CANopen Slave Device

CAN-2000C Series

Communication User's Manual

Warranty

Without contrived damage, all products manufactured by ICP DAS are warranted in one year from the date of delivery to customers.

Warning

ICP DAS revises the manual at any time without notice. However, no responsibility is taken by ICP DAS unless infringement act imperils to patents of the third parties.

Copyright

Trademark

The brand name ICP DAS as a trademark is registered, and can be used by other authorized companies.

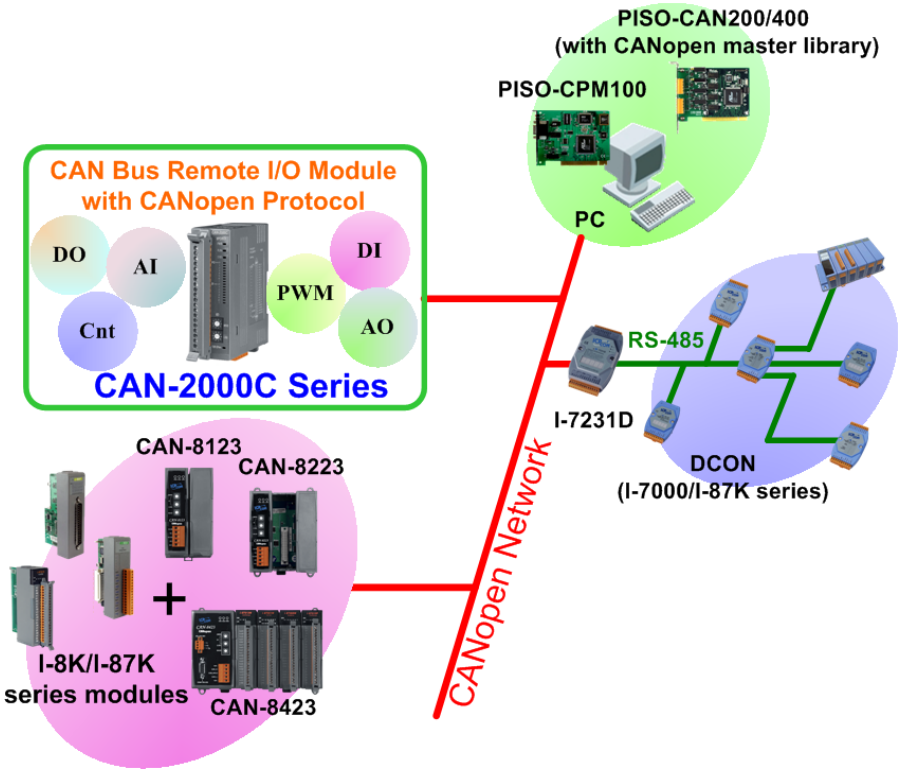
Contents

1 Introduction

1.1 Overview

CANopen, a kind of communication protocols, is an intelligent field bus (CAN bus). It has been developed as a standard embedded network with a high flexible configuration. It provides a standard communication protocol transmitting real-time data in PDO (**P**rocess **D**ata **O**bjects), configuration data in SDO (**S**ervice **D**ata **O**bjects), and network management data (NMT message, and Error Control), even supports the special functions (Time Stamp, Sync message, and Emergency message). Nowadays, CANopen is used on many applications and in specific fields, such as medical equipment, off-road vehicles, maritime electronics, public transportation, automation and so on.

The CAN-2000C series CANopen slave modules are specially designed for the slave device of the CANopen protocols. All of these CAN-2000C series modules follow the CANopen Spec DS-301 V4.02 and DS-401 V2.1, and supply a great deal of features to users, such as dynamic PDO, EMCY object, error output value, SYNC cyclic and acyclic and so forth. The general application for the CAN-2000C series CANopen slave devices architecture is as follows.



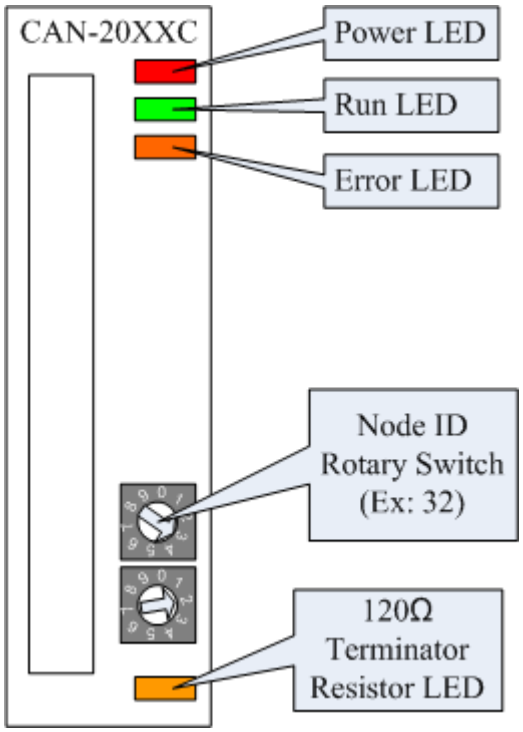
1.2 CAN-2000C General Hardware Specifications

- Built-in Watchdog
- PWR LED, RUN LED, and ERR LED
- Terminal resister LED
- 120Ω terminator resister selected by DIP-switch
- CAN bus interface: ISO 11898-2, 5-pin screw terminal with removable terminal block.
- Unregulated from +10V_{DC} ~ +30V_{DC}
- Operating Temperature:-25°C ~ +75°C
- Storage Temperature:-30°C ~ +80°C
- Humidity: 10% ~ 90% RH, non-condensing.

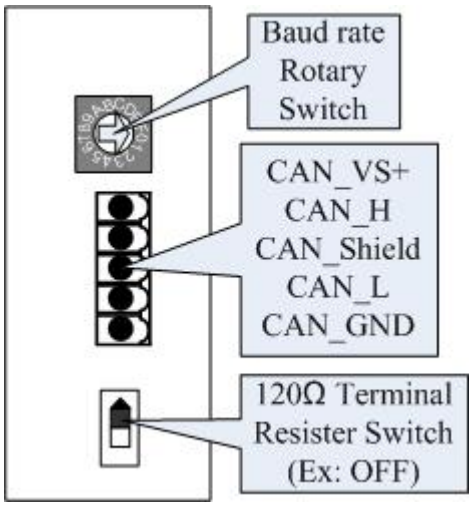
1.3 CAN-2000C Common Features

- NMT: Slave
- Error Control: Node Guarding and Heartbeat Producer
- Node ID: Setting by Rotary Switch (1 ~ 99)
- No of SDOs: 1 server, 0 client
- No. of PDOs: 10Rx, 10Tx
- PDO Modes: Event-triggered, remotely requested, cyclic and acyclic SYNC
- Support dynamic PDO mapping
- Emergency Message variable
- Support “Save” and “Load” command, can save I/O setting or restore I/O default setting.
- CANopen Version: DS-301 v4.02
- Device Profile: DS-401 v2.1
- Baud Rate selected by rotary switch from 0~7 (10K, 20K, 50K, 125K, 250K, 500K, 800K and 1M bps).
- Status LED: Power LED, RUN LED, and ERR LED indicators

2 Hardware Specification



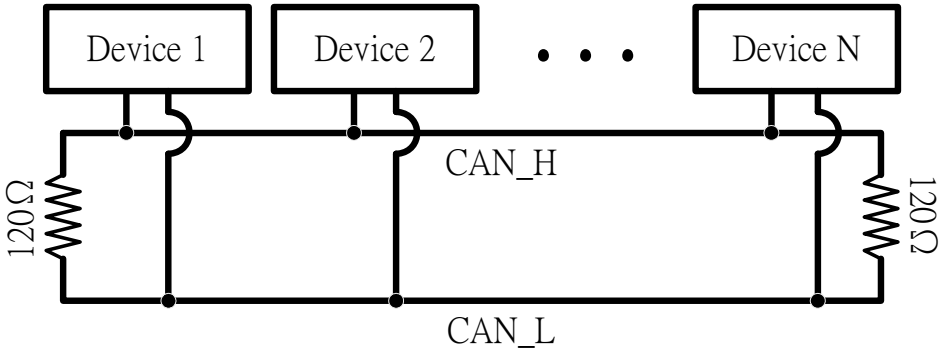
(Top View)



(Bottom View)

2.1 Wire Connection

In order to minimize the reflection on the CAN bus line, the CAN bus line has to be terminated at both ends by two terminal resistances as shown in the following. According to the ISO 11898-2 spec, each terminal resistance is 120Ω (or other between 108Ω~132Ω). The length related resistance has to reach 70 mΩ/m. At this circumstance, users would better check the resistances of the CAN bus before installing a new CAN network.



Moreover, to minimize the voltage drop, value of the terminal resistance must be higher than the one defined in the ISO 11898-2. The following table is for users' reference.

Bus Length (meter)	Bus Cable Parameters		Terminal Resistance (Ω)
	Length Related Resistance (mΩ/m)	Cross Section (Type)	
0~40	70	0.25(23AWG)~ 0.34mm^2(22AWG)	124 (0.1%)
40~300	< 60	0.34(22AWG)~ 0.6mm^2(20AWG)	127 (0.1%)
300~600	< 40	0.5~0.6mm^2 (20AWG)	150~300
600~1K	< 20	0.75~0.8mm^2 (18AWG)	150~300

In the CAN-2000C series module, the 120Ω terminal resistance is supplied as a standard accessory. In the following figure, the position SW1 allowed to build in a terminal resistance.

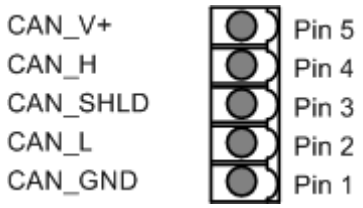
The figure indicates two conditions, "Disable (Up)" and "Enable (Down)". And if the terminal resister is enabled, the terminal resister LED will on.

The bus length determines the CAN bus baud rate. In the following the table provides users a relationship between the baud rate and the bus length.

Baud rate (bit/s)	Max. Bus length (m)
1 M	25
800 K	50
500 K	100
250 K	250
125 K	500
50 K	1000
20 K	2500
10 K	5000

Note: When the bus length is greater than 1000m, the bridge or repeater devices may be needed.

The pin descriptions of the CAN bus connectors on the CAN-2000C are shown below.



Pin No.	Signal	Description
1	CAN_GND	Ground (0V)
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_SHLD	Optional CAN Shield
4	CAN_H	CAN_H bus line (dominant high)
5	CAN_V+	CAN external positive supply

2.2 Power LED

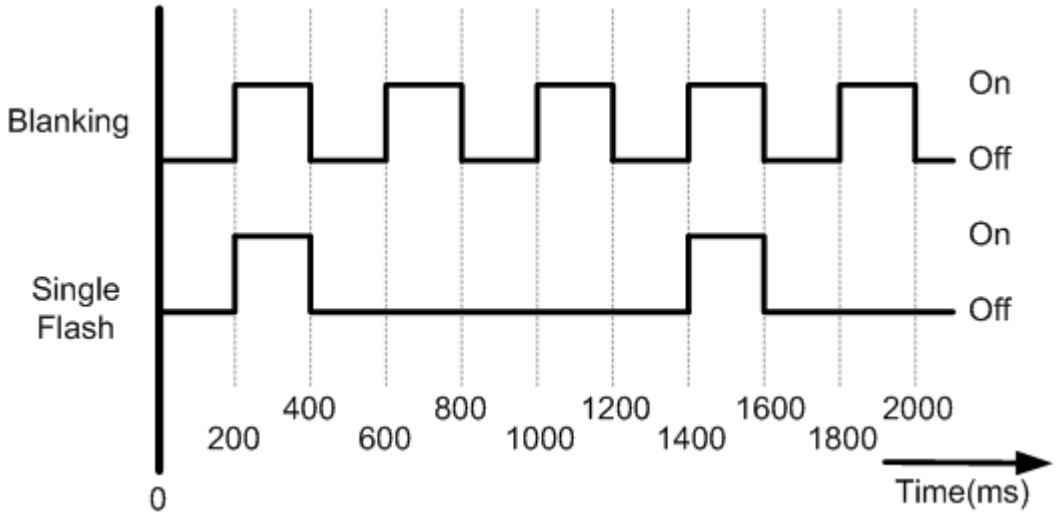
The CAN-2000C series products need 10 to 30 VDC power supplies. Under a normal connection, a good power supply and a correct voltage selection, as the unit is turned on, the LED will light up in red. If it can't work, please check with local agents or resellers for more help.

2.3 CANopen Status LED

Each one CAN-2000C module has two LED indicators. One is the Error LED (lighting in orange) and the other one is the RUN (Performing) LED (lighting in green). The Error LED and the Run (Performing) LED information are presented in the CANopen specifications. When the CANopen communication carries out, these indicators will glitter in different time. The following descriptions will show meanings of the glittering signal as these indicators are being triggered.

2.3.1 The RUN LED

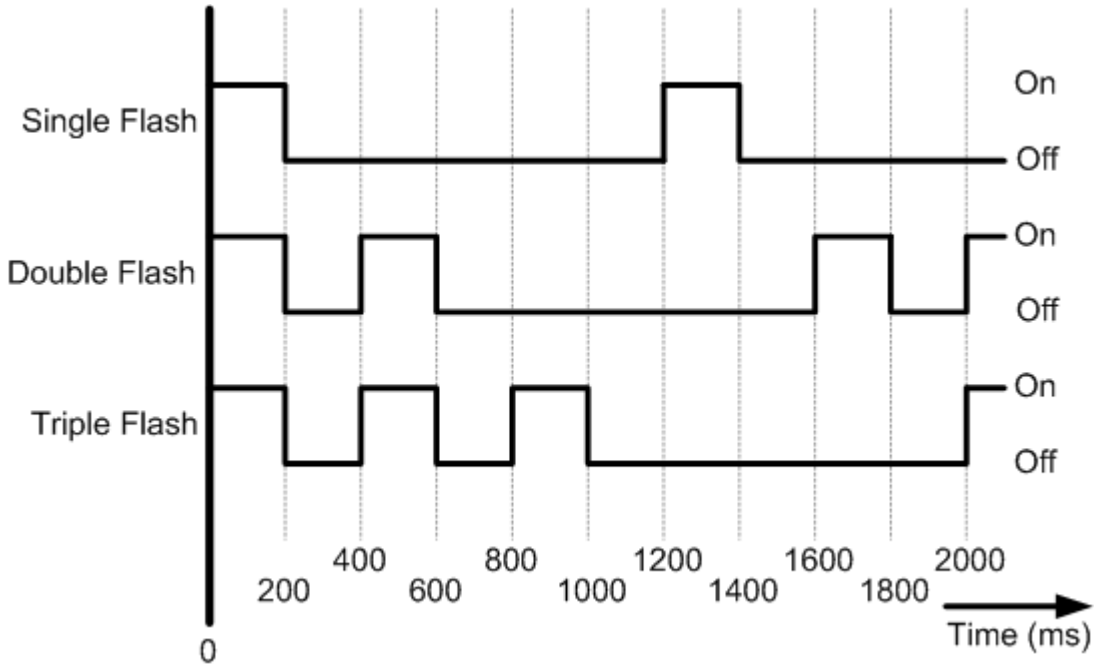
The RUN LED relates to the physical mechanism on the CANopen that will be discussed later. The data state and the signal state description are respectively shown in the following figure and table.



No.	Signal	State	Description
1	No Light	Non-operation	Malfunction or Power Supply /Connection not ready
2	Single Flash	Stopped	The device is in Stopped state
3	Blinking	Pre-operation	The device is in the pre-operational state
4	Continuing Light	Operation	The device is in the operational state

2.3.2 The ERR LED

The ERR LED relates to the state of missing messages at the CAN physical layer (These missing messages might be SYNC or Guard messages). The data state and the signal state description are respectively shown in the following figure and table.

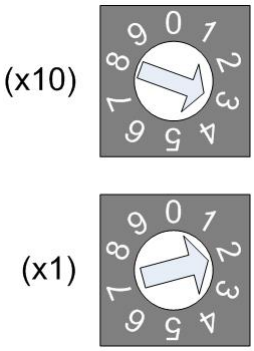


No.	Signal	State	Description
1	No Light	No error	The device is in working condition.
2	Single Flash	Error Reminding when Warning Level is Reached	At least one of the error counters of the CAN controller has reached or exceeded the warning level (too many error frames).
3	Double Flash	Error Reminding when Events happen.	A guard event (NMT-Slave or NMT-master).
4	Continuing Light	Bus Off	The CAN controller is in a bus off condition.

Note: If several errors occur at the same time, the most severe error will have high priority to show its signal first. For example, if NMT Error (No. =3) and Bus Off Error (No. =4) occur, the Bus off error signal will indicate.

2.4 The Node ID & the Baud rate Rotary Switch

The rotary switches for node ID configure the node ID of CAN-2000C module. These two switches are for the tens digit and the units digit of node ID. The node ID value of this demo picture is 32.



Node ID rotary switch

The rotary switch for baud rate handles the CAN baud rate of the CAN-2000C module. The relationship between the rotary switch value and the practical baud rate is presented in the following table.

Baud rate rotary switch

Rotary Switch Value	Baud rate (K BPS)
0	10
1	20
2	50
3	125
4	250
5	500
6	800
7	1000

2.5 Module Support

The CAN-2000C series modules include many kinds of DI, DO, AI and AO types series modules. Please refer to the web to get more detail information. The web site http://www.icpdas.com/products/Remote_IO/can_bus/can-2000.htm

3 CANopen Application

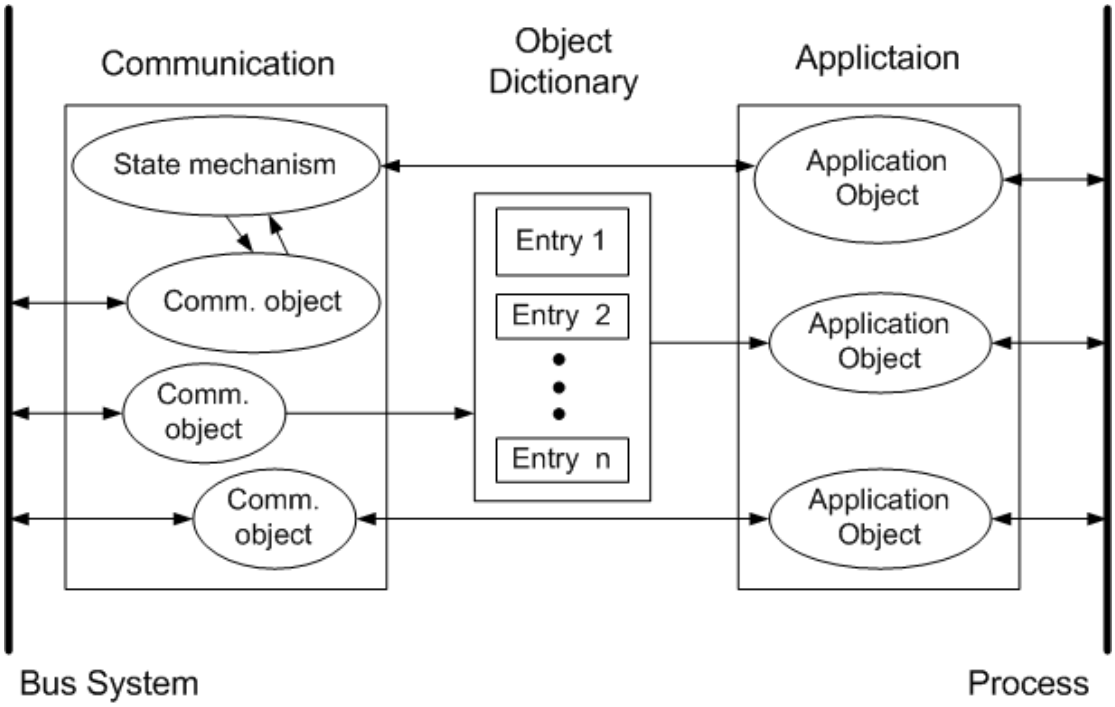
The CANopen is a kind of network protocols evolving from the CAN bus, used on car control system in early days, and has been greatly used in various applications, such as vehicles, industrial machines, building automation, medical devices, maritime applications, restaurant appliances, laboratory equipment & research.

3.1 CANopen Introduction

CANopen provides not only the broadcasting function but also the peer-to-peer data exchange function between every CANopen node. The network management function instructed in the CANopen simplifies the program design. In addition, users can also implement and diagnose the CANopen network, including network start-up, and error management by standard mechanisms (CANopen device), i.e. the CANopen device can effectively access the I/O values and detect node states of other devices in the same network. Generally, a CANopen device can be modeled into three parts.

- Communication
- Object Dictionary
- Application program

The functions and general concepts for each part are shown as follows.



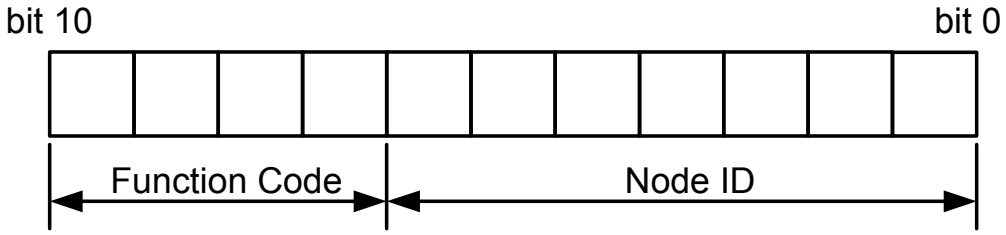
Communication

The communication part provides several communication objects and appropriate functionalities to transmit CANopen messages via the network structure. These objects include PDO (Process Data Object), SDO (Service Data Object), NMT (Network Management Objects), SYNC (Synchronous Objects)…etc. Each communication object has its relative communication model and functionality. For example, the communication objects for accessing the device object dictionary is SDO, using the Client/Server structure as its communication model (section 3.2). Real-time data or I/O values can be accessed quickly without any protocol by means of PDO communication objects. The PDO's communication model follows the Producer/Consumer structure. It is also named the Push/Pull model (section 3.3). NMT communication objects are used for controlling and supervising the state of the nodes in the CANopen network, and it follows a Master/Slave structure (section 3.5). No matter which kind of communication object is used, the transmitted message will comply with the data frame defined in the CAN 2.0A spec. Generally, it looks like the following figure.

ID	RTR	Data Length	8-byte Data

11-bit data is limited in the ID field. It is useful in the arbitration mechanism. The RTR, limited in 1-bit data, is used for remote-transmitting requests as the value is set to 1. The data length, limited in 4-bit data, shows the valid data number stored in the 8-byte data field. The last field, 8-byte data, is applied to store the message data.

In the CANopen specifications the 4-bit function code and 7-bit node ID are assumed to combine the 11-bit ID of CAN message, and named the communication object ID (COB-ID). The COB-ID structure is displayed below.



The COB-IDs are used for recognizing where the message comes from or where the message is sent to, as well deciding the priority of the message transmission around node network. According to the arbitration mechanism rule of the CAN bus, the CAN message with the lower COB-ID will get the higher priority to be transmitted. In the CANopen specifications some COB-IDs are reversed for specific communication objects, and can't be defined arbitrarily by users. The following list shows these reversed COB-IDs.

Reversed COB-ID (Hex)	Used by object
0	NMT
1	Reserved
80	SYNC
81~FF	EMERGENCY
100	TIME STAMP
101~180	reversed
581~5FF	Default Transmit-SDO
601~67F	Default Receive-SDO
6E0	reversed
701~77F	NMT Error Control
780~7FF	reversed

In addition, the other COB-IDs shown in the following table can be used if necessary.

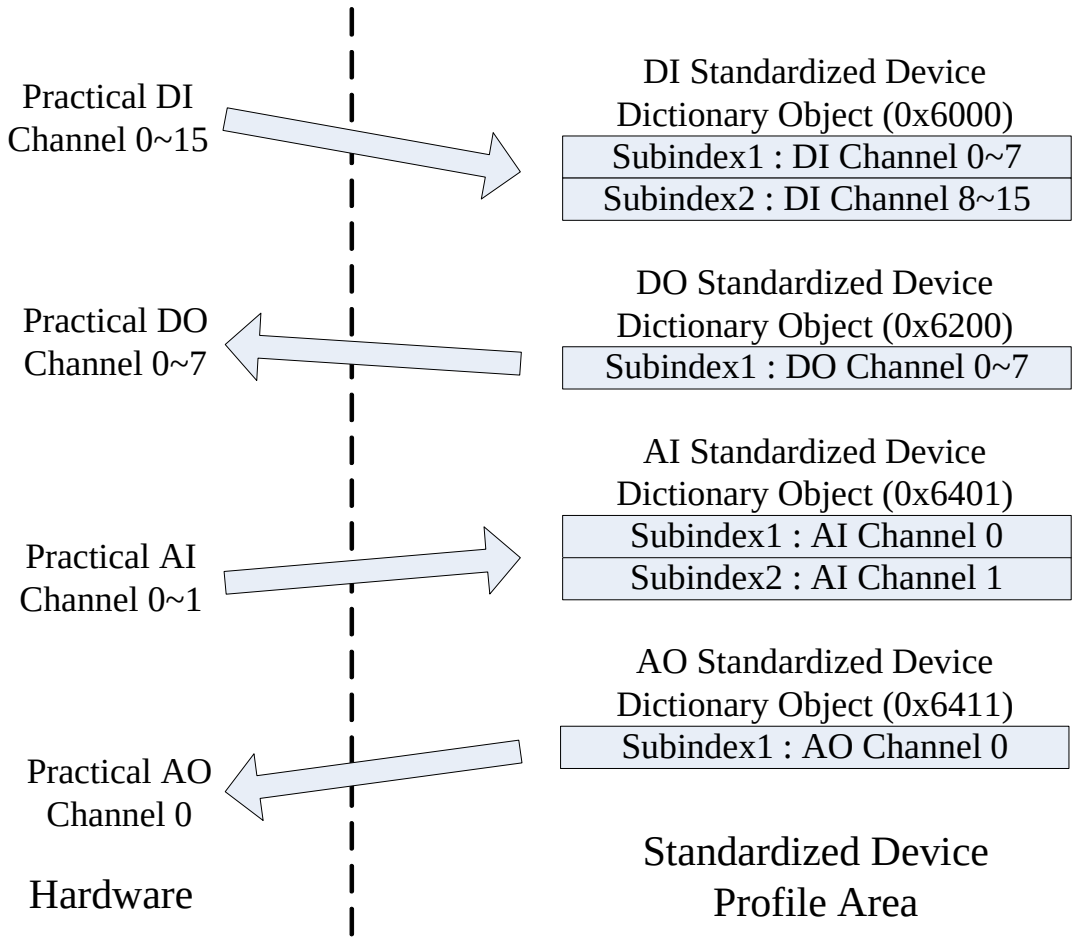
(Bit10~Bit7) (Function Code)	(Bit6~Bit0)	Communication object Name
0000	0000000	NMT
0001	0000000	SYNC
0010	0000000	TIME STAMP
0001	Node ID	EMERGENCY
0011/0101/0111/1001	Node ID	TxPDO1/2/3/4
0100/0110/1000/1010	Node ID	RxPDO1/2/3/4
1011	Node ID	SDO for transmission (TxSDO)
1100	Node ID	SDO for reception (RxSDO)
1110	Node ID	NMT Error Control

Object Dictionary

The object dictionary collects a lot of important information which can affect device's reaction, such as the data accessing through I/O channels, the communication values and the network states. Essentially, the object dictionary consists of a group of entry objects, and these entries can be accessed via the node network in a pre-defined method. Each object entry within the object dictionary has its own function, for example communication parameters, device profile, data type (ex. 8-bit Integer, 8-bit unsigned…), and access type (read only, write only …). All of them are addressed in a 16-bit index and an 8-bit sub-index. The overall profile of the standard object dictionary is shown below.

Index	Object
0x0000	Reserved
0x0001 - 0x001F	Static Data Types
0x0020 - 0x003F	Complex Data Types
0x0040 - 0x005F	Manufacturer Specific Complex Data Types
0x0060 - 0x007F	Device Profile Specific Static Data Types
0x0080 - 0x009F	Device Profile Specific Complex Data Types
0x00A0 - 0x0FFF	Reserved for further use
0x1000 - 0x1FFF	Communication Profile Area
0x2000 - 0x5FFF	Manufacturer Specific Profile Area
0x6000 - 0x9FFF	Standardized Device Profile Area
0xA000 - 0xBFFF	Standardized Interface Profile Area
0xC000 - 0xFFFF	Reserved for further use

About the standardized device profile area, it stores the DI, DO, AI and AO channels of CANopen, especially the entries with indexes 0x6000, 0x6200, 0x6401, and 0x6411. When the CANopen device obtains the input value, these values will be stored in the 0x6000 and 0x6401 indexes. Furthermore, the values stored in the 0x6200 and 0x6411 indexes will also output to the DO and AO channels. The example of basic concept is presented as follows.

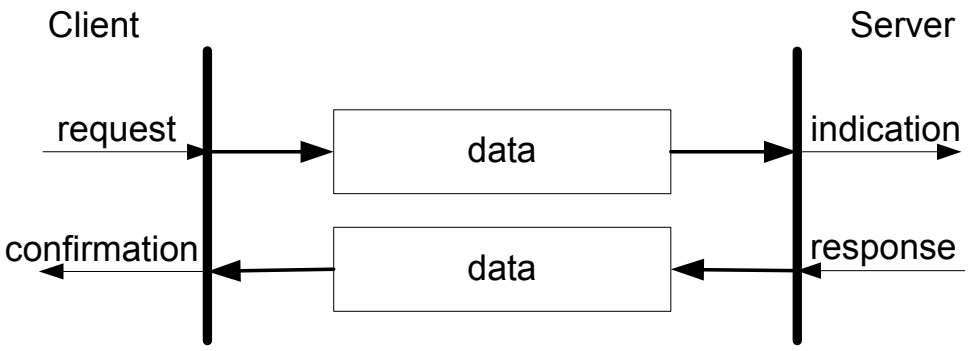


Application

The application objects control all of the device functions, related to the interaction with the process environment. It's just like a medium between the object dictionary and practical process, such as the analog I/O, digital I/O….

3.2 SDO Introduction

In order to access the entries in a device object dictionary, service data objects (SDOs) are provided. By means of the SDO communication method, a peer-to-peer communication bridge between two devices is established, and its transmission follows the client-server relationship. The general concept is shown in the figure below.



The SDO has two kinds of the COB-IDs, RxSDOs and TxSDOs. They can be viewed in the CANopen device. For example, users send a SDO message to the CAN-2000C modules by using RxSDO. On the contrary, the devices CAN-2000C transmit a SDO message by using TxSDOS.

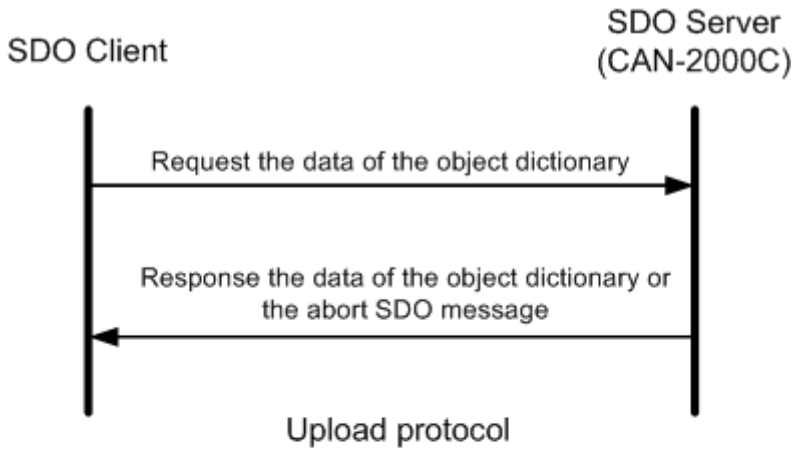
Before the SDO has been used, only the client can take the active requirement for a SDO transmission. When the SDO client starts to transmit a SDO, it is necessary to choose a proper protocol.

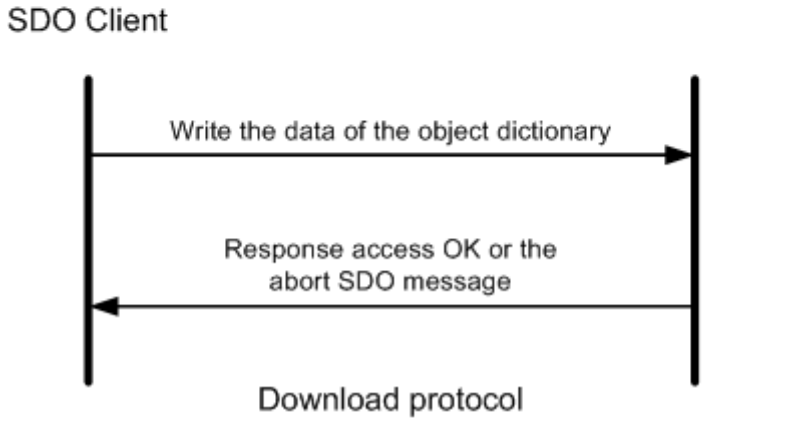
If the SDO client has to get the information from the device object dictionary and from the SDO server, the segment upload protocol or block upload protocol will be applied.

It is worth to be mentioned, the front protocol is used for transmitting fewer data; the latter protocol is used for transmitting larger data. Both the segment download protocol and block download protocol will work when the SDO client wants to modify the object dictionary to the SDO server. The differences between the segment download protocol and the block download protocol are similar to the differences between the segment upload protocol and the block upload protocol. Because of the different access types in the object dictionary, not all accessing action of the object dictionary via the SDO transmission is allowed. If the SDO client trends to modify the entries of the SDO server object dictionary which uses the read-only access type, the abort SDO transfer

protocol will be given, and the SDO transmission will also be stopped.

The CAN-2000C only supports the SDO server. Therefore, it can be passive and wait for requests from clients. The general concept figure of the upload and download protocol with the CAN-2000C is shown as follows.





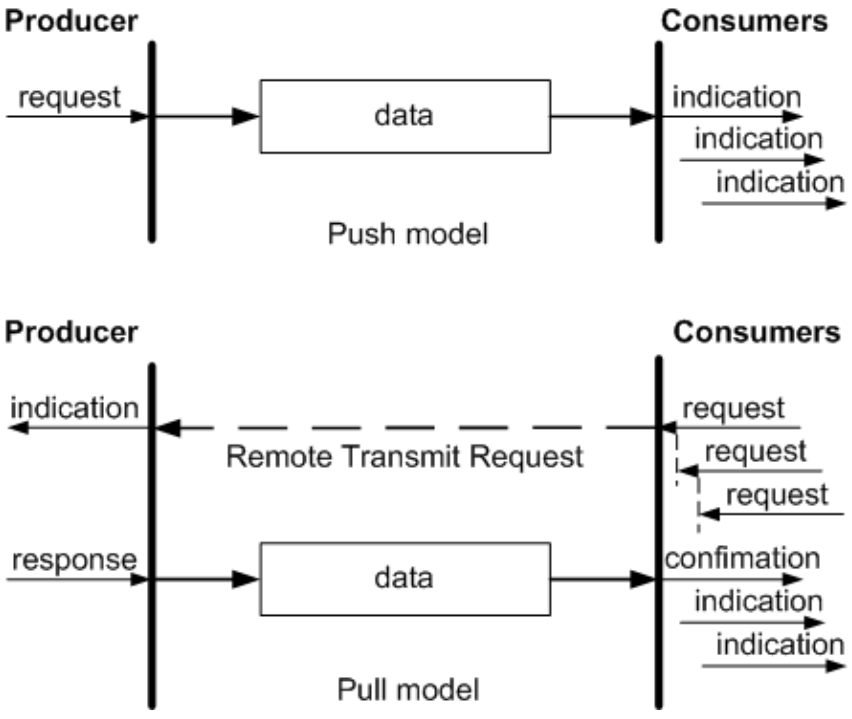
3.3 PDO Introduction

Based on the transmission data format of the CAN bus, the PDO can transmit eight bytes of process data at one time. Because of the PDO messages without overheads, it is more efficient than other communication objects within CANopen and therefore used for real-time data transfer, such as DI, DO, AI, AO, etc.

Communication Modes for the PDO

PDO reception or transmission is implemented via the producer/consumer communication model (also called the push/pull model). When starting to communicate in the PDO push mode, it needs one CANopen device to play the role of PDO producer, and non device or more than one device to play the role of PDO consumer.

The PDO producer sends out the PDO message after it reached the CAN bus arbitration. Afterwards, each PDO consumer will receive this PDO message respectively, and then message is processed by each device to check whether it is needed or not (be dropped). In the PDO pull mode, one of the PDO consumers needs to send out a remote transmit request to the PDO producer. According to this remote request message, the PDO producer responds the corresponding PDO message for each PDO consumer in the CAN bus. The PDO communication structure figure is shown below.



For the CANopen device, the TxPDO specializes in transmitting data, and usually is applied on DI/AI channels. The COB-ID of the PDO for receiving data is RxPDO COB-ID, and it is usually applied on DO/AO channels. Take the CAN-2000C as an example. If a PDO producer sends a PDO message to the CAN-2000C, it needs to use the RxPDO COB-ID of the CAN-2000C because it is a PDO reception action viewed from the CAN-2000C. Inversely, when some PDO consumers send remote transmit requests to the CAN-2000C, it must use the TxPDO COB-ID of the CAN-2000C because it is a PDO transmission action viewed from the CAN-2000C.

Trigger Modes Of PDO

For PDO producers, PDO transmission messages can be trigged by three conditions. They are the event driven, timer driven and remote request conditions. All of them are described below.

Event Driven

PDO transmission can be triggered by a specific driven event, including the following conditions. Under the cyclic synchronous transmission type, the event is driven by the expiration of the specified transmission period, synchronized by the reception of the SYNC message.

Moreover, under the acyclic synchronous or asynchronous transmission type, the PDO transmission can also be triggered or driven by a device-specified event in the CANopen specification DS-401 v2.1, i.e. by following this spec, the PDO will be triggered by any change in the DI-channel states when the transmission type of this PDO is set to acyclic synchronous or asynchronous.

Timer Driven

PDO transmissions are also triggered by a specific time event, even if a specified time elapsed without occurrence of an event. For example, the PDO transmission of the CAN-2000C can be triggered by the event timer of the PDO communication parameters, which is set by users.

Remote Request

The PDO transmission can be triggered by receiving a remote request from any other PDO consumer with under the asynchronous or RTR setting.

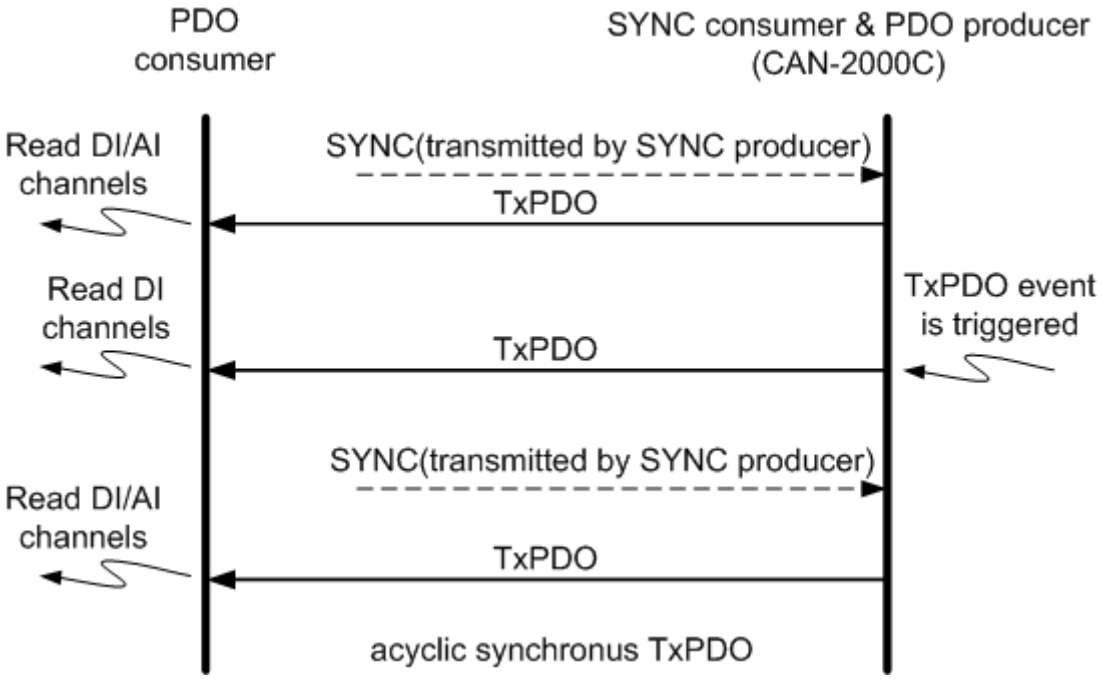
PDO Transmission Types

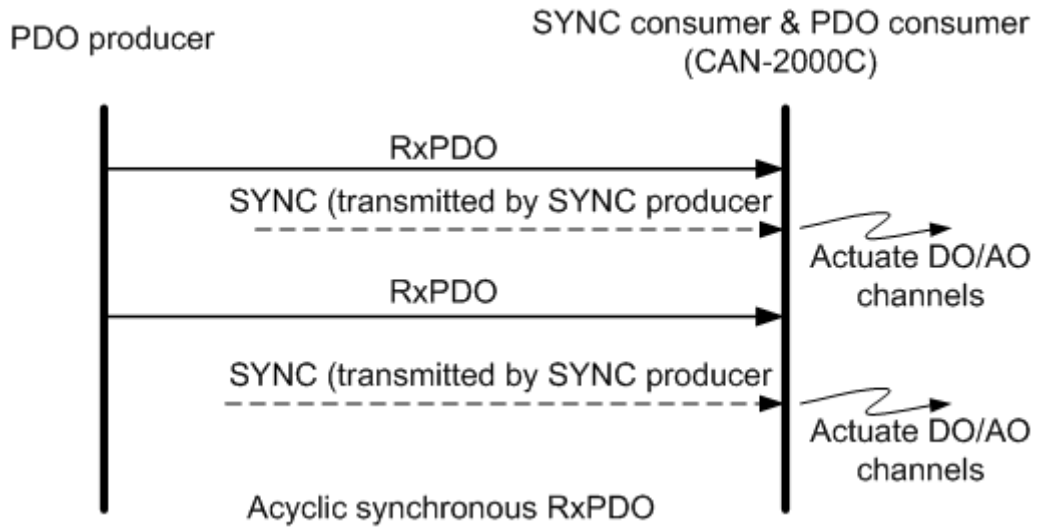
Generally there are two kinds of PDO transmission modes, synchronous and asynchronous. For the PDO in a synchronous mode, it must be triggered by the reception of a SYNC message.

The synchronous mode can be further distinguished into three kinds of transmission(s), acyclic synchronous, cyclic synchronous and RTR-only synchronous. The acyclic synchronous can be triggered by both the reception of a SYNC message and the driven event mentioned above.

Acyclic synchronous

For the TxPDO object, after receiving an object from the SYNC producer, the CAN-2000C will respond with a pre-defined TxPDO message to the PDO consumers. For the RxPDO object, the CAN-2000C needs to receive the SYNC objects to actuate the RxPDO object, which is received before the SYNC object. The following figures indicate how the acyclic synchronous transmission type works on the RxPDO and the TxPDO.

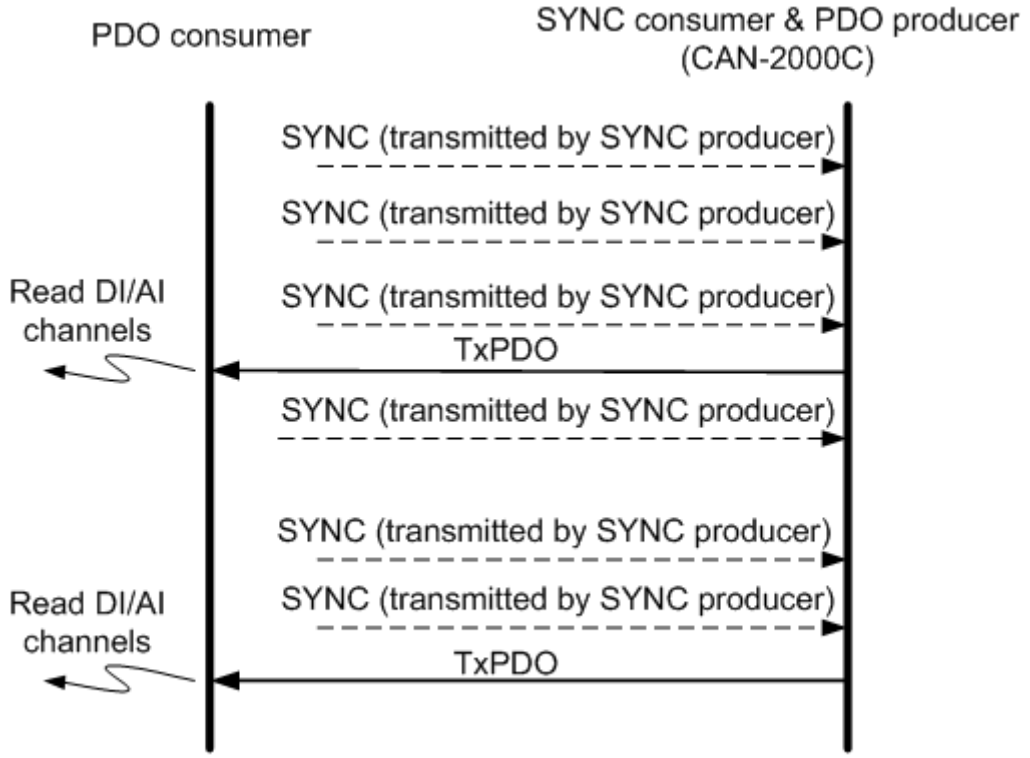




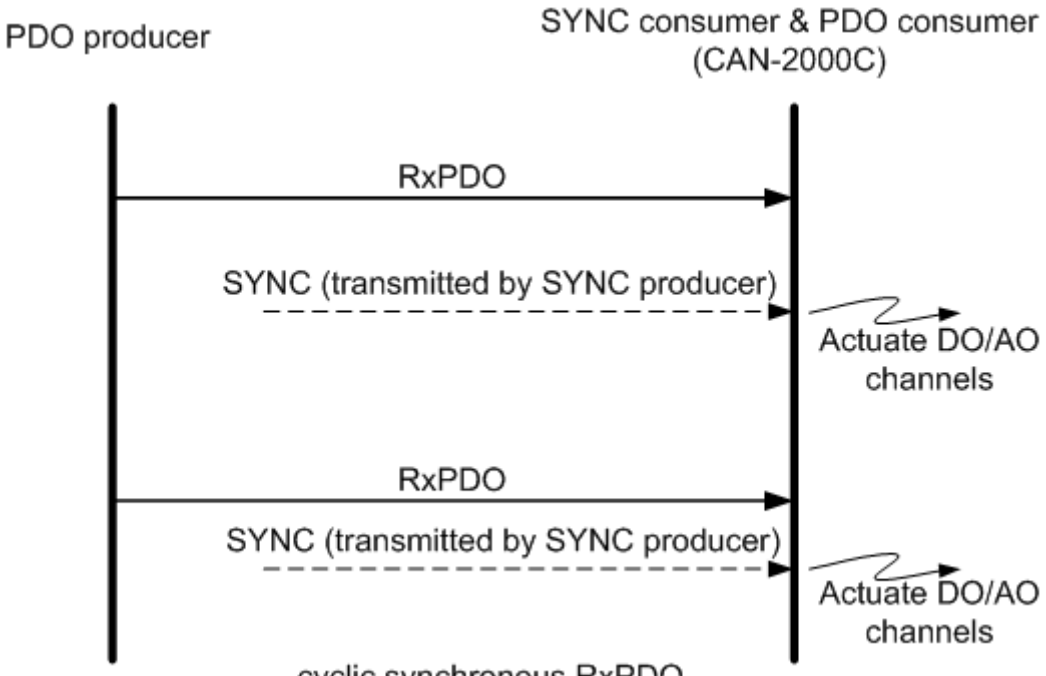
Cyclic synchronous

Inversely, the cyclic synchronous transmission mode is triggered by the reception of an expected number of SYNC objects, and the max number of

expected SYNC objects can be 240. For example, if the TxPDO is set to response when receiving 3 SYNC objects, the CAN-2000C will feed back the TxPDO object according to the set. For the RxPDO, actuating the DO/AO channels by the RxPDO is independent of the number of SYNC objects. These concepts are shown in the figures below.



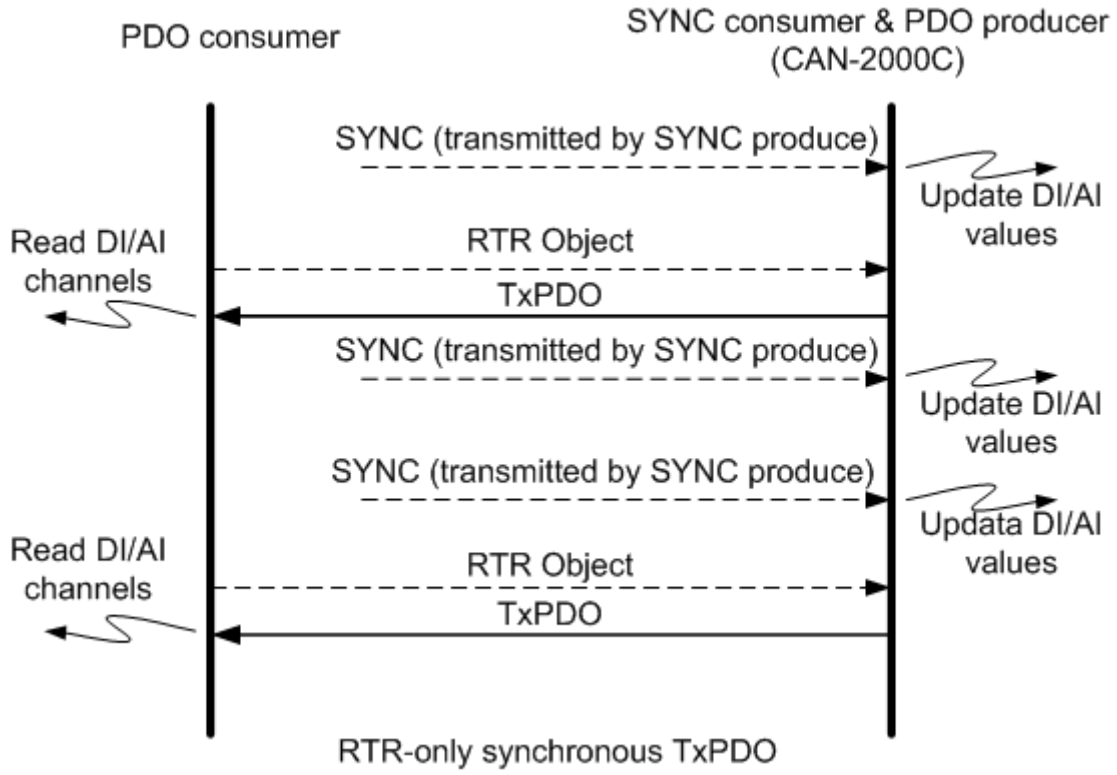
cyclic synchronous TxPDO



cyclic synchronous RxPDO

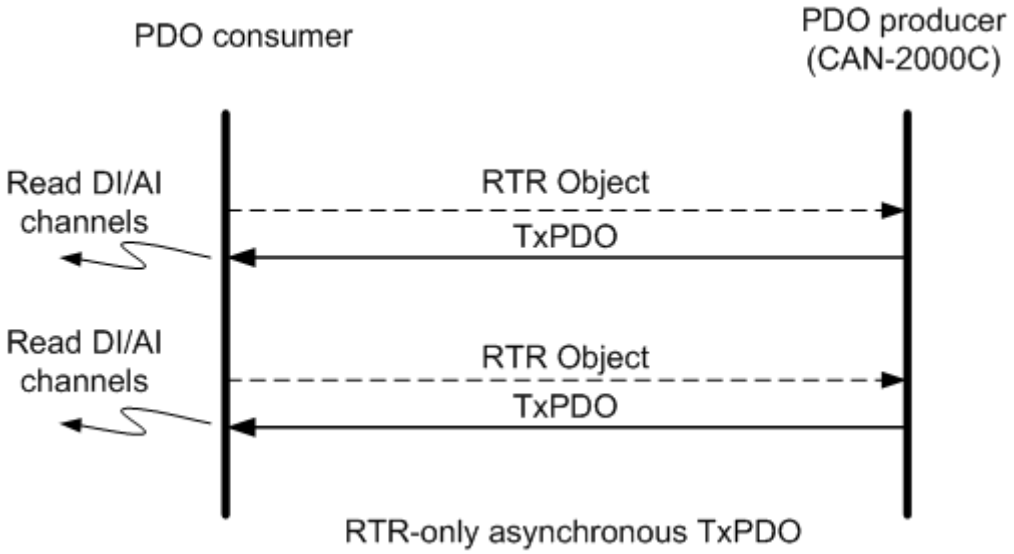
RTR-only synchronous

The RTR-only synchronous mode is activated when receiving a remote-transmit-request message, i.e. SYNC objects. This transmission type is only useful for TxPDO. In this situation, the CAN-2000C will update the DI/AI value when receiving any SYNC object. And, if the RTR object is received, the CAN-2000C will respond to the TxPDO object. The following figure shows the mechanism of this transmission type.



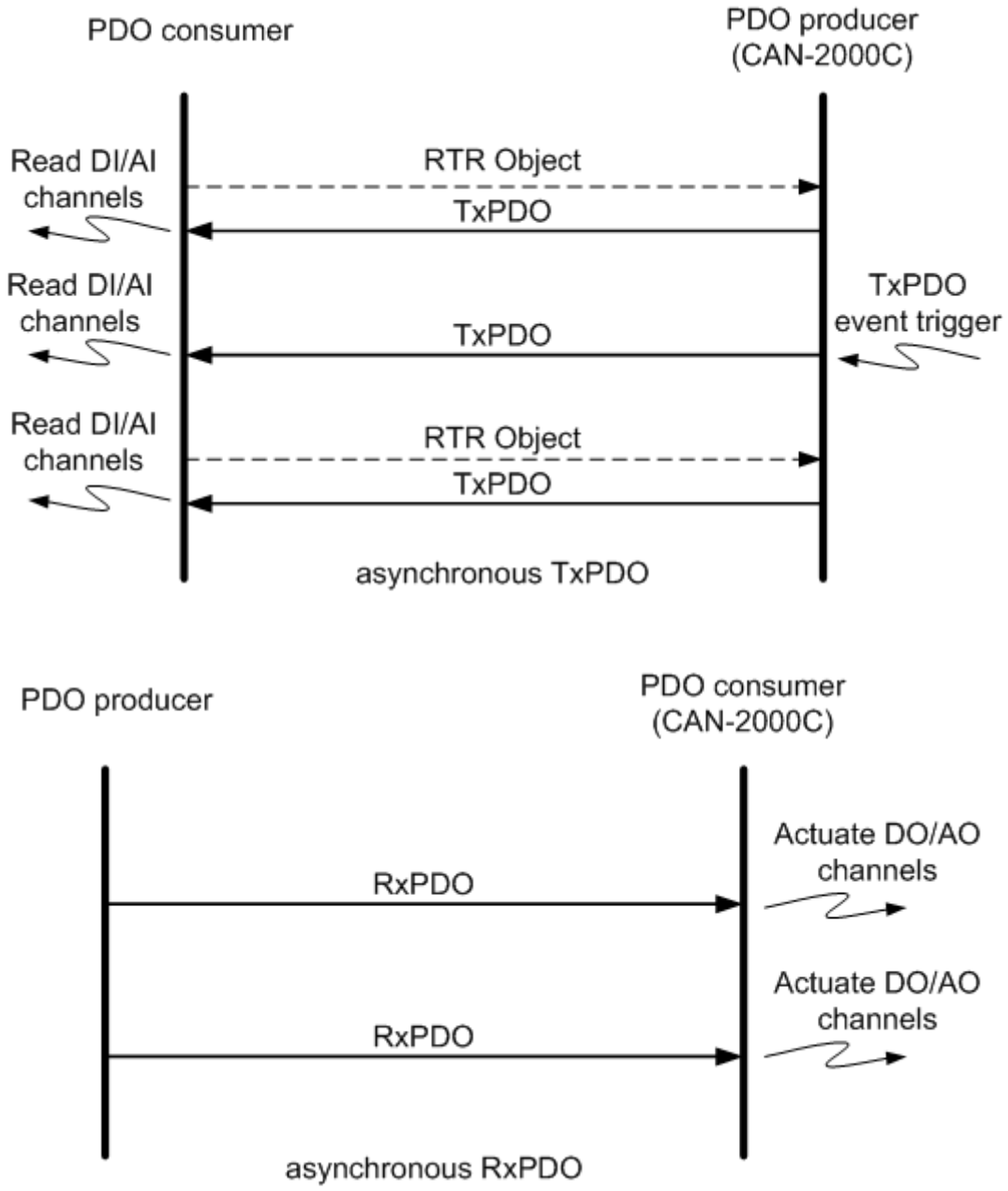
RTR-only asynchronous

The asynchronous mode is independent of the SYNC object. This mode can also be divided into two parts. There are RTR-only asynchronous transmission type and asynchronous transmission type. The RTR-only transmission type is only for supporting TxPDO transmissions, only triggered by receiving the RTR object from the PDO consumer. This action is depicted below.



Asynchronous

The other part is the asynchronous transmission type. Under this type, the TxPDO message can be triggered by receiving the RTR object and the device-specified event mentioned in the event driven paragraph. Furthermore, the DO/AO channels can act directly by receiving the RxPDO object. This transmission type is the default value when the CAN-2000C boots up. The concept of the asynchronous type is shown as follows.



Inhibit Time

Because of the arbitration mechanism of the CAN bus, the CANopen communication object ID in small size has a higher transmission priority than the bigger one. For example, there are two nodes on the CAN bus, the one needs to transmit the CAN message with the COB-ID 0x181, and the other has to transmit the message with COB-ID 0x182. When these two nodes transmit the CAN message to the CAN bus simultaneously, only the message containing COB-ID 0x181 can be successfully sent to the CAN bus because of the higher transmission priority. So the message with COB-ID 0x182 will be held to transmit until the message with COB-ID 0x181 is successfully transmitted. This arbitration mechanism can guarantee the successful transmission for one node when a transmission conflict occurs.

However, if the message with COB-ID 0x181 is continually transmitted, the message with COB-ID 0x182 will be postponed to be transmitted. In order to avoid the occupation of the transmission privilege by the message with the lower COB-ID, the inhibit time parameters for each of the PDO objects are supported to define a minimum time interval between each PDO message transmission, which has a multiple of 1ms. During this time interval, the PDO message will be inhibited from transmission.

Event Timer

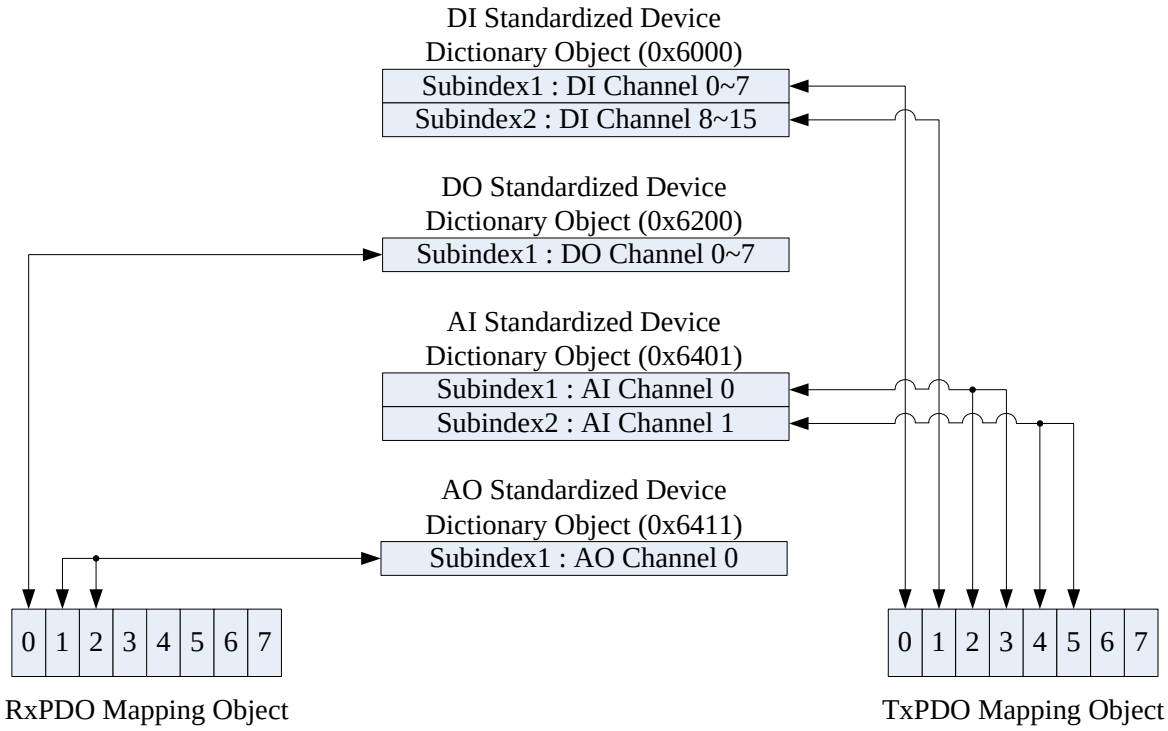
This parameter setting on the event timer is only used for TxPDO. If the parameter of the event timer is not equal to 0 under the transmission type in asynchronous mode, the expiration of this time value can be just considered to be an event. This event will cause the TxPDO message transmission. The event timer parameter is defined as a multiple of 1ms.

PDO Mapping Objects

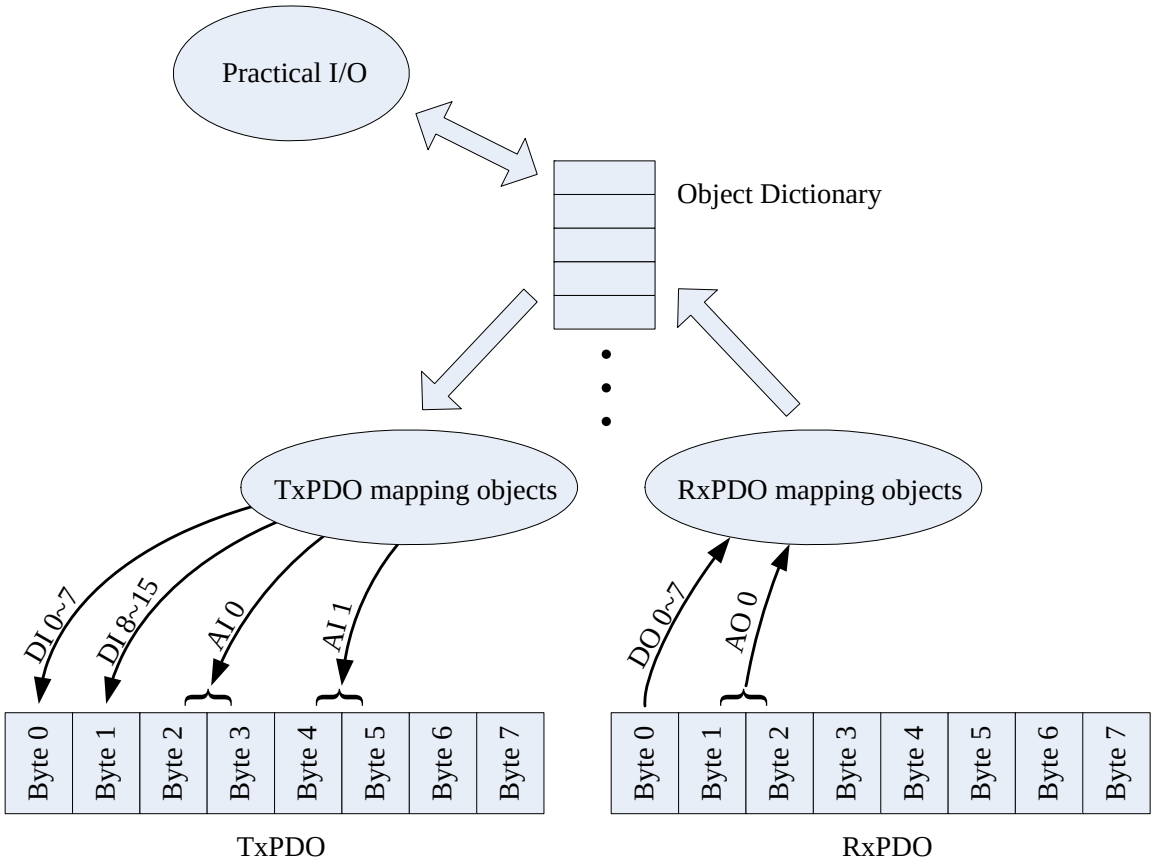
The PDO mapping objects are provided to the interface which is for PDO messages and real I/O data in the CANopen device. They define the meanings for each byte in the PDO message, and may be changed by using a SDO message. All of the PDO mapping objects are arranged in the Communication Profile Area. In the CANopen spec (see DS 401), RxPDO and TxPDO default mapping objects will specify something as follows:

- There shall be up to 4 TxPDO mapping objects and up to 4 RxPDO mapping objects with default mappings.
- The 1st RxPDO and TxPDO mapping objects are used for digital outputs and inputs to each other.
- The 2nd, 3rd, and 4th RxPDO and TxPDO mapping objects are respectively assigned to record the value of analog outputs and inputs.
- If a device supports too many digital input or output channels over 8 channels, the related analog default PDO mapping objects remaining the additional unused digital I/Os will use its additional objects. This rule with the same concept is used on the additional analog channels. Take the RxPDO as an example; there are 11 DO and 13 AO object entries in the object dictionary. Generally in the CAN-2000C, the first 8 DO object entries will be mapped to the first RxPDO mapping object because one DO object entry needs one byte space. The last 3 DO object entries will be assigned to the 5th RxPDO according to the above rules the 2nd and the 3rd. Furthermore, one AO object entry needs 2 bytes of space. Therefore, the second RxPDO mapping object has been occupied by the first 4 AO object entries. The following 4 AO object entries will be assigned to the third RxPDO mapping object, as well to the 4th RxPDO mapping object. Because the 5th RxPDO mapping object has been occupied by the DO object entries, the last AO object entry shall be assigned into the 6th RxPDO mapping object.

Before applying the PDO communications, the PDO producer and the PDO consumers must have mutual PDO mapping information. On the one hand, the PDO producers need PDO mapping information to decide how to assign the expected practical I/O data to PDO messages. Besides, PDO consumers need the PDO mapping information to recognize each byte of received PDO message, i.e. when a PDO producer transmits a PDO object to PDO consumers, the consumers will contrast this PDO message with PDO mapping entries, previously obtained from the PDO producer, and then interpret the meanings of these values from the received PDO object. For example, if a CANopen device has 16 DI, 8 DO, 2 AI, and 1 AO channels. The input or output values of these channels will be mutually stored into several specific entries.



According to the PDO mapping objects in the figure above, if this CANopen device gets the RxPDO message in three bytes, the first byte is for the output value from the DO channels 0~7, and the following two bytes are for the analog output value. After interpreting the data of the RxPDO message, the device will actuate the DO and AO channels by the received RxPDO message. It is worth to mention that TxPDO also operate in the same procedure as RxPDO message. When the TxPDO trigger events occur, the CANopen device will send the TxPDO message to the PDO consumers. The values of the bytes assigned in the TxPDO message will follow the TxPDO mapping object as shown in the above figure. The first two bytes of the TxPDO message are respectively for the values from the DI channels 0~7 and channel 8~15. The following third and forth bytes of the TxPDO message is for the value 0 of the AI channel. And the fifth and sixth bytes are for the value 1 of the AI channel. The relationships among the object dictionary, the PDO mapping object and the PDO message are given below.

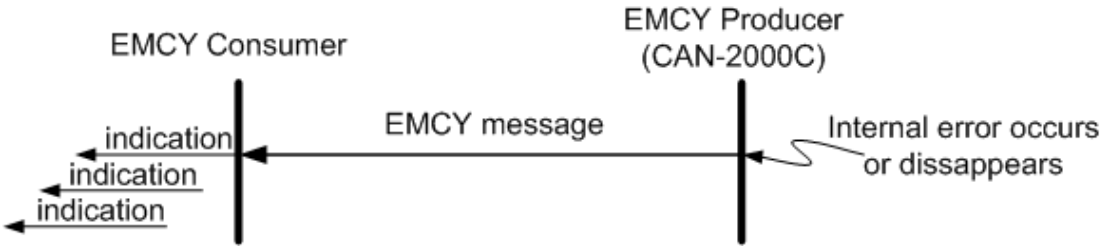

Practical I/O
Object Dictionary
TxPDO mapping objects
RxPDO mapping objects
DI 0~7
DI 8~15
AI 0
AI 1
DO 0~7
AO 0
Byte 0
Byte 1
Byte 2
Byte 3
Byte 4
Byte 5
Byte 6
Byte 7
Byte 0
Byte 1
Byte 2
Byte 3
Byte 4
Byte 5
Byte 6
Byte 7
TxPDO
RxPDO

3.4 EMCY Introduction

EMCY messages are triggered when a device internal error occurs, i.e. after a CANopen device detects the internal error, an emergency message will be transmitted to the EMCY consumers per time per error event. But the EMCY message will not be transmitted again if the same error repeatedly occurs. When error reasons are gone, an emergency message containing the emergency error code “00 00” will only respond to the specific error fields. So, by checking the EMCY message, users can understand what happened in the CAN-2000C, and then do something about the error event.

Please note that only the emergency consumers can receive the EMCY object, and only the CAN-2000C can support functions of the emergency producer.

The general concept regarding EMCY communications is shown below.



An emergency message containing 8-byte of data is called emergency object data. The abbreviated diagram is shown below, and all fields in the emergency object data will be described in section 5.3.

Byte	0	1	2	3	4	5	6	7
Content	Emergency Error Code		Error register	Manufacturer specific Error Field				

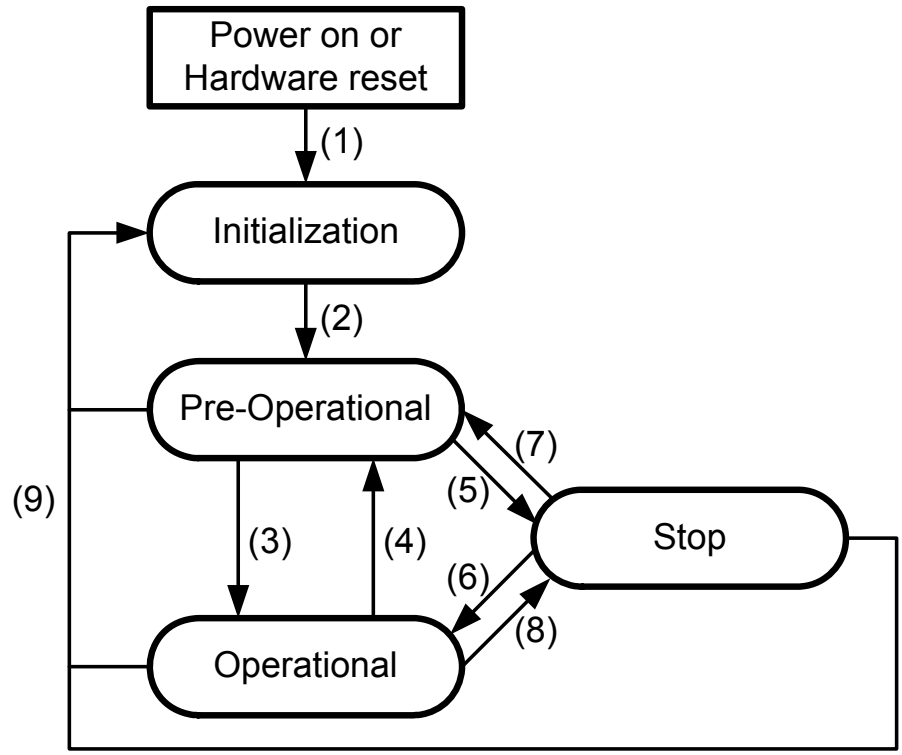
3.5 NMT Introduction

The Network Management (NMT) follows the node-oriented structure and the master-slave relationship. In the same CAN bus network, only one CANopen device is allowed to execute the function of NMT master. Each CANopen node is regarded as a unique NMT slave identified by its node ID from 1 to 127.

The NMT service supplies two protocols, the module control protocol and the error control protocol. Through the module control protocol, the nodes can be controlled to several kinds of status, such as installing, pre-operational, operational, and stopped. According to the NMT slave can present in different statuses, it has different privileges to carry out the communication protocol. Through the error control protocol, users are enabling to detect the remote error in the network in order to confirm whether the node still works or not.

3.5.1 Module Control Protocols

Before introducing the modules control protocols, the architecture of the NMT state mechanism needs to be mentioned. The diagram shows the process and the relationships among each NMT state and the mechanism.



State Mechanism Diagram

(1)	Under “Power on” or “Hardware Reset”, the initialization state will be loaded automatically.
(2)	As the Initialization accomplished, Pre-Operational state will be entered automatically
(3),(6)	Indication of starting remote node
(4),(7)	Indication of entering Pre-Optional State
(5),(8)	Indication of stopping remote node
(9)	Indication of the “Reset Node” or the “Reset Communication”

Devices will directly lead to the Pre-Operational state after finishing the device initialization. Then, the nodes will be switched into different state by receiving a specific indication. By the way, each different NMT state will consider a specific communication method. For example, the PDO message can only do the transmission and receiving in the operational state. In the following table, the relationship among each NMT state and communication objects is given.

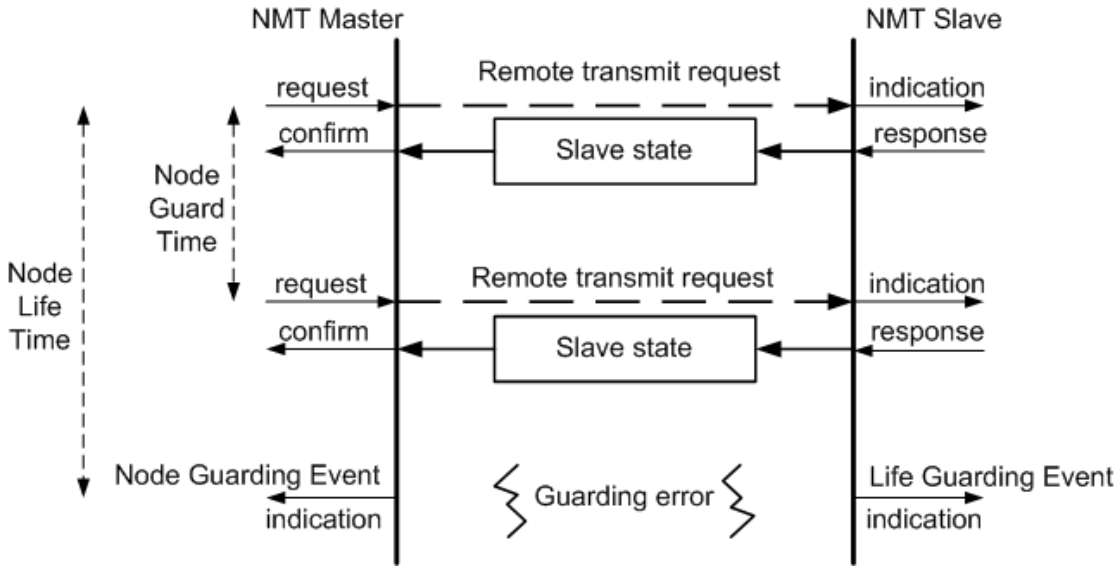
	Installing	Pre-operational	Operational	Stopped
PDO			**O**	
SDO		**O**	**O**	
SYNC Object		**O**	**O**	
Time Stamp Object		**O**	**O**	
EMCY Object		**O**	**O**	
Boot-Up Object	**O**			
NMT		**O**	**O**	**O**

3.5.2 Error Control Protocols

There are two kinds of protocols defined in the error control protocol. According to the CANopen spec, one device is not allowed to use the following error control mechanisms at the same time, Node Guarding Protocol and Heartbeat Protocol. In addition, the CAN-2000C provides the salve function of the Node Guarding Protocol for practical applications. Therefore, only node guarding protocols will be highlighted here, and described below.

Node Guarding Protocol

The Node Guarding Protocol follows the Master/Slave relationship. It helps users monitoring the node in the CAN bus. The communication method of node guarding protocol is defined as follows.

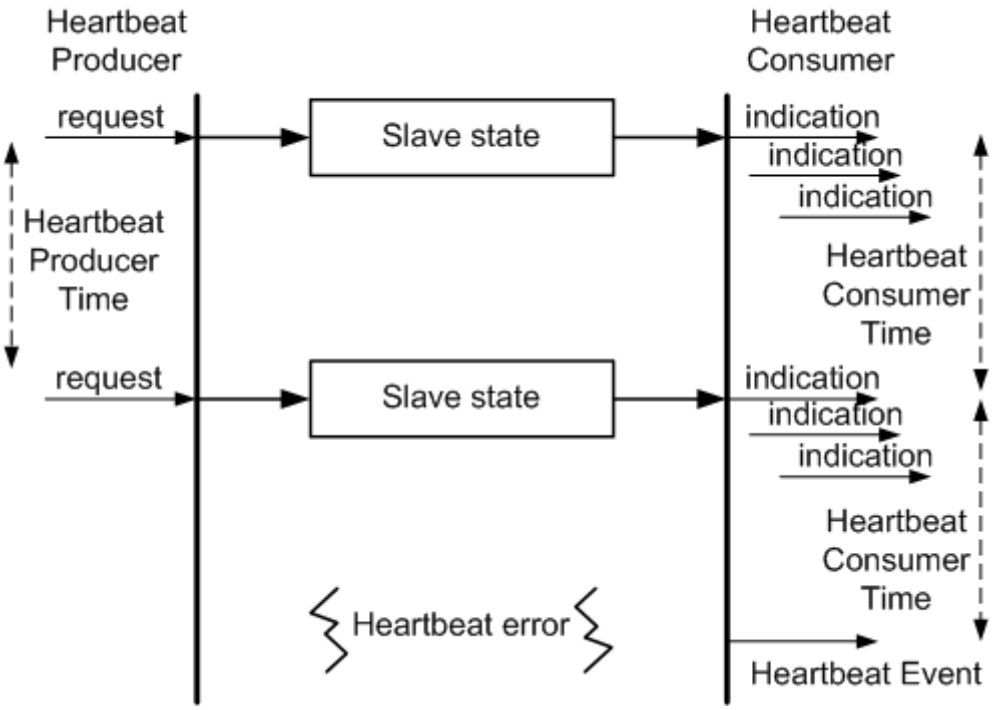


The NMT master will inspect each NMT slave at regular time intervals. This time-interval is called the node guard time, given by the “guard time * life time factor”, and may be different from each NMT slave. And the response of the NMT slave contains the state of that NMT slave, which may be in a "**Stopped**", "**Operational**", or "**Pre-operational**" state. The node life time factor can also be different for each NMT slave. If the NMT slave has not been inspected during its life time, a remote node error will be given, and indicate through the "Life Guarding Event" service.

In addition, the reported NMT slave state, which does not match the expected state, will also produce the “Node Guarding Event”. This event may occur in the DO and AO channels, and output the error mode value, recorded in the object with index 0x6207 and index 0x6444. Moreover, the object with index 0x6206 and 0x6443 can control the error mode value of the DO or AO channels in enabling or disabling when the “Lift Guarding Event” has been indicated. For more information about objects with index 0x6206, 0x6207, 0x6443, and 0x6444, please refers to the chapter 6.

Heartbeat Protocol

The Heartbeat Protocol follows the Producer/Consumer relationship. It helps users monitoring the node in the CAN bus. The communication method of node guarding protocol is defined as follows.



The Heartbeat Protocol defines an Error Control Service without need of the remote frames. A Heartbeat Producer transmits a Heartbeat message cyclically. One or more Heartbeat Consumer receives the indication. The relationship between producer and consumer is configurable via the object dictionary. The Heartbeat Consumer guards the reception of the Heartbeat within the Heartbeat Consumer Time. If the Heartbeat is not received within the Heartbeat Consumer Time a Heartbeat Event will be generated.

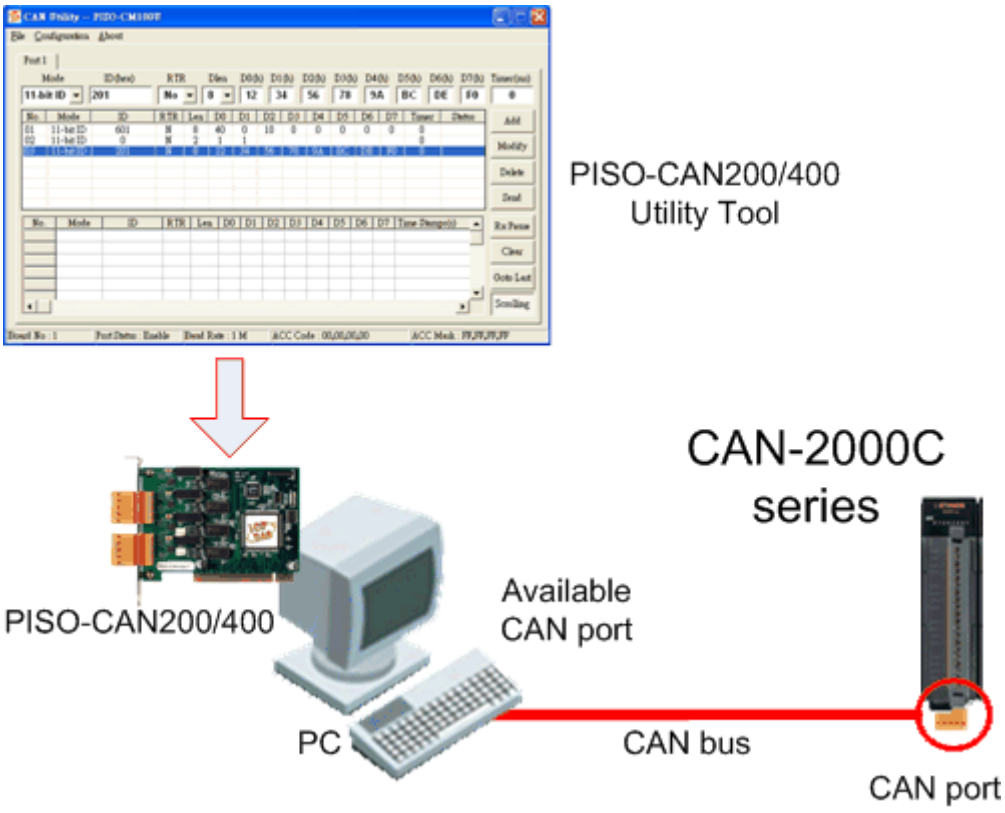
If the Heartbeat Producer Time is configured on a device, the Heartbeat Protocol begins immediately. It is not allowed for one device to use both error control mechanisms Guarding Protocol and Heartbeat Protocol at the same time. If the Heartbeat Producer Time is unequal to 0, the Heartbeat Protocol is used.

4 Getting Start

When gets a CAN-2000C series module, turn off it and set the proper node ID and baud rate by using the rotary switch first. Afterwards, reboot the module and if a CAN tool has received the boot up message of the CAN-2000C module, it means the module boot up successful. And if not, please check the baud rate and connect of the module.

5 CANopen Communication Set

In the following section, several CANopen communication protocols are described. Each protocol description has one corresponding example. Before the example, users must have one CAN interface device to send out the CAN command. Therefore, the PISO-CAN200/400 CAN interface card with a 2/4 CAN port PCI will be requested. It provides an easy-to-use utility tool to sending the CAN 2.0A or 2.0B command. The relationship between the software and the hardware is shown as follows.



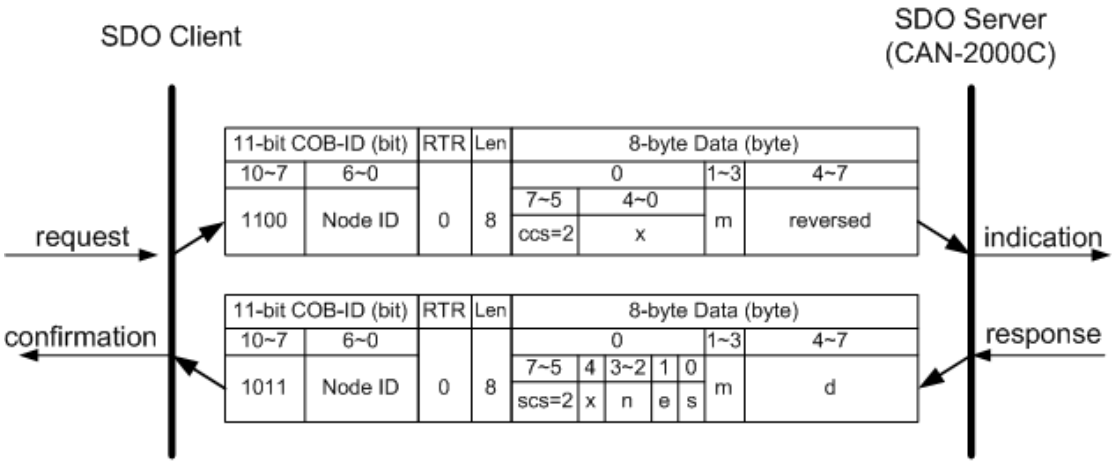
Please refer to the PISO-CAN200/400 user manual to know how to use its Utility Tool.

5.1 SDO Communication Set

5.1.1 Upload SDO Protocol

Initiate SDO Upload Protocol

Before transferring the SDO segments, the client and server need to communicate with each other by using the initiate SDO upload protocol. Via the initiate SDO upload protocol, the SDO client will inform the SDO server what object the SDO client wants to request. As well, the initiate SDO upload protocol is permitted to transmit up to four bytes of data. Therefore, if the data length of the object, which the SDO client can read, is equal to or less than the permitted data amount, the SDO communication will be finished only by using the initial SDO upload protocol, i.e. if the data upload is less enough to be transmitted in the initiate SDO upload protocol, then the upload SDO segment protocol will not be used. The communication process of this protocol is shown as follows.



Initiate SDO Upload Protocol

ccs : client command specified
2: initiate upload request

scs : server command specified
2: initiate upload response

n : Only valid if **e** = 1 and **s** = 1, otherwise 0. If valid, it indicates the number of bytes in **d** that do not contain data. Bytes [8-**n**, 7] do not contain segment data.

e : transfer type
0: normal transfer
1: expedited transfer
If the **e**=1, it means that the data of the object are equal or less than 4 bytes, and only initiate SDO upload protocol is needed. If **e**=0, the upload SDO segment protocol is necessary.

s : size indicator
0: Data set size is not indicated.
1: Data set size is indicated.

m : multiplexer
It represents the index/sub-index of the data to be transfer by the SDO. The first two bytes are the index value and the last byte is the sub-index value.

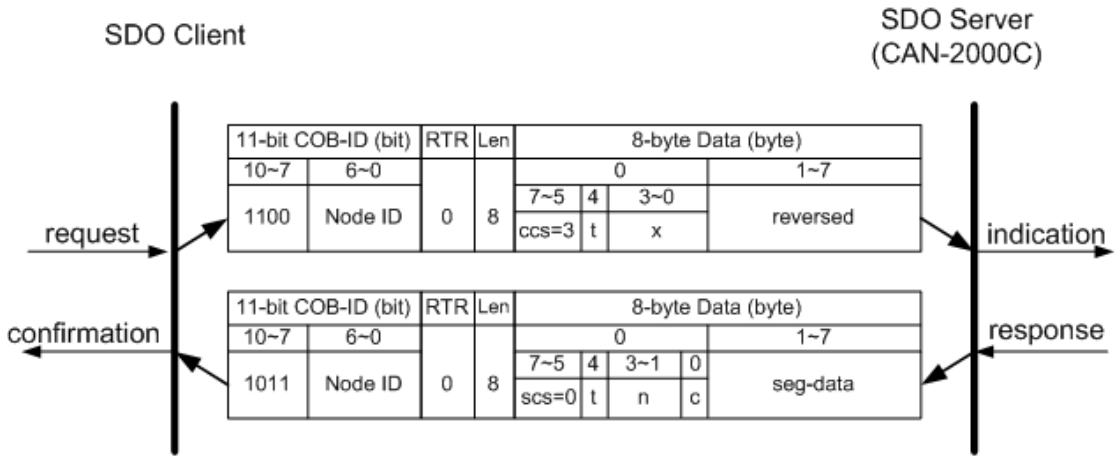
d : data
e=0, **s**=0: **d** is reserved for further use.
e=0, **s**=1: **d** contains the number of bytes to be uploaded, and byte 4 contains the least significant bit, and byte 7 contains the most significant bit.
e=1, **s**=1: **d** contains the data of length 4-**n** to be uploaded, the encoding depends on the type of the data referenced by index and sub-index.
e=1, **s**=0: **d** contains unspecified number of bytes to be uploaded.

x : not used, always 0

reserved : reserved for further use , always 0

Upload SDO Segment Protocol

When the upload data length is over 4 bytes, the upload SDO segment protocol will be needed. After finishing the transmission of the initiate SDO upload protocol, the SDO client will start to upload the data. The upload SDO segment protocol will comply with the process shown below.



Upload SDO Segment Protocol

ccs : client command specified
3: upload segment request

scs : server command specified
0: upload segment response

t : toggle bit
This bit must alternate for each subsequent segment that is uploaded. The first segment will have the toggle bit set to 0. The toggle bit will be equal for the request and the response message.

c : indicates whether there are still more segments to be uploaded
0: more segments to be uploaded.
1: no more segments to be uploaded.

seg-data : It is at most 7 bytes of segment data to be uploaded. The encoding depends on the type of the data referenced by index and sub-index.

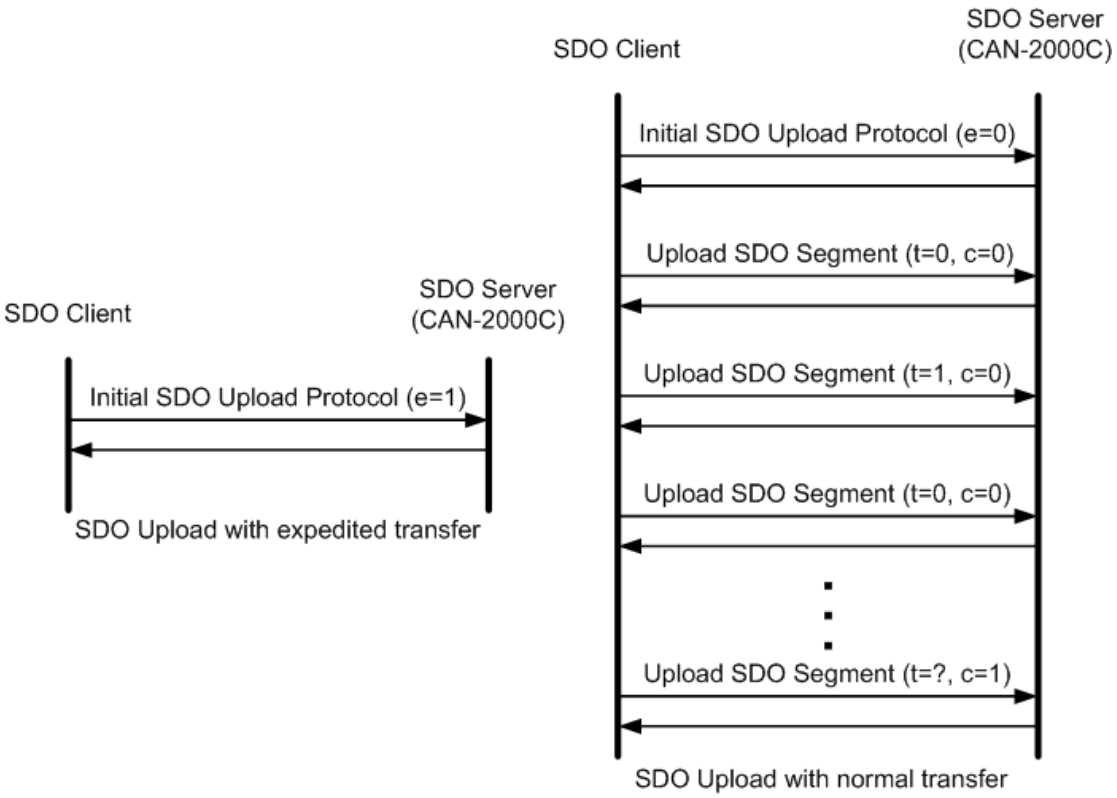
n : It indicates the number of bytes in **seg-data** that do not contain segment data. Bytes [8-**n**, 7] do not contain segment data. **n** = 0 if no segment size is indicated.

x : not used, always 0

reserved : reserved for further use , always 0

SDO Upload Example

The practical application of the SDO upload is illustrated as below.



In the following paragraph, both expedited transfer and normal transfer are given according to the procedure described above. In addition, the method of how to get the value stored in the object dictionary is also presented. As to the initiate SDO upload protocol, users can obtain how many sub-indexes the object with index 0x1400 can support. This information is in the object with index 0x1400 with sub-index 00. As well, users can get the string in the object with index 0x1008 via the initiate SDO upload protocol and the upload SDO segment protocol.

- **Example for expedited transfer**

Step 1. SDO message will be sent to the CAN-2000C to obtain the object entry with index 0x1400 and sub-index 00 stored in the communication profile area. The message structure is as follows. Moreover, the node ID of the CAN-2000C set to 1, and the information about the object entry with index 0x1400 will be described in the chapter 6.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	00	14	00	00	00	00	00

SDO client → **SDO server (CAN-2000C)**

ccs : 2

m : 00 14 00

According to the low byte has the higher transferred sequence, the first byte “00” will get the priority than the second byte “14”. Here the last byte “00” means the sub-index 00.

Step 2. The CAN-2000C will reply to the data stored in the object entry with index 0x1400 and sub-index 00.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	5	4F	00	14	00	02	--	--	--

SDO client ← **SDO server (CAN-2000C)**

scs : 2

n : 3

e : 1

s : 1

m : 00 14 00

d : 02 -- -- --

Because of the **n**=3, only the 4th byte is valid. Therefore, the feedback value is 02.

- **Example for normal transfer**

Step 1. Send the RxSDO message to the CAN-2000C to obtain the object entry with index 0x1008 and sub-index 00 stored in the communication profile area. The message structure is as follows. Moreover, the node ID for the CAN-2000C set to 1, and the information about object entry with index 0x1008 will be described in the chapter 6.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	08	10	00	00	00	00	00

SDO client → **SDO server (CAN-2000C)**

ccs : 2
m : 08 10 00

Step 2. The CAN-2000C will respond to the SDO message with the indication of how many bytes will be uploaded from the CAN-2000C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	41	08	10	00	09	00	00	00

SDO client ← **SDO server (CAN-2000C)**

scs : 2
n : 0
e : 0
s : 1
m : 08 10 00
d : 09 00 00 00

Because of the **e**=0 and **s**=1, the **d** means how many data users will upload from the CAN-2000C. The byte “09” is the lowest byte in the data length with a long format. Therefore, the data “09 00 00 00” means that users will upload 9 bytes data from CAN-2000C.

Step 3. The CAN-2000C is requested to start the data transmission.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	60	00	00	00	00	00	00	00

SDO client → **SDO server (CAN-2000C)**

ccs : 3
t : 0

Step 4. The CAN-2000C will respond to the first 7 bytes in the index 0x1008 and sub-index 00 object entries.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	00	43	41	4E	2D	32	30	38

SDO server (CAN-2000C) → **SDO client**

scs : 0
t : 0
n : 0
c : 0
seg-data : 43 41 4E 2D 32 30 38

Users can check the chapter 6 to know that the object entry with index 0x1008 and sub index 00 has the data type “VISIBLE_STRING”. Therefore, users need to transform these data values into the corresponding ASCII character. For example, after transformation, they become “CAN-208”.

Step 5. The CAN-2000C is requested to transmit the rest of the data.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	70	00	00	00	00	00	00	00

SDO client ⟶ **SDO server (CAN-2000C)**

ccs : 3
t : 1

Step 6. The rest of the data will be received from the SDO server.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	3	1B	38	00	--	--	--	--	--

SDO client ⟵ **SDO server (CAN-2000C)**

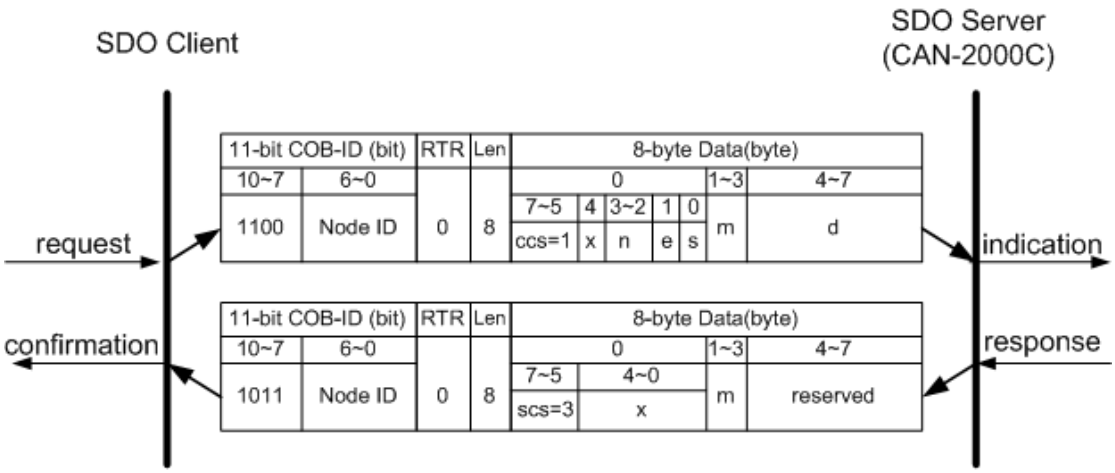
scs : 0
t : 1
n : 5
c : 1
seg-data : 38 00 -- -- -- -- --

Because of the **n**=5, and only the first two bytes are valid, the value of 0x38 and 0x00 will be transferred to the corresponding ASCII character. After transformation, it became “8 ”. So the example object had shown “CAN-2088” string.

5.1.2 Download SDO Protocol

Initiate SDO Download Protocol

The download modes are similar to the upload modes, but different in some parameters of the SDO messages. They are also separated into two steps. If the download data length is less than 4 bytes, the download action will finish in the download initialization protocol. Otherwise, the download segment protocol will be needed. These two protocols are shown below.



Initiate SDO Download Protocol

ccs : client command specified
1: initiate download request

scs : server command specified
3: initiate download response

n : Only valid if **e** = 1 and **s** = 1, otherwise 0. If valid, it indicates the number of bytes in **d** that do not contain data. Bytes [8-**n**, 7] do not contain segment data.

e : transfer type
0: normal transfer
1: expedited transfer
If the **e**=1, it means that the data of the object are equal or less than 4 bytes, and only initiate SDO download protocol is needed.
If **e**=0, the download SDO protocol is necessary.

s : size indicator
0: data set size is not indicated
1: data set size is indicated

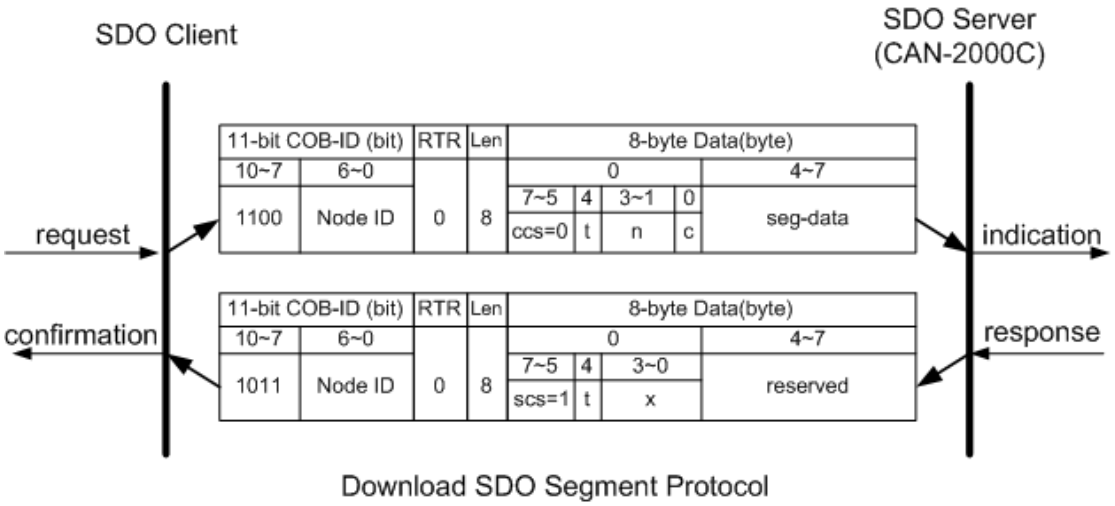
m : multiplexer
It represents the index/sub-index of the data to be transfer by the SDO.

d : data
e=0,**s**=0: **d** Is reserved for further use.
e=0,**s**=1: **d** contains the number of bytes to be downloaded, and byte 4 contains the least significant bit, and byte 7 contains the most significant bit.
e=1,**s**=1: **d** contains the data of length 4-**n** to be downloaded, the encoding depends on the type of the data referenced by index and sub-index.
e=1,**s**=0: **d** contains unspecified number of bytes to be downloaded.

x : not used, always 0

reserved : reserved for further use , always 0

Download Segment Protocol



Download SDO Segment Protocol

ccs : client command specified
0: download segment request

scs : server command specified
1: download segment response

seg-data : It is at most 7 bytes of segment data to be downloaded. The encoding depends on the type of the data referenced by index and sub-index.

n : It indicates the number of bytes in **segment data** that do not contain segment data. Bytes [8-**n**, 7] do not contain segment data. **n** = 0 if no segment size is indicated.

c : It indicates whether there are still more segments to be downloaded.
0 more segments to be downloaded
1: no more segments to be downloaded

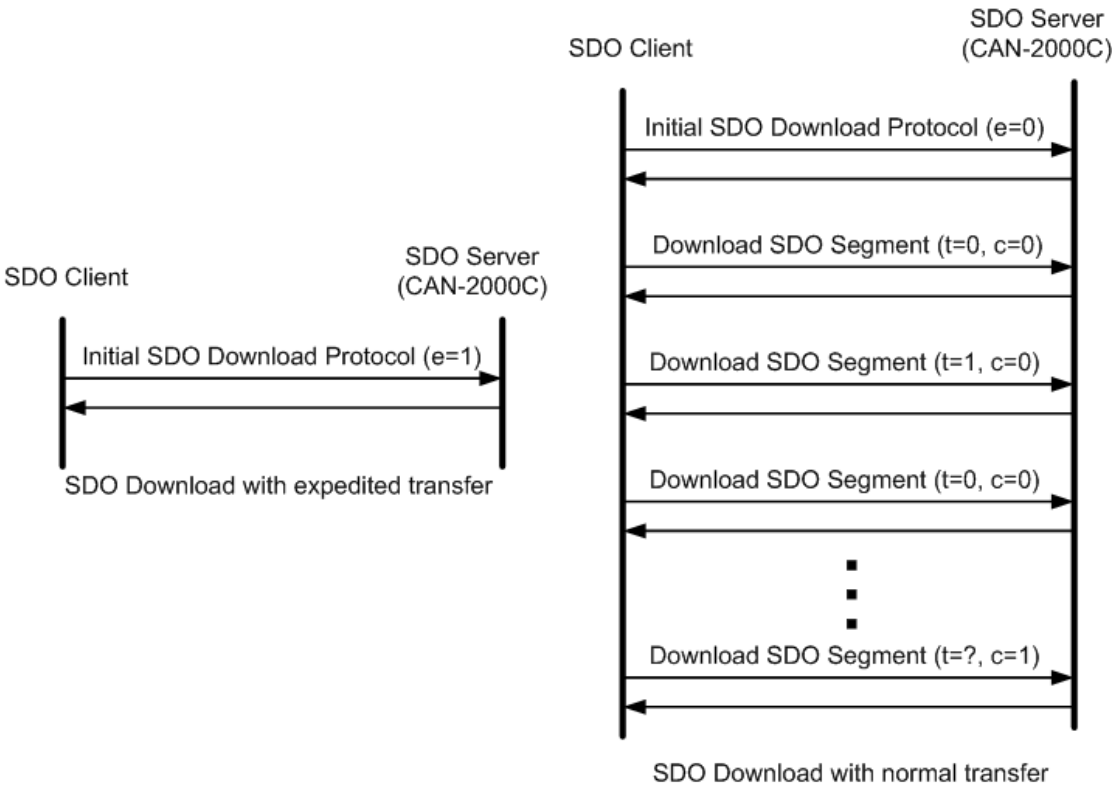
t : toggle bit
This bit must alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

x : not used, always 0

reserved : reserved for further use , always 0

SDO Download Example

When the SDO download example has been applied, the procedure in the below figure may be applied.



Since all of those object entries, which can be written, in the CAN-2000C are equal or less than 4 bytes, we can only provide the example for expedited transfer.

- **Example for expedited transfer**

Step 1. The Rx SDO message is sent to the CAN-2000C to access the object entry with index 0x1400 and sub-index 02 stored in the communication profile area. For example, the value of this object entry is changed to 5, as the node ID for the CAN-2000C is set to 1.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	2F	00	14	02	05	00	00	00

SDO client ⟶ **SDO server (CAN-2000C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 14 02
d : 05 00 00 00
Because the **n**=3, only the 4th byte is valid. Therefore, the feedback value is 05.

Step 2. The CAN-2000C will reply with the message to finish the data download. Then, users can use the upload methods to read back the value.

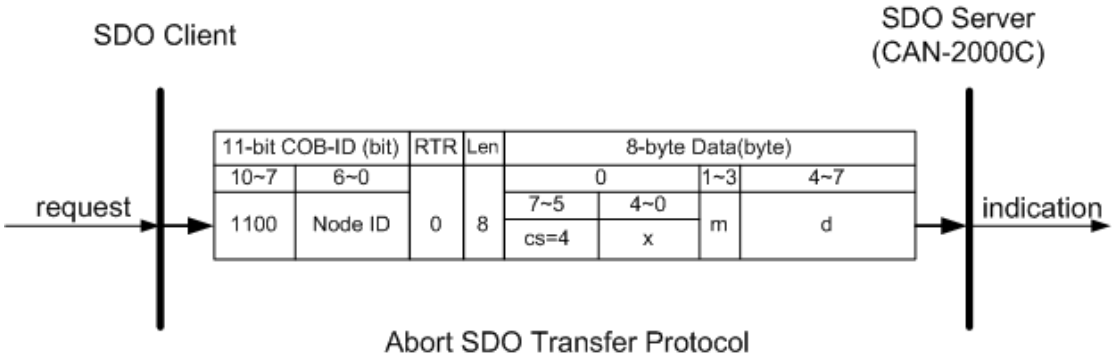
11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	4	60	00	14	02	--	--	--	--

SDO client ⟵ **SDO server (CAN-2000C)**

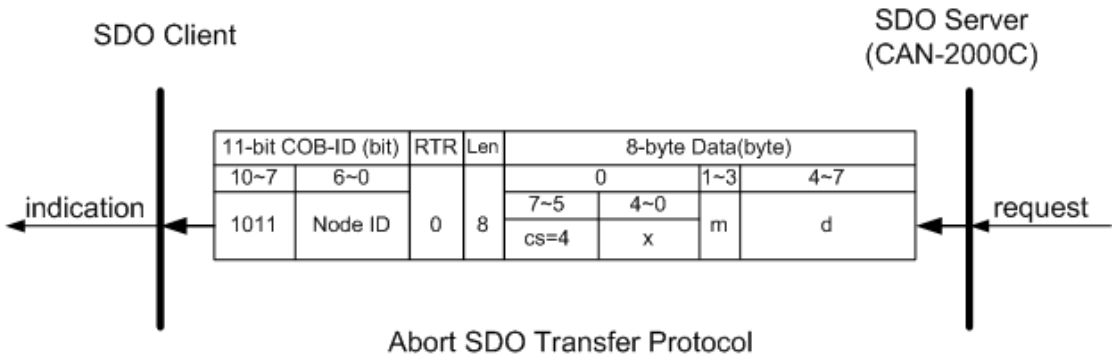
scs : 3
m : 00 14 02

5.1.3 Abort SDO Transfer Protocol

In some conditions, the SDO client or SDO server will terminate the SDO transmission. For example, the value of entries that users want to modify does not exist or is read-only, even users wouldn't continue the uncompleted SDO protocol under some special situations. When these conditions occur, both the client and the server can be activated to send the Abort SDO Transfer message. The Abort SDO Transfer protocol is shown below.



Abort SDO Transfer Protocol



Abort SDO Transfer Protocol

cs : command specified
4: abort transfer request

x : not used, always 0

m : multiplexer
It represents index and sub-index of the SDO

d : contains a 4-byte "Abort Code" about the reason for the abort

Abort Code	Description
0503 0000h	Toggle bit not alternated.
0504 0000h	SDO protocol timed out.
0504 0001h	Client/server command specified not valid or unknown.
0504 0002h	Invalid block size (block mode only).
0504 0003h	Invalid sequence number (block mode only).
0504 0004h	CRC error (block mode only).
0504 0005h	Out of memory.
0601 0000h	Unsupported access to an object.
0601 0001h	Attempt to read a write only object.
0601 0002h	Attempt to write a read only object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility in the device.
0606 0000h	Access failed due to a hardware error.
0607 0010h	Data type does not match, length of service parameter does not match
0607 0012h	Data type does not match, length of service parameter too high
0607 0013h	Data type does not match, length of service parameter too low
0609 0011h	Sub-index does not exist.
0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0609 0036h	Maximum value is less than minimum value.
0800 0000h	General error.
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of a file error).

Abort SDO Transfer Example

The object index 0x1008 doesn't support the sub-index 01 entry. Therefore, if users read the object entry with index 0x1008 and sub-index 01, the CAN-2000C will reply the Abort SDO Transfer message. The example is figured as follows.

Step 1. The Rx SDO message will be sent to the CAN-2000C in order to get the object entry with index 0x1008 and sub-index 01. The following example is assumed that the node ID for the CAN-2000C is set to 1.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	08	10	01	00	00	00	00

SDO client ⟶ **SDO server (CAN-2000C)**

ccs : 2
m : 08 10 01

Step 2. The CAN-2000C will reply to the Abort SDO message as shown below.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	80	08	10	01	11	00	09	06

SDO client ⟵ **SDO server (CAN-2000C)**

cs : 4
m : 08 10 01
d : 11 00 09 06
According to the low byte data have the transferring priority, the data will be converted to "06 09 00 11". Therefore, after searching the Abort Code table described above, this Abort Code can be interpreted as "Sub-index does not exist".

5.2 PDO Communication Set

5.2.1 PDO COB-ID Parameters

Before the real-time data are transmitted by the PDO, it is necessary to check the COB-ID parameter of this PDO in the PDO communication objects. This parameter setting controls the COB-ID of the PDO communication, which is in 32 bits, and each bit with its meaning is given in the table follow.

Bit Number	Value	Meaning
31 (MSB)	0	PDO exits (PDO is valid)
	1	PDO does not exist (PDO is not valid)
30	0	RTR allowed on this PDO
	1	No RTR allowed on this PDO
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28-11	0	If bit 29=0
	x	If bit 29=1: 28-11 bits of 29-bit COB-ID
10-0 (LSB)	x	10-0 bits of COB-ID

Note: Only CAN-2000C supports CAN 2.0A.

In the following table, it's regarding the default PDO COB-ID parameters.

Number of PDO	Default COB-ID of PDO	
	Bit10~Bit7 (Function Code)	Bit6~Bit0
TxPDO1	0011	Node ID
TxPDO2	0101	Node ID
TxPDO3	0111	Node ID
TxPDO4	1001	Node ID
RxPDO1	0100	Node ID
RxPDO2	0110	Node ID
RxPDO3	1000	Node ID
RxPDO4	1010	Node ID

Note: 1. Users can also define the PDO COB-ID by themselves. Actually, all COB-ID can be defined by users except the reserved COB-ID described in the table of the section 3.1. It is important to avoid the conflict with the defined COB-ID used in the same node.

2. The PDO COB-ID parameters cannot be changed if the PDO is valid (bit 31 =0).

5.2.2 Transmission Type

The transmission type is one of the several parameters defined in PDO communication objects with sub-index 02. Each PDO has its own transmission type. The transmission type can indicate the transmission or reception character for its corresponding PDO. The following table describes the relationship between the value of the transmission type and the PDO character. For example, if users used transmission type 0 for the first TxPDO, the CANopen device will follow the rule of the acyclic and synchronous PDO transmission.

Transmission Type	PDO Transmission method				
	cyclic	acyclic	synchronous	asynchronous	RTR only
0		O	O		
1-240	O		O		
241-251	----------reversed----------				
252			O		O
253				O	O
254				O	
255				O	

Note:

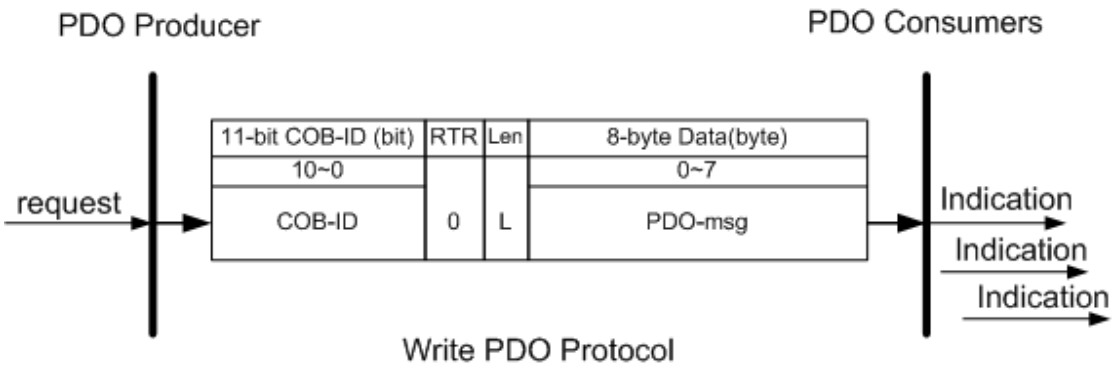
1. The transmission type 1-240 indicates how many SYNC objects the TxPDO will be triggered. The RxPDO is always triggered by the following SYNC upon reception of data independent of the transmission types 0-240.
2. The transmission type 252 and 253 are only used for TxPDO. The transmission type 252 means that the data is updated (but not sent) immediately after reception of the SYNC object. For these two transmission types, the PDO is only transmitted on remote transmission requests.
3. For the transmission types 254 and 255, the event timer will be used in the TxPDO. The PDO, including the DI value, will be sent when the DI value is changed. And both transmission types will directly trigger an update of the mapped data when receiving the RxPDO.

5.2.3 PDO Communication Rule

The PDO related objects are indicated from index 0x1400 to 0x1BFF. For the CAN-2000C, RxPDO communication objects are from index 0x1400 to index 0x140F, and RxPDO mapping objects are from index 0x1600 to index 0x160F. The ranges of the TxPDO communication objects and the mapping objects are from index 0x1800 to index 0x180F and from index 0x1A00 to index 0x1A0F respectively. Moreover, each PDO communication object has its own PDO mapping object.

For example, the first RxPDO communication object is stored in the entry with index 0x1400, and the corresponding mapping object is stored in an entry with index 0x1600. The object with index 0x1401 and the object with index 0x1601 are a group, and so on. The TxPDO also follows the same rules. The first TxPDO communication object is stored in the entry with 0x1800, and the corresponding mapping object is in the 0x1A00 entry, and so on. Therefore, before users access the practical I/O channels via PDO communication, each parameter for the PDO communications and mapping objects must be controlled.

Besides, only PDO communications can be used in the NMT operational state. Users can use the NMT module control protocol to change the NMT state of the CAN-2000C. It is described in the section 5.4. Besides, during communication via the PDO messages, the data length of the PDO message must match with the PDO mapping object. If the data length ‘L’ of the PDO message exceeds the total bytes ‘n’ of the PDO mapping object entries, only the first 'n' bytes of the PDO message are used by the PDO consumer. If ‘L’ is less than 'n', the PDO message will not be disposed by the PDO consumer, and an Emergency message with error code 8210h will be transmitted to the PDO producer. The PDO communication set is shown as follows.

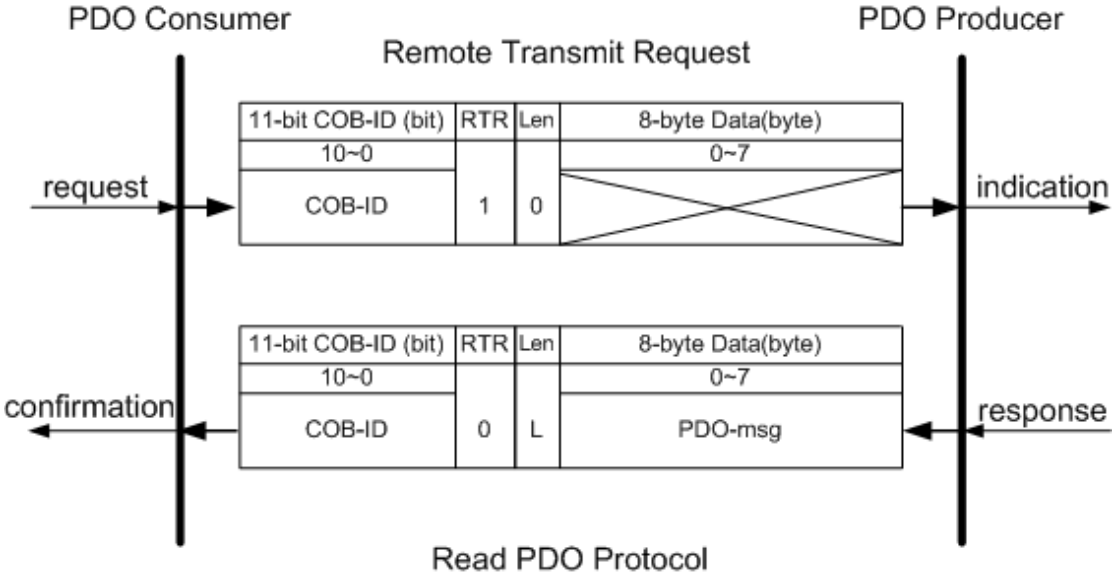


Write PDO Protocol

COB-ID : the default PDO COB-ID, or the PDO COB-ID can be defined by user

L : the data length about how many bytes the PDO message has

PDO-msg : the real-time data or the data which can be mapped into the PDO mapping objects



Read PDO Protocol

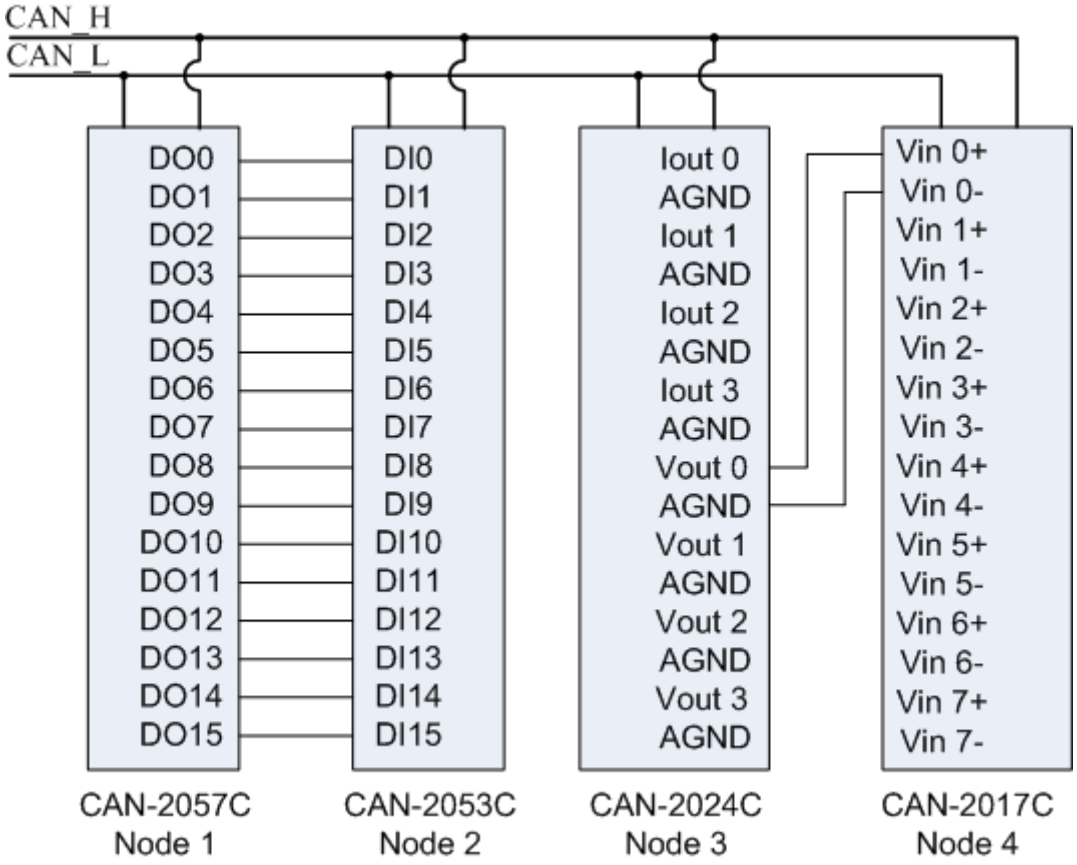
COB-ID : the default PDO COB-ID, or the PDO COB-ID defined by users

L : the data length about how many bytes the PDO message has

PDO-msg : the real-time data or the data which can be mapped into the PDO mapping objects

PDO Communication Example

To take a look at a PDO communication demo, some CAN-2000C modules will be needed. They are CAN-2057C, CAN-2053C, CAN-2024C and CAN-2017C. Connect each I/O channels for these modules as following figure.



Please use rotary switch to set the node ID from 1 to 4, and CAN bus baud rate to 125Kbps. Moreover, refer to manuals of these modules to set the CAN-2024C and CAN-2017C input/output range to -10V~+10V. The four modules' information is as below.

PDO information:

Node	Name	PDO Type	COB-ID	Transmission	Inhibit Time	Event Timer
1	CAN-2057C	Rx PDO	0x201	0xFF	0x00	0x00
3	CAN-2024C	Rx PDO	0x303	0xFF	0x00	0x00
2	CAN-2053C	Tx PDO	0x182	0xFF	0x00	0x00
4	CAN-2017C	Tx PDO	0x284	0xFF	0x00	0x00
4	CAN-2017C	Tx PDO	0x384	0xFF	0x00	0x00

PDO I/O mapping information:

COB-ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x201	DO 0~7	DO 8~15						
0x303	AO 00_L	AO 0_H	AO 1_L	AO 1_H	AO 2_L	AO 2_H	AO 3_L	AO 3_H
0x182	DI 0~7	DI 8~15						
0x284	AI 0_L	AI 0_H	AI 1_L	AI 1_H	AI 2_L	AI 2_H	AI 3_L	AI 3_H
0x384	AI 4_L	AI 4_H	AI 5_L	AI 5_H	AI 6_L	AI 6_H	AI 7_L	AI 7_H

SDO I/O channels Information:

Name	CAN-2057C	CAN-2024C	CAN-2053C	CAN-2017C
Node	1	3	2	4
Index	0x6200	0x6411	0x6000	0x6401
Description	DO	AO	DI	AI
Sub-Index 0	2	4	2	8
Sub-Index 1	00 ~ 07 ch	00 ch	00 ~ 07 ch	00 ch
Sub-Index 2	07 ~ 15 ch	01 ch	07 ~ 15 ch	01 ch
Sub-Index 3		02 ch		02 ch
Sub-Index 4		03 ch		03 ch
Sub-Index 5				04 ch
Sub-Index 6				05 ch
Sub-Index 7				06 ch
Sub-Index 8				07 ch

After concluding the above preparations, the several functions of PDO communication will be introduced as follows.

- The function of accessing digital/analog I/O with asynchronous PDO.
- The function by using Event Timer to obtain the input value.
- The function of the acyclic and synchronous RxPDO.
- The function of the acyclic and synchronous TxPDO.
- The function of the cyclic and synchronous TxPDO.
- The function of the synchronous and RTR-only TxPDO.
- The function of the asynchronous and RTR-only RxPDO.
- The function of the dynamic PDO mapping for DI/AI/DO/AO channels

Before describing the example, the step0 must be checked. And the default COB-ID for each communication object is assumed to be being used.

Step0: The following message must be sent in order to change the NMT state of the CAN-2000C first, because only the PDO communication can run under the NMT Operational state.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	8	01	01	00	00	00	00	00	00

NMT master → **NMT slave (CAN-2000C)**

cs : 1

Node ID : 1

- **Access Digital I/O & Analog I/O**

Step 1. In order to change the DO value of the CAN-2057C to be 0x1234, users must send the PDO message by using the first RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	8	34	12	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2057C)**

COB-ID : 0x201

L : 8

PDO-msg : 34 12 00 00 00 00

Only the first two bytes are valid, even if the **L** is set to 8, because the data in the first RxPDO contains only two bytes. According to the PDO mapping table shown above, the first byte is for the DO0~DO7 channel values of the CAN-2057. The second byte is for the DO8~DO15 channel values of the CAN-2057C.

Step 2. Because of the change of the DI-channel status, the TxPDO is transmitted automatically when the transmission type is 255, based on the CANopen spec 401. Then, users will receive the 1st TxPDO message.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	34	12	--	--	--	--	--	--

PDO consumer ← **PDO consumer (CAN-2053C)**

COB-ID : 0x182

L : 2

PDO-msg : 34 12 -- -- -- -- -- --

Because the data length is 2, only the first two bytes are valid. The DI value will be 1 if the DI is OFF, according to the character of the CAN-2053C DI channels. Therefore, the first byte indicates that the DI2, DI4, and DI5 of the CAN-2053C are in ON state. The second byte shows that the DI9 and DI12 of the CAN-2053C are in ON state.

Step 3. In order to output 5V to the AO0 of the CAN-2024C, users must send the PDO message by using the 2nd RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	1	0	0	0	0	0	0	1	1	0	8	FF	3F	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2024C)**

COB-ID : 0x303

L : 8

PDO-msg : FF 3F 00 00 00 00 00 00

The first two bytes are for AO channel 0, and the others are for AO channel 1, 2 and 3. Users need to transfer the float value to hex format because only the CAN-2000C supports the hex format. The output range of the CAN-2024C is -10V~10V. According to the transformation table stored in the appendix table, the mapping hex-format range is from 0x8000 (-32768) to 0x7FFF (32767). Therefore, the 5V is mapped to the 0x3FFF by applying following equation.

$$HexValue = \left(\frac{5V - (-10V)}{10V - (-10V)} \right) * \left(32767 - (-32768)\right) + (-32768)$$
$$= 16383.25 \approx 16383 = 0x3FFF$$

The first two bytes of the PDO message will be filled with “FF” and “3F”. All the other AI channels are set to 0V. Then, the value “00 00” will be given for these channels. For more details about how to transfer the value between the hex and float, please refer to the user manual of the I/O module of CAN-2000C series products.

Step 4. Even the AI input value has been changed according the AO value, the RxPDO will not respond automatically in the CAN-2017C. Therefore, users need to use the RTR message from the 2nd TxPDO to read back the AI value.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	1	0	0	0	0	1	0	0	1	0	00	00	00	00	00	00	00	00

PDO consumer → **PDO consumer (CAN-2017C)**

COB-ID : 0x284

Step 5. The feedback value for AI is 5V.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	1	0	0	0	0	0	0	1	0	8	00	40	FD	FF	FD	FF	FD	FF

PDO consumer ← **PDO consumer (CAN-2017C)**

COB-ID : 0x284

L : 8

PDO-msg : 00 40 FD FF FD FF FD FF

The first two bytes are for AI channel 0. The others are for AI channel 1, 2, and 3. The feedback AI0 value is 0x4000. All the other AI channels are 0xFFFD. Users need to transfer this AI0 value to float. The CAN-2017C's input float range is set to -10V ~ +10V and the input hex range is from 0x8000 (-32768) to 0x7FFF (32767). The value 0x4000 (16384) can be transferred by using the following equation.

$$FloatValue = \left(\frac{16384-(-32768)}{32767-(-32768)}\right) * (10V-(-10V)) + (-10V)$$

$$\approx 5.001V$$

- **Event Timer Functionality**

Step 6. Users can use the SDO to change the event timer of the 2nd TxPDO to 1000, stored in index 0x1801 with sub-index 5. In addition, the value 1000 means 1 second according to the event timer is ms,

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	1	0	0	0	8	2B	01	18	05	E8	03	00	00

SDO client → **SDO server (CAN-2017C)**

ccs : 1
n : 2
e : 1
s : 1
m : 01 18 05
d : E8 03 00 00
The value 0x03E8 is equal to 1000. Because the **n**=2, the last two bytes “00 00” is useless.

Step 7. The CAN-2017C will response the message to finish the data download.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	1	0	0	0	4	60	01	18	05	--	--	--	--

SDO client ← **SDO server (CAN-2017C)**

scs : 3
m : 01 18 05

Step 8. After changing the value of the event timer, the AI value will be automatically transmitted per second. The example below shows that at the first time the 2n TxPDO message is received.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	1	0	0	0	0	1	0	0	0	8	00	40	FD	FF	FF	FF	FF	FF

PDO consumer ← **PDO producer (CAN-2017C)**

COB-ID : 0x284

L : 8

PDO-msg : 00 40 FD FF FF FF FF FF

Step 9. The following example shows that at the second time the 2nd TxPDO message is received.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	1	0	0	0	0	1	0	0	0	8	06	40	FF	FF	FF	FF	FF	FF

PDO consumer ← **PDO producer (CAN-2017C)**

COB-ID : 0x284

L : 8

PDO-msg : 06 40 FF FF FF FF FF FF

The value of 0x4006 is equal to 5.002V. The AI value is changed because of the noise disturbance or other factors.

Step 10. It shows that at the third time for the 2nd TxPDO message is received.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	1	0	0	0	0	1	0	0	0	8	00	40	FF	FF	FD	FF	FF	FF

PDO consumer ← **PDO producer (CAN-2017C)**

COB-ID : 0x284
L : 8
PDO-msg : 00 40 FF FF FD FF FF FF

Step 11. Users can set the event timer to 0 to finish the event timer test.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	1	0	0	0	8	2B	01	18	05	00	00	00	00

SDO client → **SDO server (CAN-2017C)**

ccs : 1
n : 2
e : 1
s : 1
m : 01 18 05
d : 00 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	1	0	0	0	4	60	01	18	05	--	--	--	--

SDO client ← **SDO server (CAN-2017C)**

scs : 3
m : 01 18 05

- **Transmission Type 0 for the first RxPDO**

Step 12. Users can set the transmission type of the first RxPDO to 0.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	2F	00	14	02	00	00	00	00

SDO client → **SDO server (CAN-2057C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 14 02
d : 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	4	60	00	14	02	--	--	--	--

SDO client ← **SDO server (CAN-2057C)**

scs : 3
m : 00 14 02

Step 13. Change the DO value of the CAN-2057C to be 0x5678 by using the 1st RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	8	78	56	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2057C)**

COB-ID : 0x201
L : 8
PDO-msg : 78 56 00 00 00 00 00 00

Step 14. The DO value isn't changed immediately according to the character of the transmission type 0. Meanwhile, the SYNC message is needed to trigger the action of the 1st RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0	0	0	--	--	--	--	--	--	--	--

SYNC producer → **SYNC consumer (CAN-2057C)**

COB-ID : 0x80

The message of the SYNC object is always fixed as the format described above. The COB-ID of the SYNC object can be changed arbitrarily. It complies with the producer/consumer relationship.

Step 15. After transmitting the SYNC object, the 1st RxPDO is triggered. The DI value is also changed at the same time. Hence, users can receive the 1st TxPDO from CAN-2053C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	78	56	--	--	--	--	--	--

PDO consumer ← **PDO producer (CAN-2053C)**

COB-ID : 0x182

L : 2

PDO-msg : 78 56 -- -- -- -- -- --

Step 16. Users can set the transmission type of the first RxPDO to 255 to finish the test.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	2F	00	14	02	FF	00	00	00

SDO client → **SDO server (CAN-2057C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 14 02
d : FF 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	4	60	00	14	02	--	--	--	--

SDO client ← **SDO server (CAN-2057C)**

scs : 3
m : 00 14 02

- **Transmission Type 0 for the first TxPDO**

Step 17. Users can set the transmission type of the first TxPDO to 0.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	0	0	8	2F	00	18	02	00	00	00	00

SDO client ⟶ **SDO server (CAN-2053C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 18 02
d : 00 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	0	0	4	60	00	18	02	--	--	--	--

SDO client ⟵ **SDO server (CAN-2053C)**

scs : 3
m : 00 18 02

Step 18. Users can change the DO value of the CAN-2057C to be 0x90AB by using the first RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	8	AB	90	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2057C)**

COB-ID : 0x201
L : 8
PDO-msg : AB 90 00 00 00 00 00 00

Step 19. The first TxPDO will not be transmitted immediately even if the DI value is changed according to the character of the transmission type 0. In addition, the SYNC message is needed to trigger the action of the first TxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0	0	0	---	---	---	---	---	---	---	---

SYNC producer → **SYNC consumer (CAN-2053C)**

COB-ID : 0x80

Step 20. After transmitting the SYNC object, the 1st TxPDO is triggered, and users can receive the 1st TxPDO from CAN-2053C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	90	AB	--	--	--	--	--	--

PDO consumer ← **PDO producer (CAN-2053C)**

COB-ID : 0x182
L : 2
PDO-msg : 90 AB -- -- -- -- -- --

Step 21. Users can send the SYNC message again.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0	0	0	---	---	---	---	---	---	---	---

SYNC producer → **SYNC consumer (CAN-2053C)**

SYNC COB-ID : 0x80

Step 22. Nothing happened because the DI values were not changed. This is the main difference between transmission type 0 and 1. (Under the transmission type 1, the TxPDO is always transmitted no matter whether the DI values are changed or not, when the CAN-2000C modules receives the SYNC object.)

- **Transmission Type 3 for the first TxPDO**

Step 23. Users can set the transmission type of the first TxPDO to 3.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	0	0	8	2F	00	18	02	03	00	00	00

SDO client → **SDO server (CAN-2053C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 18 02
d : 03 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	0	0	4	60	00	18	02	--	--	--	--

SDO client ← **SDO server (CAN-2053C)**

scs : 3
m : 00 18 02

Step 24. Users can change the DO value of the CAN-2057C to be 0xCDEF by using the first RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	8	EF	CD	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2057C)**

COB-ID : 0x201

L : 8

PDO-msg : EF CD 00 00 00 00 00 00

Step 25. The SYNC message has to be transmitted in 3 times according to the character of transmission type 3.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0	0	0	---	---	---	---	---	---	---	---

SYNC producer → **SYNC consumer (CAN-2053C)**

COB-ID : 0x80

Step 26. After finishing the transmission of the three SYNC objects, the first TxPDO will be triggered, and users will receive the first TxPDO from CAN-2053C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	EF	CD	---	---	---	---	---	---

PDO consumer ← **PDO producer (CAN-2053C)**

COB-ID : 0x182

L : 2

PDO-msg : EF CD -- -- -- -- -- --

- **Transmission Type 252 for the first TxPDO**

Step 27. Users can set the transmission type of the first TxPDO to 252.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	0	0	8	2F	00	18	02	FC	00	00	00

SDO client ⟶ **SDO server (CAN-2053C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 18 02
d : FC 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	0	0	4	60	00	18	02	--	--	--	--

SDO client ⟵ **SDO server (CAN-2053C)**

scs : 3
m : 00 18 02

Step 28. Users can change the DO value of the CAN-2057C to be 0x1234 by using the first RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	8	34	12	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2057C)**

COB-ID : 0x201

L : 8

PDO-msg : 34 12 00 00 00 00 00 00

Step 29. The fist TxPDO will not be transmitted immediately according to the transmission type 252. Meanwhile, it will send the RTR message of the first TxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	1	0	---	---	---	---	---	---	---	---

PDO consumer → **PDO producer (CAN-2053C)**

COB-ID : 0x182

Step 30. The feedback DI values is out-of-date. (Users can see the LEDs status on the CAN-2053C to confirm the practical DI values).

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	34	12	00	00	00	00	00	00

PDO consumer ← **PDO producer (CAN-2053C)**

COB-ID : 0x182
L : 2
PDO-msg : 34 12 00 00 00 00 00 00

Step 31. Transmit a SYNC message.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0	0	0	---	---	---	---	---	---	---	---

SYNC producer → **SYNC consumer (CAN-2053C)**

COB-ID : 0x80

Step 32. Users can send the RTR message of the first TxPDO again.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	1	0	---	---	---	---	---	---	---	---

PDO consumer → **PDO producer (CAN-2053C)**

COB-ID : 0x182

Step 33. The feedback DI values will be the real DI values.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	34	12	---	---	---	---	---	---

PDO consumer ← **PDO producer (CAN-2053C)**

COB-ID : 0x182

L : 2

PDO-msg : 34 12 -- -- -- -- -- --

- **Transmission Type 253 for the first TxPDO**

Step 34. Users can set the transmission type of the first TxPDO to 253.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	0	0	8	2F	00	18	02	FD	00	00	00

SDO client → **SDO server (CAN-2053C)**

ccs : 1
n : 3
e : 1
s : 1
m : 00 18 02
d : FD 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	0	0	4	60	00	18	02	--	--	--	--

SDO client ← **SDO server (CAN-2053C)**

scs : 3
m : 00 18 02

Step 35. Users can change the DO value of the CAN-2057C to be 0x5678 by using the first RxPDO.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	8	78	56	00	00	00	00	00	00

PDO producer → **PDO consumer (CAN-2057C)**

COB-ID : 0x201
L : 8
PDO-msg : 78 56 00 00 00 00 00 00

Step 36. According to the transmission type 253, only the first TxPDO can be transmitted when receiving the RTR message. So, users can send the RTR message to get DI values. Then, the CAN-2053C will reply with digital input status.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	1	0	---	---	---	---	---	---	---	---

PDO consumer ⟶ **PDO producer (CAN-2053C)**

COB-ID : 0x182

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	0	1	0	0	2	78	56	---	---	---	---	---	---

PDO consumer ⟵ **PDO producer (CAN-2053C)**

COB-ID : 0x182

L : 2

PDO-msg : 78 56 -- -- -- -- -- --

Step 37. Set the transmission type of the 1st TxPDO to 255 to finish the test.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	0	0	8	2F	00	18	02	FF	00	00	00

SDO client ⟶ **SDO server (CAN-2053C)**

ccs : 1

n : 3

e : 1

s : 1

m : 00 18 02

d : FF 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	0	0	4	60	00	18	02	--	--	--	--

SDO client ← **SDO server (CAN-2053C)**

scs : 3

m : 00 18 02

- **Dynamic PDO Mapping for DI/AI/DO/AO Channels**

Step 38. Users can use the 5th TxPDO of CAN-2017C to create a new PDO communication with PDO COB-ID 0x333, which is useless for others CANopen device for AI channel 2 and 5. Before setting the COB-ID of a PDO, users have to check the bit 31 of the COB-ID first. Only the COB-ID with the value 0 on the bit 31 can be changed. So the COB-ID can be configured directly according to the 5th TxPDO is invalid.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	1	0	0	0	8	23	05	18	01	33	03	00	00

SDO client → **SDO server (CAN-2017C)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 18 01
d : 33 03 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	1	0	0	0	4	60	05	18	01	---	---	---	---

SDO client ← **SDO server (CAN-2017C)**

scs : 3
m : 05 18 01

Step 39. Users can create a new PDO mapping object for the 5th TxPDO. Before getting the device objects into the index 0x1A05, users have to check the value of the index 0x1A05 with sub-index 00. If the value is not equal to 0, any modification will be rejected. In this case, it is necessary to have the value in 0. Therefore, users have to fill the AI2 and AI5 of the CAN-2017C into the index 0x1A05 with sub-index 01 and 02.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	1	0	0	0	8	23	05	1A	01	10	03	01	64

SDO client ⟶ **SDO server (CAN-2017C)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 1A 01
d : 10 03 01 64

The value “10 03 01 64” means the mapped object is stored in the index 0x6401 with sub-index 03. It is a 16-bit data unit. Users can check this object in the Standardize object mapping table described above. It is mapped according to the AI2 of the CAN-2017C. In CAN-2017C, all analog channels are presented by 16-bit value.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	1	0	0	0	4	60	05	1A	01	---	---	---	---

SDO client ⟵ **SDO server (CAN-2017C)**

scs : 3
m : 05 1A 01

Step 40. According to the purposes, users have to fill the AI5 of the CAN-2017C into the index 0x1A05 with sub-index 02 respectively.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	1	0	0	0	8	23	05	1A	02	16	06	01	64

SDO client ⟶ **SDO server (CAN-2017C)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 1A 02
d : 10 06 01 64

The value “10 06 01 64” means the mapped object is stored in the index 0x6401 with sub-index 06. It is a 16-bit data unit. Users can check this object in the Standardize object mapping table described above. It is mapped according to the AI5 of the CAN-2017C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	1	0	0	0	4	60	05	1A	02	---	---	---	---

SDO client ⟵ **SDO server (CAN-2017C)**

scs : 3
m : 05 1A 02

Step 41. In order to use this PDO mapping object normally, the value of the index 0x1A05 with sub-index 00 must be changed to 2. The value 2 means there are 2 objects mapped to the 5th TxPDO. They are the index 0x6401 with sub-index 03 and index 0x6401 with sub-index 06.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	1	0	0	0	8	2F	05	1A	00	02	00	00	00

SDO client → **SDO server (CAN-2017C)**

ccs : 1
n : 3
e : 1
s : 1
m : 05 1A 00
d : 02 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	1	0	0	0	4	60	05	1A	00	---	---	---	---

SDO client ← **SDO server (CAN-2017C)**

scs : 3
m : 05 1A 00

Step 42. Users can use the 5th RxPDO of CAN-2024C to create a new PDO communication with PDO COB-ID 0x222 for AO2 and AO5, and create the RxPDO mapping object in the index 0x1605 because the COB-ID 0x222 is not available for the CAN-2024C. This procedure is similar to the steps 37 to 40.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	1	0	8	23	05	14	01	22	02	00	00

SDO client → **SDO server (CAN-2024C)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 14 01
d : 22 02 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	1	0	4	60	05	14	01	---	---	---	---

SDO client ← **SDO server (CAN-2024C)**

scs : 3
m : 05 14 01

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	1	0	8	23	05	16	01	16	03	11	64

SDO client → **SDO server (CAN-2024C)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 16 01
d : 10 03 11 64

The value “10 03 11 64” means the mapped object is stored in the index 0x6411 with sub-index 03. It is a 16-bit data unit. Users can check this object in the Standardize object mapping table described above. It is mapped to the AO2 for CAN-2024C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	1	0	4	60	05	16	01	---	---	---	---

SDO client ⟵ **SDO server (CAN-2024C)**

scs : 3
m : 05 16 01

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	1	0	8	23	05	16	02	16	06	11	64

SDO client ⟶ **SDO server (CAN-2024C)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 16 02
d : 10 06 11 64

The value “10 06 11 64” means the mapped object is stored in the index 0x6411 with sub-index 06. It is a 16-bit data unit. Users can check this object in the Standardize object mapping table described above. It is mapped according to the AO5 of the CAN-2024C.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	1	0	4	60	05	16	02	---	---	---	---

SDO client ⟵ **SDO server (CAN-2024C)**

scs : 3
m : 05 16 02

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	1	1	0	8	2F	05	16	00	02	00	00	00

SDO client → **SDO server (CAN-2024C)**

ccs : 1
n : 3
e : 1
s : 1
m : 05 16 00
d : 02 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	1	1	0	4	60	05	16	00	---	---	---	---

SDO client ← **SDO server (CAN-2024C)**

scs : 3
m : 05 16 00

Step 43. Transform the AO2 and AO5 of CAN-2024C to be 0V and 5V respectively.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	1	0	0	0	1	0	0	4	00	00	FF	3F	---	---	---	---

PDO consumer → **PDO producer (CAN-2024C)**

COB-ID : 0x222
PDO-msg : 00 00 FF 3F -- -- -- --

The first two bytes are assigned to the value 0x0000 of the AO2 of the CAN-2024C. The 3rd and 4th bytes are assigned to the value 0x3FFF for the AO5 of the CAN-2024C. Total bytes of this PDO message are 4.

Step 44. Users can send a RTR message to get the AI value with COB-ID 0x333.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	1	0	0	1	1	0	0	1	1	1	0	---	---	---	---	---	---	---	---

PDO consumer ⟵ **PDO producer (CAN-2017C)**

COB-ID : 0x333

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	1	1	0	0	0	0	1	0	1	0	4	FF	FF	00	40	---	---	---	---

PDO consumer ⟵ **PDO producer (CAN-2017C)**

COB-ID : 0x333

L : 4

PDO-msg : FF FF 00 40 -- -- -- --

The first two bytes are assigned to the value 0xFFFF for the AI2 of the CAN-2017C. The 3rd and 4th bytes are assigned to the value 0x4000 for the AI5 of the CAN-2017C. After transferring, the input value of the AI2 is -0.001V and AI5 is 5.000V.

5.3 EMCY Communication Set

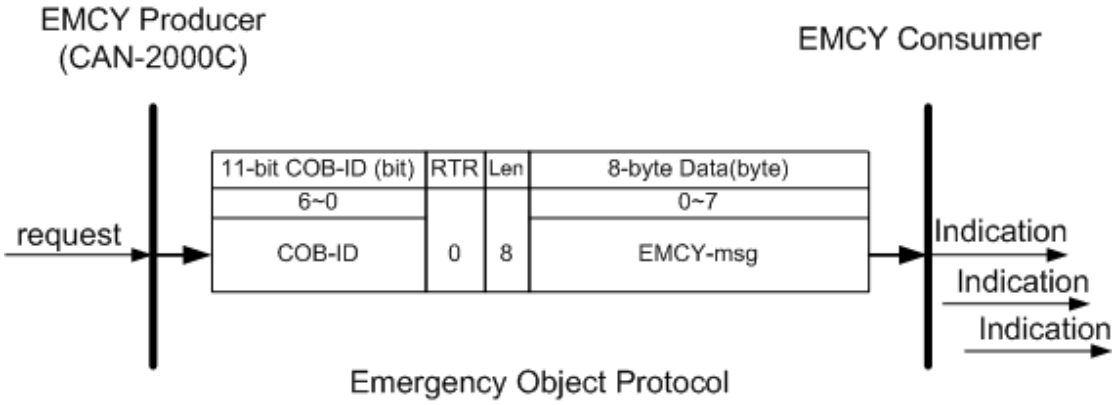
5.3.1 EMCY COB-ID Parameter

The EMCY COB-ID is similar to the PDO COB-ID. It can be a default value or can be the value defined by users via SDO communication methods. This COB-ID is stored in the object 0x1014, and the data format is shown in the following table. Before using the EMCY mechanism, bit 31 of the EMCY COB-ID needs to be confirmed.

Bit Number	Value	Meaning
31 (MSB)	0	EMCY exits (EMCY is valid)
	1	EMCY does not exist (EMCY is not valid)
30	0	reserved (always 0)
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28-11	0	If bit 29=0
	x	If bit 29=1: 28-11 bits of 29-bit COB-ID
10-0 (LSB)	x	10-0 bits of COB-ID

5.3.2 EMCY Communication

The EMCY message is triggered when some internal error occurs. After the transmission of one EMCY message, the object with index 0x1003 will record this EMCY event. Therefore, users can track the error's occurrences. The CAN-2000C supports the maximum of 5 records stored in the index 0x1003 object. The sub-index 1 of this object will store the last EMCY event, and sub-index 5 will record the most previous EMCY event. The EMCY communication set is given below.



COB-ID : the EMCY COB-ID
The EMCY COB-ID can be defined by users. This situation is similar to the PDO COB-ID. The default value is 4-bit function code "0001" with 7-bit node ID.

EMCY-msg : record the type or the class of the occurrence error

The data format of the emergency object data complies with the structure bellows.

Byte	0	1	2	3	4	5	6	7
Content	Emergency Error Code		Error register	Manufacturer specific Error Field				

Each bit on the error register is defined as follows. Only the CAN-2000C supports bit 0, bit 4 and bit 7.

Bit	Meaning
0	generic error
1	current
2	voltage
3	temperature
4	communication error (overrun, error state)
5	device profile specific
6	reserved (always 0)
7	manufacturer specific

The emergency error codes and the error register are specified in the following table.

Emergency Error Code		Error Register	Manufacturer Specific Error Field			Description
High Byte	Low Byte		First Two Byte		Last Three Byte	
00	00	00	00	00	00 00 00	Error Reset or No Error
10	00	81	01	00	00 00 00	CAN Controller Error Occur
50	00	81	02	00	00 00 00	EEPROM Access Error
81	10	11	04	00	00 00 00	Soft Rx Buffer Overrun
81	10	11	05	00	00 00 00	Soft Tx Buffer Overrun
81	10	11	06	00	00 00 00	CAN Controller Overrun
81	30	11	07	00	00 00 00	Lift Guarding Fails
81	40	11	08	00	00 00 00	Recover from bus off
82	10	11	09	00	00 00 00	PDO Data length Error
FF	00	80	0A	00	00 00 00	Request to reset Node or communication

After producing the EMCY message, the emergency object data will be saved to the object with index 0x1003, and the error register of the emergency object data will be mapped to object 0x1001. Therefore, users can use these two objects to view what happened in the CAN-2000C and check the error history.

EMCY Communication Example

Before starting the example, a CAN-2000C module with more than 8-channels DO or 1-channel AO, like CAN-2057C is needed. Here, the same hardware configuration shown in the PDO example is used for the EMCY communication.

Step 1. In order to generate the emergency event, it's necessary to send the data to RxPDO1 with data length 1.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	1	00	---	---	---	---	---	---	---

PDO consumer → **PDO producer (CAN-2057C)**

COB-ID : 0x201

L : 1

PDO-msg : 00

Step 2. Then, the CAN-2057C will reply to an emergency message based on the PDO data length of TxPDO1 doesn't correspond to the value defined in the PDO mapping object.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	1	0	8	10	82	11	09	00	00	00	00

EMCY consumer ← **EMCY producer (CAN-2057C)**

COB-ID : 0x81

EMCY-msg : 10 82 11 09 00 00 00 00

The first two bytes "10 82" are for the emergency error codes. The 3rd byte "11" is for the error register, i.e. the CAN-2057C has either a communication or a generic error. The last five bytes "09 00 00 00 00" are for the manufacturer specific errors. This emergency message means that the data length of TxPDO doesn't correspond to the value defined in the PDO mapping object.

Step 3. After recognizing the 0x1003 object with sub-index 01, users will get emergency error codes of the emergency object data recording in this object.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	03	10	01	00	00	00	00

SDO client → **SDO server (CAN-2057C)**

ccs : 2
m 03 10 01

Step 4. The CAN-8423 will reply to the ending message.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	43	03	10	01	10	82	09	00

SDO client ← **SDO server (CAN-2057C)**

scs : 2
n : 0
e : 1
s : 1
m : 03 10 01
d : 10 82 09 00

Step 5. Users have to check the object 0x1001, and make sure that the communication and generic errors on the error register are indicated.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	01	10	00	00	00	00	00

SDO client → **SDO server (CAN-2057C)**

ccs : 2
m 01 10 00

Step 6. The communication and generic errors on the error register are indicated in the received message.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	5	4F	01	10	00	11	---	---	---

SDO client ⟵ **SDO server (CAN-2057C)**

scs : 2
n : 3
e : 1
s : 1
m : 01 10 00
d : 11 -- -- --

Step 7. Users can send the data to RxPDO1 with data length 2. Then, the EMCY message containing the error reset information will be received. Because the value of TxPDO is the same with the previous one, the DO channels will not change.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0	0	1	0	2	00	00	---	---	---	---	---	---

PDO consumer ⟶ **PDO producer (CAN-2057C)**

COB-ID : 0x201
L : 2
PDO-msg : 00 00 -- -- -- -- -- --

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	1	0	8	00	00	00	00	00	00	00	00

NMT master ⟵ **NMT slaver (CAN-2057C)**

EMCY-msg : 00 00 00 00 00 00 00 00
The data “00 00 00 00 00 00 00 00” are for the error reset EMCY message, i.e. CAN-2057C has no error now.

Step 9. Users have to check the index 0x1003 with sub-index 01 again. Then, the error reset emergency code should be recorded.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	03	10	01	00	00	00	00

SDO client ⟶ **SDO server (CAN-2057C)**

ccs : 2
m 03 10 01

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	43	03	10	01	---	---	---	---

SDO client ⟵ **SDO server (CAN-2057C)**

ccs : 1
n : 0
e : 1
s : 1
m : 03 10 01
d : -- -- -- --

Step 10. Users have to check the index 0x1003 with sub-index 02. Then, the received emergency error code had been recorded in the emergency object data.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	03	10	02	00	00	00	00

SDO client ⟶ **SDO server (CAN-2057C)**

ccs : 2
m : 03 10 02

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	43	03	10	02	10	82	09	00

SDO client ← **SDO server (CAN-2057C)**

ccs : 1
n : 0
e : 1
s : 1
m : 03 10 02
d : 10 82 09 00

Step 11. Users have to confirm the error register stored in index 0x1001. The value should be 0 now.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	40	01	10	00	00	00	00	00

SDO client → **SDO server (CAN-2057C)**

ccs : 2
m 01 10 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	8	4F	01	10	00	00	---	---	---

SDO client ← **SDO server (CAN-2057C)**

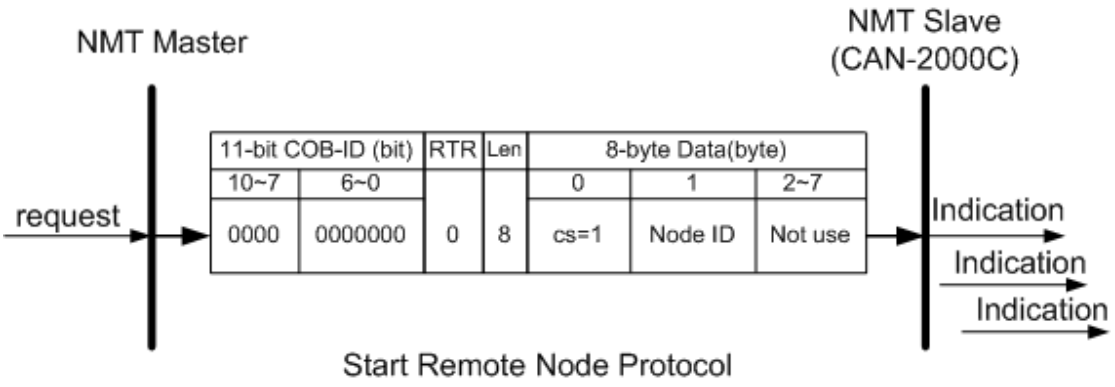
ccs : 1
n 2
e 1
s 1
m : 01 10 00
d : 00 -- -- --

5.4 NMT Communication Set

5.4.1 Module Control Protocol

The NMT communication set can be applied for changing the NMT slave status. The following figure shows how to change the different NMT statuses for the CAN-2000C.

Start Remote Node Protocol

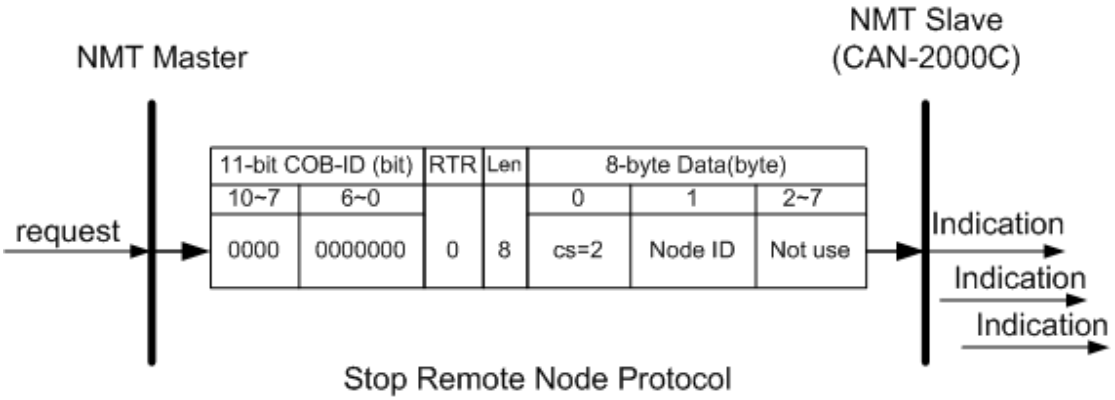


Start Remote Node Protocol

cs : NMT command specified
1: start

Node ID : the node ID of the NMT slave device

Stop Remote Node Protocol

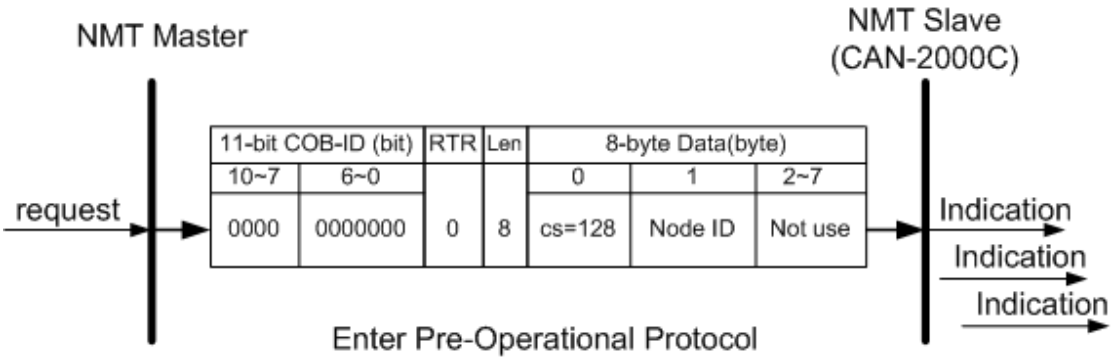


Stop Remote Node Protocol

cs : NMT command specified
2: stop

Node ID : the node ID of the NMT slave device

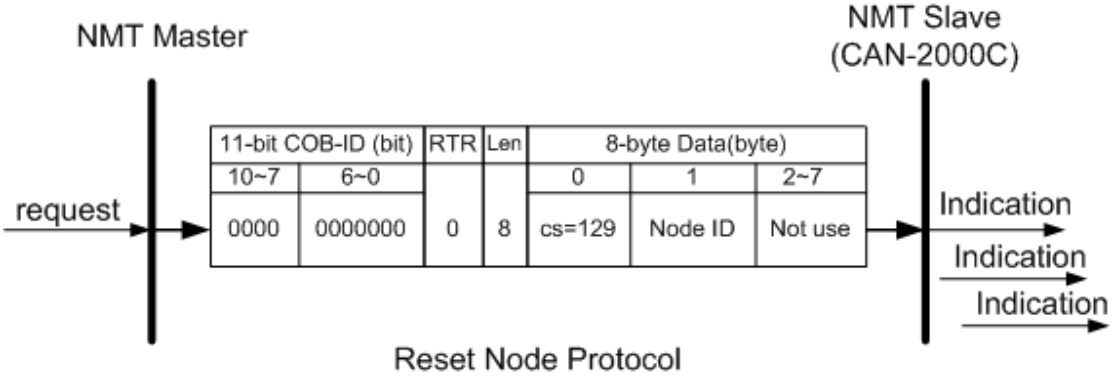
Enter Pre-Operational Protocol



cs : NMT command specified
128: enter PRE-OPERATIONAL

Node ID : the node ID of the NMT slave device

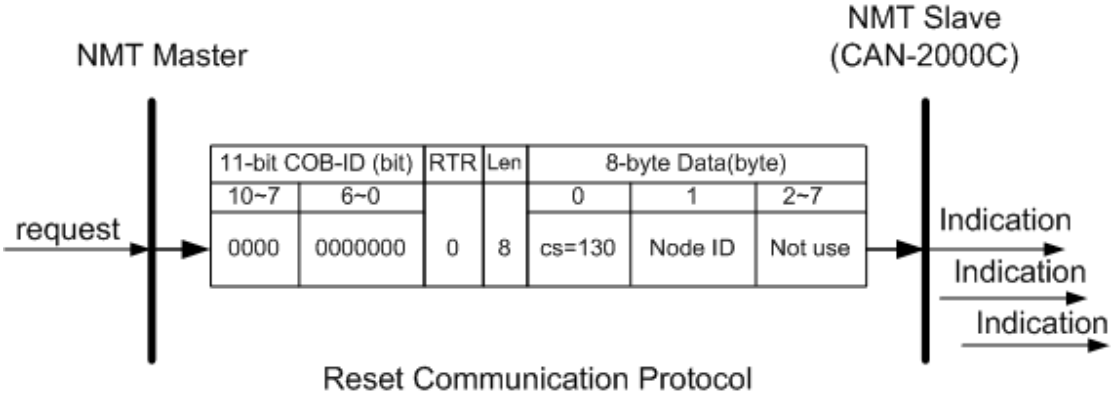
Reset Node Protocol



cs : NMT command specified
129: Reset_Node

Node ID : the node ID of the NMT slave device

Reset Communication Protocol



cs : NMT command specified
130: Reset_Communication
Node ID : the node ID of the NMT slave device

Module Control Protocol Example

If the CAN-2000C node ID is set to 5 as an example, the following steps would be…

Step1. Turn off the CAN-2000C.

Step2. Then, turn it on. After initialized, the CAN-2000C will automatically enter the Pre_Operational state. Users will note the RUN LED flashing twice per second.

Step3. Users can send the NMT module control protocol, and control the CAN-2000C to enter the operational state.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	8	01	05	00	00	00	00	00	00

NMT master → **NMT slave (CAN-2000C)**

cs : 1
Node ID : 5

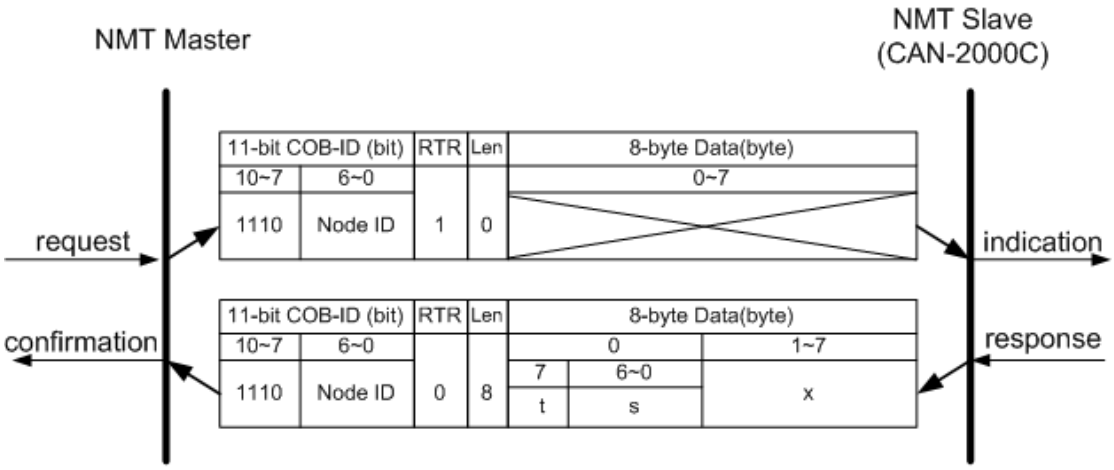
5.4.2 Error Control Protocol

Error Control Protocol is a kind of the solution to check whether the CANopen device is still alive or not. There are two protocols supported for Error Control Protocol, Node Guarding Protocol and Heartbeat Protocol. The related objects of Node Guarding Protocol include 0x100C and 0x100D. And the related objects of Heartbeat Protocol include 0x1017.

Note: It is not allowed for one CAN-2000C module to use both Node Guarding Protocol and Heartbeat Protocol at the same time.

Node Guarding Protocol

The 0x100C is the guard time, and the 0x100D is the life time factor. The node life time is the guard time multiplied by the life time factor. The Node Guarding timer of the CAN-2000C will start to count after receiving the first RTR message for the guarding identifier. The communication set of the Node Guarding Protocol is displayed below.



Node Guarding Protocol

t : toggle bit
The value of this bit will be alternatively changed between two consecutive responses from the NMT slave. After the node Guarding protocol becomes active, the value of the toggle-bit of the first response will be 0.

s : the state of the NMT Slave
4: STOPPED
5: OPERATIONAL
127: PRE-OPERATIONAL

Node Guarding Protocol Example

The default EMCY function code and node ID 1 for the CAN-2000C are used as an example on the error control protocol. The steps will be as follows.

Step 1. Turn off the CAN-2000C. Then, turn it on. The CAN-2000C will be in the pre_operational state.

Step 2. Users can set the guard time value to 250. This value will be stored in index 0x100C with sub-index 00.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	2B	0C	10	00	FA	00	00	00

SDO client → **SDO server (CAN-2000C)**

ccs : 1
n : 2
e : 1
s : 1
m : 0C 10 00
d : FA 00 00 00

Step 3. The CAN-2000C will reply with the ending message.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	4	60	0C	10	00	---	---	---	---

SDO client ← **SDO server (CAN-2000C)**

scs : 3
m : 0C 10 00

Step 4. Users can set the life-time factor value to 4. This value will be stored in the index 0x100D with sub-index 00. Then, the ending message from CAN-2000C will be received.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	2F	0D	10	00	04	00	00	00

SDO client ⟶ **SDO server (CAN-2000C)**

ccs : 1
n : 3
e : 1
s : 1
m : 0D 10 00
d : 04 00 00 00

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	4	60	0D	10	00	---	---	---	---

SDO client ⟵ **SDO server (CAN-2000C)**

scs : 3
m : 0D 10 00

Step 5. Users can send the node guarding protocol to start the mechanism of the node guard. The life time here is equal to 1000 ms (guard time * life time factor =250*4=1000),

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	1	0	0	0	0	0	0	0	1	1	0	---	---	---	---	---	---	---	---

NMT master ⟶ **NMT slaver (CAN-2000C)**

COB-ID : 0x701

Step 5. Then, users will receive the message, recording the NMT state of the CAN-2000C. For the reason that life time is equal to 1000 ms (guard time * life time factor =250*4=1000), users will transmit the node guarding protocol again.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	1	0	0	0	0	0	0	0	1	0	1	7F	---	---	---	---	---	---	---

NMT master ⟵ **NMT slaver (CAN-2000C)**

COB-ID : 0x701

t : 1

s 7F

The value 7F means that the CAN-2000C is in the NMT pre_operational state.

Step 6. Since the life time is equal to 1000 ms (guard time * life time factor =250*4=1000), users will transmit the node guarding protocol again.

Step 7. If the transmission is not available, an error event will be triggered, and an EMCY message for guarding failure will be received. Moreover, all values from the output channels will be changed according to index 0x6206, index 0x6207, index 0x6443, and index 0x6444.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	1	0	1	0	8	30	81	11	07	00	00	00	00

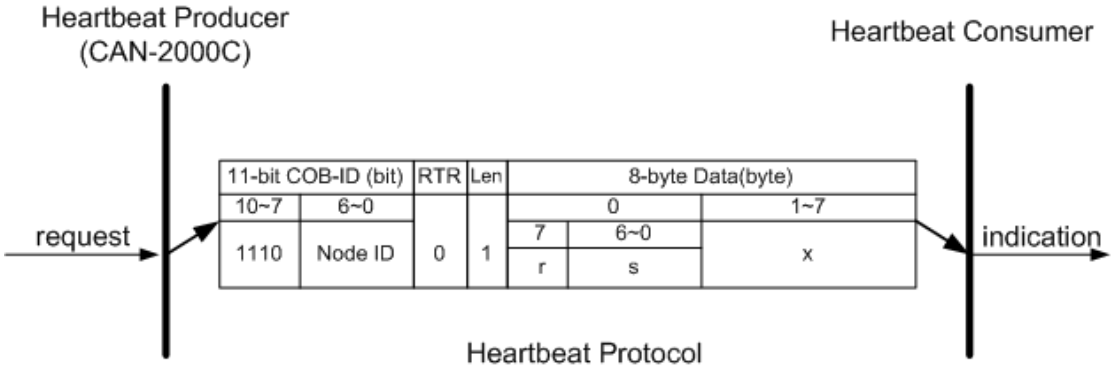
EMCY consumer ⟵ **EMCY producer (CAN-2000c)**

EMCY-msg : 30 81 11 07 00 00 00 00

The first two bytes “30 81” are for the emergency error code. The 3rd byte “11” is for the error register. The last five bytes “07 00 00 00 00” are for the manufacturer specific error values. This emergency message indicates a life guard error.

Heartbeat Protocol

The 0x1017 is the heartbeat time, and the Heartbeat Protocol defines an Error Control Service without need for remote frames. A CAN-2000C module will transmit Heartbeat message cyclically when the object 0x1017 is unequal to 0. The communication set of the Heartbeat Protocol is displayed below.



r : Reserved (always 0).

s : the state of the NMT Slave

4: STOPPED

5: OPERATIONAL

127: PRE-OPERATIONAL

Heartbeat Protocol Example

The default EMCY function code and node ID 1 for the CAN-2000C are used as an example on the error control protocol. The steps will be as follows.

Step 1. Turn off the CAN-2000C. Then, turn it on. The CAN-2000C will be in the pre_operational state.

Step 2. Users can set the heartbeat time value to 250 (ms). This value will be stored in index 0x1017 with sub-index 00.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	0	0	0	0	0	0	0	0	1	0	8	2B	17	10	00	FA	00	00	00

SDO client → **SDO server (CAN-2000C)**

ccs : 1
n : 2
e : 1
s : 1
m : 17 10 00
d : FA 00 00 00

Step 3. The CAN-2000C will reply with the ending message.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0	0	0	1	0	4	60	17	10	00	---	---	---	---

SDO client ← **SDO server (CAN-2000C)**

scs : 3
m : 17 10 00

Step 5. Then, users will receive the message, recording the NMT state of the CAN-2000C. For the reason that Heartbeat time is equal to 250 ms, and users can change the heartbeat time any time.

11-bit COB-ID (bit)											RTR	Data Length	8-byte Data (byte)							
Func Code				Node ID																
10	9	8	7	6	5	4	3	2	1	0			0	1	2	3	4	5	6	7
1	1	1	0	0	0	0	0	0	0	1	0	1	7F	---	---	---	---	---	---	---

Heartbeat consumer ← **Heartbeat producer (CAN-2000C)**

COB-ID : 0x701
t : 1
s 7F
The value 7F means that the CAN-2000C is in the NMT pre_operational state.

6 Object Dictionary of CAN-2000C

6.1 Communication Profile Area

The following tables are regarding each entry of the communication profile area is defined in CAN-2000C. For the convenient purpose, all communication entries are divided into several tables. They are “General Communication Entries”, “RxPDO Communication Entries”, “RxPDO Mapping Communication Entries”, “TxPDO Communication Entries”, and “TxPDO Mapping Communication Entries”.

Please note that In the table header with “Idx”, “Sidx” and “Attr” represent “index”, “sub-index”, and “attribute” respectively. The sign “---” in the default field means that the default is not defined or can be defined conditionally by the firmware built in CAN-2000C. In the table, the number accompanying letter “h” indicates that this value is in the hex format.

General Communication Entries

Idx	Sidx	Description	Type	Attr	Default
1000h	0h	device type	UNSIGNED 32	RO	---
1001h	0h	error register	UNSIGNED 8	RO	---
1003h	0h	largest sub-index supported for “predefine error field”	UNSIGNED 8	RO	0h
	1h	actual error (the newest one)	UNSIGNED 32	RO	---
	...	...	...	...	---
	5h	actual error (the oldest one)	UNSIGNED 32	RO	---
1005h	0h	COB-ID of Sync message	UNSIGNED 32	RW	80h
1008h	0h	manufacturer device name	VISIBLE_STRING	RO	---
1009h	0h	manufacturer hardware version	VISIBLE_STRING	RO	---
100Ah	0h	manufacturer software version	VISIBLE_STRING	RO	---
100Ch	0h	guard time	UNSIGNED 16	RW	0h
100Dh	0h	life time factor	UNSIGNED 8	RW	0h
1010h	0h	largest sub-index supported for “store parameters”	UNSIGNED 8	RO	1h
1010h	1h	save all hardware parameter	UNSIGNED 32	RW	---
1011h	0h	largest sub-index supported for	UNSIGNED 8	RO	1h

		“restore default parameters”			
1011h	1h	restore all default parameters	UNSIGNED 32	RW	---
1014h	0h	COB-ID of EMCY	UNSIGNED 32	RW	80h+Node-ID
1017h	0h	producer heartbeat time	UNSIGNED 16	RW	0
1018h	0h	largest sub-index supported for “identity object”	UNSIGNED 8	RO	4
	1h	vender ID	UNSIGNED 32	RO	---
	2h	product code	UNSIGNED 32	RO	---
	3h	revision number	UNSIGNED 32	RO	---
	4h	serial number	UNSIGNED 32	RO	---

Note:

1. The object with index 0x1000 has the following data format:

Additional information		General Information
bit 31~ bit 24	bit 23 ~ bit16	bit 15 ~ bit 0
Specific functionality	I/O functionality	Device profile number

For CAN-2000C, the specific function is always in 0. The I/O function defines what kind of CAN-2000C is. Bit 16, 17, 18, 19, 20, 21, 22 present the DI, DO, AI, AO, Counter, PWM, DIO respectively. For example, if bit 16 is 1, it means that the CAN-2000C has DI channels. If both bit 16 and 17 are 1, the CAN-2000C will have both DI and DO channels. Bit 23 is always in 0. The general information is 0x191 (0x191=401), it means that the CAN-2000C complies with the CANopen spec DS401.

2. About the objects with index 0x1001 and 0x1003, please refer to the section 5.3.2.

3. The object with index 0x1005 stores the SYNC COB-ID. In the CAN-2000C, this object is used to receive the SYNC COB-ID. The following table shows the data format of the SYNC.

Bit Number	Value	Meaning
31 (MSB)	x	do not care
30	0	Device does not generate SYNC message
	1	Device generates SYNC message
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28-11	0	If bit 29=0
	x	If bit 29=1: 28-11 bits of 29-bit COB-ID
10-0 (LSB)	x	10-0 bits of COB-ID

The CAN-2000C doesn’t support the SYNC generation, therefore 29-bit ID, bit 30 and bit

31 are always in 0.

4. The object with index 0x1008, 0x1009 and 0x100A records the CAN-2000C product information. When interpreting these objects, the ASCII table will be needed.

5. The range of the 0x100c is from 0 to 65535 in CAN-2000C. For more information of the object with index 0x100C and 0x100D, please refer to the section 5.4.2.

6. The object with index 0x1010 can store the setting of the module and 0x1011 can restore the default setting of the module.

7. For the object with index 0x1014, please refer to the section 5.3.1.

8. The object with index 0x1017 stores the heartbeat time. The producer heartbeat time is 0 if it not use. The time has to be a multiple of 1ms. For more information please refer to the section 5.4.

SDO Communication Entries

Idx	Sidx	Description	Type	Attr	Default
1200h	0h	largest sub-index supported for "server SDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID form client to server (RxSDO)	UNSIGNED 32	RO	600h+Node-ID
	2h	COB-ID form server to client (TxSDO)	UNSIGNED 32	RO	580h+Node-ID

RxPDO Communication Entries

Idx	Sidx	Description	Type	Attr	Default
1400h	0h	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	200h+Node-ID
	2h	transmission type	UNSIGNED 8	RW	FFh
1401h	0h	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	300h+Node-ID
	2h	transmission type	UNSIGNED 8	RW	FFh

1402h	0h	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	400h+Node-ID
	2h	transmission type	UNSIGNED 8	RW	FFh
1403h	0h	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	500h+Node-ID
	2h	transmission type	UNSIGNED 8	RW	FFh
1404h	0h	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	80000000h
	2h	transmission type	UNSIGNED 8	RW	FFh
...	...	...	...	...	...
1409h	0h	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	2
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	8000 0000h
	2h	transmission type	UNSIGNED 8	RW	FFh

TxPDO Communication Entries

Idx	Sidx	Description	Type	Attr	Default
1800h	0	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	5
	1	COB-ID used by PDO (Tx)	UNSIGNED 32	RW	180h+Node-ID
	2	transmission type	UNSIGNED 8	RW	FFh
	3	inhibit time	UNSIGNED 16	RW	0
	4	reversed	---	---	---
	5	event timer	UNSIGNED 16	RW	0
1801h	0	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	5
	1	COB-ID used by PDO (Tx)	UNSIGNED 32	RW	280h+Node-ID
	2	transmission type	UNSIGNED 8	RW	FFh
	3	inhibit time	UNSIGNED 16	RW	0
	4	reversed	---	---	---
	5	event timer	UNSIGNED 16	RW	0
1802h	0	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	5
	1	COB-ID used by PDO (Tx)	UNSIGNED 32	RW	380h+Node-ID

	2	transmission type	UNSIGNED 8	RW	FFh
	3	inhibit time	UNSIGNED 16	RW	0
	4	reversed	---	---	---
	5	event timer	UNSIGNED 16	RW	0
1803h	0	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	5
	1	COB-ID used by PDO (Tx)	UNSIGNED 32	RW	480h+Node-ID
	2	transmission type	UNSIGNED 8	RW	FFh
	3	inhibit time	UNSIGNED 16	RW	0
	4	reversed	---	---	---
	5	event timer	UNSIGNED 16	RW	0
1804h	0	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	5
	1	COB-ID used by PDO (Tx)	UNSIGNED 32	RW	80000000h
	2	transmission type	UNSIGNED 8	RW	FFh
	3	inhibit time	UNSIGNED 16	RW	0
	4	reversed	---	---	---
	5	event timer	UNSIGNED 16	RW	0
...	...	...	...	...	...
1809h	0	largest sub-index supported for "receive PDO parameter"	UNSIGNED 8	RO	5
	1	COB-ID used by PDO (Tx)	UNSIGNED 32	RW	80000000h
	2	transmission type	UNSIGNED 8	RW	FFh
	3	inhibit time	UNSIGNED 16	RW	0
	4	reversed	---	---	---
	5	event timer	UNSIGNED 16	RW	0

6.2 Module Device Profile Area

Every CAN-2000C series module has different function. So about the device profile of the module, please refer to the module's manual.