Library

Step 1: Insert the product CD into the CD-ROM and find the path \CANopen\Master\PISO-CAN200_400. Then execute the setup.exe to install the CANopen Master Library.

Step 2: Wait until the install wizard has prepared.



Step 3: Click “Next” to start the CANopen Master Library installation.



Step 4: Select the folder where the CANopen Master Library setup will be installed and click “Next” button to continue.

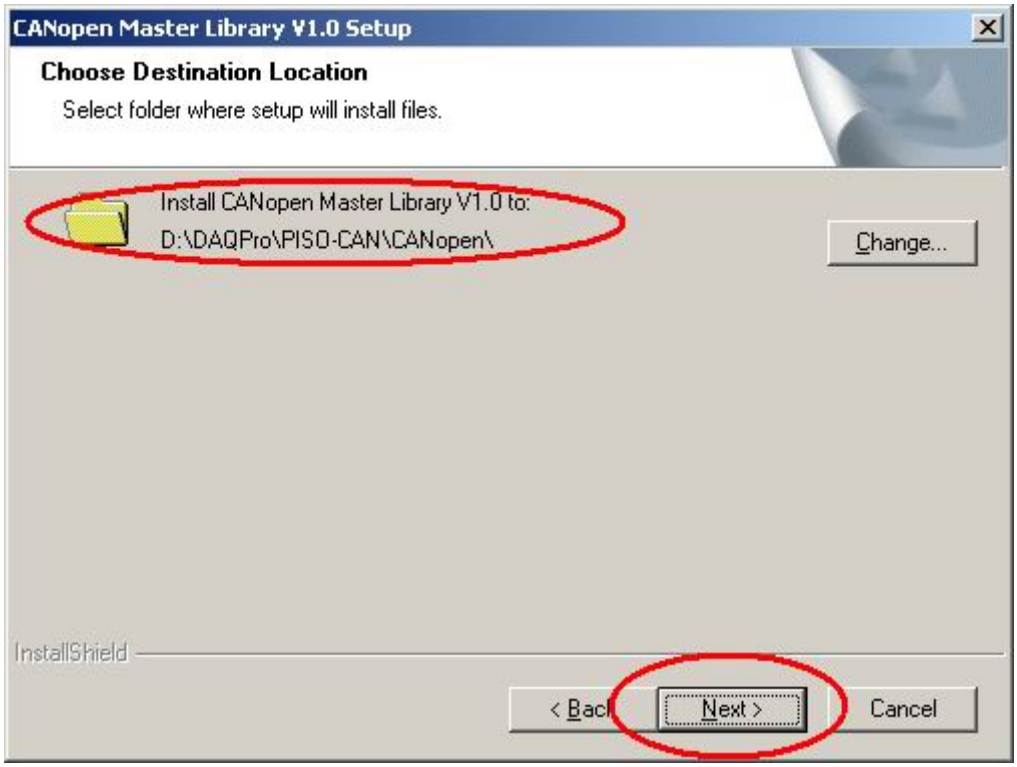
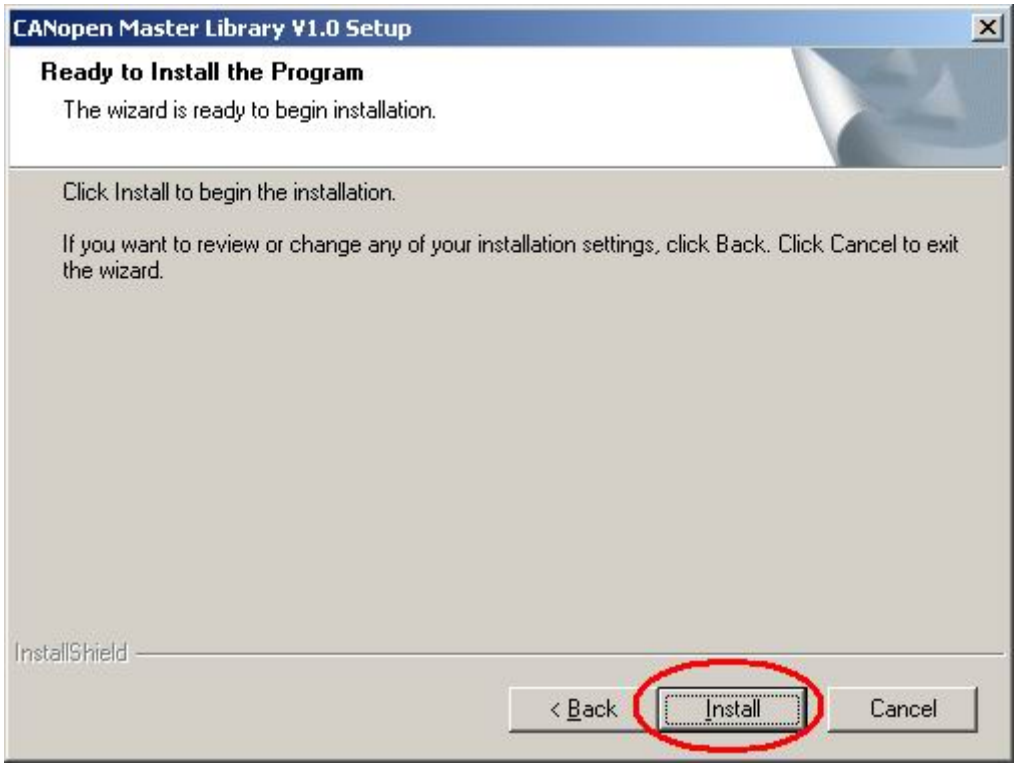
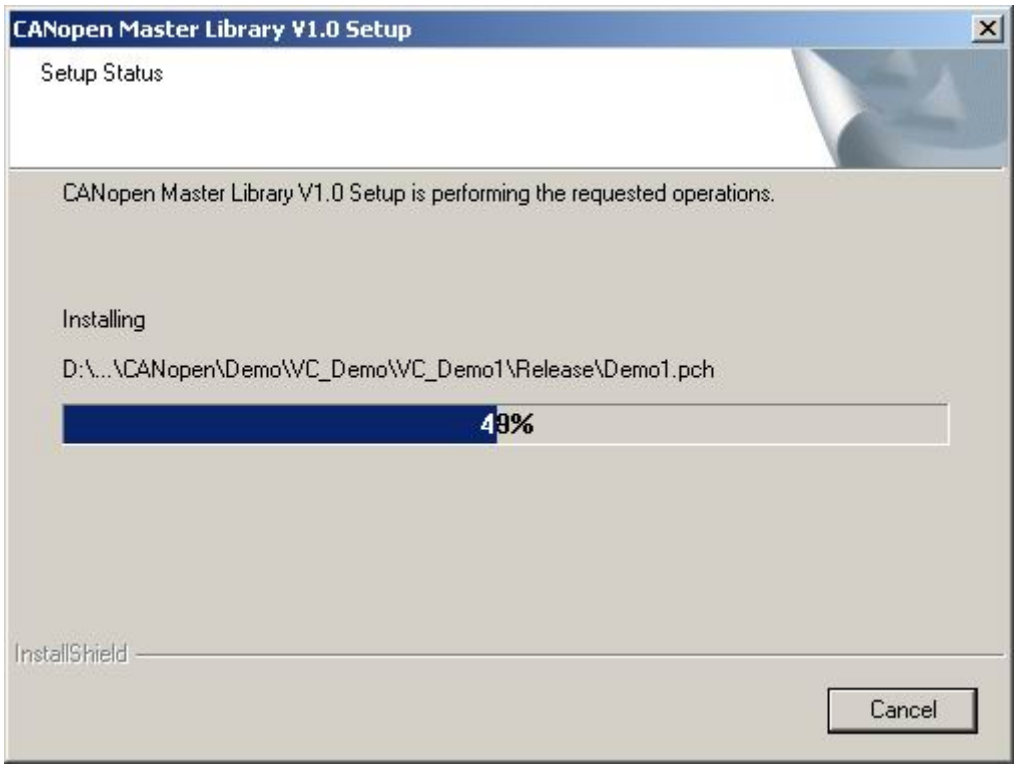


Figure 2.5

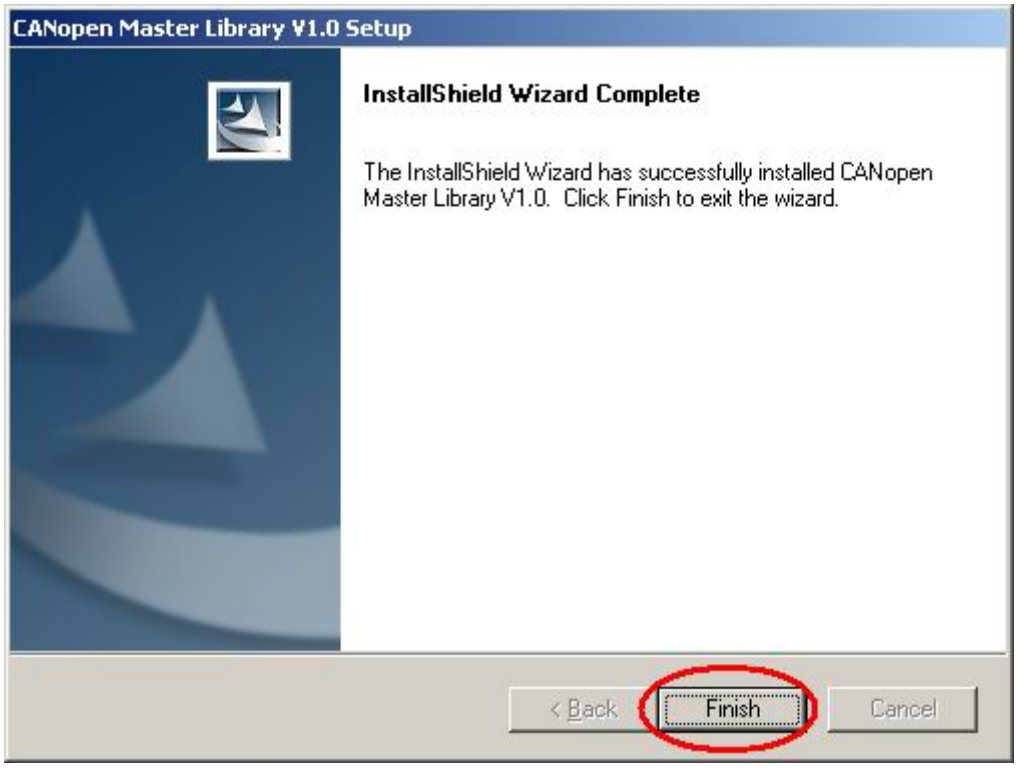
Step 5: Click the button “Install” to continue.



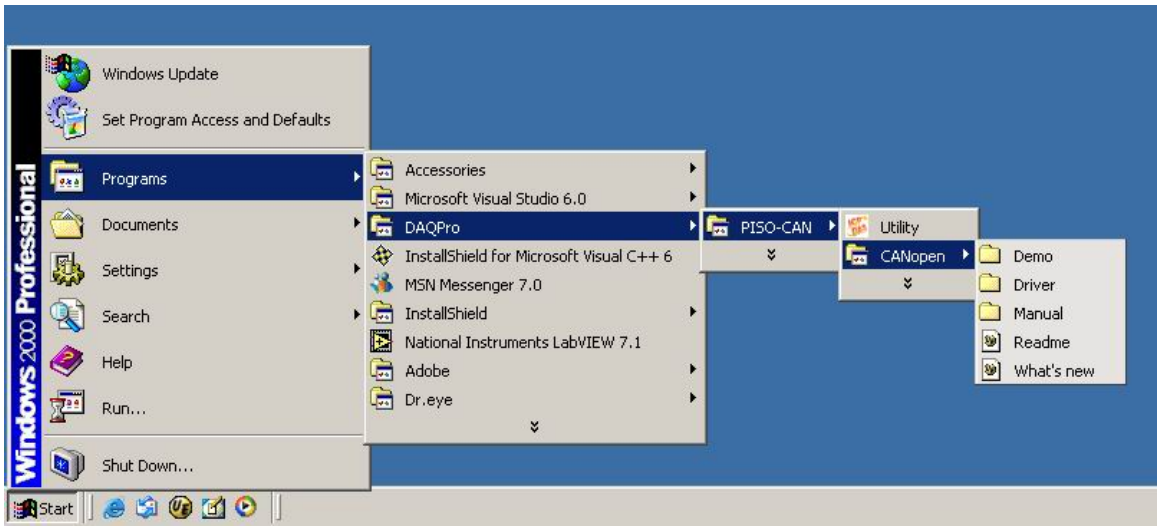
Step 6: Wait for the CANopen Master Library installation.



Step 7: After finishing the process, click “Finish” button to complete the installation.

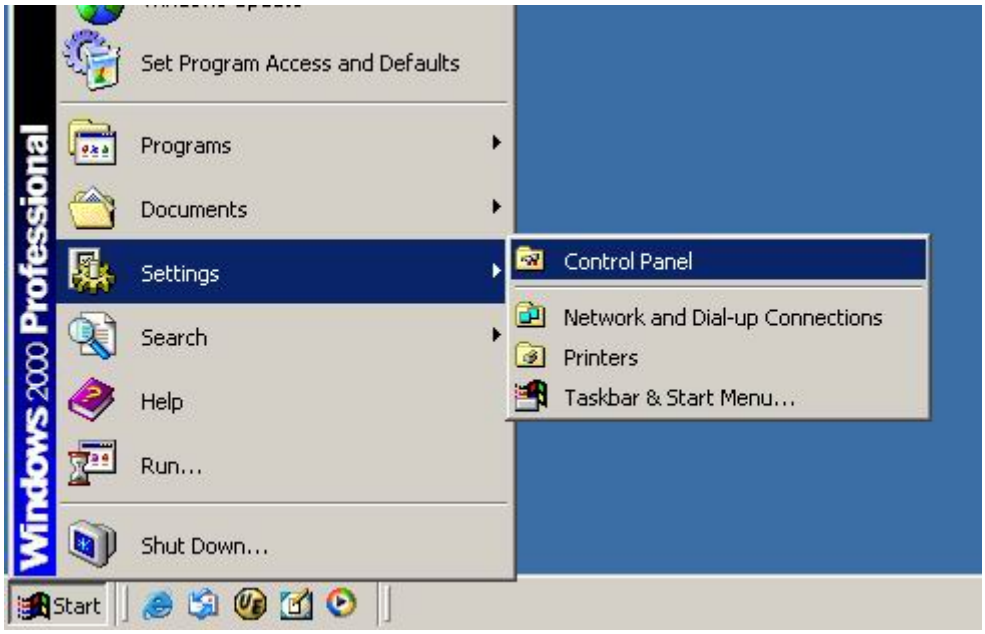


When finishing the CANopen Master Library installation. The CANopen folder will be found at the Start menu shown as below.



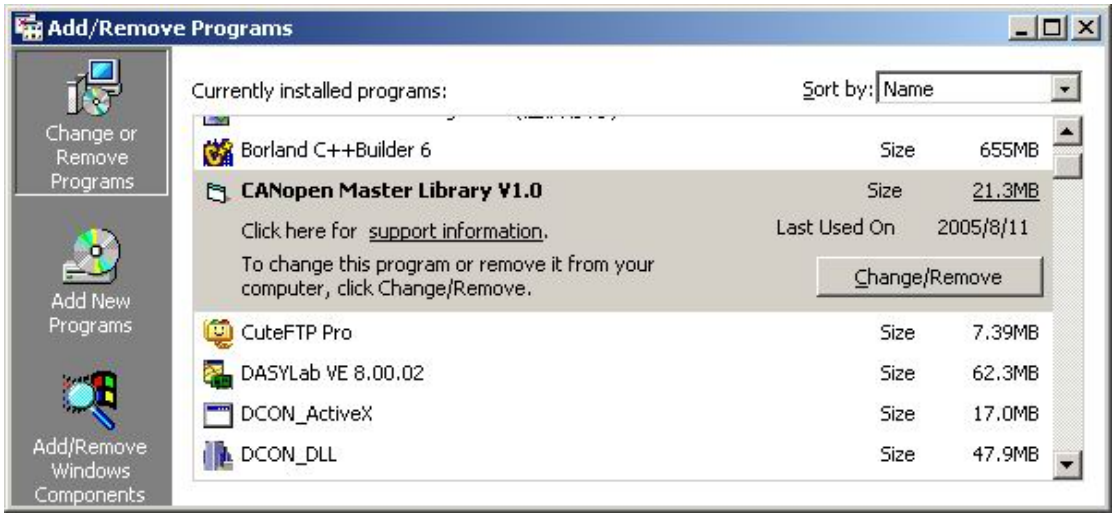
Remove the CANopen Master Library

Step 1: Click “Start” in the task bar. Select the Setting/Control Panel as shown in the following figure.

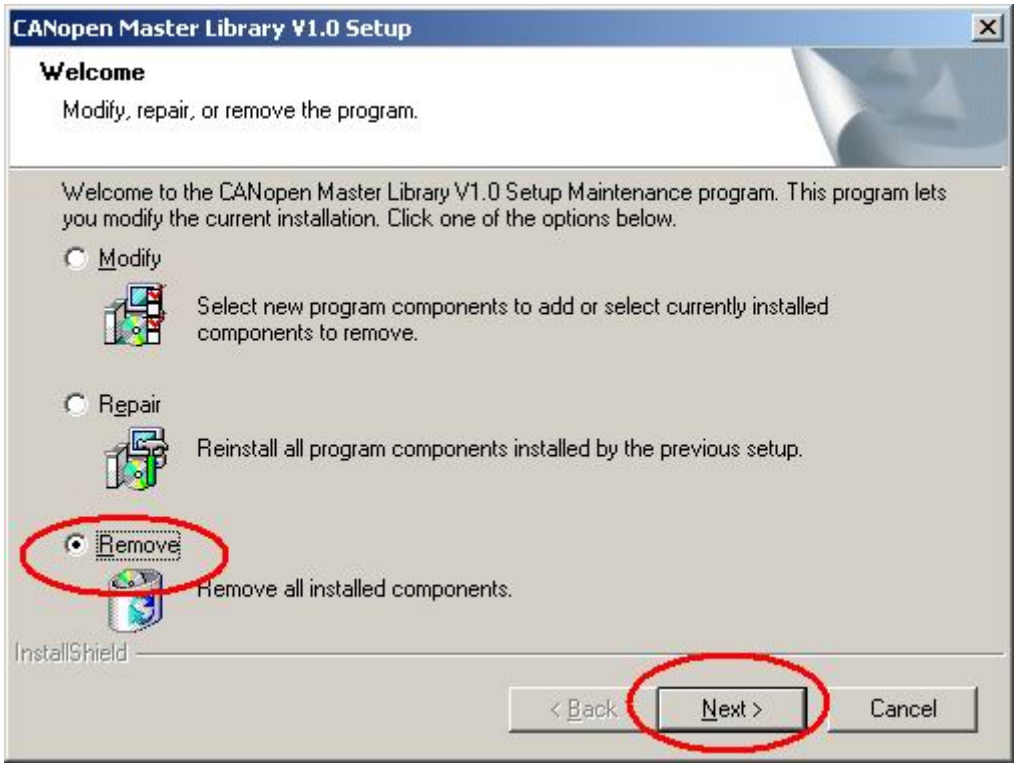


Step 2: Click the “Add/Remove Programs” icon to open the dialog.

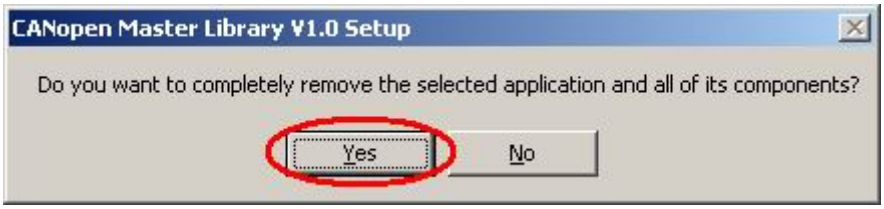
Step 3: Find out the CANopen Master Library, and click the button “Change/Remove”.



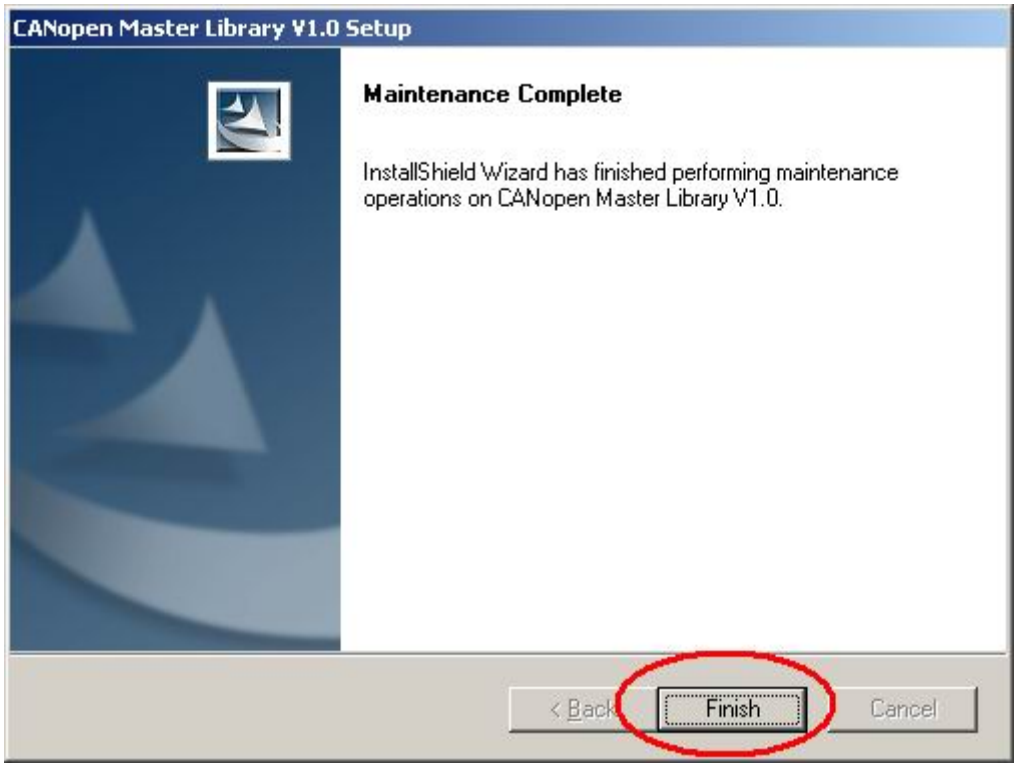
Step 4: Choose “Remove” option and click the button “Next” to remove the software.



Step 5: Click the button “Yes” to remove the software.



Step 6: Finally, click the button “Finish” to finish the uninstall process.



3 Function Description

3.1 DLL Function Definition and Description

All the functions provided by the PISOCANCPM.dll are listed in the following table. The detail information for each function is presented in the section 4.2. In order to make the descriptions more simply and clear, the attributes for the both input and output parameter functions are given as [input] and [output] respectively. They are shown in the table 3.1.

Keyword	Set parameter by user before calling this function?	Get the data from this parameter after calling this function?
[input]	Yes	No
[output]	No	Yes

Table 3.1

Functions Table

No.	Function Name	Description
1	CPM_GetVersion	Get the version of the PISOCANCPM.dll
2	CPM_ActiveBoard	Activate the PISOCAN200/400 CAN card
3	CPM_CloseBoard	Stop and close the kernel driver
4	CPM_InitPort	Initialize port and return CAN channel Handle
5	CPM_InitMaster	Initialize CANopen master parameters
6	CPM_ShutdownMaster	Stop the CANopen master function
7	CPM_AddNode	Add one node into the CANopen network
8	CPM_RemoveNode	Remove one node from the CANopen network
9	CPM_NMTChangeState	Change the CANopen node state

10	CPM_NMTGetState	Get CANopen node state
11	CPM_NMTGuarding	Start to the node guarding function
12	CPM_SDOReadData	Read data by upload SDO protocol
13	CPM_SDOReadSegment	Read data by upload SDO segment data
14	CPM_SDOReadBlock	Read data by upload SDO block protocol
15	CPM_SDOWriteData	Write data by download SDO protocol
16	CPM_SDOWriteSegment	Write data by download SDO segment data
17	CPM_SDOWriteBlock	Write data by download SDO block protocol
18	CPM_SDOAbortTransmission	Send SDO abort message
19	CPM_InstallPDO	Install and enable the specific PDO
20	CPM_MappingPDO	Setting the PDO mapping object
21	CPM_RemovePDO	Remove the PDO object
22	CPM_WritePDO	Use PDO to write data to the device
23	CPM_RemotePDO	Use PDO to get data from the remote device
24	CPM_ResponsePDO	Read the PDO data responded from slave
25	CPM_ResPDOCount	Get PDO message count
26	CPM_SendSYNC	Send SYNC message
27	CPM_ReadEMCY	Read EMCY message
28	CPM_WriteDO	Use SDO to output data to index 0x6200
29	CPM_WriteAO	Use SDO to output data to index 0x6411
30	CPM_ReadDI	Use SDO to input data from index 0x6000
31	CPM_ReadAI	Use SDO to input data from index 0x6401

Table 3.2 Description of functions

3.2 Function Return Code

The following table interprets all the return code returned by the CANopne Master Library function.

Return Code	Error ID	Comment
0	CPM_NoError	OK
1	CPM_DriverError	CAN card kernel driver can't be opened
2	CPM_ActiveBoardError	This board can't be activated.
3	CPM_BoardNumberError	The board number isn't correct or exceeds the maximum board number, 7.
4	CPM_PortNumberError	The port number isn't correct exceeds the maximum port number.
5	CPM_ResetError	CAN controller hardware reset error
6	CPM_SoftResetError	CAN controller software reset error
7	CPM_InitError	Initiate the CAN controller failure
8	CPM_ConfigError	Port has not been configured successfully.
9	CPM_SetACRError	Set the Acceptance Code Register error
10	CPM_SetAMRError	Set the Acceptance Mask Register error
11	CPM_SetBaudRateError	Set Baud Rate error
12	CPM_EnableRxIrqFailure	Enable CAN controller receive interrupt failure
13	CPM_DisableRxIrqFailure	Disable CAN controller receive interrupt failure
14	CPM_InstallIrqFailure	Install PCI board IRQ failure
15	CPM_RemoveIrqFailure	Removing PCI board IRQ failure
16	CPM_TransmitBufferLocked	CAN controller transmit buffer is locked
17	CPM_TransmitIncomplete	Previously transmission is not yet completed
18	CPM_ReceiveBufferEmpty	CAN controller Rx FIFO is empty
19	CPM_DataOverrun	Data was lost because there was not enough space in Rx FIFO
20	CPM_ReceiveError	Receive data is not completed
21	CPM_SoftBufferIsEmpty	Software buffer is empty
22	CPM_SoftBufferIsFull	Software buffer is full

50	CPM_MasterInitErr	CANopen Master initializes error
52	CPM_NodeAddErr	Add a CANopen node failure
53	CPM_NodeRemoveErr	Remove a CANopen node failure
54	CPM_NodeIDDoesExist	The node id has existed
55	CPM_NodeIDDoesNotExist	The node id is not existed
56	CPM_NodeIdOverRange	The id is out of the range (1 ~ 127)
60	CPM_CreateThreadErr	Create receive or transmit data thread error
61	CPM_ResumeThreadErr	Receive or transmit data thread resumed error
62	CPM_SuspendThreadErr	Receive or transmit data thread suspended error
63	CPM_TerminateThreadErr	Receive or transmit data thread terminated error
64	CPM_GetExitCodeThreadErr	Gat exit code of thread error
65	CPM_CloseHandleErr	Close receive or transmit data thread error
70	CPM_CobIdDoesExist	The COB-ID has been used
71	CPM_CobIdDoesNotExist	The COB-ID has not been used
72	CPM_EMCYFIFOisEmpty	No EMCY message in the buffer
73	CPM_SlaveStateErr	CANopen node state error
74	CPM_AllPDOareUsing	All PDO objects are occupied
75	CPM_PDOInstallErr	The COB-ID can’t be install to PDO
76	CPM_PDOSetError	PDO mapping parameters error
77	CPM_PDORemoveErr	The PDO can’t be disable
78	CPM_PDODoesNotExist	The PDO is not exist
79	CPM_CobIdNotRxPDO	The COB-ID is not Rx PDO
80	CPM_CobIdNotTxPDO	The COB-ID is not Tx PDO
81	CPM_MappingEnableErr	Set PDO mapping object failure
82	CPM_NoResponse	PDO message don’t respond
90	CPM_ReadSegment	Upload SDO segment protocol need to be used
91	CPM_WriteSegment	Download SDO segment protocol need to be used
92	CPM_ReadBlock	Upload SDO block protocol need to be used
93	CPM_WriteBlock	Download SDO block protocol need to be used
95	CPM_DataReSend	Segment or block transmit failure, users need to resend the data

96	CPM_ReadDataError	Upload Segment or block transmit error or slave didn’t support block protocol
97	CPM_WriteDataError	Download Segment or block transmit error or slave didn’t support block protocol
98	CPM_DataSizeRangeErr	Data length can’t be 0
100	CPM_Timeout	Response message is timeout

Table 3.3 Description of return code

3.3 CANopen Master Library Application Flowchart

In this section, it describes that the operation procedure about how to use the CANopen Master Library to build users application. This information is helpful for users to apply the CANopen Master Library easily. Besides, the CANopen operation principles must be obeyed when build a CANopen master application. For example, if the CANopen node is in the pre-operational status, the PDO communication object is not allowed to use. For more detail information, please refer to the demo programs in section 4.

Main Programming Sequence

When users program apply the CANopen Master Library functions, the function CPM_ActiveBoard must be call first. The function CPM_InitPort is used to initialize the CAN controller. It can be called after calling the function CPM_ActiveBoard.

After initializing the CAN interface successfully, the function CPM_InitMaster will be used to arrange the resources, create threads, and configure CAN controller. Then, users need to use the function CPM_AddNode to install at least one CANopen device into the node list.

If the function CPM_InitMaster and CPM_AddNode has been executed, the communication services (NMT, SYNC, EMCY, SDO, and PDO services) can be used at any time before calling the function CPM_ShutdownMaster because the function CPM_ShutdownMaster will stop all process created by the function CPM_InitMaster. Finally, when users want to finish their program, don’t forget to execute the function CPM_CloseBoard to release all resources and stop using the CAN card.

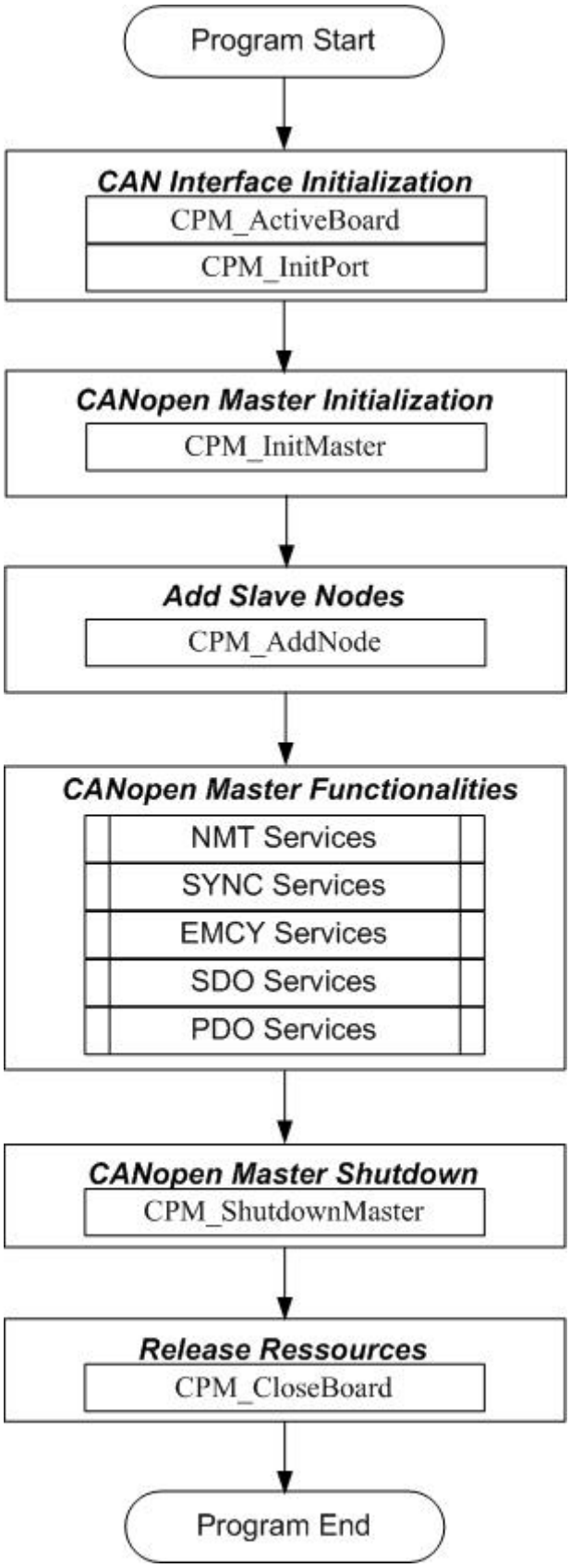


Figure 3.1 Main programming sequences

CAN Interface Initialization and Resource Release

Before using the CANopen master Initialization functionality, users need to call the functions CPM_ActiveBoard and CPM_InitPort firstly. If the CANopen master application is closed, the function CPM_CloseBoard is needed to release the system resources.

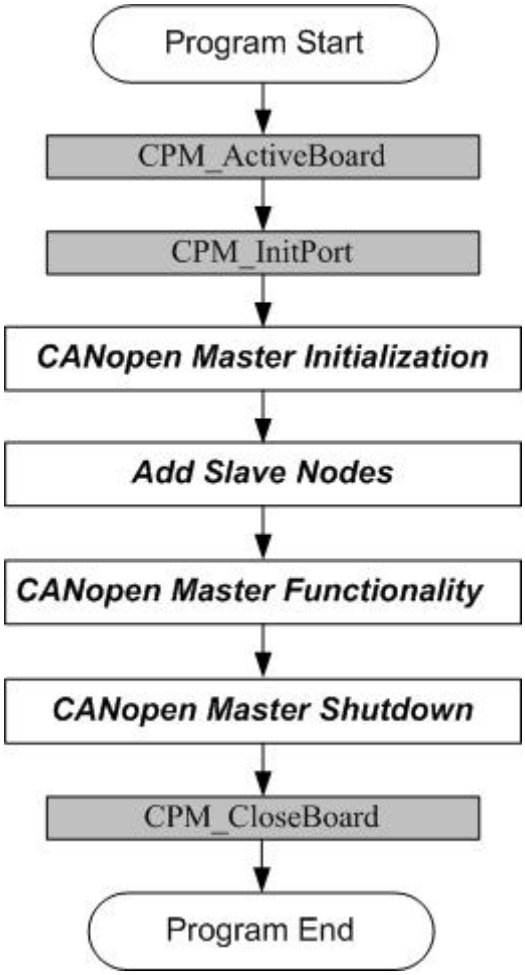


Figure 3.2 Initialize interface flow chart

Initialization and Shutdown of the CANopen Master

After initializing the CAN interface, the CANopen master functionalities are started up by using the function CPM_InitMaster. Calling the function CPM_InitMaster will configure the CAN controller of CAN card and initialize all of the components that will be used in the CANopen master functionalities. users need to use the function CPM_AddNode to install at least one CANopen

device into the node list. When users use the function CPM_ShutdownMaster, all CANopen nodes in the node list will be removed. Then, the CANopen master based on the PISO-CAN200/400 CAN card will stop to work.

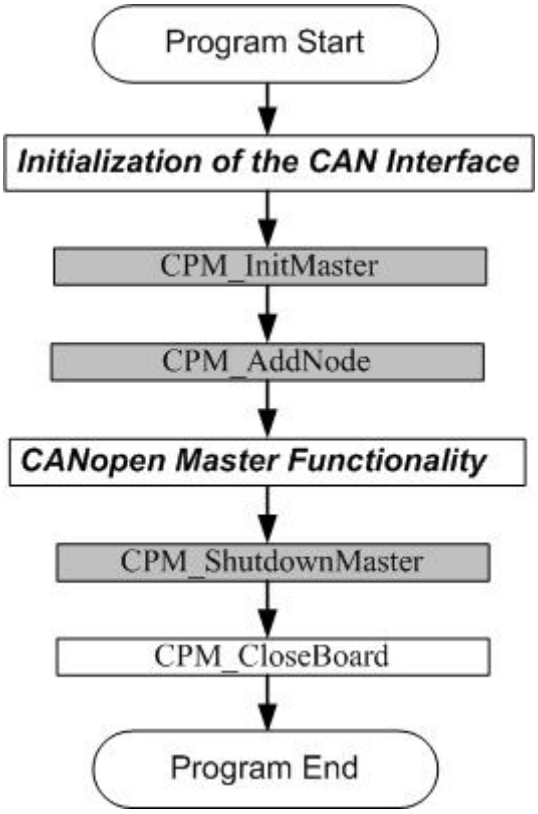


Figure 3.3 CANopen Master Initialization/shutdown interface flowchart

3.4 Communication Services Introduction

NMT Services

The CANopen Master Library provides several NMT services functions, such as the functions CPM_AddNode, CPM_RemoveNode, CPM_NMTChangeState, CPM_NMTGetState, and CPM_NMTGuarding. As the prerequisite for the master, the slave nodes have to be registered by CPM_AddNode function with providing its Node-ID. The registered slave nodes can be individually removed from the node list by the function CPM_RemoveNode. Through NMT services, the NMT Master controls the state of the slave. Table 4.1 is the command value and corresponding NMT command for the input parameters of the function CPM_NMTChangeState. When using the function CPM_NMTGetState, the slave status value and their descriptions are shown in the table 4.2. The Node Guarding protocol is implemented via the function CPM_NMTGuarding. If the slave nodes are in the node list, users can change the node guarding parameters defined in the slave nodes by calling the function CPM_NMTGuarding.

Command Value	Description
1 (0x01)	Enter Operational
2 (0x02)	Stop
128 (0x80)	Enter Pre-Operational
129 (0x81)	Reset_Node
130 (0x82)	Reset_Communication

Table 3.4 NMT Command Specifier

State of Slave	Description
4 (0x04)	STOPPED
5 (0x05)	OPERATIONAL
127 (0x7F)	PRE-OPERATIONAL

Table 3.5 The State of The Slave

SDO Services

Initiate SDO download or Initiate SDO upload protocol is used when SDO data length ≦ 4 bytes. These two protocols are handled by the functions CPM_SDOReadData and CPM_SDOWriteData with parameter bBlock = 0. If the SDO message data length ＞ 4 bytes, using these two functions described above will return the value CPM_ReadSegment or CPM_WriteSegment. In this case, users must use the SDO segment protocol to finish the SDO transmission. The SDO segment protocol can be handled by using the functions CPM_SDOReadSegment and CPM_SDOWriteSegment. If users call the function CPM_SDOReadData or CPM_SDOWriteData with parameter bBlock = 1, these two functions will return the values CPM_ReadBlock or CPM_WriteBlock. Then, the function CPM_SDOReadBlock or CPM_SDOWriteBlock must be used to handle the SDO block protocol whether the SDO data length exceeds 4 bytes or not.

CPM_SDOAbortTransmission function can abort a pending SDO transfer at any time. Applying the abort service will have no confirmation from the salve device.

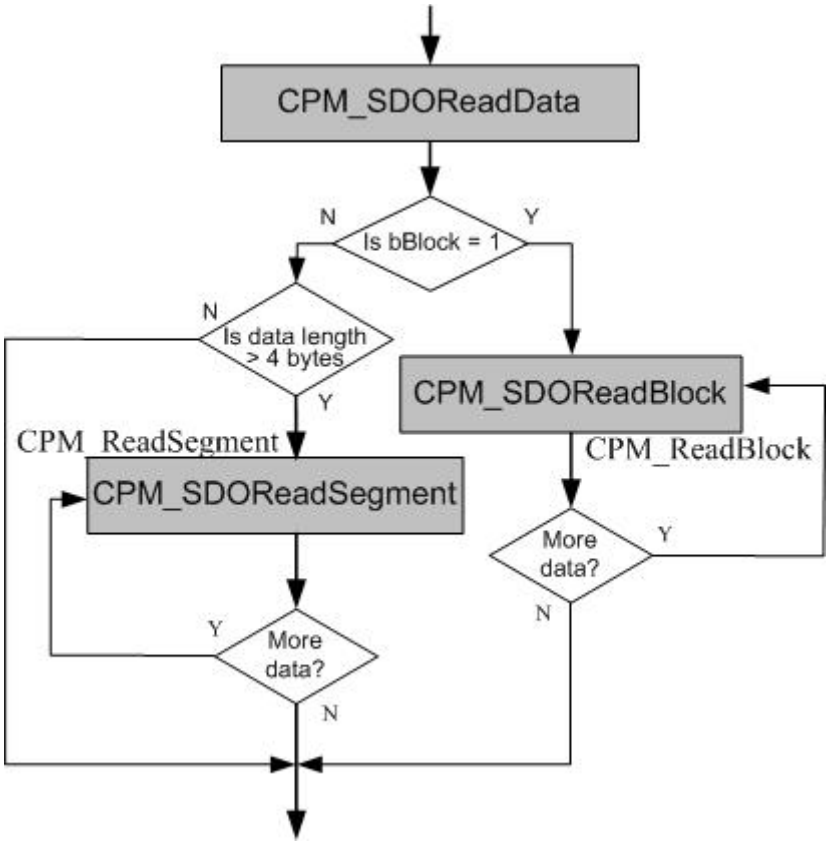


Figure 3.4 SDO upload flow chart

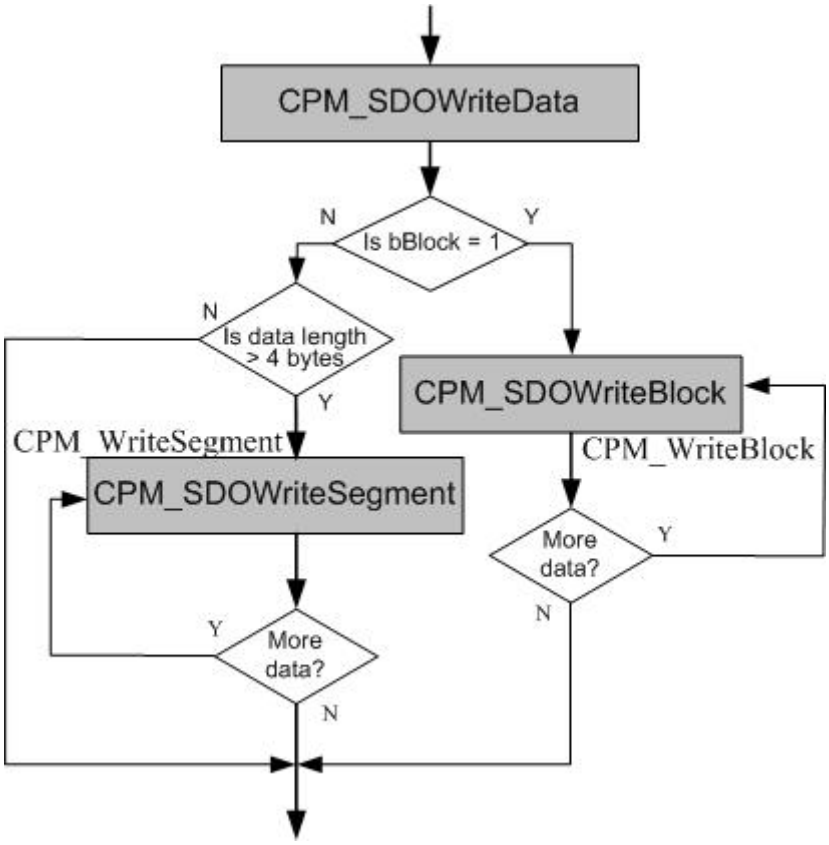


Figure 3.5 SDO download flow chart

PDO Services

The function CPM_MappingPDO is used for setting TPDOs or RPDOs mapping object. Each PDO object supports 0~8 application objects. These application objects defined in the CANopen specification DS401 are mapped to the DI/DO/AI/AO channels. After calling the function CPM_MappingPDO, users need to implement the function CPM_InstallPDO to activate the PDO communication object. If the PDO communication object is not needed no more, use the function CPM_RemovePDO to remove it.

The PDOs data are written to the PDO buffer by using the function CPM_WritePDO. This function can write all PDO 8-byte data or write some part of PDO 8-byte data. If users write some part of the PDO data, the other part of the PDO data will not be changed. When some device response the PDO data, users can use the function CPM_ResponsePDO to read these data stored in the PDO buffer.

In CANopen specification, users can get the TxPDOs data by applying the remote transmit request CAN frame. In this case, the function CPM_RemotePDO is needed.

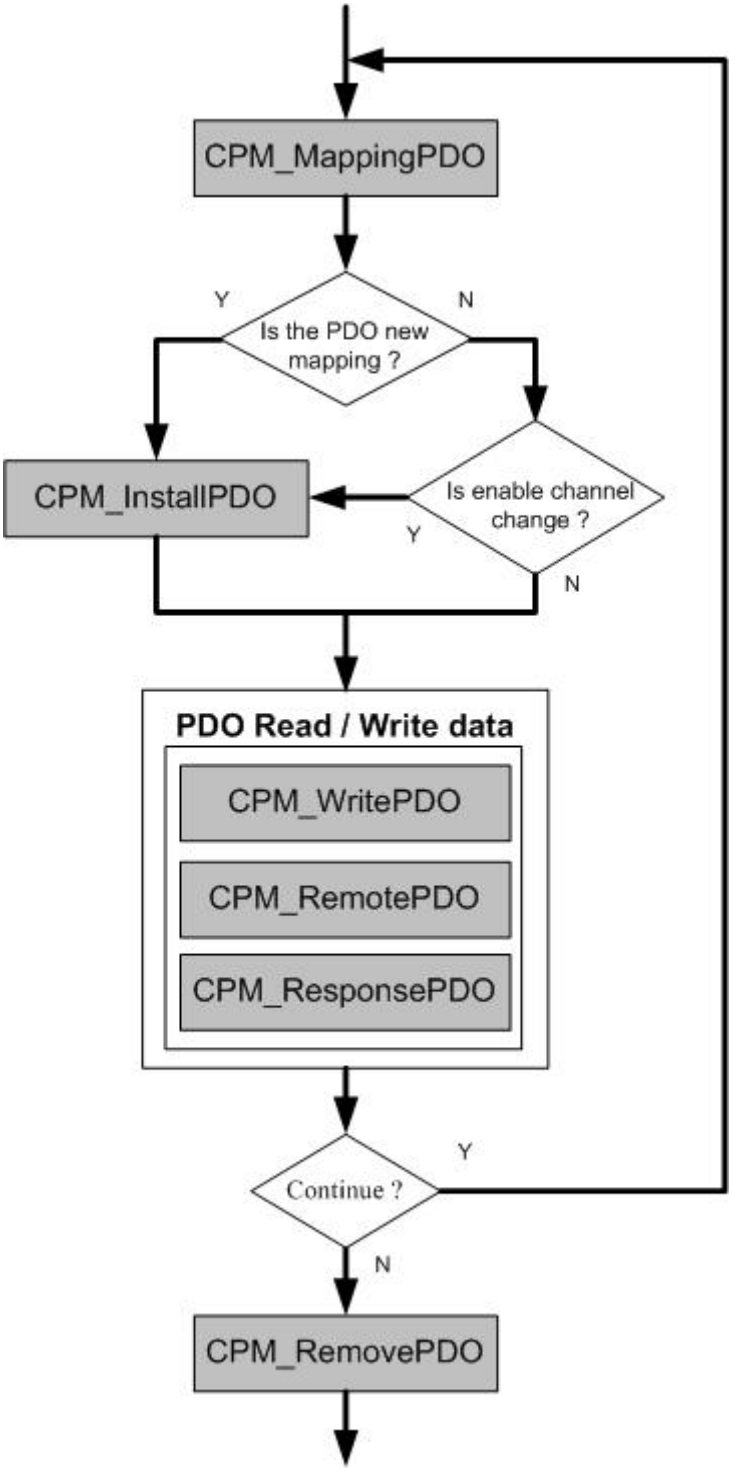


Figure 3.6 PDO flow chart

SYNC Services

Calling the function CPM_SendSYNC starts the SYNC object transmission. The parameter dwSyncCycle of the function CPM_SendSYNC can adjust the cycle period of SYNC COB-ID sent by master. This parameter range is from 0 to 4294967295ms. If the parameter dwSyncCycle is set to 0, the SYNC object transmission will be stopped.

EMCY Services

Emergency objects are triggered by the occurrence of a device internal error situation. When the master receives the emergency messages (EMCY messages) from the slaves, the EMCY message will be saved in the software buffer. Then, users need to call the function CPM_ReadEMCY to get the EMCY messages from the software buffer.

3.5 Function Description

3.5.1 CPM_GetVersion

- **Description:**

  This function is used to obtain the version information of PISOCANCPM.dll driver.

- **Syntax:**

  float CPM_GetVersion(void)

- **Parameter:**

  None

- **Return:**

  DLL library version information.

3.5.2 CPM_ActiveBoard

- **Description:**

  The function can activate the PISO-CAN 200/400. It must be called once before using other functions of PISOCANCPM.dll.

- **Syntax:**

  WORD CPM_ActiveBoard(BYTE bBoardNo)

- **Parameter:**

  bBoardNo: [input] PISO-CAN200/400 board number (0~7)

- **Return:**

  CPM_NoError
  CPM_DriverError
  CPM_BoardNumberError
  CPM_ActiveBoardError
  CPM_InstallIrqFailure

3.5.3 CPM_CloseBoard

- **Description:**

  The function can close the kernel driver and release the system resource. It must be called once before exiting the users' master application program. Or, the system resources may be occupied, and the next time to call the function CPM_ActiveBoard may fail.

- **Syntax:**

  WORD CPM_CloseBoard(BYTE bBoardNo)

- **Parameter:**

  bBoardNo: [input] PISO-CAN200/400 board number (0~7)

- **Return:**

  CPM_NoError
  CPM_DriverError
  CPM_BoardNumberError
  CPM_ActiveBoardError
  CPM_DisableRxIrqFailure
  CPM_RemoveIrqFailure

3.5.4 CPM_InitPort

- **Description:**

  The function can initiate the CAN controller and return a handle value for this CAN channel.

- **Syntax:**

  WORD CPM_InitPort(BYTE bBoardNo, BYTE bPort,
  CANChannelStruct *Handle)

- **Parameter:**

  bBoardNo: [input] PISO-CAN200/400 board number (0~7)
  bPort: [input] CAN port number (0~1 for PISO-CAN200 CAN card, 0~3 PISO-CAN400 CAN card)
  ***Handle:** [output] The pointer of the structure CANChannelStruct. This structure is defined as following

```
typedef struct
{
    BYTE bBoardNo;
    BYTE bPort;
} CANChannelStruct;
```

  bBoardNo**:** Save the parameter **bBoardNo** of the function CPM_InitPort to the struct CANChannelStruct.
  bPort**:** Save the parameter **bPort** of the function CPM_InitPort to the struct CANChannelStruct.

- **Return:**

  CPM_NoError
  CPM_DriverError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CPM_InitError

3.5.5 CPM_InitMaster

- **Description:**

  This function must be applied when configuring the CAN controller and initialize the master. After calling this function, the CAN receive interrupt will be enable. Then, the CANopen master can start to control the CANopen slave devices.

- **Syntax:**

```
WORD CPM_InitMaster(CANChannelStruct *Handle,
                    CPMConfigStruct *CanConfig,
                    WORD wTimeOut)
```

- **Parameter:**

***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

WTimeOut: [input] SDO timeout value. It is calculated since master sends a SDO message until slave responses the SDO message.

***CanConfig:** [input] The pointer of the structure CPMConfigStruct. This structure is defined as following

```
typedef struct cpmconfig
{
    BYTE AccCode[4];
    BYTE AccMask[4];
    BYTE BaudRate;
} CPMConfigStruct;
```

AccCode[4]: Acceptance code for CAN controller.
AccMask[4]: Acceptance mask for CAN controller.

BaudRate:

Parameter	1	2	3	4
Baud rate	10Kbps	20Kbps	50Kbps	125Kbps
Parameter	5	6	7	8
Baud rate	250Kbps	500Kbps	800Kbps	1Mbps

- **Return:**

CPM_NoError
CPM_DriverError
CPM_BoardNumberError
CPM_PortNumberError
CPM_ActiveBoardError
CPM_InitError
CPM_SoftResetError
CPM_SetACRError
CPM_SetAMRError
CPM_SetBaudRateError
CPM_ConfigError
CPM_EnableRxIrqFailure
CPM_MasterInitErr
CPM_CreateThreadErr
CPM_ResumeThreadErr

3.5.6 CPM_ShutdownMaster

- **Description:**

  Execute the function CPM_ShutdownMaster to release the resource used by the CANopen master program.

- **Syntax:**

  WORD CPM_ShutdownMaster (CANChannelStruct *Handle)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

- **Return:**

  CPM_NoError
  CPM_DriverError
  CPM_ActiveBoardError
  CPM_SuspendThreadErr

3.5.7 CPM_AddNode

- **Description:**

  The function CPM_AddNode can add a CANopen slave with Node ID, bNodeID, into the master node list. The added node can be removed from the master node list by the function CPM_RemoveNode. By the way, all of the added nodes are removed automatically when the function CPM_ShutdownMaster is used.

- **Syntax:**

  WORD CPM_AddNode(CANChannelStruct *Handle, BYTE bNodeID)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

  BNodeID: [input] Slave device Node-ID (1~127).

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CPM_ConfigError
  CPM_TransmitBufferLocked
  CPM_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeAddErr
  CPM_NodeIDDoesExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_Timeout

3.5.8 CPM_RemoveNode

- **Description:**

  The function CPM_RemoveNode removes the slave with the Node-ID bNodeID from node list of the node manager. It requires a valid Node-ID, which has installed by the function CPM_AddNode before.

- **Syntax:**

  WORD CPM_RemoveNode(CANChannelStruct *Handle,
  BYTE bNodeID)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  BNodeID: [input] Slave device Node-ID (1~127).

- **Return:**

  CPM_NoError
  CPM_ActiveBoardError
  CPM_MasterInitErr
  CPM_NodeRemoveErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange

3.5.9 CPM_NMTChangeState

- **Description:**

  The function CMP_NMTChangeState is used to change the state of a slave. If the slave state is changed to reset node state by using this function, this slave node-id will be removed from node list.

- **Syntax:**

  WORD CPM_NMTChangeState(CANChannelStruct *Handle,
  BYTE bNodeID, BYTE bState)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

  BNodeID: [input] Slave device Node-ID (1~127). Set this parameter to 0 to indicate all slave devices.

  BState: [input] NMT command specifier.

  1: start
  2: stop
  128: enter PRE-OPERATIONAL
  129: Reset_Node
  130: Reset_Communication

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange

3.5.10 CPM_NMTGetState

- **Description:**

  The function CMP_NMTGetState can get the NMT state from slave.

- **Syntax:**

  WORD CPM_NMTGetState(CANChannelStruct *Handle,
  BYTE bNodeID, BYTE *bState)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

  BNodeID: [input] Slave device Node-ID (1~127).

  ***bState:** [output] The state of the slave.

  4/132: STOPPED

  5/133: OPERATIONAL

  127/255: PRE-OPERATIONAL

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_Timeout

3.5.11 CPM_NMTGuarding

- **Description:**

  Use the function CMP_NMTGuarding to set Guard Time and Life Time Factor of the specific slave with node-ID bNodeID. Then, the master will send the Guarding message to slave device automatically. If the master or slave device doesn't receive the Guarding message in the Node Life time (Node Life time = dwGuardTime * bLifeTimeFactor), both the master and slave will be triggered to send the EMCY message.

- **Syntax:**

```
WORD CPM_NMTGuarding(CANChannelStruct *Handle,
                     BYTE bNodeID,
                     WORD wGuardTime,
                     BYTE bLifeTimeFactor,
                     DWORD dwSendGuardTime)
```

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

  BNodeID: [input] Slave device Node-ID (1~127).

  WGuardTime: [input] Guard Time (1 ~ 65535).

  BLifeTimeFactor: [input] Life Time Factor (1 ~ 255).

  DwSendGuardTime: [input] The Guard message send from master.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist

CPM_NodeIdOverRange

CPM_CreateThreadErr

CPM_SlaveStateErr

CPM_Timeout

3.5.12 CPM_SDOReadData

- **Description:**

Before calling the function CPM_SDOReadDat, the function CPM_InitMaster is needed. The function CPM_SDOReadData is useful to the SDO upload from a specific slave. When users use this function, pass the salve device node-id, bNodeID, and upload mode into this function. When calling this function, the pre-define SDO communication object ID will be used to send out a SDO message.

The upload mode parameter bBlock of the function CPM_SDOReadData has two different kind of value. There are shown below.

1、 bBlock = 0: If the total data size does not exceed 4 bytes. The function CPM_SDOReadData will return the value CPM_Noerror. Then, the SDO transmission is finished. If the data size does exceed 4 bytes, it will return the value CPM_ReadSegment. In this case, users need to use the function CPM_SDOReadSegment to continue the SDO transmission.

2、 bBlock = 1: This function will return the value CPM_ReadBlock, and the SDO Block upload protocol is used. Therefore, users need to call the function CPM_SDOReadBlock to continue the SDO transmission.

- **Syntax:**

```
WORD CPM_SDOReadData(CANChannelStruct *Handle,
                     BYTE bNodeID, WORD wIndex,
                     BYTE bSubIndex, PacketStruct *pRData,
                     DWORD *Rsize, BYTE bBlock)
```

- **Parameter:**

***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

BNodeID: [input] Slave device Node-ID (1~127).

WIndex: [input] SDO index in the object dictionary.

BSubIndex: [input] SDO subindex in the object dictionary.

***pRData:** [output] SDO data respond from the specific slave device.

***Rsize:** [output] Total data size.

bBlock: [input] Using SDO Block protocol.

- ## Return:

CPM_NoError
CPM_BoardNumberError
CPM_PortNumberError
CPM_ActiveBoardError
CAN_ConfigError
CAN_TransmitBufferLocked
CAN_TransmitIncomplete
CPM_MasterInitErr
CPM_NodeIDDoesNotExist
CPM_NodeIdOverRange
CPM_SlaveStateErr
CPM_ReadSegment
CPM_ReadBlock
CPM_Timeout

3.5.13 CPM_SDOReadSegment

- **Description:**

  The method is called when the value CPM_ReadSegment is returned from the functions CPM_SDOReadData or CPM_SDOReadSegment. If SDO data size is more than 7 bytes, the CPM_SDOReadData or CPM_SDOReadSegment function will also return the value CPM_ReadSegment.

- **Syntax:**

```
WORD CPM_SDOReadSegment(CANChannelStruct *Handle,
                        BYTE bNodeID,
                        PacketStruct *pRData)
```

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  BNodeID: [input] Slave device Node-ID (1~127).
  ***pRData:** [output] Segment data uploaded from the slave device.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_ReadSegment
  CPM_Timeout

3.5.14 CPM_SDOReadBlock

- **Description:**

  When the parameter bBlock of the function CPM_SDOReadData is 1, the function CPM_SDOReadData will return the value CPM_ReadBlock. However, if users send or receive the SDO message whose data size is more then 127 segments (127 * 7 bytes), the function CPM_SDOReadBlock also returns the value CPM_ReadBlock. In these two cases, the function CPM_SDOReadBlock is needed.

- **Syntax:**

  WORD CPM_SDOReadBlock(CANChannelStruct *Handle,
  BYTE bNodeID, BYTE *pRData)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  BNodeID: [input] Slave device Node-ID (1~127).
  ***pRData:** [output] At most 127 * 7 bytes of block data to be uploaded.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_ReadBlock
  CPM_ReadDataError
  CPM_Timeout

3.5.15 CPM_SDOWriteData

- **Description:**

The function CPM_SDOWriteData can send out a SDO message to specific salve device. This procedure is also called download SDO message. The parameter bNodeID of the function CPM_SDOWriteData is used to point which slave device will receive this SDO message. Because the data length of each object dictionary's object is different, users need to use different kind of SDO protocol when reading the object dictionary's object. The function CPM_SDOWriteData has two kinds of SDO download mode. The SDO download mode is decided by the parameter bBlock, and these two modes are shown below:

1、bBlock = 0: If the total data size of the object dictionary's object does not exceed 4 bytes. The function CPM_SDOWriteData will return the value CPM_Noerror. Otherwise, it will return the value CPM_WriteSegment.

2、bBlock = 1: The SDO Block download protocol is used and the function CPM_SDOWriteData returns the value CPM_WriteBlock.

Calling CPM_SDOWriteData requires a master (CPM_InitMaster).

- **Syntax:**

```
WORD CPM_SDOWriteData(CANChannelStruct *Handle,
                      BYTE bNodeID, WORD wIndex,
                      BYTE bSubIndex, DWORD dwDataSize,
                      PacketStruct *pRData, BYTE *bpData,
                      BYTE bBlock)
```

- **Parameter:**

***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

bNodeID: [input] Slave device Node-ID (1~127).

wIndex: [input] The index value of the object dictionary's object.

bSubIndex: [input] The subindex value of the object dictionary's object.

dwDataSize: [input] Total data size to be written.

***pRData:** [output] SDO data respond from the specific slave device.

***bpData:** [input] The SDO Data which will be downloaded. (This parameter is useless when the parameter dwDataSize is over than 4 or the parameter bBlock is set to1).

bBlock: [input] SDO download mode.

- **Return:**

CPM_NoError
CPM_BoardNumberError
CPM_PortNumberError
CPM_ActiveBoardError
CAN_ConfigError
CAN_TransmitBufferLocked
CAN_TransmitIncomplete
CPM_MasterInitErr
CPM_NodeIDDoesNotExist
CPM_NodeIdOverRange
CPM_SlaveStateErr
CPM_WriteSegment
CPM_WriteBlock
CPM_DataSizeRangeErr
CPM_Timeout

3.5.16 CPM_SDOWriteSegment

- **Description:**

  When the function CPM_SDOWriteData returns the value CPM_WriteSegment, the function CPM_SDOWriteSegment must be called to continue the SDO segment download protocol. Afterwards, if the function CPM_SDOWriteSegment still returns the value CPM_WriteSegment, the function CPM_SDOWriteSegment must be used again.

- **Syntax:**

```
WORD CPM_SDOWriteSegment(CANChannelStruct *Handle,
                         BYTE bNodeID,
                         PacketStruct *pRData,
                         BYTE *bpData)
```

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127).
  ***pRData:** [output] SDO segment data responded from the slave device.
  ***bpData:** [input] The SDO Data which will be downloaded. The SDO data length can't exceed one segment.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange

CPM_WriteSegment

CPM_SlaveStateErr

CPM_Timeout

3.5.17 CPM_SDOWriteBlock

- **Description:**

  When the parameter bBlock of the function CPM_SDOWriteData is set to 1, the function CPM_SDOWriteData will return the value CPM_WriteBlock, and the function CPM_SDOWriteBlock must be used to continue the SDO block download protocol. If the data length of download SDO data is more then one block size defined by the function CMP_SDOWriteData, this function must be called again.

- **Syntax:**

  WORD CPM_SDOWriteBlock(CANChannelStruct *Handle,
  BYTE bNodeID, BYTE *pAckseq,
  BYTE *bpData)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127).
  ***pAckseq:** [output] Last segment sequence number that was received successfully during the last block download.
  ***bpData:** [input] The SDO Data which will be downloaded. The SDO data length can't exceed one block.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange

CPM_SlaveStateErr

CPM_WriteBlock

CPM_WriteDataError

CPM_DataSizeRangeErr

CPM_Timeout

3.5.18 CPM_SDOAbortTransmission

- **Description:**

  Call the function CMP_SDOAbortTransmission to cancel the SDO transmission. The parameter bNodeID is used to specify which SDO communication will be terminated between the master and the specific salve device. The function will not give any acknowledge to the master.

- **Syntax:**

  WORD CPM_SDOAbortTransmission(CANChannelStruct *Handle,
  BYTE bNodeID)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127).

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr

3.5.19 CPM_InstallPDO

- **Description:**

  After CPM_PDOMapping function, call the function CPM_InstallPDO to install a PDO object in the CANopen Master Library stack. If the slave device has defined the default PDO object, it will be installed when the function CPM_AddNode is called.

- **Syntax:**

```
WORD CPM_InstallPDO(CANChannelStruct *Handle, BYTE bNodeID,
                    BYTE TxRxType, DWORD dwCobId,
                    BYTE bTransmitType, WORD wInhibitime,
                    WORD wEventTimer, BYTE bEnableChannel)
```

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Salve device Node-ID (1~127).
  TxRxType: [input] PDO type (0: for receive PDO, 1: for transmit PDO).
  dwCobId: [input] COB-ID used by the PDO object.
  bTransmitType: [input] PDO transmission type (0 ~ 255).
  wInhibitime: [input] PDO inhibit time (0 ~ 65535 ms)(not used for RPDO).
  wEventTimer: [input] PDO event timer (0 ~ 65535 ms).
  bEnableChannel: [input] Number of mapped application objects in PDO (0 ~ 8).

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete

CPM_MasterInitErr

CPM_NodeIDDoesNotExist

CPM_NodeIdOverRange

CPM_AllPDOareUsing

CPM_PDOInstallErr

CPM_CobIdNotRxPDO

CPM_CobIdNotTxPDO

CPM_MappingEnableErr

CPM_SlaveStateErr

CPM_Timeout

3.5.20 CPM_MappingPDO

- **Description:**

  This function can set the PDO mapping objects. If users use this function to define the PDO mapping object with the PDO COB-ID that is not installed by using the function CPM_InstallPDO, this PDO mapping object will be useless.

- **Syntax:**

```
WORD CPM_MappingPDO(CANChannelStruct *Handle,
                    BYTE bNodeID, BYTE TxRxType,
                    DWORD dwCobId, BYTE bChannel,
                    BYTE *MappingData)
```

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127).
  TxRxType: [input] Receive or Transmit PDO parameter (0: Rx, 1: Tx).
  dwCobId: [input] COB-ID used by the PDO object.
  bChannel: [input] PDO mapping for the nth application object to be mapped (at most is 8).
  ***MappingData:** [input] 4 bytes data of mapped.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesExist
  CPM_NodeIdOverRange

CPM_AllPDOareUsing

CPM_PDOSetError

CPM_SlaveStateErr

CPM_Timeout

3.5.21 CPM_RemovePDO

- **Description:**

  The function CPM_RemovePDO can remove a TxPDO or RxPDO installed by the CPM_InstallPDO.

- **Syntax:**

  WORD CPM_RemovePDO(CANChannelStruct *Handle,
  BYTE bNodeID, DWORD dwCobId)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127).
  dwCobId: [input] COB-ID used by the PDO object.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesExist
  CPM_NodeIdOverRange
  CPM_PDOSetError
  CPM_PDORemoveErr
  CPM_PDODoesNotExist
  CPM_SlaveStateErr
  CPM_Timeout

3.5.22 CPM_WritePDO

- **Description:**

Call the function CPM_WritePDO to send out a PDO message to the specific salve device. Before using this function, users need to use the function CPM_InstallPDO to install the PDO object, and change the target slave device NMT state to the operational by using the function CPM_ChangeState. If users just want to change some part of the PDO data sent before, use the parameter bOffset to set the data byte position which need to be modify, and use the parameters *Data and bDataLen to point the data and data length which users want to modify.

- **Syntax:**

```
WORD CPM_WritePDO(CANChannelStruct *Handle,
                  DWORD dwCobId, BYTE *Data,
                  BYTE bOffset, BYTE bDataLen)
```

- **Parameter:**

***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
dwCobId: [input] COB-ID used by the PDO object.
***Data:** [input] Data pointer to point the PDO data.
bOffset: [input] The first PDO data byte position(0 ~ 7).
bDataLen: [input] data size of PDO Data (bDataLen + bOffset $\leqq$ 8).

- **Return:**

CPM_NoError
CPM_BoardNumberError
CPM_PortNumberError
CPM_ActiveBoardError
CAN_ConfigError
CAN_TransmitBufferLocked
CAN_TransmitIncomplete
CPM_MasterInitErr
CPM_PDODoesNotExist

3.5.23 CPM_RemotePDO

- **Description:**

  Use the function CPM_RemotePDO to send a remote request PDO message to the slave device.

- **Syntax:**

  WORD CPM_RemotePDO(CANChannelStruct *Handle,
  DWORD dwCobId, PacketStruct *pRData)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  dwCobId: [input] COB-ID used by the PDO object.
  ***pRData:** [output] The PDO message received from the remote salve device.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_PDODoesNotExist
  CPM_Timeout

3.5.24 CPM_ResponsePDO

- **Description:**

  When users set the salve device to some PDO transmission type, the slave will return the PDO message automatically if some event is triggered or the event timer is time up. Call the function CPM_ResponsePDO to read the PDO message which is received from the slave device and stored in the PDO software buffer. The buffer size of the PDO software buffer is 10 records. If the received PDO messages exceed 10 records without reading by the function CPM_ResponsePDO, the oldest PDO message will be covered by the newest one. These received PDO messages include the PDO messages produced by Event Driven, Timer Driven, Remotely requested.

- **Syntax:**

  WORD CPM_ResponsePDO(CANChannelStruct *Handle,
  PacketStruct *pRData)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  ***pRData:** [output] The PDO data responded from the slave device.

- **Return:**

  CPM_NoError
  CPM_ActiveBoardError
  CPM_NoResponse

3.5.25 CPM_ ResPDOCount

- **Description:**

  To get the message count of PDO message from PDO buffer.

- **Syntax:**

  WORD CPM_ResponsePDO(CANChannelStruct *Handle)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

- **Return:**

  The un-read PDO message count

3.5.26 CPM_SendSYNC

- **Description:**

  Use the function CMP_SendSYNC to send a SYNC message with specific COB-ID cyclically. If the parameter dwSyncCycle is 0, the SYNC message will be stopped.

- **Syntax:**

  WORD CPM_SendSYNC(CANChannelStruct *Handle,
  DWORD dwCobid, DWORD dwSyncCycle)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  dwCobId: [input] COB-ID used by the SYNC object.
  dwSyncCycle: [input] SYNC message transmission period.

- **Return:**

  CPM_NoError
  CPM_ActiveBoardError
  CPM_MasterInitErr
  CPM_CreateThreadErr

3.5.27 CPM_ReadEMCY

- **Description:**

  When the salve detects an internal error, it will send an emergency message. The master library will receive this emergency message and store it in the software buffer. The function CPM_ReadEMCY can return the emergency message stored in the buffer.

- **Syntax:**

  WORD CPM_ReadEMCY(CANChannelStruct *Handle,
  PacketStruct *pRData)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.

  ***pRData:** [output] Emergency message data stored in the buffer.

- **Return:**

  CPM_NoError
  CPM_ActiveBoardError
  CPM_MasterInitErr
  CPM_EMCYFIFOisEmpty

3.5.28 CPM_WriteDO

- **Description:**

  Use this function to output index 0x6200 with SDO protocol.

- **Syntax:**

  WORD CPM_ReadEMCY(CANChannelStruct *Handle, BYTE bNodeID
  , BYTE channel, BYTE value)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127)..
  channel: [input] Output DO channel. The parameter is from 1.
  value: [input] Output data.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_Timeout

3.5.29 CPM_WriteAO

- **Description:**

  Use this function to output index 0x6411 with SDO protocol.

- **Syntax:**

  WORD CPM_ReadEMCY(CANChannelStruct *Handle, BYTE bNodeID
  , BYTE channel, WORD value)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127)..
  channel: [input] Output AO channel. The parameter is from 1.
  value: [input] Output data.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_Timeout

3.5.30 CPM_ReadDI

- **Description:**

  Use this function to read index 0x6000 with SDO protocol.

- **Syntax:**

  WORD CPM_ReadEMCY(CANChannelStruct *Handle, BYTE bNodeID
  , BYTE channel, BYTE *value)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127)..
  channel: [input] Read DI channel. The parameter is from 1.
  ***value:** [input] Input data.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_Timeout

3.5.31 CPM_ReadAI

- **Description:**

  Use this function to read index 0x6401 with SDO protocol.

- **Syntax:**

  WORD CPM_ReadEMCY(CANChannelStruct *Handle, BYTE bNodeID
  , BYTE channel, WORD *value)

- **Parameter:**

  ***Handle:** [input] CAN channel handle pointer returned from the function CPM_InitPort.
  bNodeID: [input] Slave device Node-ID (1~127)..
  channel: [input] Read AI channel. The parameter is from 1.
  ***value:** [input] Input data.

- **Return:**

  CPM_NoError
  CPM_BoardNumberError
  CPM_PortNumberError
  CPM_ActiveBoardError
  CAN_ConfigError
  CAN_TransmitBufferLocked
  CAN_TransmitIncomplete
  CPM_MasterInitErr
  CPM_NodeIDDoesNotExist
  CPM_NodeIdOverRange
  CPM_SlaveStateErr
  CPM_Timeout

4 Demo Programs for Windows

After installing the CANopen Master Library correctly, users can see the demo programs on the folder DAQPro\PISO-CAN\CANopen\Demo. We provide the demos developed by three kinds of program development tools. These tools are BCB, VC++ and VB. When users open the Demo folder, the file architecture is shown as follows.

```
|--\Demo                    → demo program
   |--\BCB_Demo             →  for Borland C++ Builder 3
   |
   |--\VB_Demo              →  for Visual Basic 6
   |
   |--\VC_Demo              →  for Visual C++ 6
```

If users want to develop their own CANopen master programs by BCB, VC++ or VB, some library files are needed. The library files for the different program development tools locate in the different folder. They are shown below.

```
|--\Driver                  → CANopen driver
   |--\PISOCANCPM.dll       →  CANopen dynamic link library
   |
|--\Library                 → CANopen library
   |--\BCB                  →  for Borland C++ Builder 3
   |   |--\CPM.h            →  CANopen header file
   |   |--\BCBPISOCANCPM.lib →  CANopen static library
   |
   |--\VB                   →  for Visual Basic 6
   |   |--\CPM.bas          →  CANopen declaration file
   |
   |--\VC                   →  for Visual C++ 6
   |   |--\CPM.h            →  CANopen header file
   |   |--\PISOCANCPM.lib   →  CANopen static library
```

A brief of the demo programs

These demo programs are developed for demonstrating how to use the CANopen Master Library to apply the general CANopen communication protocol. These demo programs. These demos provide the SDO, PDO, NMT communication applications. Each communication protocol is achieved by using different functions of CANopen Master Library. The relationship between CANopen Master Library functions and CANopen communication protocols are display in the following description.

NMT Services: CPM_NMTChangeState, CPM_NMTGuarding

SDO Services: CPM_SDOReadData, CPM_SDOReadSegment,
CPM_SDOWriteData, CPM_SDOWriteSegment

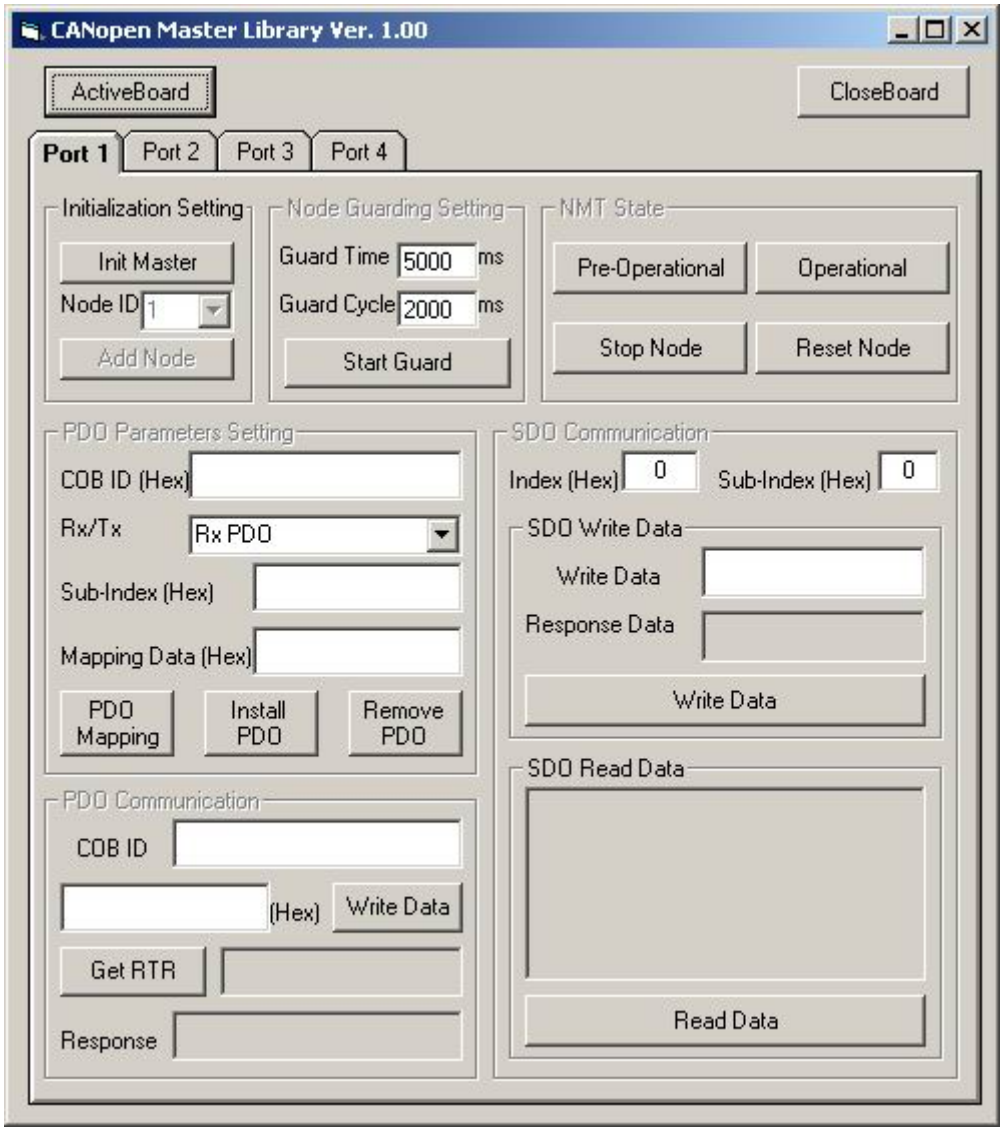
PDO Services: CPM_InstallPDO, CPM_MappingPDO,
CPM_RemovePDO, CPM_WritePDO,
CPM_RemotePDO, CPM_ResponsePDO

4.1 Brief Introduction of the demo programs

This section displays the demo program contour and the operation procedure for the demo programs. Because the BCB demo is similar with the VB demo, only the VB demo and VC++ demo will be introduced.

BCB and VB Demo:

When the demo starts to run, the demo user interface is shown below.



When the ActiveBoard button is clicked, the word of ActiveBoard will be changed into the bold word.

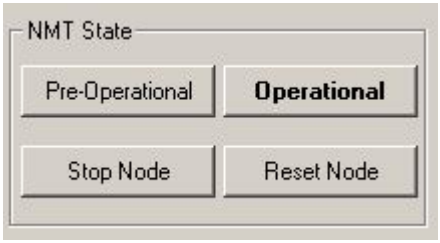
When the CloseBoard button is clicked, the word of CloseBoard will be changed into the bold word, and the word of ActiveBoard will become regular one.

After click ActiveBoard button to activate the CAN card, select one CAN port which users want to initialize. Then, click the “Init Master” button.

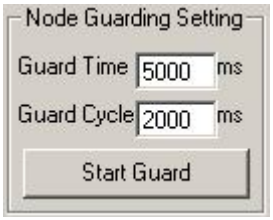
If master initialization is successful, the word of “Init Master” will be changed into the word of Shutdown. Then, the “Node ID” combo box and “Add Node” button will enable. Click Shutdown button will execute the function CPM_ShutdownMaster, and stop all the function of the CANopen master. At the same time, the word of Shutdown will return to the word of “Init Master”.

Select CANopen slave node id in the “Node ID” combo box, and click “Add Node” button, then the word of “Add Node” will be changed into the word of Remove. Click Remove button will remove the node selected previously. If users change the Node ID in the combo box, the demo will check if this node has been added before. If yes, the button with “Remove” word is shown. If the Node ID is new one, the button with “Add Node” words will be indicated.

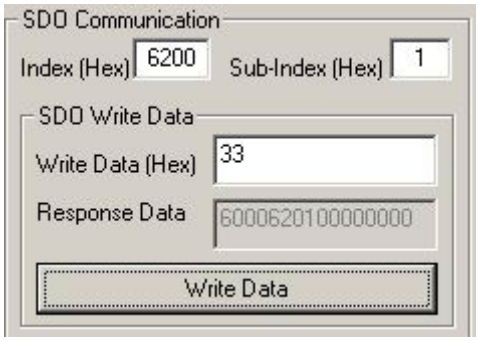
After adding the slave into slave node, the state of the slave will be shown with bold word on the state button in the NMT State frame. If users want to change the state, please click the button which you want in the NMT State frame.



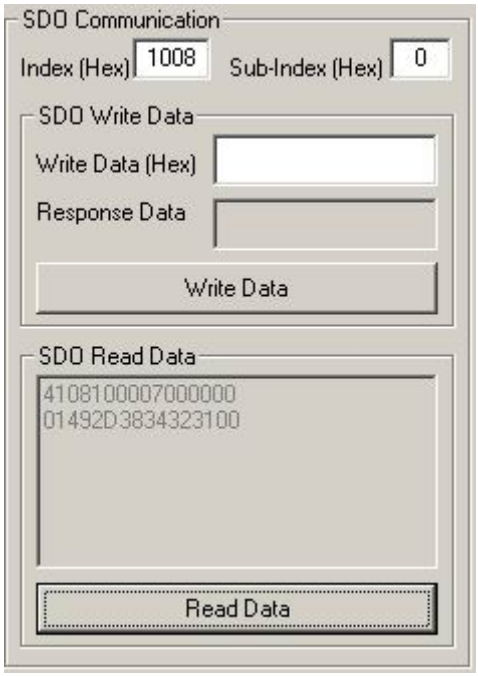
If users want to use the Node Guarding Protocol, input guard time and guard cycle. Then, click “Start Guard” button to apply the Node Guarding Protocol. The unit of guard time and guard cycle is million second.



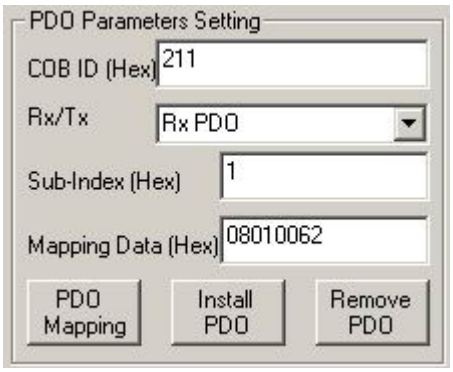
The SDO Communication frame includes the two parts. They are SDO Write Data and SDO Read Data. The SDO Write Data frame can write data to the salve node by SDO protocol. For example input 6200 for Index, 1 for sub-Index, 33 for written Data, and click “Write Data” button, then the value 0x33 will be written to the object with index 6200 and sub-index 1 of the slave node. Afterwards, the salve node response data will be shown on the “Response Data” text box.



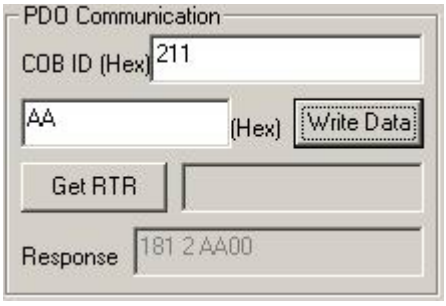
SDO Read Data frame can read data from CANopen slave node by SDO protocol. For example input 1008 for Index, 0 for sub-Index, and click the “Read Data” button to send data to salve node. Then, the slave response data will be shown on the “Read Data” text box.



PDO Parameters Setting frame includes the functions CPM_MappingPDO, CPM_InstallPDO, and CPM_RemovePDO. Input COB ID, Rx/Tx type, sub-Index value, and mapping data. Then, click PDO Mapping button to set the PDO mapping parameters. Afterwards, click “Install PDO” button will execute the function CPM_InstallPDO with the value set in the COB ID text, Rx/Tx combo box, and sub-Index text. Click “Remove PDO” button will execute the function CPM_RemovePDO to remove the PDO object with the ID set in the COB ID text.

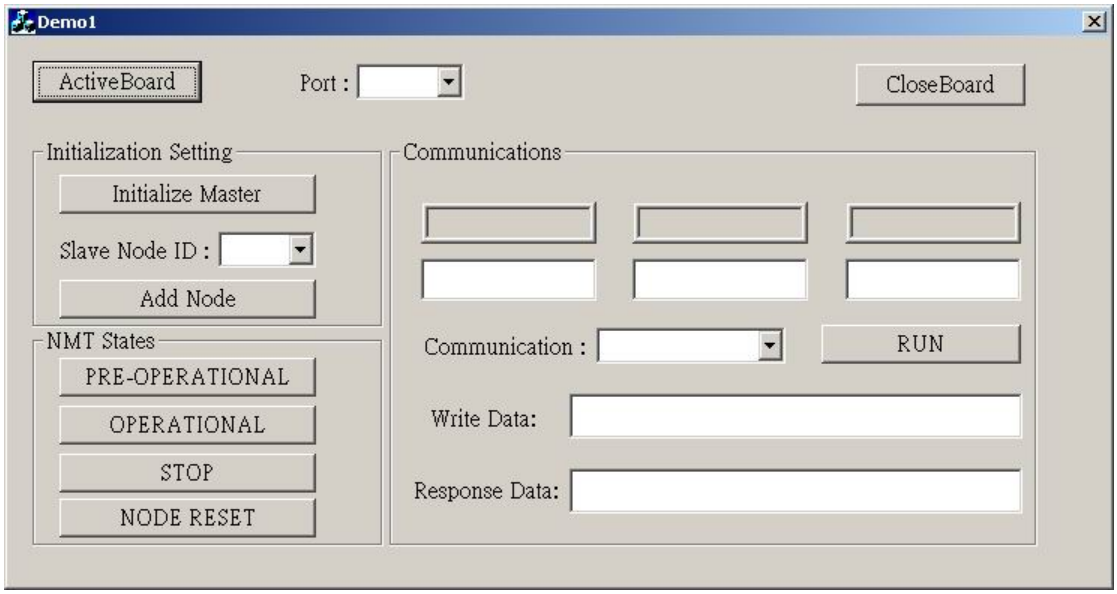


The PDO Communication frame includes the functions for PDO communication. Input COB ID 211 and data AA, then click “Write Data” button. It will write the data to the slave by PDO protocol. The response data will show in the “Response” text box if some data are responded from the slave device. “Get RTR” button can get remote data from the slave if the COB ID is set to support the remote request transmit mode.



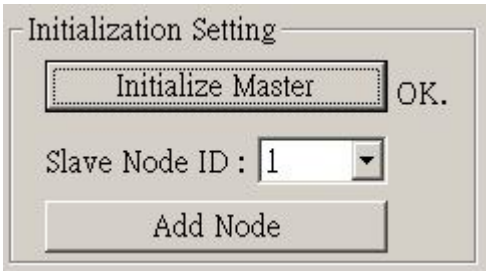
VC Demo:

When the demo starts to run, the demo user interface is shown below.

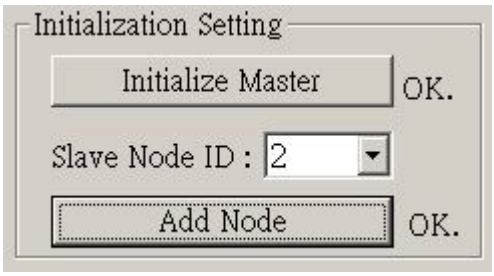


When the ActiveBoard button is clicked, the “OK” word will be shown on the right hand side of the button. If click CloseBoard button, the program will exit.

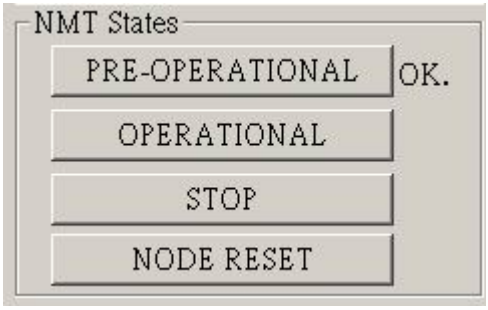
Select one CAN port and click “Initialize Master” button. If master initialization is successful, the word “OK” will be displayed on the right hand side of the button.



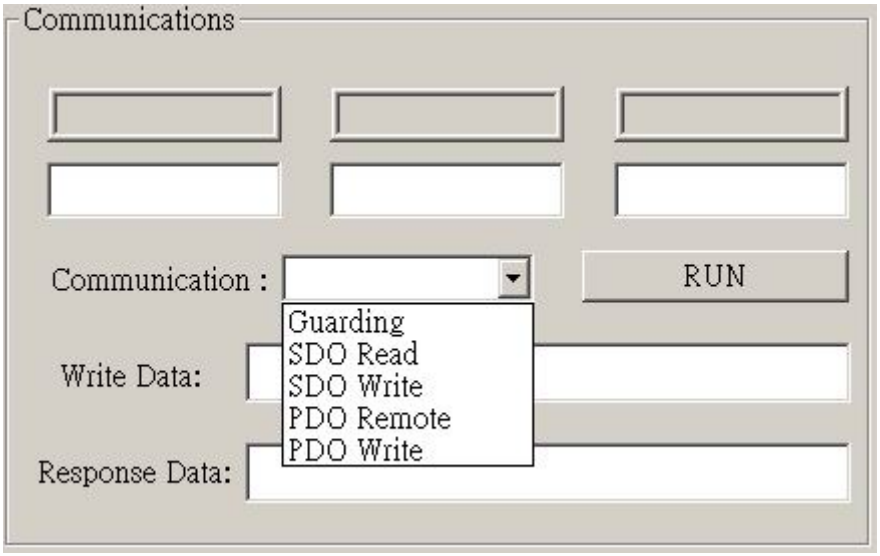
Select node id and click “Add Node” button. The word “OK” will also be shown on the side of the “Add Node” button.



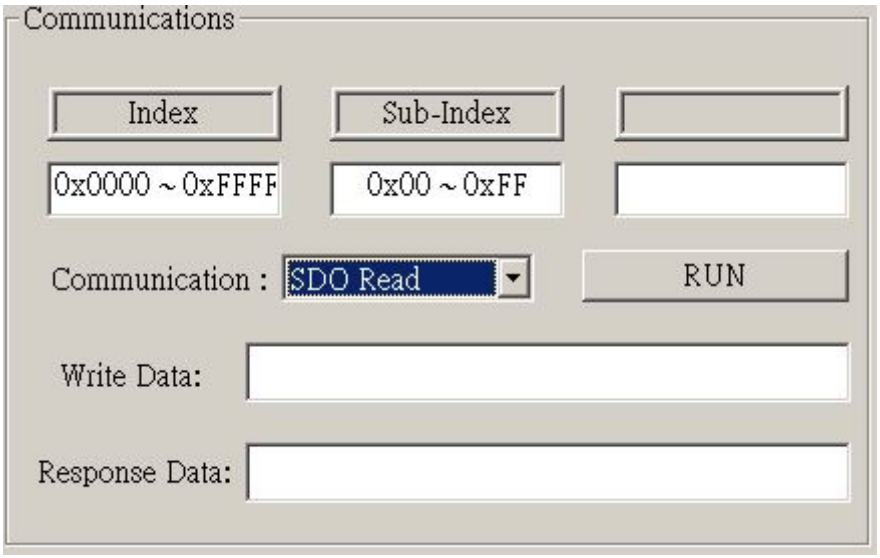
After adding a CANopen slave, the slave NMT state will be shown by the word “OK” on the right hand side of the state button. If users want to change the state, select and click the state button, which will be set to the node.



There are five communication modes to be selected in the Communication combo box.



Here, only the SDO Read mode is taken for an example. If users select the SDO Read mode, the word Index and Sub-Index will be shown in the block. In this case, the Communication frame can help for reading data from slave node by SDO protocol.



Input the value for index and sub-index, and click “Run” button to execute the function CPM_SDOReadData. The response data will be displayed in the “Response Data” text box.