I-7232D CANopen/Modbus RTU Gateway

User Manual

Warranty

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

Warning

ICP DAS assume no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

Tables of Content

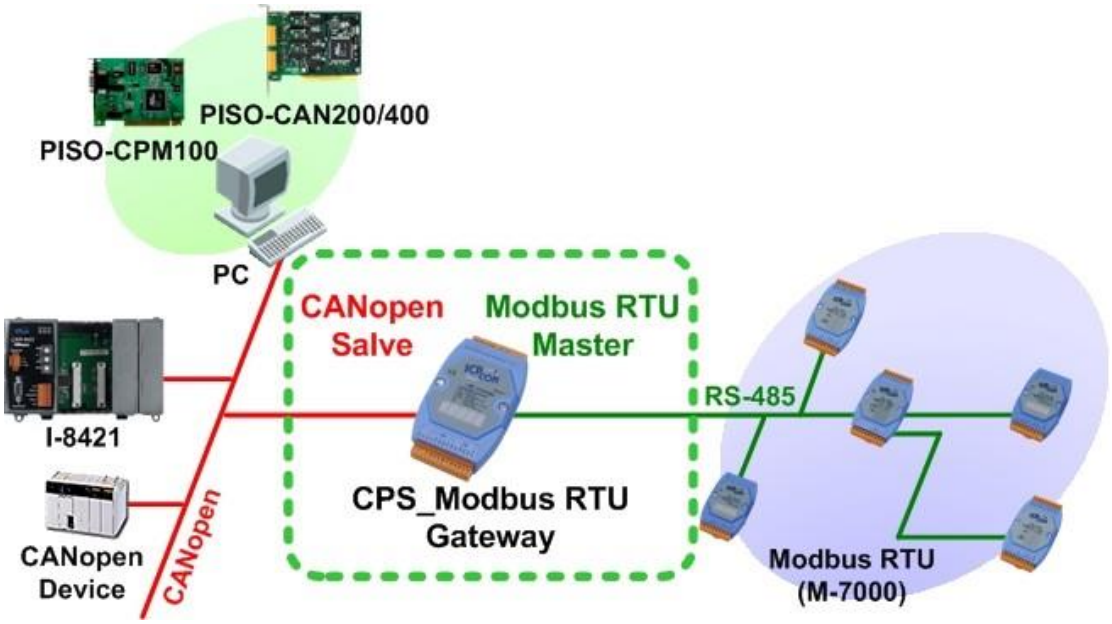
1 Introduction

1.1 Overview

CANopen and Modbus RTU are two kinds of famous protocols and are wildly used in various applications. The I-7232D is a CANopen to Modbus RTU gateway. Using I-7232D gateway, the Modbus RTU I/O modules can be

connected with the CAN bus. In CANopen protocol application, the I-7232D plays the role in a CANopen slave device. Hence, it can produce or consume the PDO messages, receive the SDO message from the SDO client, and deal with the NMT messages from NMT master. In the Modbus RTU protocol application, The I-7232D is a Modbus RTU master device. It can collect all I/O information of the Modbus RTU devices through the RS-485 port of I-7232D. When the I-7232D receives the command from CAN bus, it will do the corresponding actions to Modbus RTU I/O channels. In addition, we also provide the utility tool for users to configure the communication parameters and build EDS file for the I-7232D. Therefore, users can easily apply Modbus RTU IO modules in any CANopen master interface with EDS file via the

I-7232D.



1.2 Hardware Features

CPU:80186, 80MHz

- Philip SJA1000 CAN controller
- Philip 82C250 CAN transceiver
- SRAM:512K bytes
- Flash Memory:512K bytes
- EEPROM:2k bytes
- Real Time Clock
- Built-in Watchdog
- 16-bit Timer
- 2500 Vrms isolation on CAN side
- Power Supply:3.0W
- Unregulated +10VDC to +30VDC
- Operating Temperature:-25°C to +75°C
- Storage Temperature:-30°C to +85°C
- Humidity:5%~95%
- RUN, ERR and Overrun Led indicators

COM1

- RS-232: TXD, RXD, RTS, CTS, GND ● Communication speed: 115200 Max.
- Configure tool connection **COM2**
- RS-485: D2+, D2-

-
  - Communication speed: 115200 Max.
  - Connect to Modbus RTU IO modules

Display

- 7-segmemt LED to show operation mode, Node ID, CAN baud and RS-485 baud

1.3 I-7232D Features

NMT: Slave

- Error Control: Node Guarding
- Node ID: Setting by Utility or LSS protocol
- No of PDOs: 32 Rx, 32Tx
- PDO Modes: Event-triggered, remotely requested, cyclic and acyclic SYNC
- PDO Mapping: variable
- No of SDOs: 1 server, 0 client
- Emergency Message: Yes
- CANopen Version: DS-301 v4.01
- Device Profile: DSP-401 v2.0
- CiA DSP-305 v1.1
- Produce EDS file dynamically
- Baud Rate setting by Utility or LSS protocol: 10K, 20K, 50K, 125K, 250K, 500K, 800K and 1M bps
- CAN, ERR and Overrun LED indicators
- Support max 10 Modbus RTU series modules

-
  - Support max 10 Modbus RTU commands
  - Provide friendly Utility to configure
  - 7-segmemt LED to show operation mode, Node ID, CAN baud and RS-485 baud

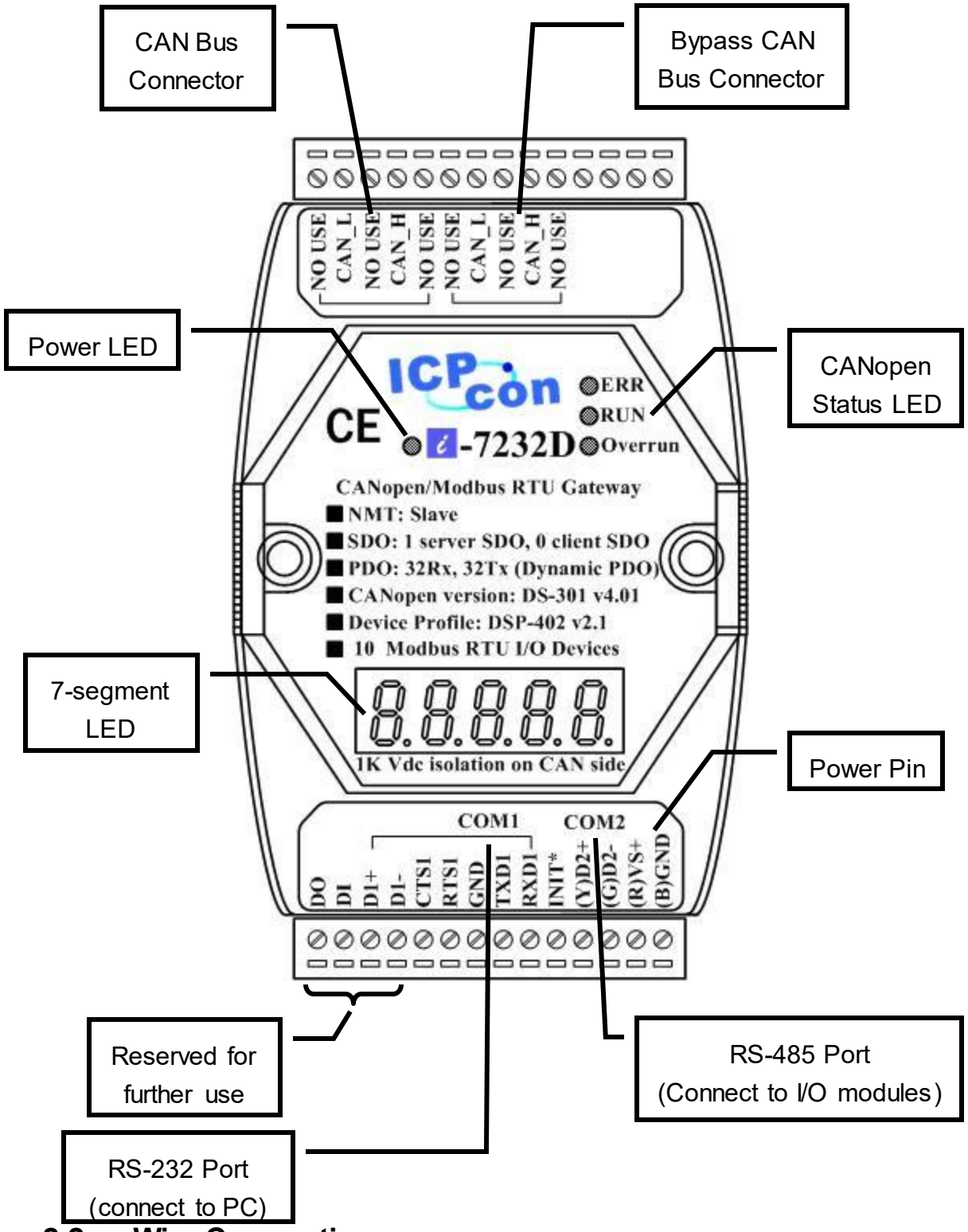
1.4 Utility Features

Support CANopen node ID, baud rate setting, and com port parameters setting

- Show Modbus RTU modules configuration
- Show Application objects configuration
- Support EDS file creating

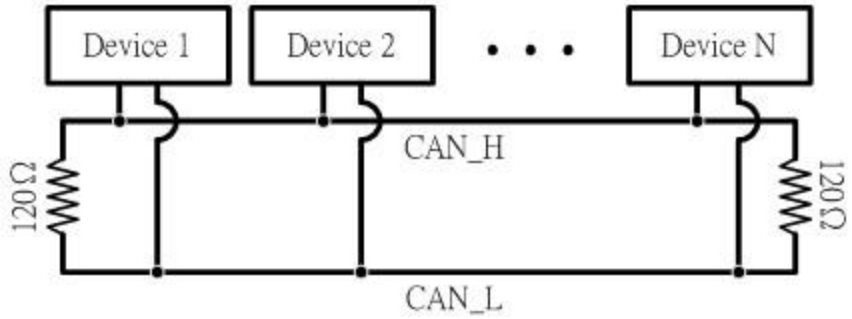
2 Hardware Specification

2.1 Hardware Structure



2.2 Wire Connection

In order to minimize the reflection effects on the CAN bus line, the CAN bus line has to be terminated at both ends by two terminal resistances as following figure. According to the ISO 11898-2 spec, each terminal resistance is 120Ω (or between 108Ω~132Ω). The length related resistance should have 70 MΩ/m. The user should check the resistances of CAN bus, before install a new CAN network.



Moreover, to minimize the voltage drop on long distance, the terminal resistance should be higher than the value defined in the ISO 11898-2. The following table could be a reference.

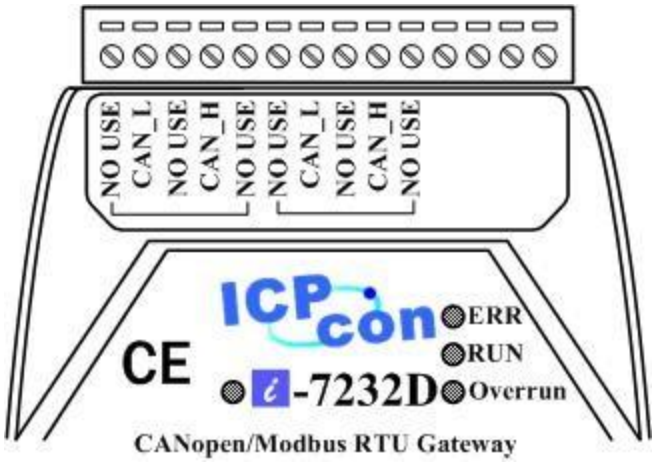
Bus Length (Meter)	Bus Cable Parameters		Terminal Resistance (Ω)
	Length Related Resistance (MΩ/m)	Cross Section (Type)	
0~40	70	0.25(23AWG)~ 0.34mm^2(22AWG)	124 (0.1%)
40~300	< 60	0.34(22AWG)~ 0.6mm^2(20AWG)	127 (0.1%)
300~600	< 40	0.5~0.6mm^2 (20AWG)	150~300
600~1K	< 20	0.75~0.mm^2 (18AWG)	150~300

The CAN bus baud rate has the high relationship with the bus length. The following table indicates the corresponding bus length on every kind of baud rate.

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

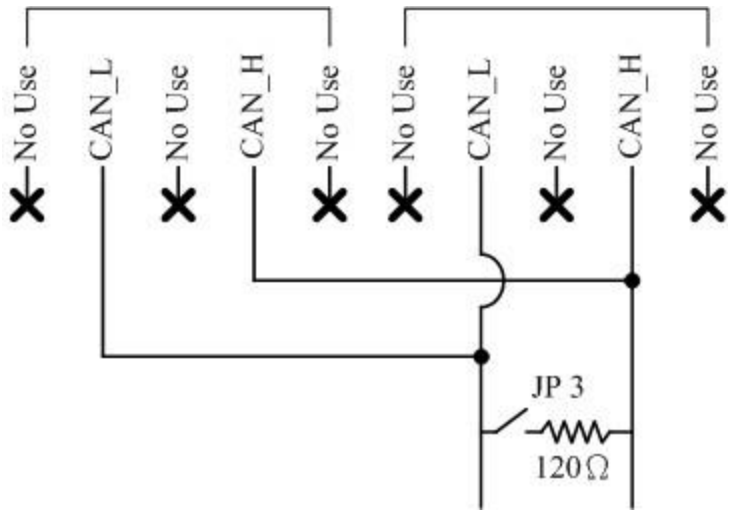
Note: When the bus length is greater than 1000m, the bridge or repeater devices may be needed.

In order to wiring conveniently, the I-7232D supplies two CAN bus connector. Each connecter built on the CANopen/Modbus RTU Gateway looks like as following figure.



Pin No.	Signal	Description
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_SHLD	Optional CAN Shield
4	CAN_H	CAN_H bus line (dominant high)

Be careful that the bypass CAN bus connector can't not be regard as another CAN channel. It is just designed for connecting to another CANopen device conveniently. The structure of the internal electronic circuit is presented as follows.



2.3 Power LED

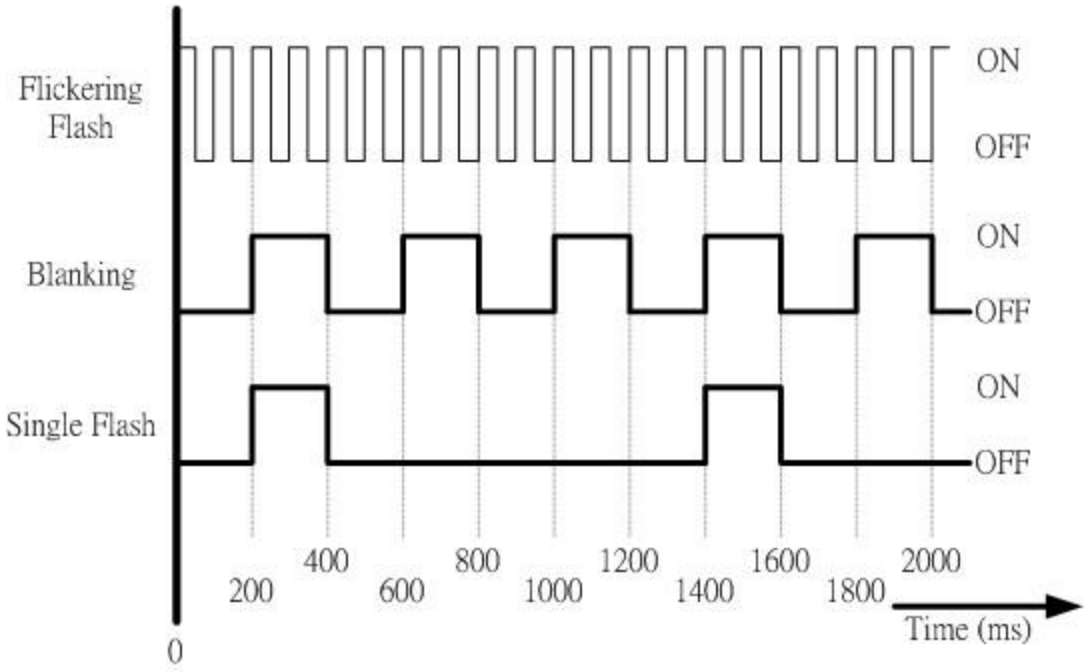
I-7232D needs 10~30 VDC power input and consumes 3.9W. The Power LED will be turn on after applying power.

2.4 CANopen Status LED

I-7232D provides three CANopen LED indicators, such as Error LED (red), RUN LED (green), and Overrun LED (red). The Error LED and Run LED are defined in the CANopen spec. When the CANopen communication events occur, these indicators will be triggered to glitter with different period. The Overrun LED is defined by ICPDAS. When the software buffer of the I-7232D is overrun, the overrun LED will turn on. Before the I-7232D finishes the preparation for the function of the Modbus RTU master or when the I-7232D executes the command to reset itself, all CANopen Status LED will be turned off (but the Power LED is still turned on). The following descriptions interpret the twinkling signal meanings when these indicators are triggered.

2.4.1 **RUN LED**

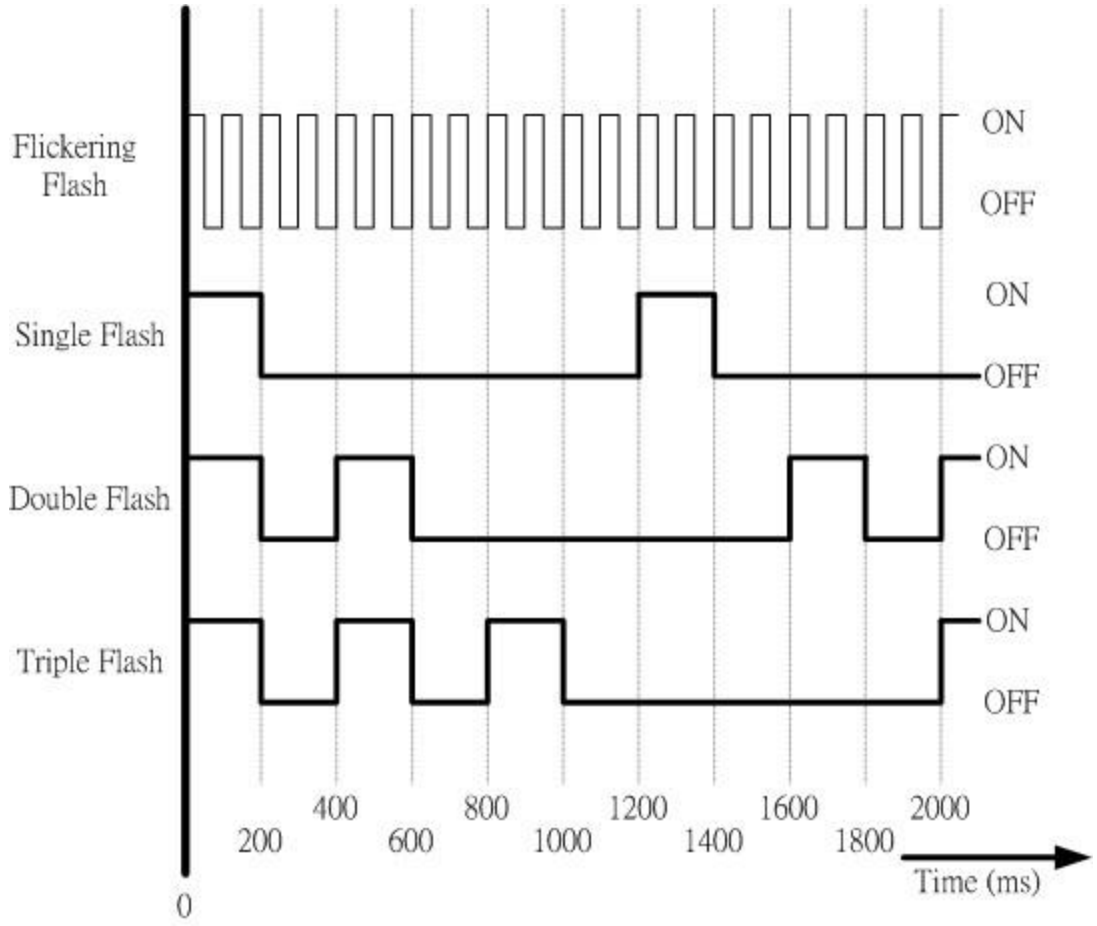
The RUN LED indicates the condition of the CANopen network state mechanism. About the information of CANopen state mechanism, please refer to the section 3.5.1. The different signal periods and related meanings are displayed respectively as following figure and table.



No.	CAN RUN LED	State	Description
1	Single Flash	Stopped	The Device is in Stopped state
2	Blinking	Pre-operational	The Device is in the pre-operational state
3	Flickering	AutoBaud/LSS	Auto Baudrate detection in progress or LSS services in progress (Alternately flickering with ERR LED)
4	On	Operational	The Device is in the operational state

2.4.2 **ERR LED**

The ERR LED indicates the status of the CAN physical layer and indicates errors due to missing CAN messages (These messages may be SYNC or Guard messages) and running LSS protocol. Each error event has different twinkling signal period, and the signal periods and related meanings are displayed respectively as following figure and table.



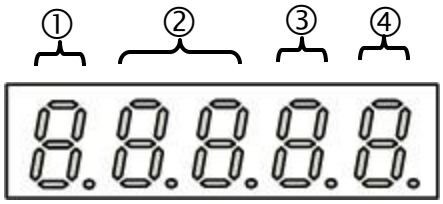
No.	Error LED	State	Description
1	Off	No error	The Device is in working condition.
2	Single Flash	Warning limit reached	At least one of the error counters of the CAN controller has reached or exceeded the warning level (too many error frames).
3	Flickering	AutoBaud/LSS	Auto Baudrate detection in progress or LSS services in progress (Alternately flickering with RUN LED)
4	Double Flash	Error Control Event	A guard event (NMT-Slave or NMT-master) or a heartbeat event (Heartbeat consumer) has occurred.
5	Triple Flash	SYNC Error	The SYNC message has not been received within the configured communication cycle period time out (see Object Dictionary Entry 0x1006).
6	On	Bus Off	The CAN controller is bus off.

Note: If several errors are present at the same duration, the error with the highest number is indicated. For example, if NMT Error (No. =3) and Sync Error (No. =4) occur, the SYNC error is indicated.

2.4.3 **Overrun LED**

This LED is useless when the I-7232D works normally. When CAN message loading is heavy and cause software buffer overrun, the overrun LED will be turned on. At the same time, an emergency message will be transmitted to the CANopen master automatically. In this case, some CAN message may be lost. After the buffer overrun condition disappears, the LED will be turned off. For further information about the emergency message, refer to the section 3.4

2.5 7-segment LED



①: Show the operation state of the I-7232D. If it works normally, the LED displays the character 'n'.

②: These two LED indicate the CANopen node ID of the I-7232D by using hex format. For example, if the CANopen node ID of the I-7232D is 31, these two LED will show the characters "1F".

③: This LED displays the CAN bus baud rate of the I-7232D by number 0~7. The meanings of these numbers are described in the table below.

7-segment LED Number	Baud rate (K BPS)
0	10
1	20
2	50
3	125
4	250
5	500
6	800
7	1000

④: The RS-485 baud rate of the I-7232D is indicated on this LED. The mapping table between LED number and RS-485 baud rate is displayed on the following table.

7-segment LED Number	Baud rate (BPS)
0	1200
1	2400
2	4800
3	9600
4	19200
5	38400
6	57600
7	115200

3 CANopen System

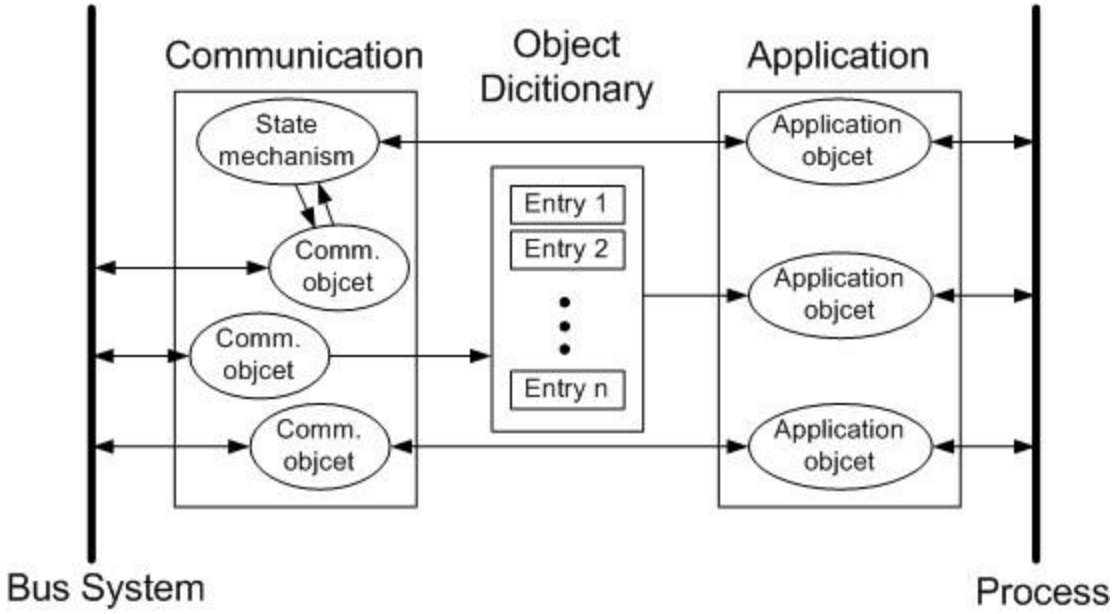
3.1 CANopen Introduction

CANopen is a kind of network protocol based on CAN bus and has been used in various applications, such as vehicles, industrial machines, building automation, medical devices, maritime applications, restaurant appliances, laboratory equipment & research. It allows for not only broadcasting but also peer to peer that data exchange between every CANopen node. The network management functions be specified in CANopen simplifies the project design. Besides, users also can implement and diagnose the CANopen network by standard mechanisms for network start-up and error management. By the device model, any CANopen device can effectively access or get the conditions relating to the I/O values and node states of other devices in the same network. Generally, a CANopen device can be modeled into three parts

- Communication
- Object Dictionary

- Application program

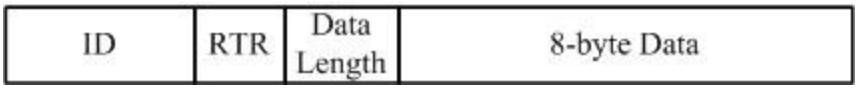
The functions and general concepts for each part are shown as follows.



Communication

The communication part provides several communication objects and appropriate functionalities to transmit CANopen messages via the underlying network structure. These objects may be PDO (Process Data Object), SDO (Service Data Object), NMT (Network Management Objects), SYNC (Synchronous Objects)…etc. Each communication object has its communication model and functionality. Take the PDO, SDO, and NMT for examples, the communication objects for accessing the device object dictionary entries is SDO, and SDO uses the Client/Server structure for its communication model (section 3.2). The real-time data or I/O value can be transmitted or received quickly without any protocol overhead by means of PDO communication objects. The PDOs 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,

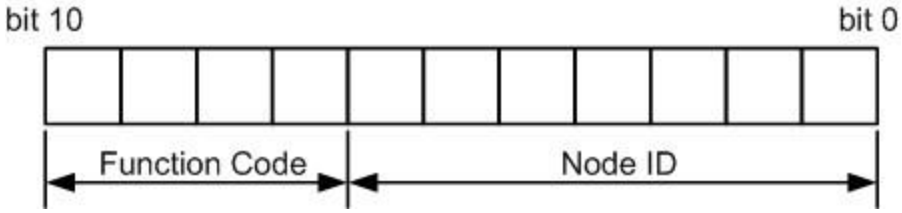
the transmitted message must obey the data frame defined in the CAN 2.0A spec. Generally, it looks like the following figure.



The ID field has 11-bit data. It is useful in the arbitration mechanism. The RTR filed has a one-bit value. If the RTR is set to 1, this message is used for remote-transmit requests. In this case, the 8-byte data is useless. The data length field is 4-bit data. It indicates that the valid data number stored in the 8-byte data field. The last field, 8-byte data, is applied to stores the message data.

CANopen spec uses the 4-bit function code and 7-bit node ID to combine the 11-bit ID of CAN message, and call it communication object ID (COB-ID).

The COB-ID structure is displayed below.



The COB-IDs are defined for recognizing where the message comes from or where the message must be sent. Also, they are used to distinguish the functionality of the transmitted or received messages, and decide the priority of the message transmission for each node on the network. According to the arbitration mechanism of the CAN bus, the CAN message with the lower value COB-ID has the higher priority to be transmitted into the CAN bus. In the CANopen spec, some COB-IDs are reversed for specific communication objects and can't be defined arbitrarily by users. The following lists are 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

Beside the COB-IDs described above, the other COB-IDs can be applied by users if need. All of the default COB-IDs used in the CANopen protocol is shown in the following table.

(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

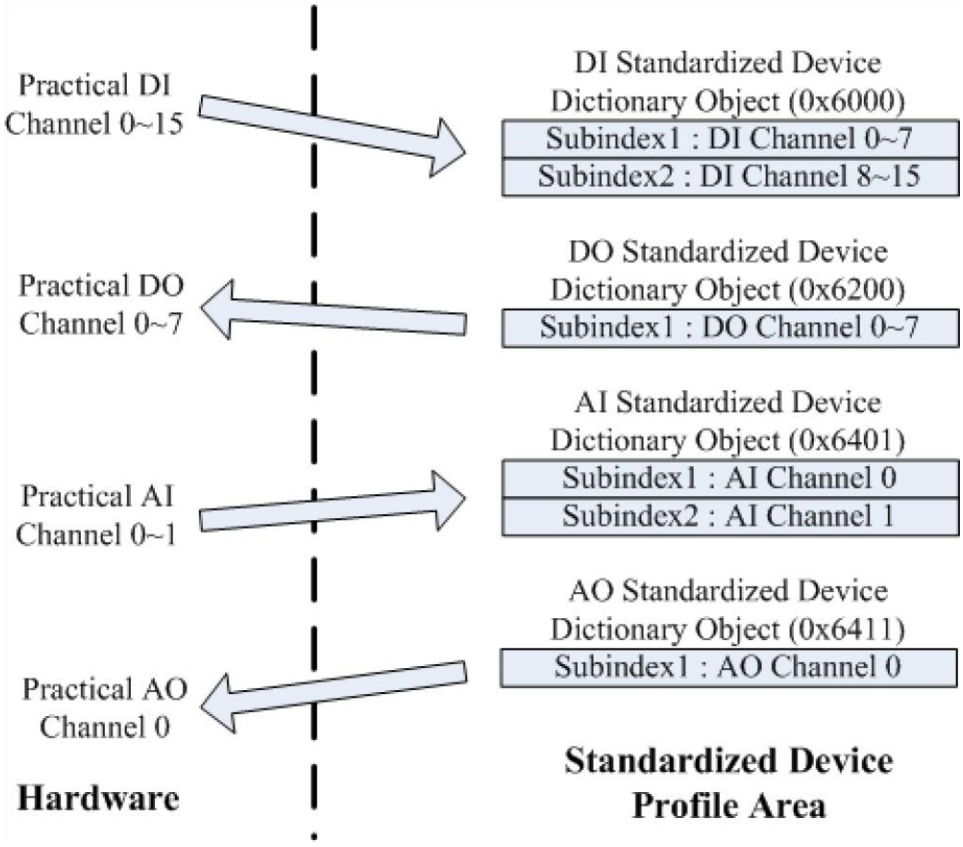
Object Dictionary

The object dictionary collects a lot of important information. This information has an influence on the device's behavior, such as the data in the I/O channels, the communication parameters and the network states. The object dictionary is essentially a group of objects. It consists of a lot of object entries, and these entries can be accessible via the network in a pre-defined method. Each object entry within the object dictionary has their own functionality (ex. 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 by a 16-bit index and an 8-bit sub-index. The overall profile of the standard object dictionary is shown below.

Index (hex)	Object
0000	Reserved
0001-001F	Static Data Types
0020-003F	Complex Data Types
0040-005F	Manufacturer Specific Data Types
0060-007F	Device Profile Specific Static Data Types
0080-009F	Device Profile Specific Complex Data Types
00A0-0FFF	Reserved for further use
1000-1FFF	Communication Profile Area
2000-5FFF	Manufacturer Specific Profile Area
6000-9FFF	Standardized Device Profile Area
A000-BFFF	Standardized Interface Profile Area
C000-FFFF	Reserved for further use

Take the standardized device profile area for an example. Assume that a CANopen device has 16 DI, 8 DO, 2AI and 1AO channels. The values of these channels will be stored into several entries in the standardized device dictionary, such as the entries with indexes 0x6000, 0x6200, 0x6401, and 0x6411. When the CANopen device obtains the input value, these values are stored in the

0x6000 and 0x6401indexes. Furthermore, the values stored in the 0x6200 and 0x6411 indexes also output to the DO and AO channels. The basic concept is depicted as follows.



Take the I-7232D as another example. There are some Modbus modules connecting to the COM 2 of the I-7232D. The related information for each module is shown below.

Module Name	Module Address	DO (ch)	AO (ch)	DI (ch)	AI (ch)
M-7052D**(Note)**	0x 01	0	0	8	0
M-7055D**(Note)**	0x 03	8	0	0	0
M-7024 **(Note)**	0x 04	0	1	0	0
M-7017R**(Note)**	0x 05	0	0	0	1

Note: The M-7000 series devices are the kinds of the Modbus RTU devices produced by ICPDAS.

After user have set the entire module's channels by using the utility tool, and boot up the I-7232D, the information of all the module's I/O channels will be collected by the I-7232D. Also, the I/O values of these channels are arranged into proper object entries one by one. The minimum data of unit is one byte, the DI and DO channels, which are not enough to fill up one byte. That will be regarded as one byte automatically. The I-7232D uses objects with the index 0x6000 to store the input values of the DI channels. The I/O values of the DO, AI, and AO channels are put into the object with the indexes 0x6200, 0x6401, and 0x6411 respectively. When data come through these I/O channels to the corresponding object, it will follow the rules below.

- The modules that are addressed from 0x1 to 0xF, it will be taken into account. The modules with any other addresses will be regarded as useless.
- The I/O channel values of the Modbus RTU modules with lower addresses are first placed into the object dictionary. After the I-7232D has filled the all I/O channels in one module, then the I-7232D will go to the next address to continue.
- Each analog channel is stored by using 2 bytes.
- The number of digital channels for one module, which can't be divided by 8 with no remainder, is stored with 1 byte.

After using the rules described above, the result of the object filling is as follows.

Index / sub-index	0x6000 (for DI)	0x6200 (for DO)	0x6401 (for AI)	0x6411 (for AO)
0x00	1	1	2	2
0x01	DI0~DI7 (MA:0x01)	DO0~DO7 (MA:0x03)	AI0 (MA:0x05)	AO0 (MA:0x04)
0x02				
0x03				
0x04				
0x05				
0x06				
0x07				
0x08				

Note: MA refers to the “Modbus RTU device address”

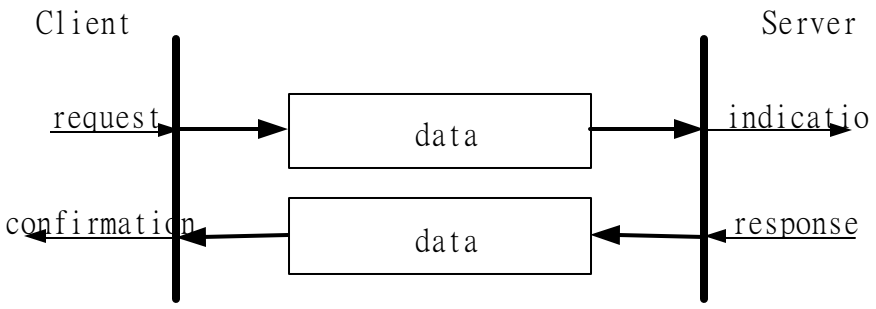
The information described above can also be viewed by using the CANopen/Modbus RTU Gateway Utility. For more details about the object dictionary and how to use the CANopen/Modbus RTU Gateway Utility, refer to chapter 6 and chapter 4.

Application

The application part handles all of the device functionalities, which respect to the interaction with the process environment. It is the bridge 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. The SDO transmission follows the client-server relationship. The general concept is shown in the figure below.



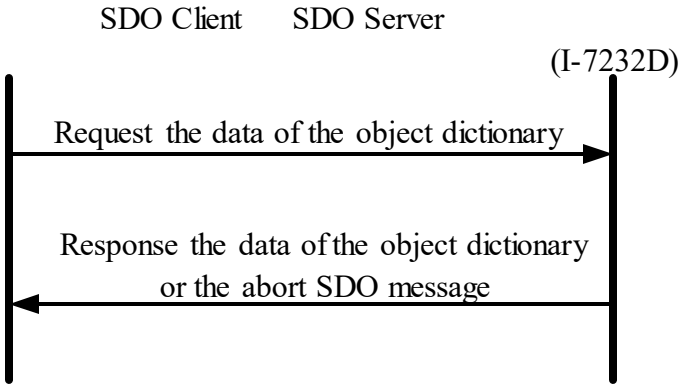
The SDO has two kinds of the COB-IDs, RxSDOs and TxSDOs. For example, from the view of the I-7232D, if users want to send a SDO message, then the I-7232D needs to receive the SDO message transmitted from users. Hence, the receive SDO (RxSDO) COB-ID of the I-7232D will be used.

If the I-7232D wants to transmit a SDO message to users, then the TxSDO COB-ID of the I-7232D will need to be utilized. 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 the proper protocol to transmit the SDO.

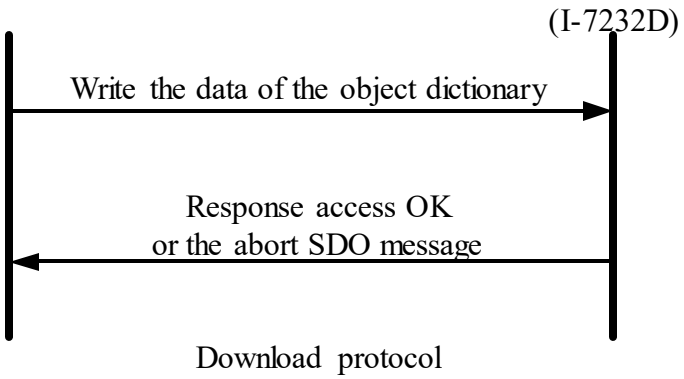
If the SDO client has to get the information of the device object dictionary from the SDO server, the segment upload protocol or block upload protocol will be applied. The former protocol is used for transmitting fewer data; the latter protocol is used for transmitting larger data. Similarly, both the segment download protocol and block download protocol will be implemented 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 entries, not all of the object dictionary entries can be allowed to access via the SDO transmission. If the SDO client trends to modify the read-only entries of the object dictionary of the SDO server, then the abort SDO transfer protocol will be given and the SDO transmission will also stop.

I-7232D only supports the SDO server. Therefore, it can only be passive and wait for the SDO client requirements. The general concept of the upload and download protocol with the I-7232D indicated in the following figure.



Upload protocol

SDO Client SDO Server



Download protocol

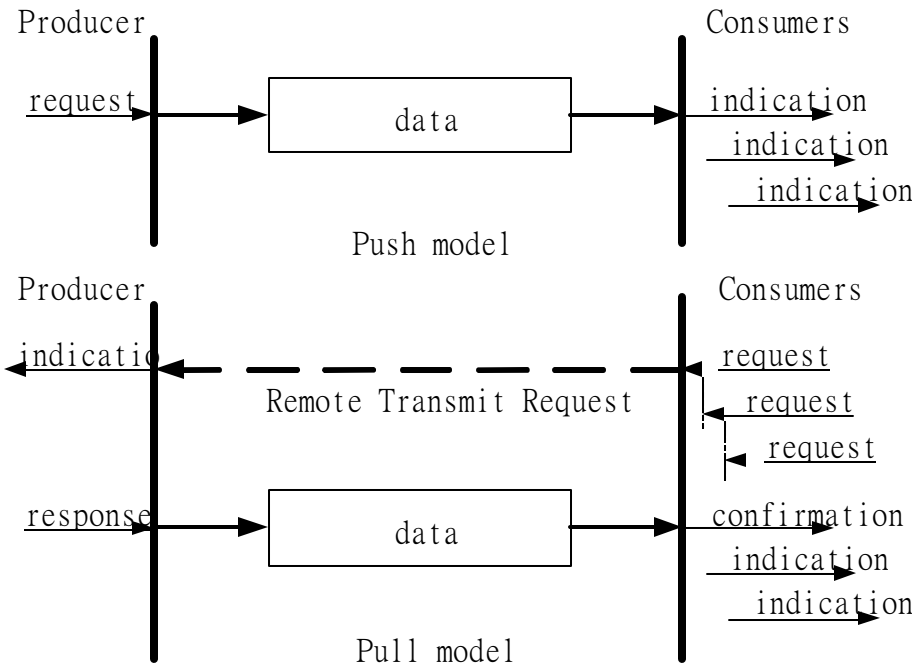
3.3 PDO Introduction

Communication Modes For The PDO

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

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 zero or more than one device to play the role of PDO consumer.

The PDO producer sends out the PDO message after it has won the CAN bus arbitration. Afterwards, each PDO consumer receives this PDO message respectively, and then checks this message if it will be processed or be dropped. In the PDO pull mode, one of the PDO consumers need 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.



From the view of the CANopen device, the TxPDO is used to transmit data from a CANopen device. Therefore, it is usually 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 I-7232D for an example; if a PDO producer sends a PDO message to the I-7232D, it needs to use the RxPDO COB-ID of the I-7232D because it is a PDO reception action viewed from the I-7232D. Inversely, when some PDO consumer send remote transmit requests to the I-7232D, it must use the TxPDO COB-ID of the I-7232D because it is a PDO transmission action viewed from the I-7232D.

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. All of them are described below.

Event Driven

PDO transmission can be triggered by the occurrence of an object specific event. For PDOs of the cyclic synchronous transmission type, this is the

expiration of the specified transmission period, which is synchronized by the exception of the SYNC message.

For PDOs of the acyclic synchronous or asynchronous transmission type, the triggering of a PDO transmission is device-specified in the CANopen spec DSP-401 v2.1. 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 the occurrence of a specific event for the device or if a specified time has elapsed without the occurrence of an event. For example, the PDO transmission of the I-7232D can be triggered by the event timer of the PDO communication parameters, which is set by the user.

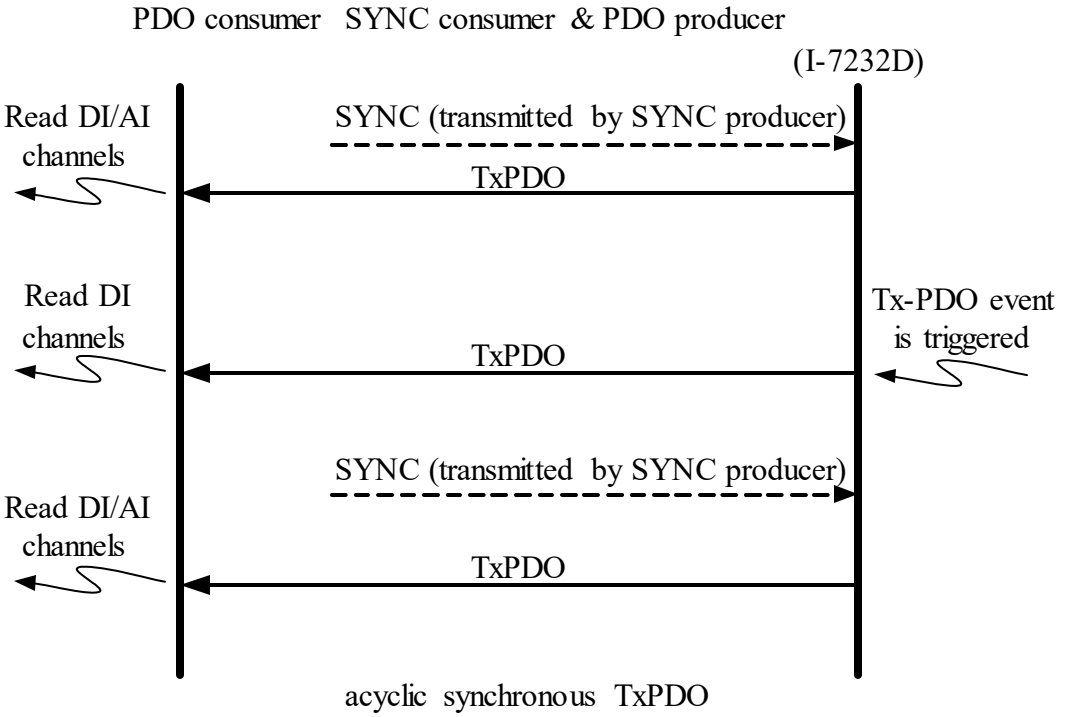
Remote Request

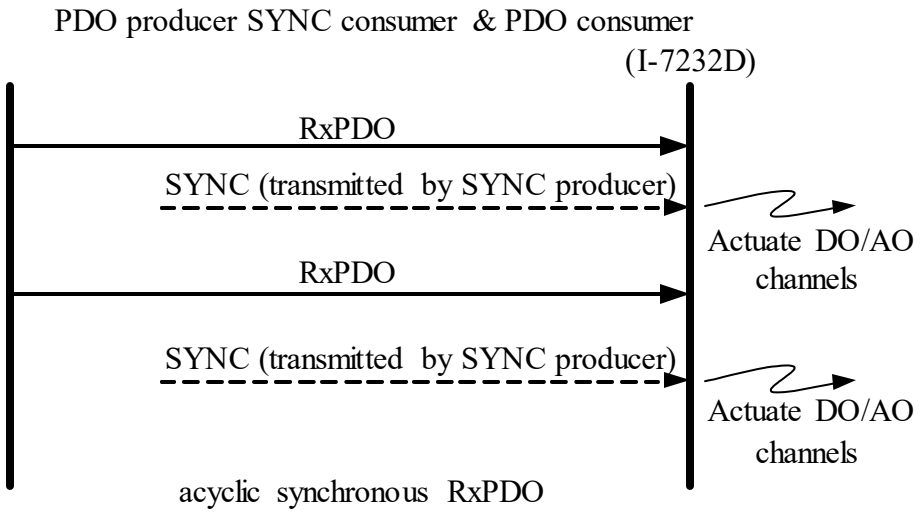
If the PDO transmission type is set to asynchronous or RTR only, the PDO transmission can only be triggered after receiving a remote transmit request from any other PDO consumer.

PDO Transmission Types

Generally speaking, 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 distinguished with more detail into three kinds of transmission.

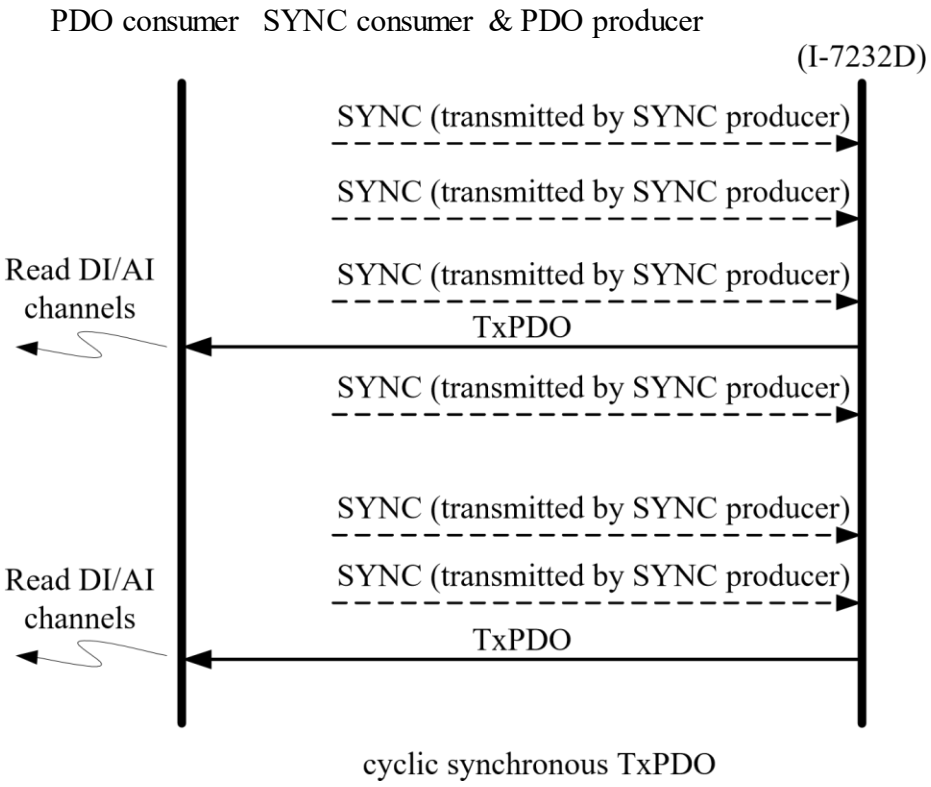
These are the acyclic synchronous, cyclic synchronous and RTR-only synchronous. The acyclic synchronous can be triggered by both the reception of a SYNC message and the occurrence of an event defined by an event driver mentioned above. For the TxPDO object, after receiving a SYNC object from SYNC producer, the I-7232D will respond with a predefined TxPDO message to the CANopen PDO consumers. For the RxPDO object, the I-7232D needs to receive the SYNC object 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.



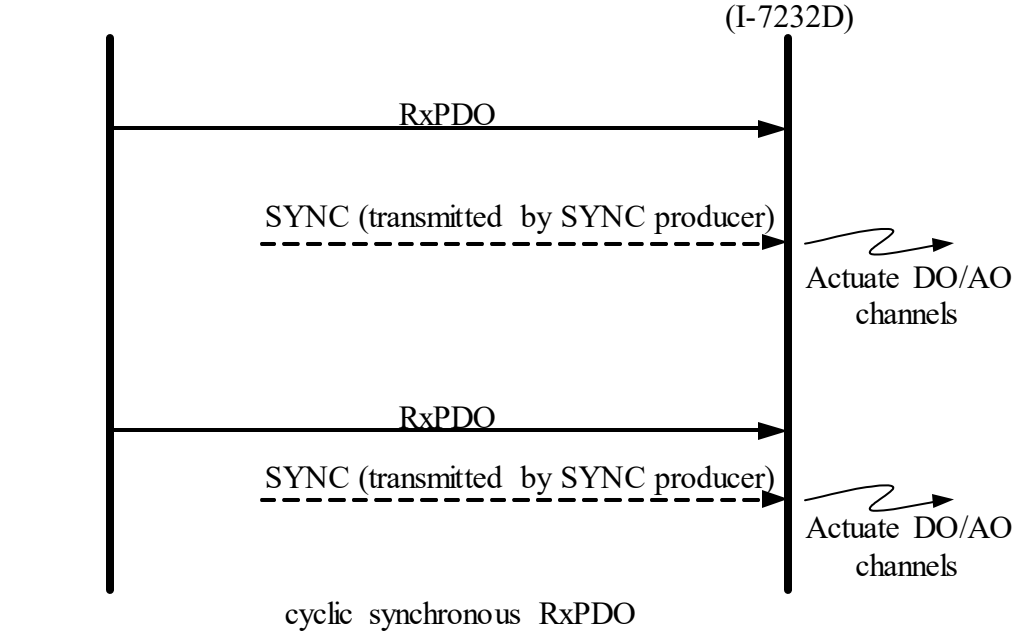


The cyclic synchronous transmission mode is triggered by the reception of an expected number of SYNC objects, and the max number of expected SYNC objects can be 240. For example, if the TxPDO is set to react when receiving 3 SYNC objects, the I-7232D will feedback the TxPDO object after receiving 3

SYNC objects. 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.

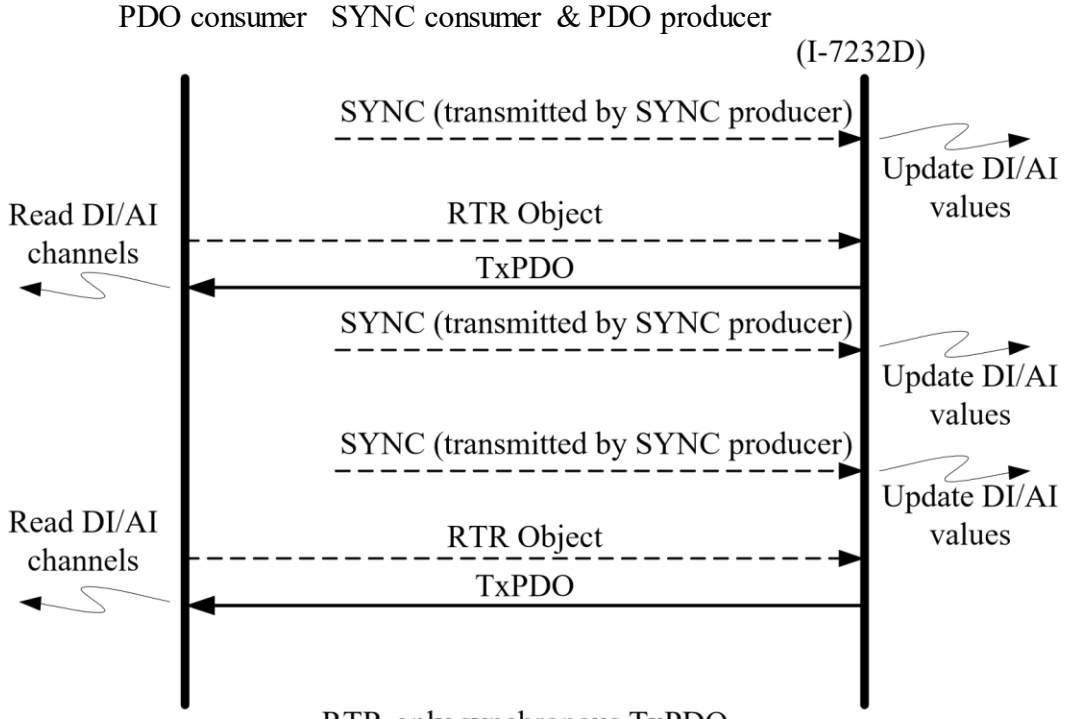


cyclic synchronous TxPDO



cyclic synchronous RxPDO

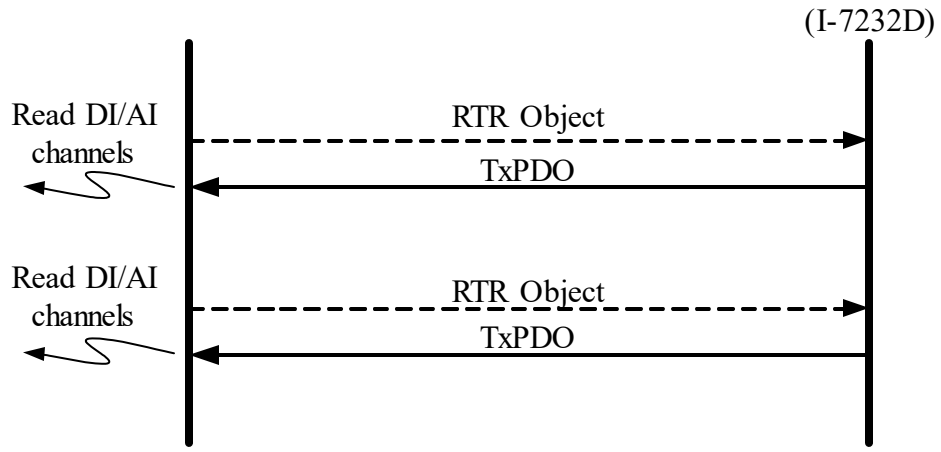
The RTR-only synchronous mode is activated when receiving a remote-transmit-request message and SYNC objects. This transmission type is only useful for TxPDO. In this situation, the I-7232D will update the DI/AI value when receiving the SYNC object. And, if the RTR object is received, the I-7232D will respond to the TxPDO object. The following figure shows the mechanism of this transmission type.



RTR-only synchronous TxPDO

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

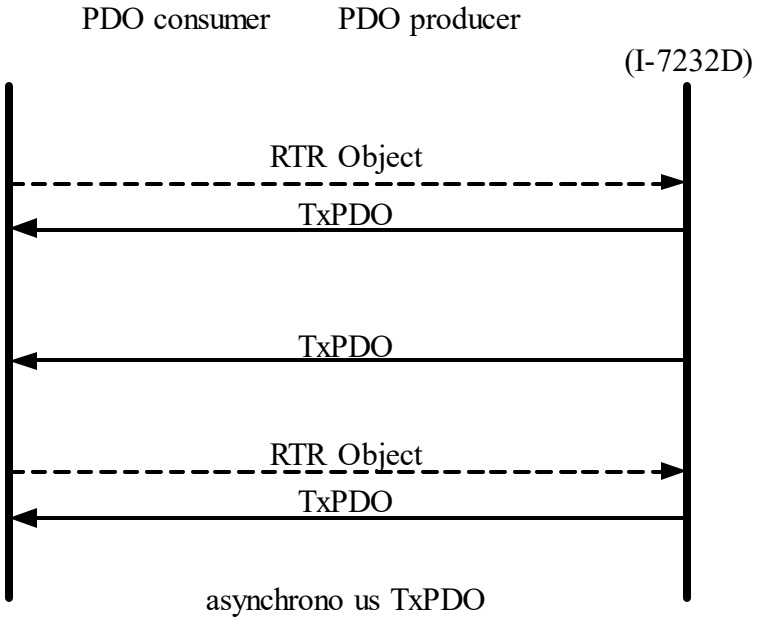
PDO consumer PDO producer



RTR-only asynchronous TxPDO

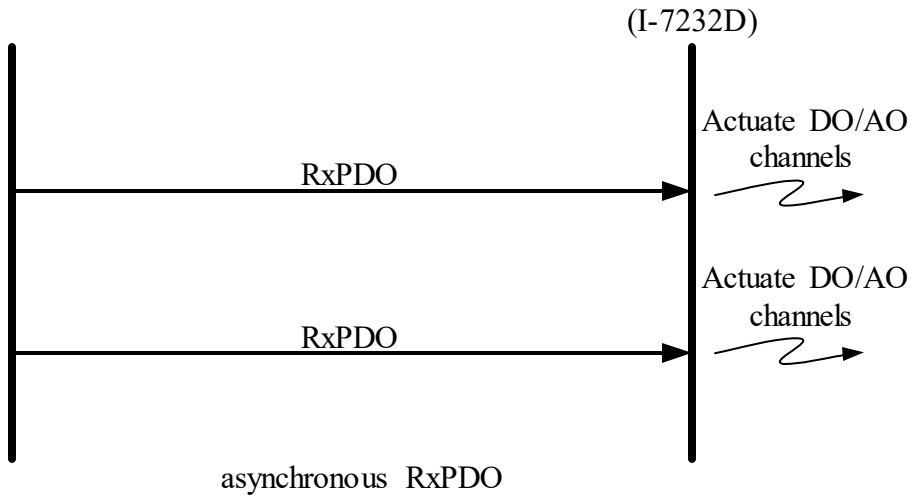
The other part of the asynchronous mode is the asynchronous transmission type. Under this transmission type, the TxPDO message can be triggered not only by receiving the RTR object but also by the occurrence of TxPDO events described in the event driver paragraph described above. Furthermore, the DO/AO channels can act directly by receiving the RxPDO object. This transmission type is the default value when the I-7232D boots up.

The concept of the asynchronous type is illustrated as follows.



asynchrono us TxPDO

PDO producer PDO consumer



Inhibit Time

Because of the arbitration mechanism of the CAN bus, the smaller CANopen communication object ID 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 sent to the CAN bus successfully because of the higher transmission priority. The message with COB-ID 0x182 needs to hold the transmission until the message with COB-ID 0x181 is transmitted successfully. 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 transmitted again and again, the message with COB-ID 0x182 will never get a chance to be transmitted. Therefore, the disadvantage of this arbitration mechanism is that the lower priority of a CAN message is never transmitted successfully if the higher priority message is sent continuously. In order to avoid the occupation of the transmission privilege by the message with a lower COB-ID, the inhibit time parameters for each of the PDO objects define a minimum time interval between each PDO message transmission, which has a multiple of 100us.

During this time interval, the PDO message will be inhibited from transmission.

Event Timer

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

PDO Mapping Objects

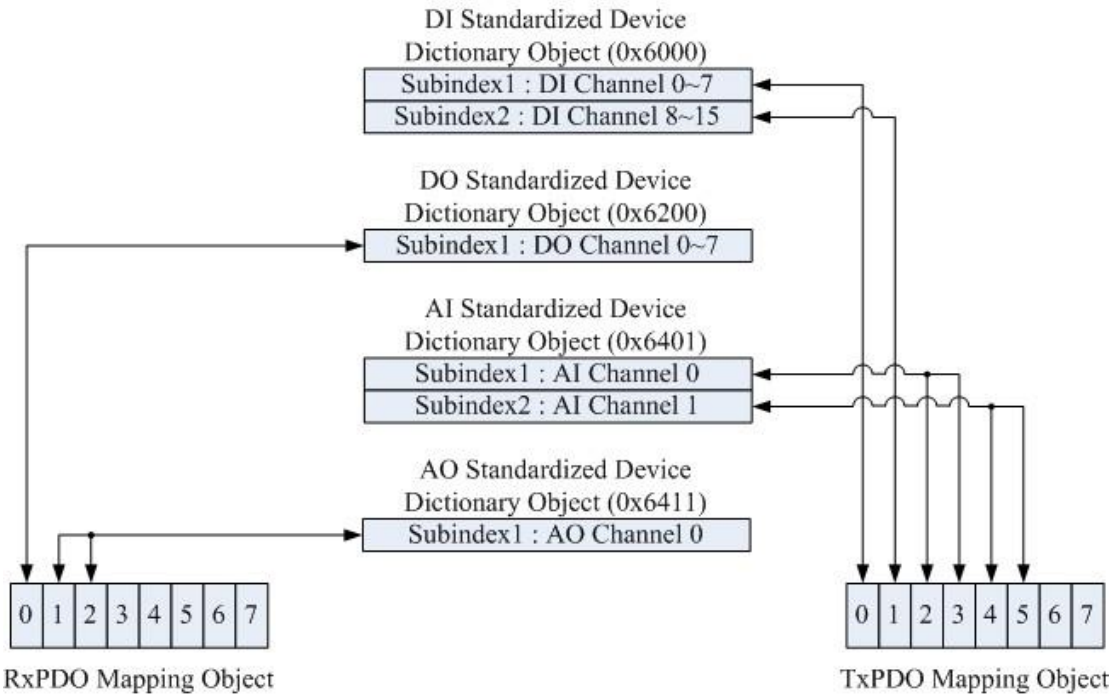
The PDO mapping objects provide the interface between PDO messages and real I/O data in the CANopen device. They define the meanings for each byte in the PDO message, and may be changed by using a SDO message. All of the PDO mapping objects are arranged in the Communication Profile Area. In the CANopen spec (CiA DS401), RxPDO and TxPDO default mapping objects may be specified as follows:

- There shall be up to 4 enabled TxPDO mapping objects and up to 4 RxPDO mapping objects with default mappings.
- 1st RxPDO and TxPDO mappings are used for digital outputs and inputs to each other.
- 2nd, 3rd, and 4th RxPDO and TxPDO mapping objects are assigned to record the value of analog outputs and inputs respectively. If a
- device supports too many digital input or output channels which exceed the 8 channels, the related analog default PDO mapping objects shall remain unused and the additional digital I/Os may use additional PDO mapping objects. This rule shall also be obeyed for the additional analog channels. Take the RxPDO for example; there are 11 DO object entries and 13 AI object entries in the object dictionary. In the default situation for the I-7232D, the first 8 DO object entries will be mapped to the first RxPDO mapping object because one DO object entry needs one bit space. The last 3 DO object entries will be assigned into the 5th RxPDO because of the 2nd and 3rd rule described above. One AO object entry needs 2 bytes of space. Therefore, the second

RxPDO mapping object loads the first 4 AO object entries. The following 4 AO object entries are packed into the third RxPDO mapping object, and so is 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.

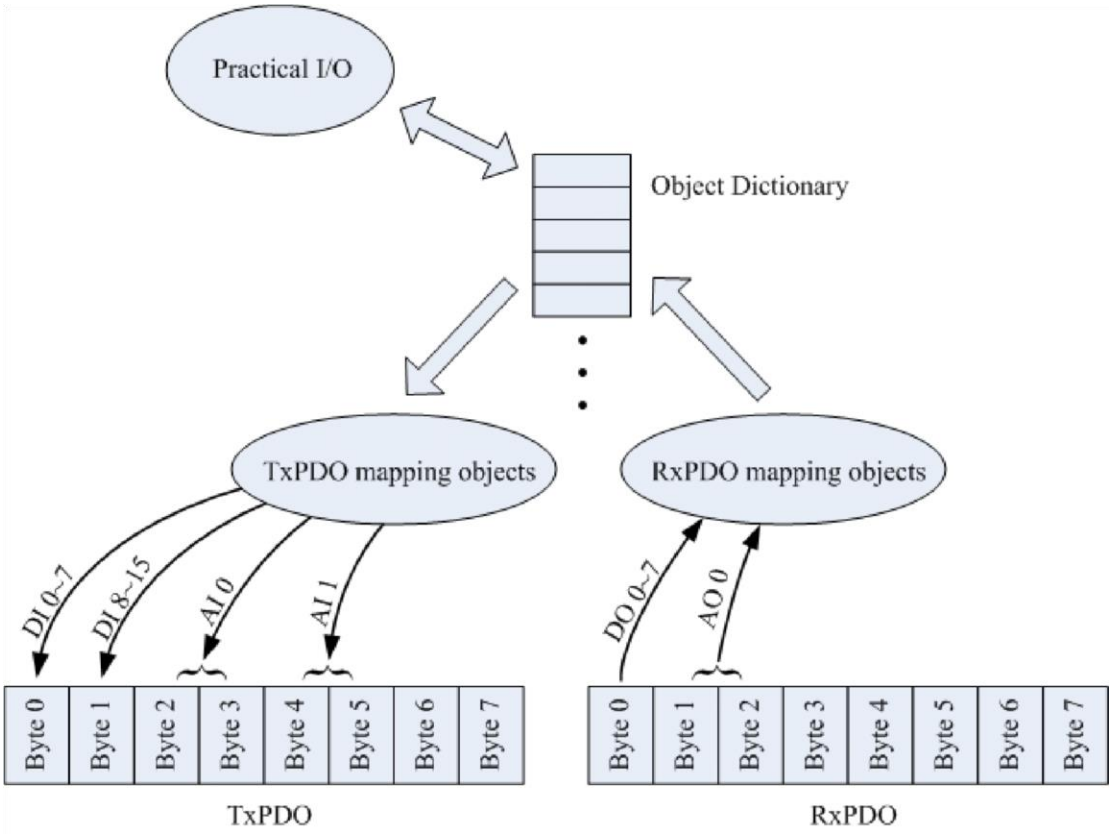
Before applying the PDO communications, the PDO producer and the PDO consumers need to have their PDO mapping information for each other. On the one hand, the PDO producers need PDO mapping information to decide how to assign the expected practical I/O data into PDO messages. On the other hand, PDO consumers need the PDO mapping information to know the meaning of each byte of received PDO message.

That is to say that when a PDO producer transmits a PDO object to PDO consumers, the consumers contrast this PDO message with PDO mapping entries which are previously obtained from the PDO producer. 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 stored into several specific entries for each other. If the user-defined PDO mapping objects have been used, then general concept for these PDO mapping objects, which have been depicted may be very useful.



According to the PDO mapping objects in the figure above, if this CANopen device gets the RxPDO message including three bytes, the first byte is interpreted as the output value of the DO channels 0~7 and the following two bytes are the analog output value.

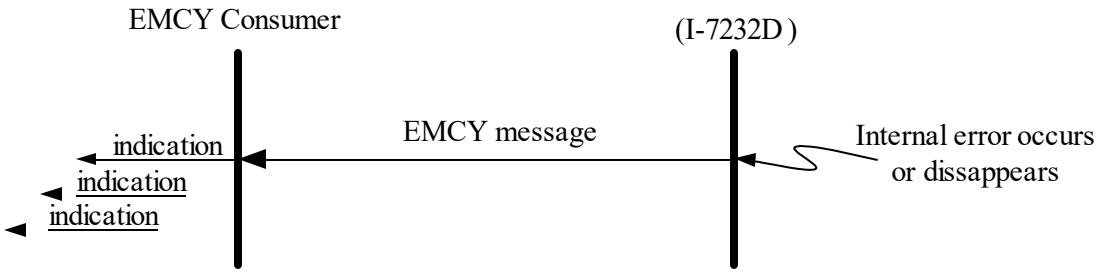
After interpreting the data of the RxPDO message, the device will actuate the DO and AO channels with the received RxPDO message. This situation is the same for TxPDO. When the TxPDO trigger events occur, the CANopen device will send the TxPDO message to the PDO consumers. The values of the bytes assigned in the TxPDO message follow the TxPDO mapping object as in the above figure. The first two bytes of the TxPDO message are the values for the DI channels 0~7 and channel 8~15. The third and forth bytes of the TxPDO message refer to the AI channel 0 value. The fifth and sixth bytes are the values link to AI channel 1. The relationships among the object dictionary, the PDO mapping object and the PDO message are given below.



3.4 EMCY Introduction

EMCY messages are triggered by the occurrence of a device internal error. It follows the producer/consumer relationship. After a CANopen device detects the internal error, an emergency message is transmitted to the EMCY consumers only once per error event. No further emergency objects must be transmitted if no new errors occur on a device. Zero or more then one emergency consumers may receive the EMCY object. The I-7232D only supports the function of the emergency producer. The general concept behind the EMCY communications is shown below.

EMCY Producer



An emergency message contains 8-byte of data called emergency object data, and follows the structure provided bellow.

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

All the fields in the emergency object data will be described in section 5.3. Take the I-7232D for an example, if any errors occur in the I-7232D, the EMCY message will be sent out from the I-7232D. Afterwards, the EMCY message will not be transmitted again if the same error occurs repeatedly.

However, if any other different error which are detected by the I-7232D occur, it will trigger the transmission of the EMCY message again. After one but not all error reasons are gone, an emergency message containing the emergency error code “00 00” may be responded with the remaining errors in the error register and manufacturer specific error fields. Hence, by means of checking the EMCY message, users can understand what is happening in the I-7232D, and can do something for the error event.

3.5 NMT Introduction

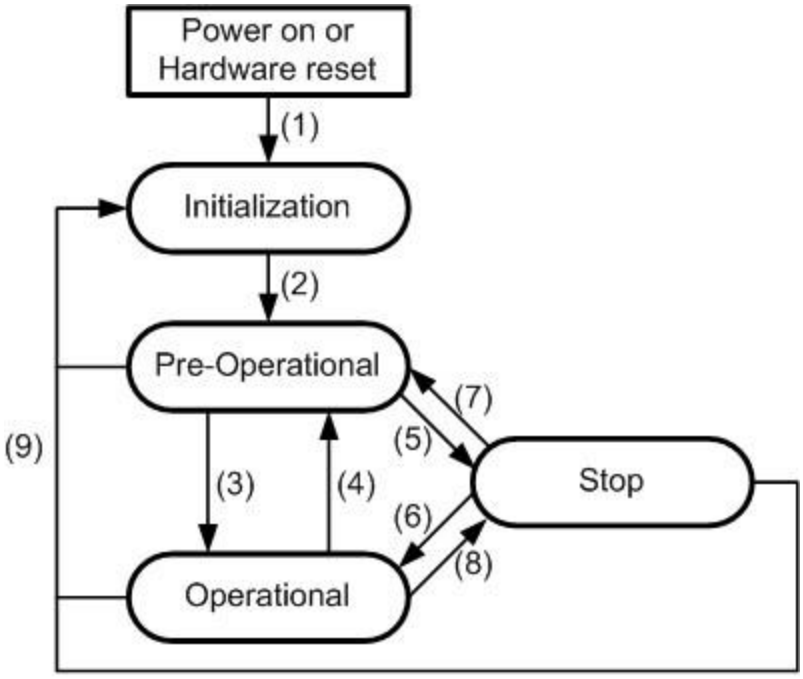
The Network Management (NMT) follows a node-oriented structure and also follows the master-server relationship. On the same CAN bus network, only one CANopen device can have the power to implement the function of NMT master. All the other CANopen nodes are regarded as NMT slaves. Each NMT slave is unique, and identified by its node ID from 1 to 127. The NMT service supplies two protocols, module control protocol and error control protocol, for different purposes. Through the NMT module control protocol, the nodes can be

controlled into several kinds of status, such as installing, pre-operational, operational, and stopped. The NMT slave in different statuses has different privileges to implement the communication protocol. The error control protocol gives users the way to detect the remote error in the network.

It can confirm if the node still lives or not.

3.5.1 **Module Control Protocols**

Before introducing the modules control protocols, let's look at the architecture of the NMT state mechanism. The following figure displays the relationships among each NMT state and the mechanism for changing the NMT state of a NMT slave.



State Mechanism Diagram

(1)	At "Power on" the initialization state is entered autonomously
(2)	Initialization finished enter Pre-Operational automatically
(3),(6)	"Start Remote Node" indication
(4),(7)	"Enter Pre-Optional State" indication
(5),(8)	"Stop Remote Node" indication
(9)	"Reset Node" or "Reset Communication" indication

Devices enter the Pre-Operational state directly after finishing the device initialization. Then, the nodes can be switched into different states by receiving an indication. Each different NMT state allows different specific communication methods. For example, the PDO message can only transmit or receive in the operational state. In the following table, the relationship among each NMT state and communication objects is given.

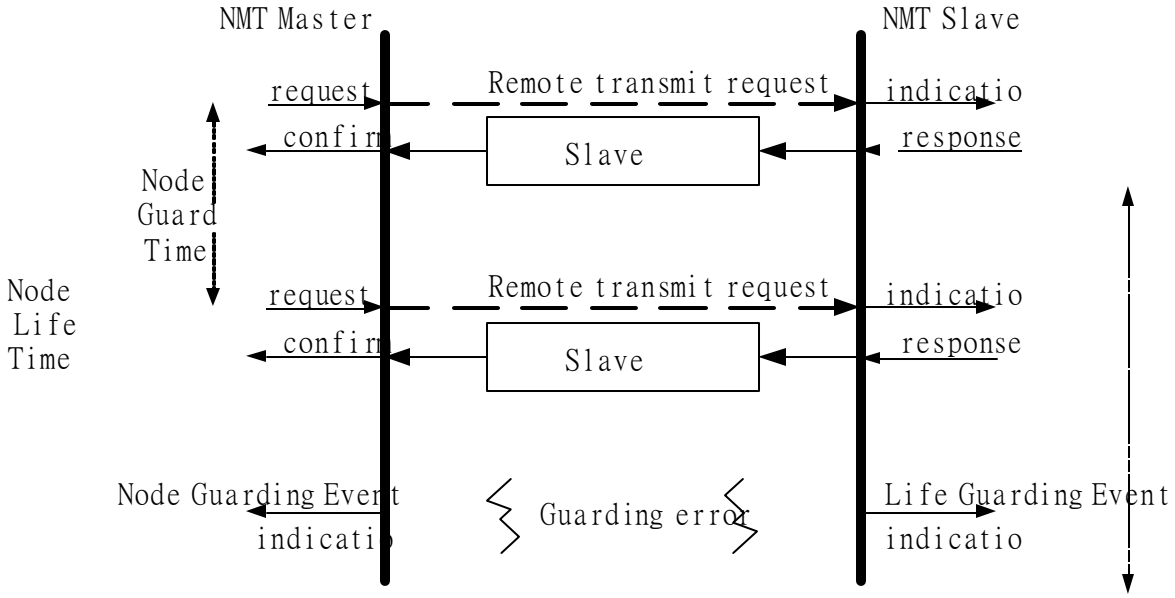
	Installing	Pre-operational	Operational	Stopped
PDO			**O**	
SDO		**O**	**O**	
SYNC Object		**O**	**O**	
Time Stamp Object		**O**	**O**	
EMCY Object		**O**	**O**	
Boot-Up Object	**O**			
NMT		**O**	**O**	**O**

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 both error control mechanisms, Guarding Protocol and Heartbeat Protocol, at the same time. I-7232D provides the salve function of the Node Guarding Protocol. Therefore, users can only use this protocol for I-7232D in practical application, and only node guarding protocols will be introduced here. The node guarding protocol of the error protocol is described below.

Node Guarding Protocol

The Node Guarding Protocol follows the Master/Slave relationship. It provides a way to help uses monitor the node in the CAN bus. The communication method of node guarding protocol is defined as follows.



The NMT master polls each NMT slave at regular time intervals. This time-interval is called the guard time and may be different for each NMT slave. 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 is given by the “guard time * life time factor”. The node life time factor can also be different for each NMT slave. If the NMT slave has not been polled during its life time, a remote node error is indicated through the "Life Guarding Event" service.

In addition, the reported NMT slave state, which does not match the expected state, also produces the “Life Guarding Event”. This event may be occurs in the DO and AO channels to output the error mode value recorded in the object with index 0x6207 and index 0x6444. The object with index 0x6026 and 0x6443 can control the error mode value of the DO or AO channels to enable or disable when the “Lift Guarding Event” has been indicated. For more

information about objects with index 0x6206, 0x6207, 0x6443, and 0x6444, please refer to chapter 6.

3.6 LSS Introduction

3.6.1 Definition

LSS protocol offers the possibility to inquire and change the settings of certain parameters of the local layers on a CANopen module with LSS Slave capabilities by a CANopen module with LSS Master Capabilities via the CANopen Network.

The following parameters can be inquired and/or changed by using LSS protocol.

- Node-ID of the CANopen slave
- Bit timing parameters of the physical layer (CAN baud)
- LSS address (index 1018H)

By using LSS, a LSS Slave can be configured for a CANopen network without using any devices like DIP-switches for setting the parameters.

3.6.1.1 LSS objects and attributes

LSS functionality is modeled using two objects. The LSS Master object exists exactly once in a CANopen network supporting LSS. The LSS Master configures layer parameters of connected CAN modules by the usage of LSS Slave objects residing on the individual modules. Communication between LSS Master and LSS Slave is accomplished by the LSS protocol.

3.6.1.2 LSS master object

The module that configures other modules via a CANopen network is called the LSS Master. There may be only one LSS Master in a network. The LSS Master has no attributes.

3.6.1.3 **LSS slave object**

The module that is configured by the LSS Master via a CANopen network is called the LSS Slave. The number of LSS Slaves in a network is not limited.

The LSS Slave has following attributes.

- **LSS address**

An LSS Slave is identified by an LSS Address. An LSS Address consists of a vendor-id, a product-code, a revision-number and a serial-number. The vendor-id, product-code and serial-number are numerical numbers. These parts are all UNSIGNED32 data format. A vendor-id is assigned to module suppliers by CiA. A product-code, revision and a serial-number are assigned by the module supplier.

For LSS-Addresses the following conditions must be met.

- ◆ The LSS address is identical to the CANopen identity object.
- ◆ The LSS address of a LSS Slave can be inquired.
- ◆ There exists no other LSS Slave in the world with the same <LSS-Address>.

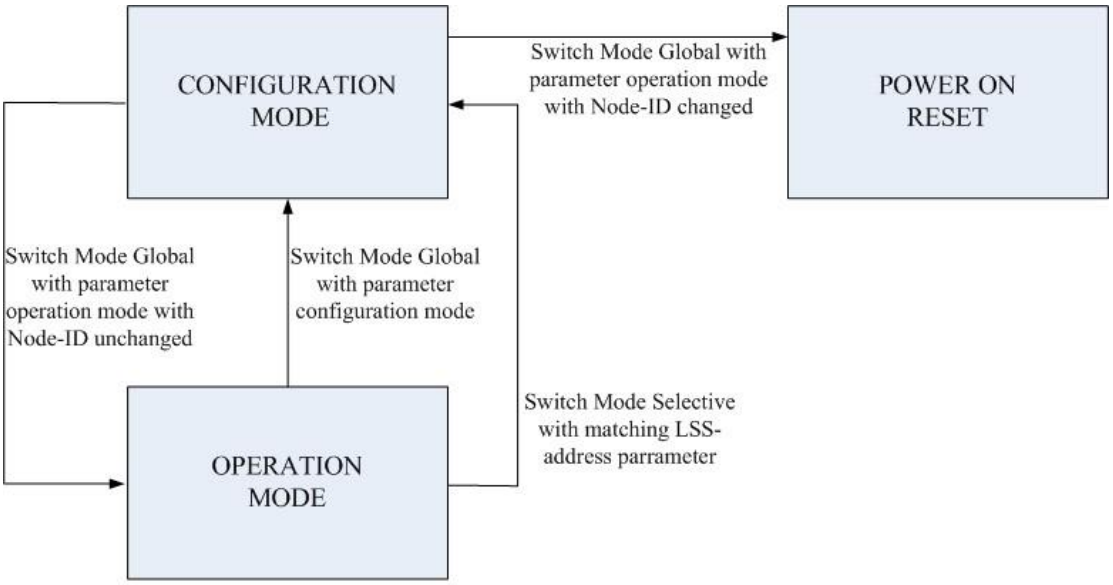
- **LSS modes**

The LSS service distinguishes between the LSS configuration phase and the operation phase of the module. Any module that is not in ‘Configuration Mode’ is in ‘Operation Mode’. In ‘Configuration Mode’ all LSS service, in ‘Operation Mode’ only the switch mode services are available.

3.6.2 **LSS MODES AND SERVICES**

LSS services can be functionally grouped in three areas:

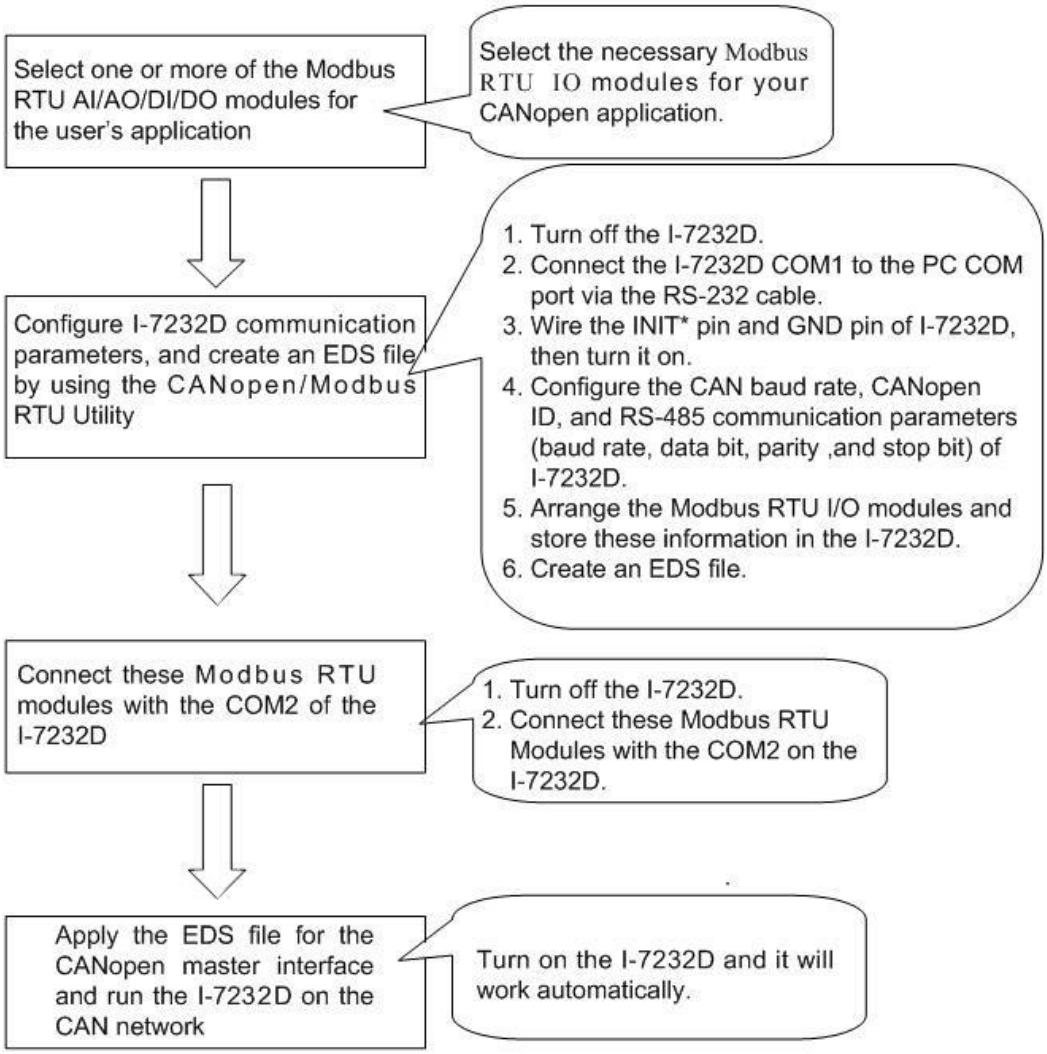
- The switch mode services provide a way to logically connect the LSS Master and LSS Slave(s) for configuration purposes. They change the LSS mode attribute of the LSS Slave.
- The configuration services perform the actual task of configuring the layer parameters of an LSS Slave. The configuration services are only available in configuration mode.
- The inquiry services provide a way for the LSS Maser to determine layer parameters. The inquiry services are available only in configuration mode.



LSS modes and switching procedure

4 CANopen System

4.1 I-7232D Configuration Flowchart



4.2 CANopen/Modbus RTU Gateway Utility Overview

The CANopen/Modbus RTU Gateway Utility is designed for the I-7232D. It provides three functions.

- Set the communication parameters of the CANopen, CAN bus and RS-485. Such as Node ID, CAN bus baud rate, RS-485 baud rate, Start Address, data length and RS-485 timeout value.
- Set the Modbus RTU modules hanging on the COM2 of the I-7232D. Then, create the EDS file to match the setting result.
- Show the important information, which is useful for the CANopen network and the RS-485 network. Such as the PDO communication objects, Modbus RTU modules information, and the standardized device objects and manufacturer specific objects defined in the I-7232D.

Before users start to use the I-7232D, they must configure the Modbus RTU I/O modules by using the CANopen/Modbus RTU Utility. During the configuration, users need to give a unique ID (0x01~0x10) for each Modbus RTU module in the RS-485 network.

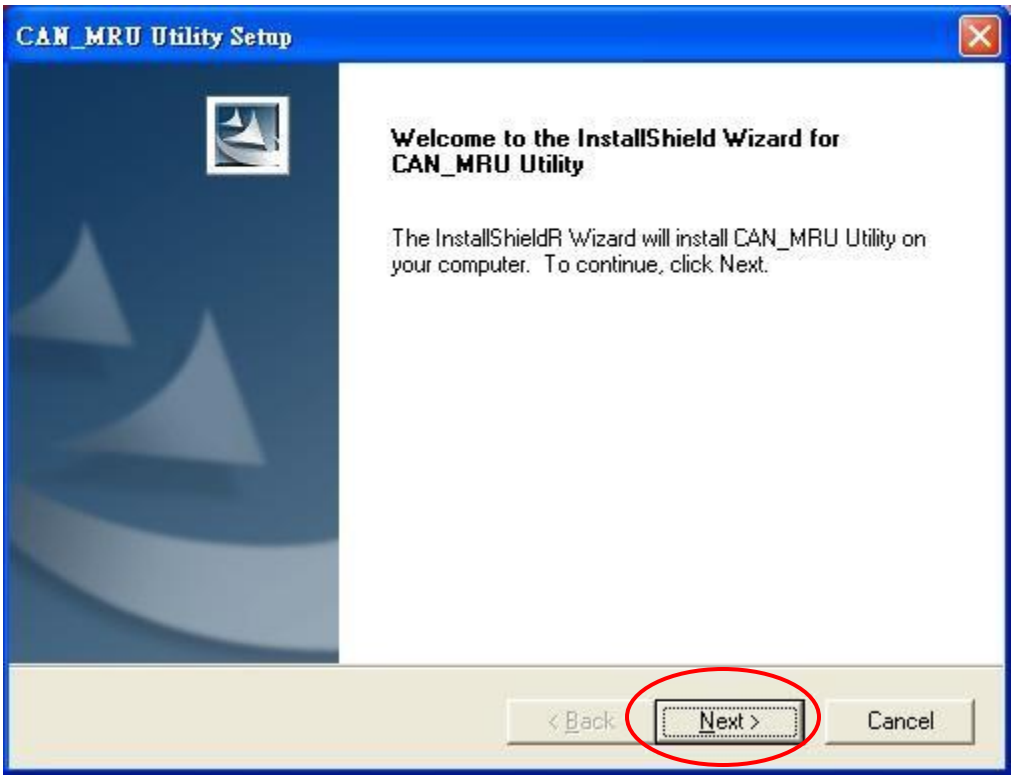
For more information about how to configure the Modbus RTU modules, please refer to the on-line help of the CANopen/Modbus RTU Utility or the user manual for the Modbus RTU modules.

4.3 CANopen/Modbus RTU gateway Utility Installation

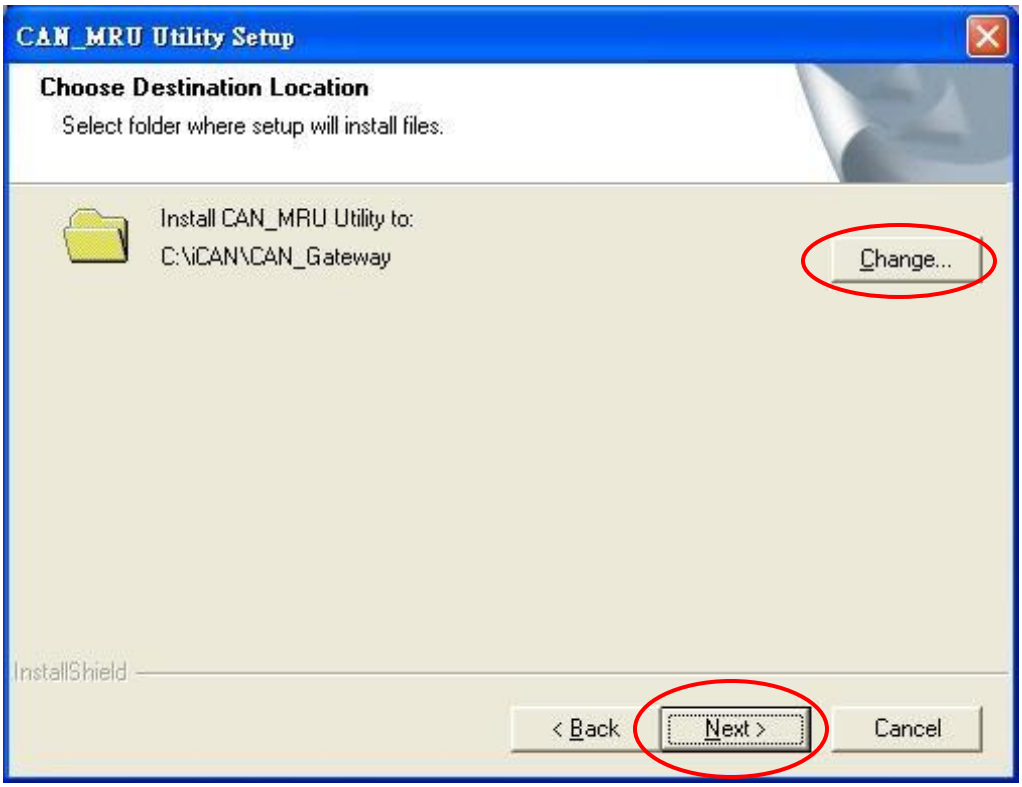
Install CAN Gateway Utility

Step 1: Download the CANopen/Modbus RTU Gateway Utility setup file from the web site http://www.icpdas.com/download/can/index.htm or CD-ROM disk following the path of "/Napdos/ iCAN/CAN_Gateway/I-7232D".

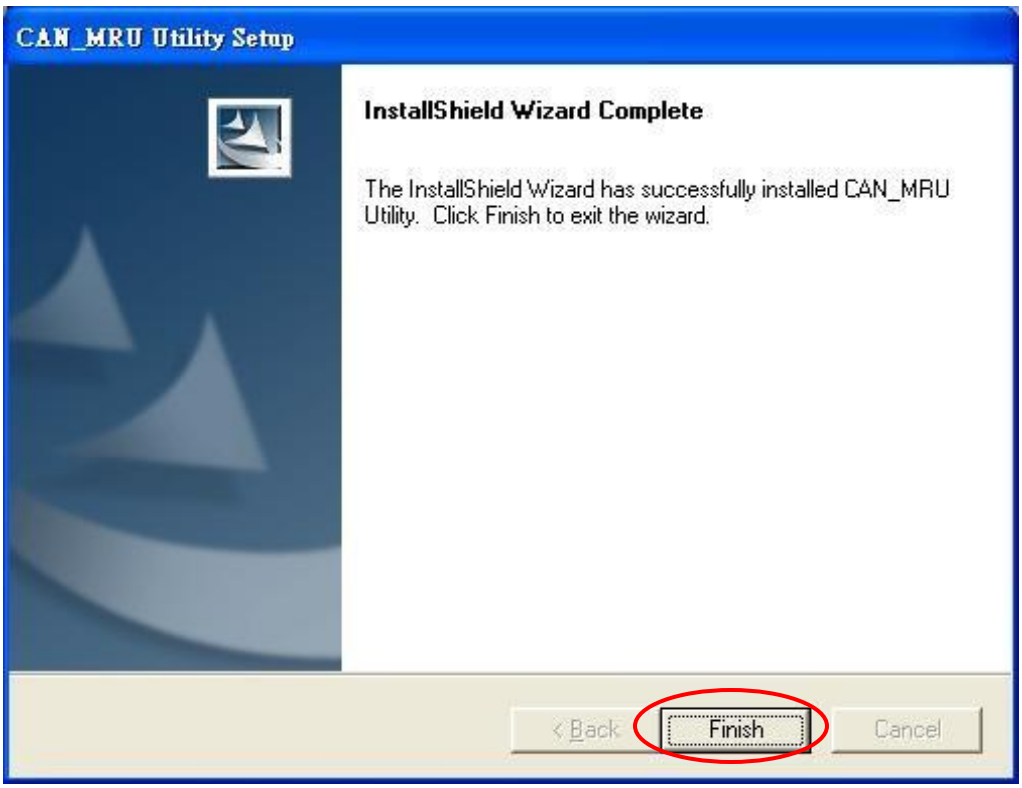
Step 2: Execute the setup.exe file to install the CANopen/Modbus RTU Gateway Utility.



Step 3: Click the “Next” button to continue. Then you will see the default path, if you want to change the installation destination, click “Change” button to set the installation path.



Step 4: Click the “Finish” button to finish the installation program.

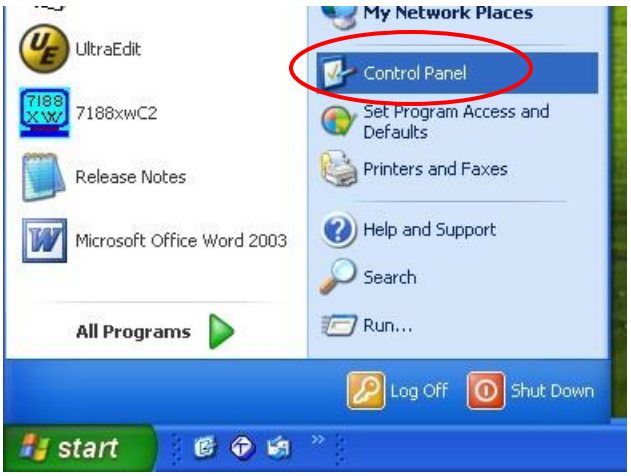


Step 5: After finishing the installation of the CANopen/Modbus RTU Gateway Utility, users can find the CAN_MRU Utility as shown in the following screen shot.

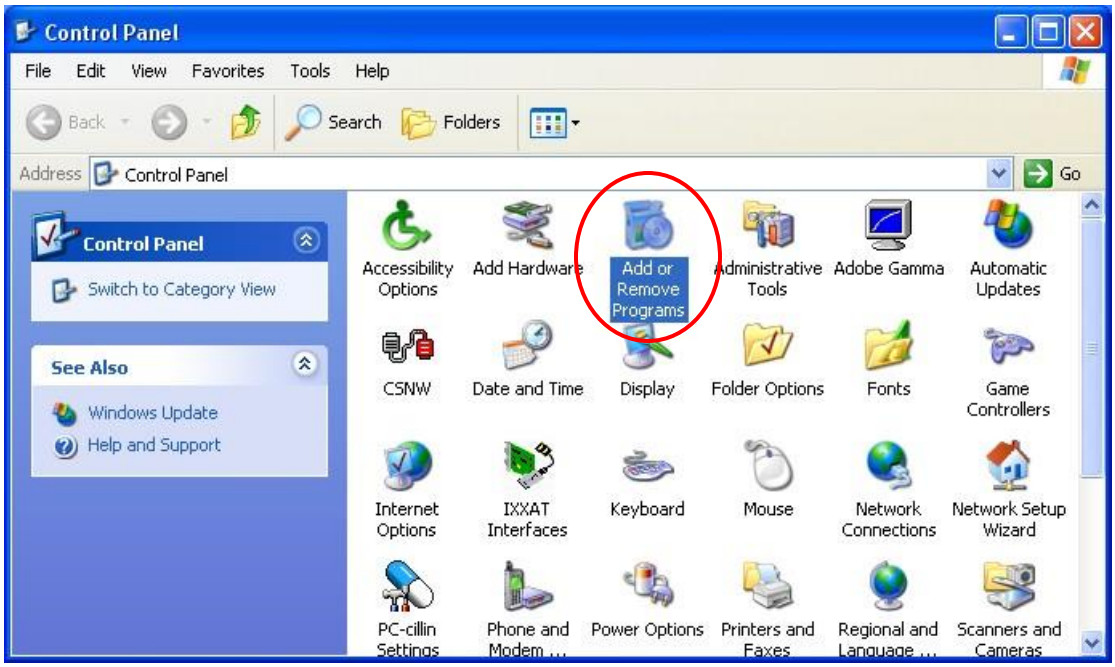
Uninstall CAN Gateway Utility

You can uninstall the CANopen/Modbus RTU Utility software from the control panel by using the following steps.

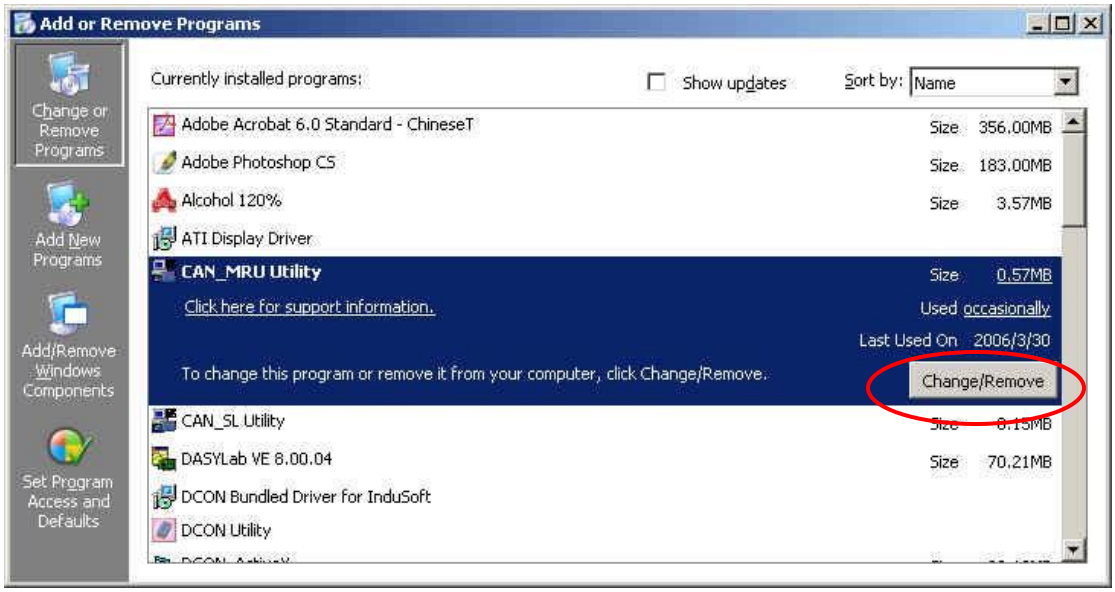
Step 1: Click “Start” in the task bar, then clicks the Control Panel as shown in the following figure.



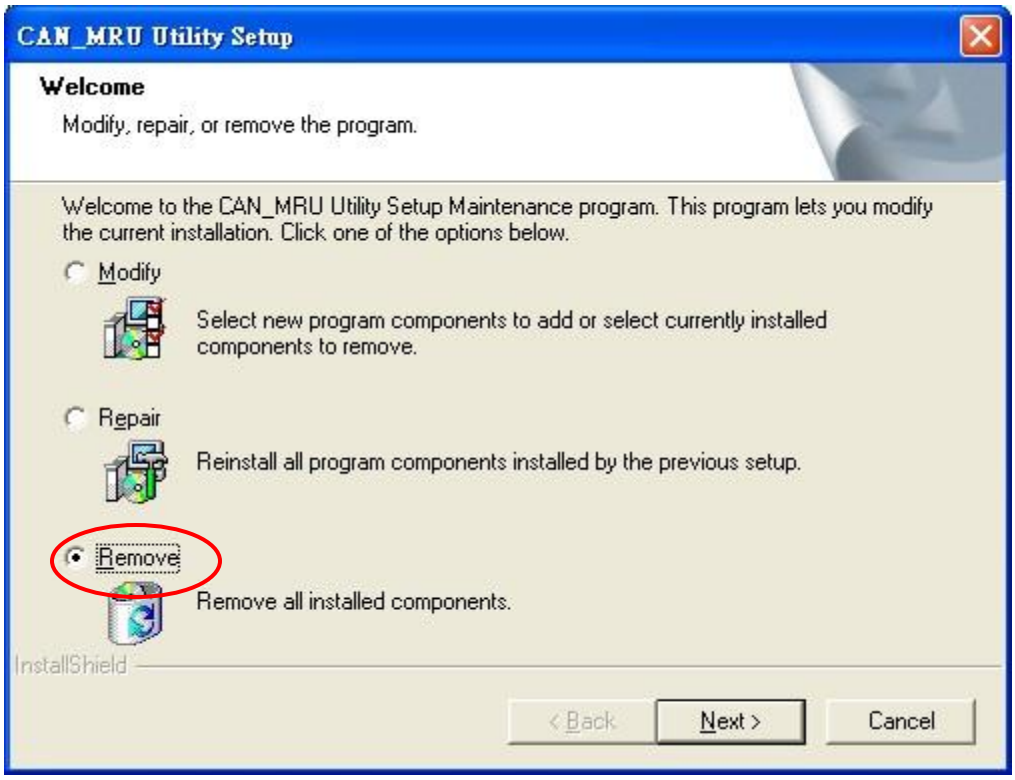
Step 2: Click the “Add/Remove” button Programs icon to open the dialog.



Step 3: Find out the CAN_MRU Utility, and click the Change/Remove button.



Step 4: Select the “Remove” item and click “Next” button to remove it.



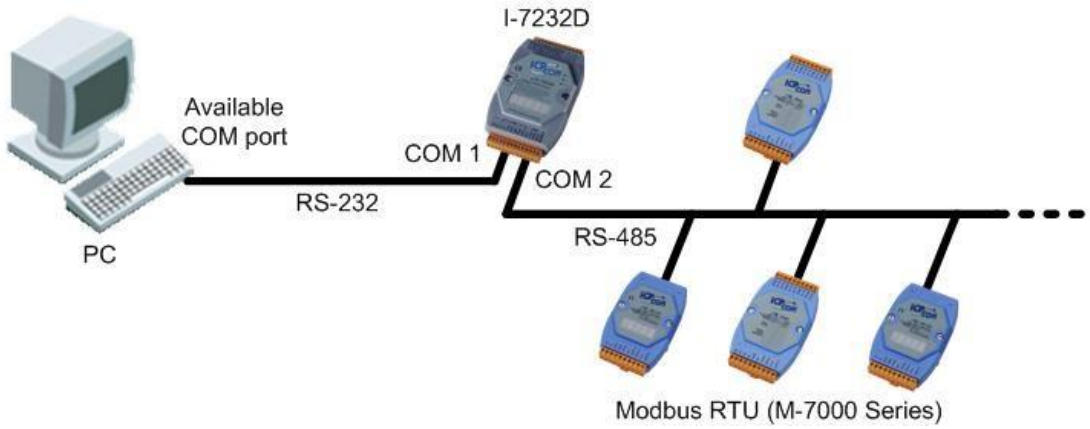
Step 5: Finally, click the button “OK” button to finish the uninstall process.


Remove Programs From Your Computer
unInstallShield will remove the software 'CAN_GW Utility' from your computer. Please wait while each of the following components is removed...
Shared program files...
Standard program files...
Folder items...
Program folders...
Program directories...
Program registry entries...
OK

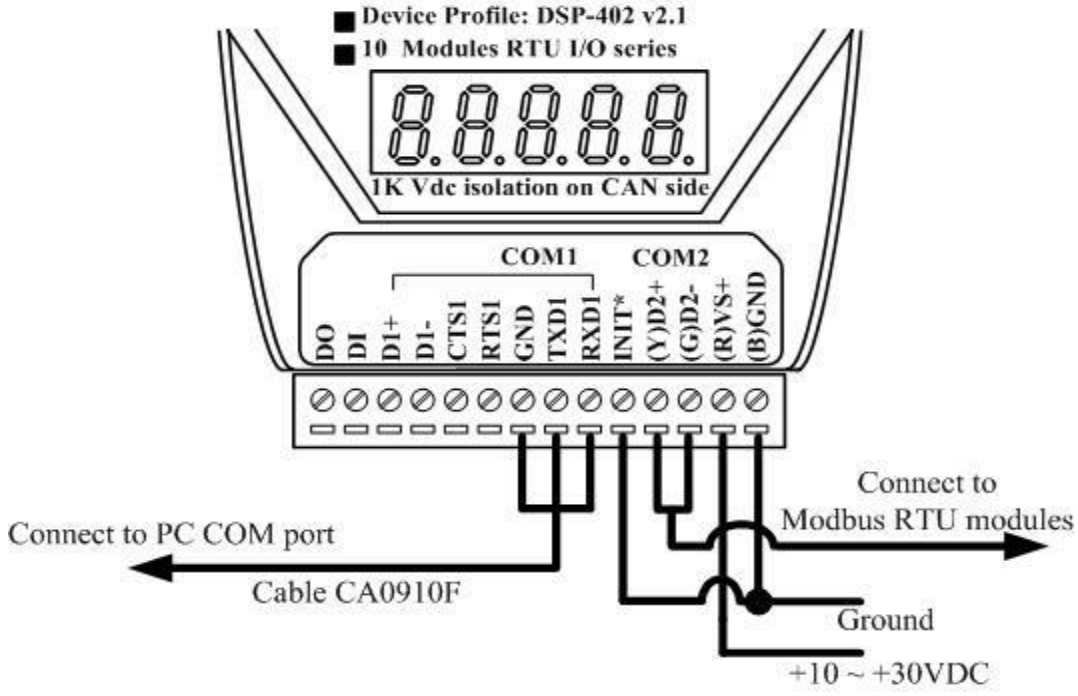
4.4 Configuration the CANopen/Modbus RTU Gateway Utility

Before using this software utility, please make sure that you have connected COM1 of the I-7232D with the available COM port on your PC. Also, connect the Modbus RTU modules with COM2 of the I-7232D. The

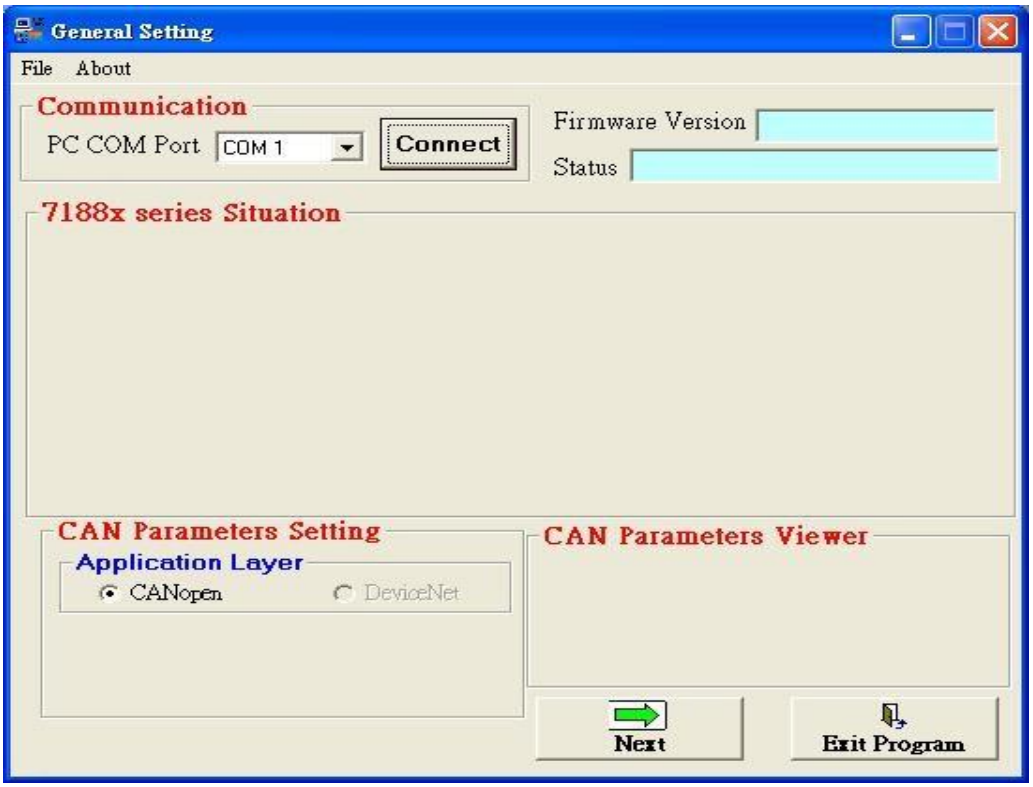
architecture is displayed in the following figure. (Note: We use the ICPDAS M-7000 series Modbus RTU modules for this demo)



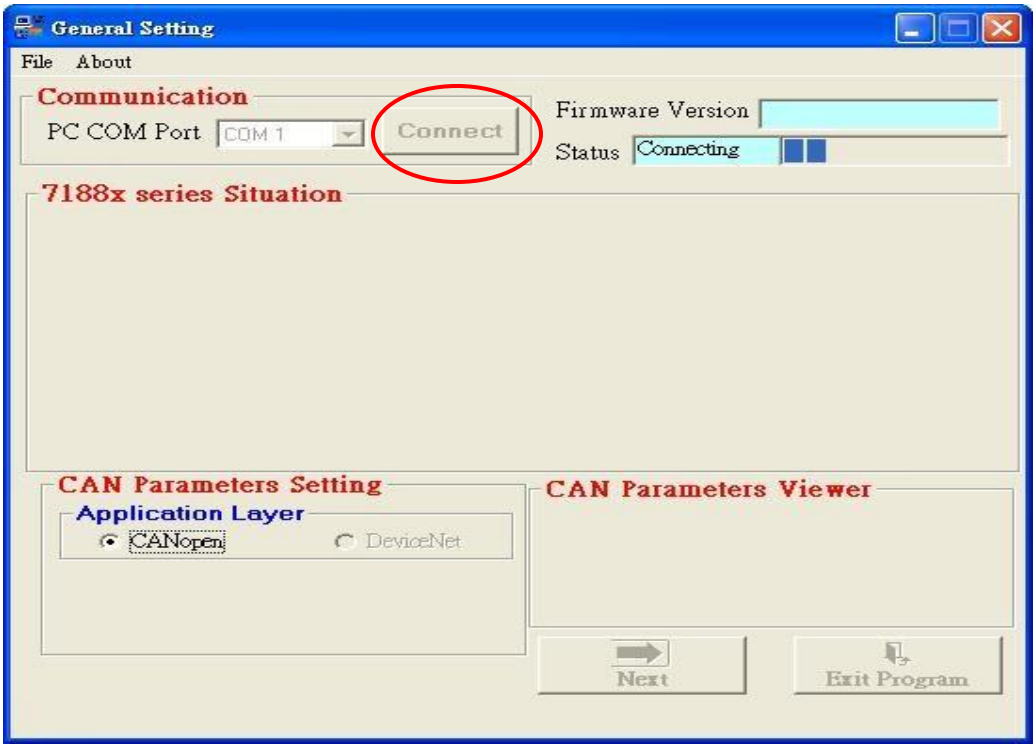
Step 1: First turn off the I-7232D. Connect the INIT* pin and GND pin on the I-7232D. Then, turn on the I-7232D.



Step 2: Execute the CAN_MRU.exe file. The following figure will be displayed.

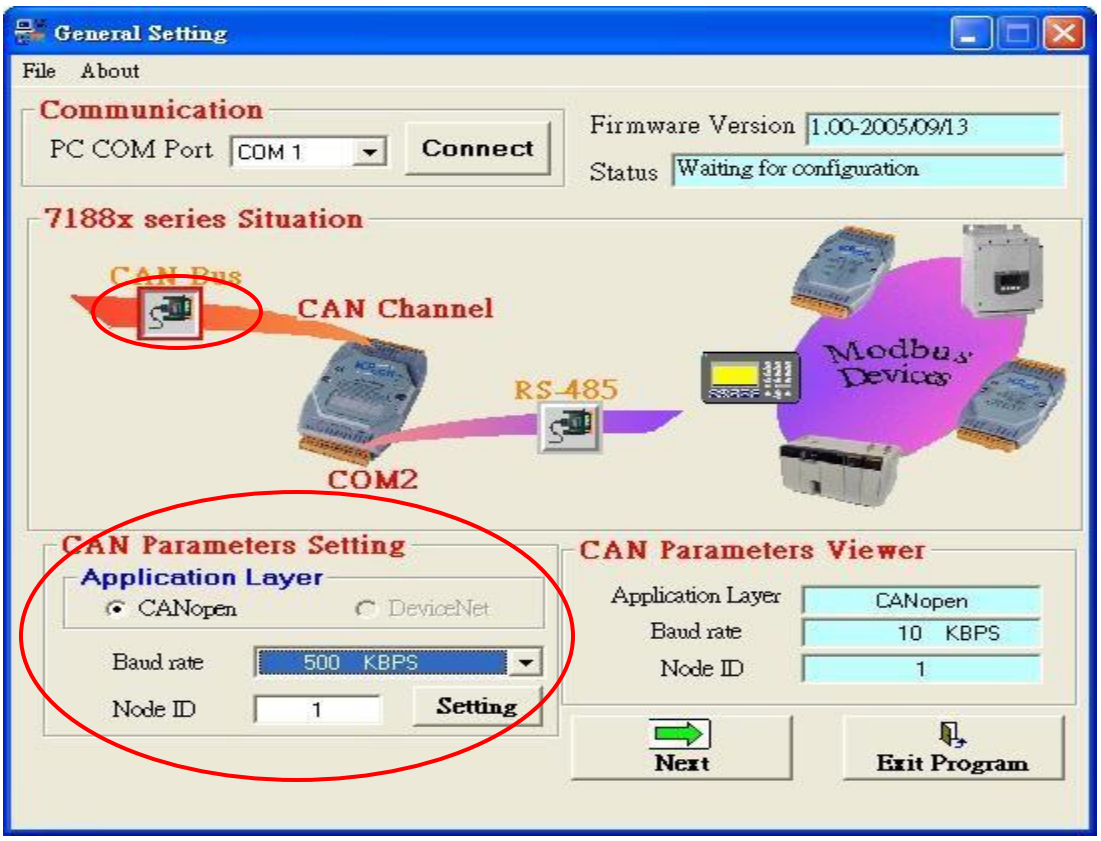


Step 3: Press the “Connect” button to connect the CANopen/Modbus RTU Gateway.

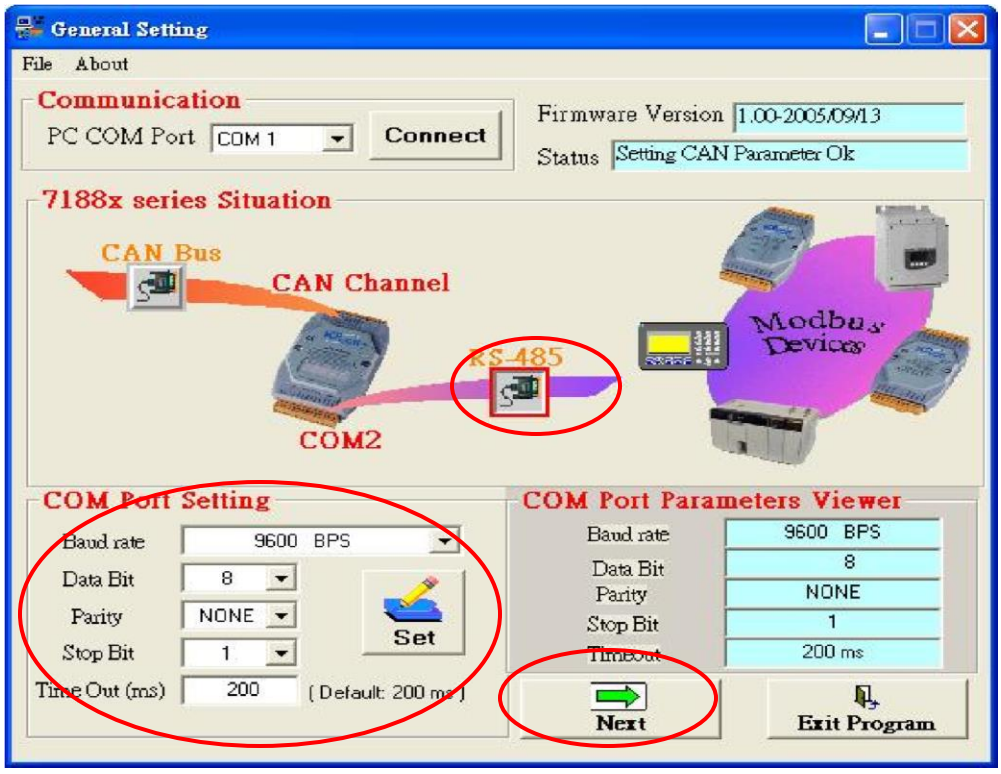


(Note: When I-7232D is not have module’s information in the EEPROM, the CANopen/Modbus RTU will response the “warning dialog”, for example when the users wire the INIT* pin connect to the GND pin, that will be happened, as following.)

Step 4: Click the “CAN Bus” button to configure the CAN parameters for the CANopen/Modbus RTU gateway.

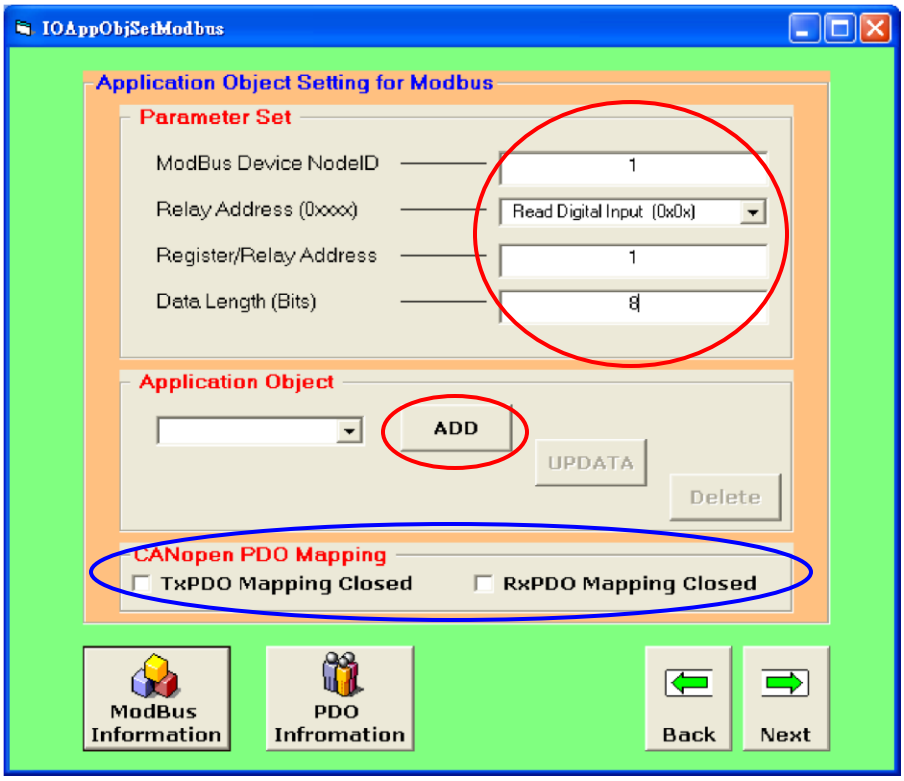


Step 5: Click the “RS-485” button to configure the RS-485 parameters for the CANopen/Modbus RTU gateway. These parameters need to match with the Modbus RTU modules communication parameters. Then click the “Next” button to next step.

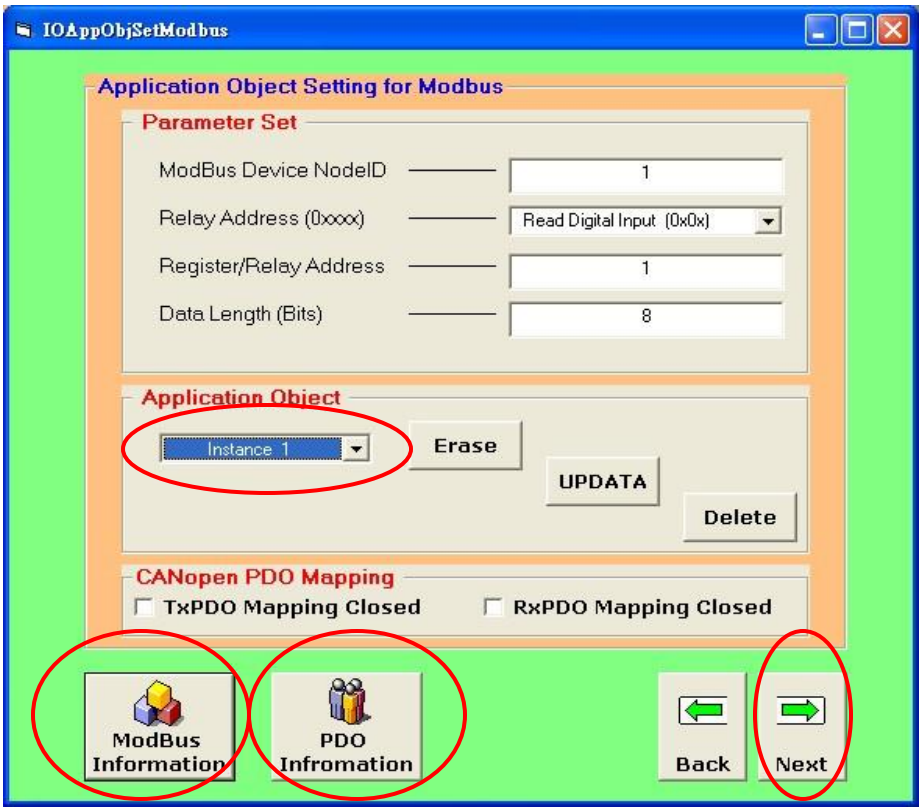


Step 6: You have to input the Modbus device NodeID, Relay Address, Register/Relay Address and Data Length for one Modbus device connected with I-7232D. These parameters are decided from this Modbus RTU device. Then click “ADD” button to save the parameters settings. Repeat the action described above if you have another Modbus device. When you finish these steps, please click “Next” button to next step.

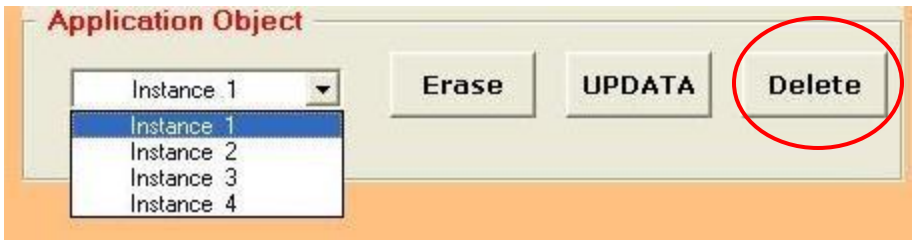
Note: If the “CANopen PDO Mapping” parameters (Tx/RxPDO Mapping Closed) are enabled, mean that the AI/AO data will be mapped from Tx/RxPDO1 (default is from Tx/RxPDO2) when the I-7232D has no any DI/DO setting.



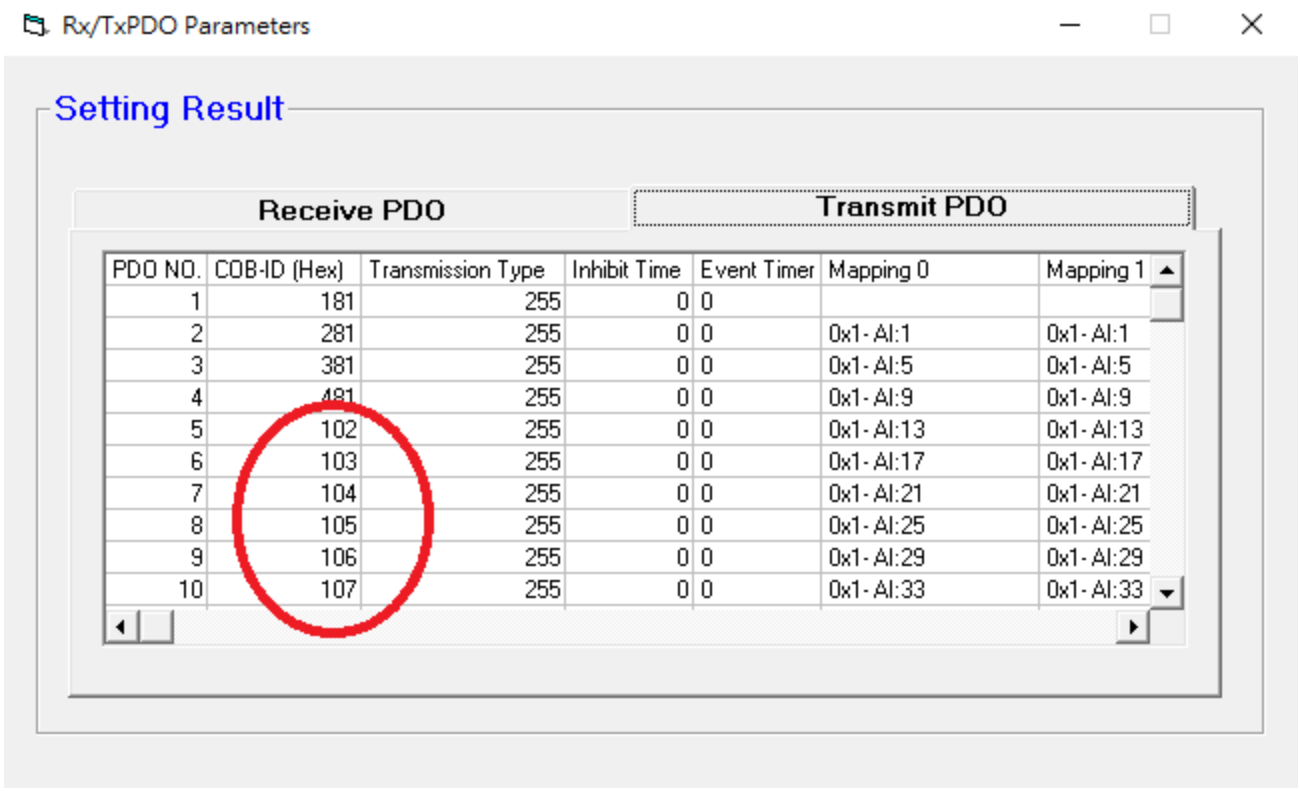
Users can click on the “PDO Information” or “Modbus Information” button to view the PDO objects or Modbus RTU devices configuration information. These information dialogs are shown below.



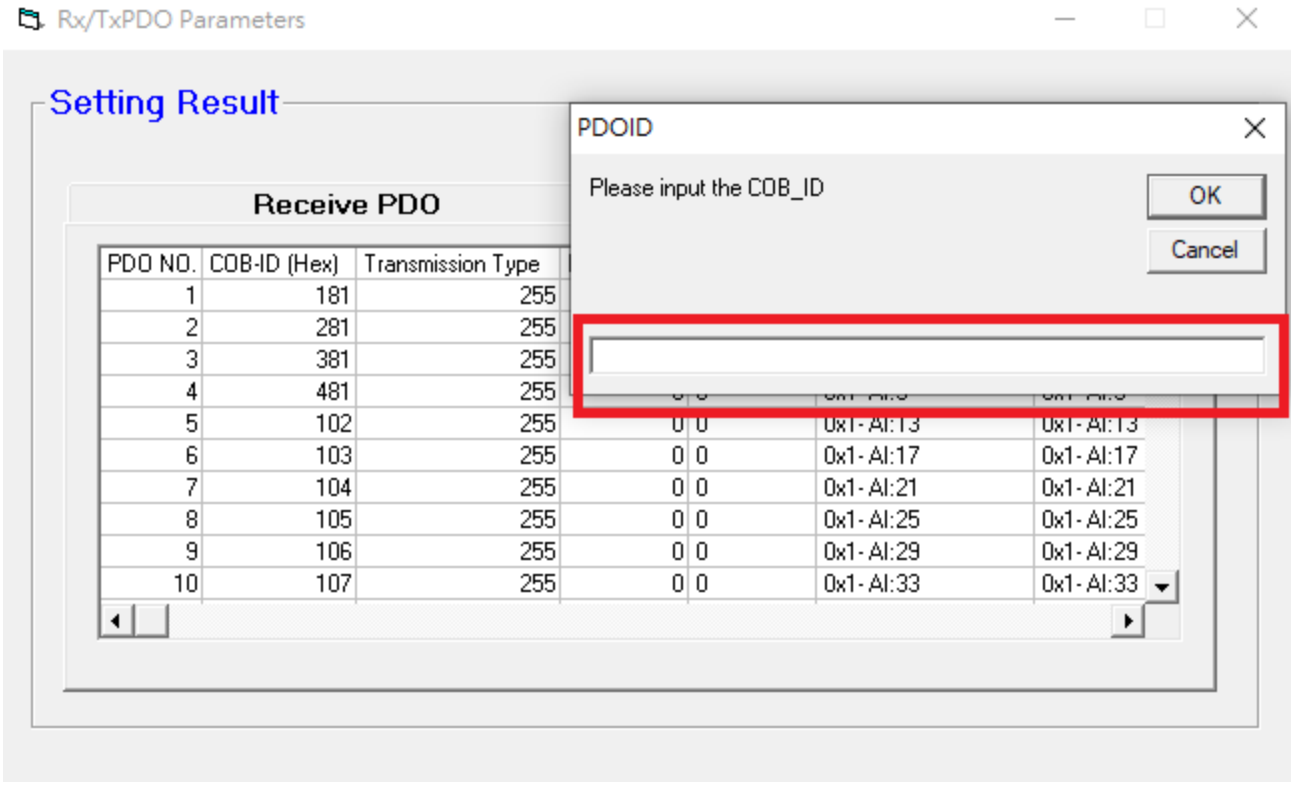
(Note: If you want to remove some device information, please select the instance number corresponding to this device, and click the “Delete” button, the device information will be removed.)



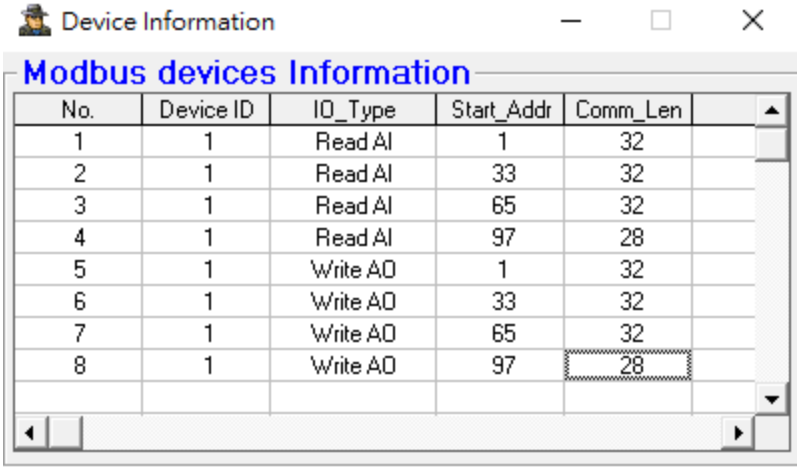
PDO Information:



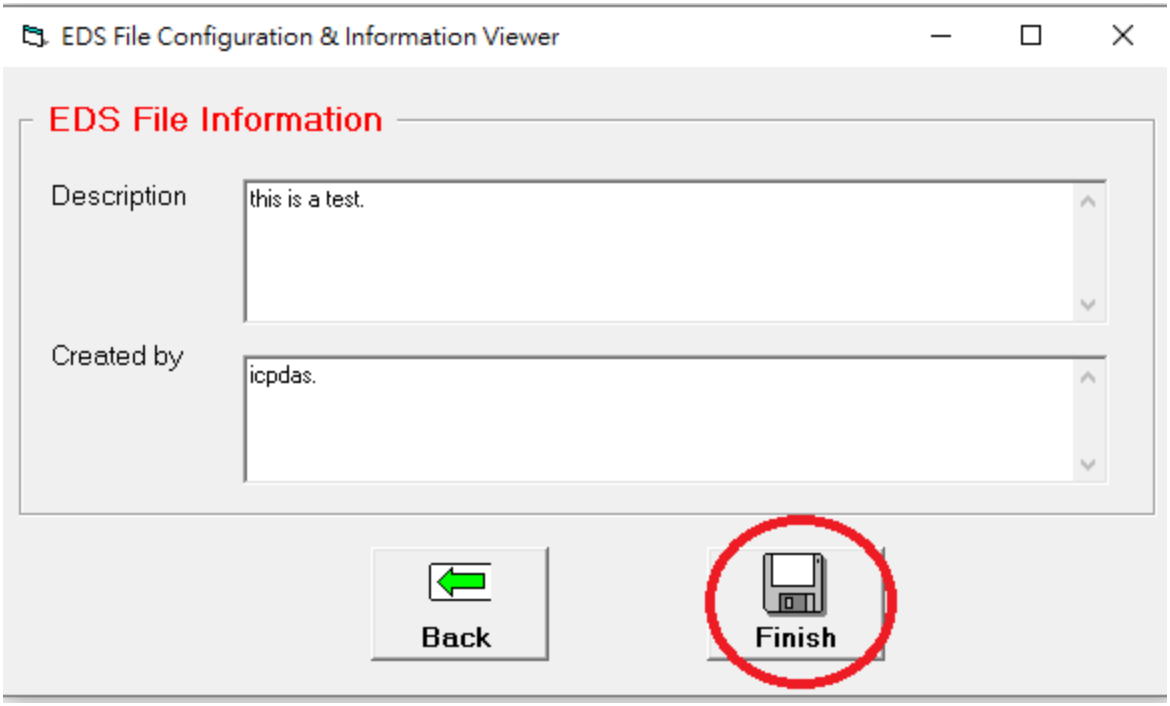
Click the column of COB-ID can change the COB-ID



Modbus Information:



Step 7: If everything is ok, click the “Finish” button to create the EDS file and save the related information into the EEPROM of the I-7232D.



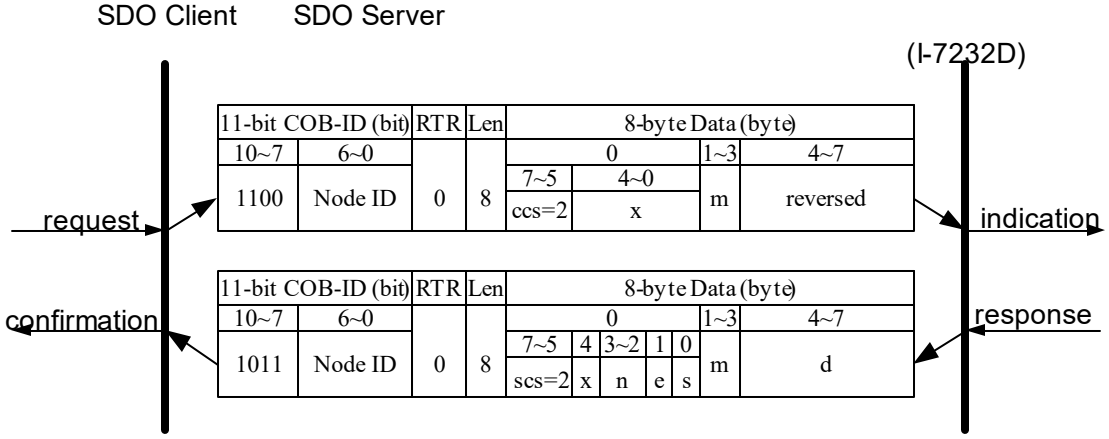
5 Configuration & Getting Start

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. During the initiate SDO upload protocol, the SDO client can tell the SDO server what object the SDO client wants to get. Also, the initiate SDO upload protocol is permitted to transfer up to four bytes of data. Therefore, if the data length of the object, which the SDO client wants to read, is equal to or less than the permitted data amount, the SDO communication can be finished by only using the initial SDO upload protocol. That is to say, 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 method of this protocol is shown as follows.



Initiate SDO Upload Protocol

ccs : client command specifier

2: initiate upload request

scs : server command specifier

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 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.

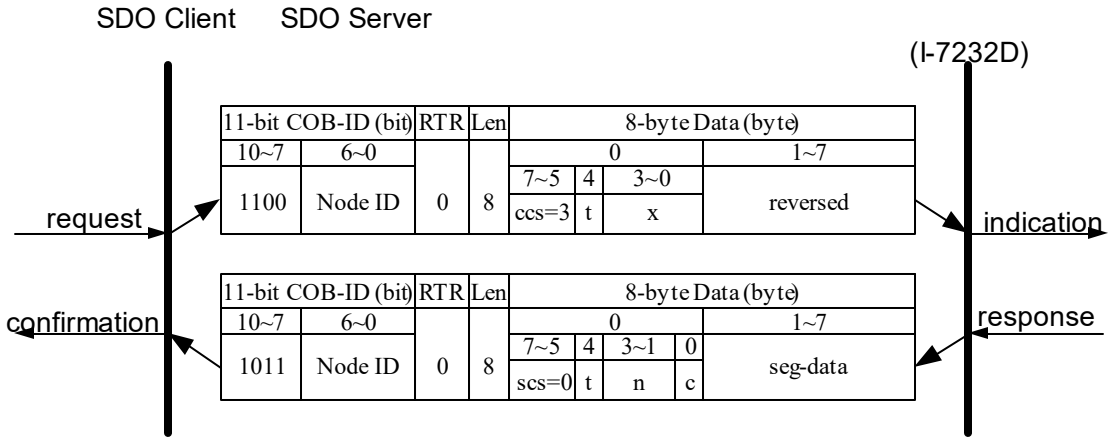
e=1, **s**=0: **d** contains unspecified number of bytes to be uploaded.

x : not used, always 0

reserved : reserved for further use , always 0

Upload SDO Segment Protocol

When the upload data length exceeds 4 bytes, the upload SDO segment protocol is needed. After finishing the transmission of the initiate SDO upload protocol, the SDO client starts to upload the data, and the upload segment protocol will follow the process shown below.



Upload SDO Segment Protocol

ccs : client command specifier

3: upload segment request **scs** :

server command specifier

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.

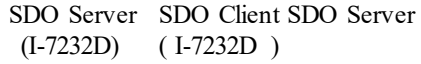
n : It indicates the number of bytes in **seg-data** that do not contain segment data. Bytes [8-**n**, 7] do not contain segment data. **n** = 0 if no segment size is indicated.

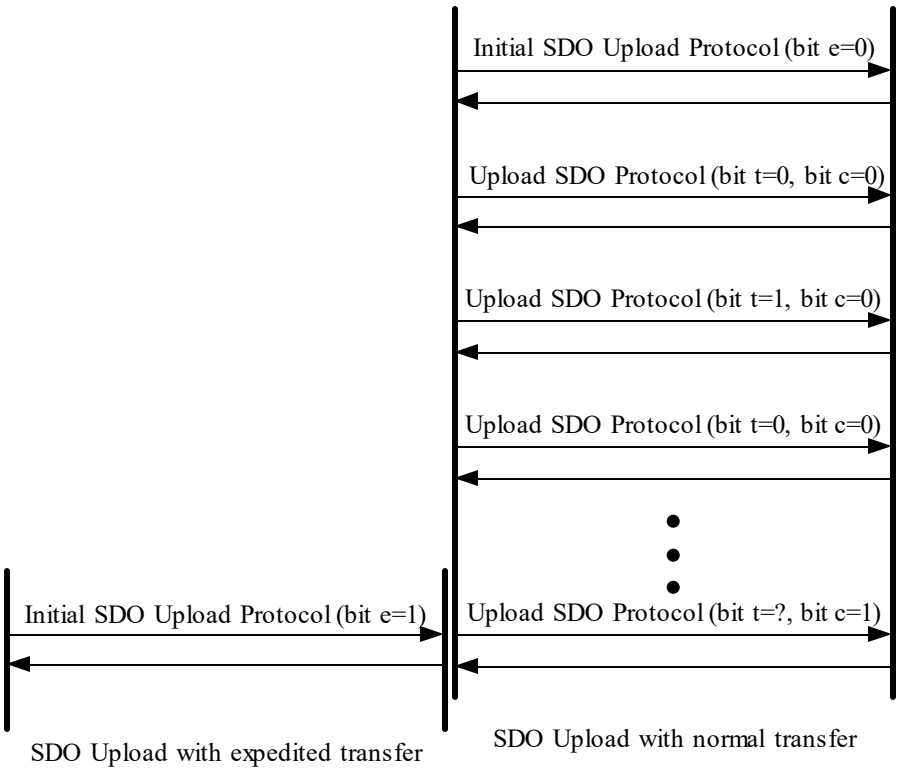
x : not used, always 0

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. The method on how to get the value stored in the object dictionary is also presented. By means of the initiate SDO upload protocol, users can obtain how many sub-indexes of the object with index 0x1400 can support. This information is located in the object with index 0x1400 with sub-index 00. Also, users can get the string located in the object with index 0x1008 by using the initiate SDO upload protocol and the upload SDO segment protocol.

- **Example for expedited transfer**

Step 1. Send the RxSDO message to the I-7232D to obtain the object entry with index 0x1400 and sub-index 00 stored in the communication profile area. The message structure is as follows. Assume that the node ID of the I-7232D is set to 1. Users can find the information about the object entry with index 0x1400 in 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 (I-7232D)**

ccs : 2

m : 00 14 00

Because low byte needs to transfer firstly, the first byte “00” is the low byte of 0x1400, the second byte “0x14” is the high byte of 0x1400, and the last byte “00” means the sub-index 00.

Step 2. I-7232D will respond 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	8	4F	00	14	00	02	00	00	00

SDO server (I-7232D) → **SDO client**

scs : 2

n : 3

e : 1

s : 1

m : 00 14 00

d : 02

Because the first byte of data indicates that only the 4th byte is valid. Therefore, the feedback value is 02.

- **Example for normal transfer**

Step 1. Send the RxSDO message to the I-7232D to obtain the object entry with index 0x1008 and sub-index 00 stored in the communication profile area. The message structure is as follows. As mentioned above, the node ID for the I-7232D is set to 1, and the information about object entry with index 0x1008 is described in 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 (I-7232D)**

ccs : 2
m : 08 10 00

Step 2. I-7232D responds to the SDO message to indicate how many bytes users will upload from the I-7232D.

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 (I-7232D)**

scs : 2
n : 0
e : 0
s : 1

m : 00 18
00

d : 09
Because the first byte from the 8-byte data indicates that only the 4th byte is valid. Therefore, the feedback value is 09, and it means that there are 9 bytes to be uploaded.

Step 3. Request the I-7232D 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 (I-7232D)**

ccs : 3 **t** : 0

Step 4. I-7232D will respond to the first 8 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	50	53	5F	44	43	4 F

SDO server (I-7232D) → **SDO client**

scs : 0 **t** : 0 **n** :
0 **c** : 0

seg-data : 43 50 53 5F 44 43 4F

Users can check chapter 6 to see that the object entry with index 0x1008 and sub index 00 has the data type “VISIBLE_STRING”. Therefore, users need to transfer these data values to the corresponding ASCII character. After transformation, they are “CPS_DCO”.

Step 5. Request the I-7232D 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 server

SDO client ⟶

(I-7232D)

ccs : 3 **t** : 1

Step 6. Receive the rest of the data 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	8	1B	4E	00	00	00	00	00	00

SDO server

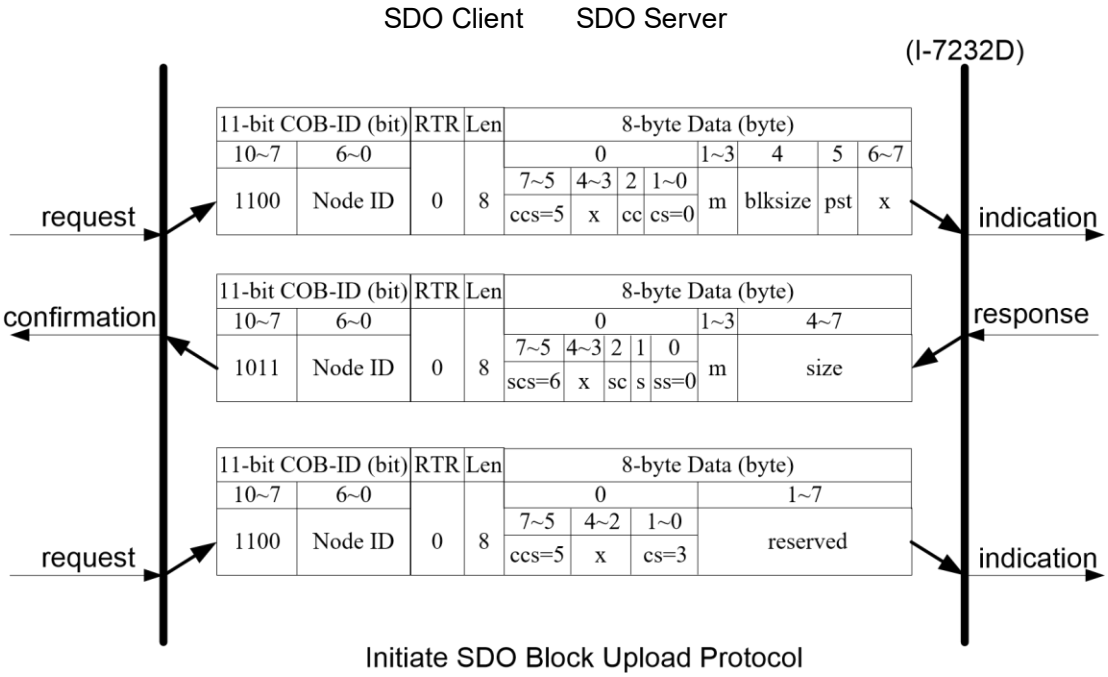
SDO client ⟵ **(I-7232D)**

scs	:	0
t	:	1
n	:	5
c	:	1
seg-data	:	4E 00 Transfer the value of 0x4E and 0x00 to the corresponding ASCII character. After transformation, it means “N ”.

5.1.2 **SDO Block Upload**

Initiate SDO Block Upload Protocol

The SDO Block Upload is usually used for large data transmission. At the beginning of the SDO Block Upload, the Initiate SDO Block Upload protocol is needed. This protocol is described below.



Initiate SDO Block Upload Protocol

ccs : client command specifier

5: block upload

scs : server command specifier 6: block upload.

cs : client subcommand

0: initiate upload request

3: start upload

ss : server subcommand 0: initiate upload response

m : multiplexor

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

cc : client CRC support **cc**=0: Client does not support generating CRC on data. **cc**=1: Client supports generating CRC on data.

sc : server CRC support **sc**=0: Server does not support generating CRC on data. **sc**=1: Server supports generating CRC on data.

pst : Protocol Switch Threshold in bytes to change the SDO transfer protocol **pst**=0: change of transfer protocol not allowed **pst**>0: If the size of the data in bytes that has to be uploaded is less or equal **pst**, the server can optionally switch to the 'SDO Upload Protocol' by transmitting the server response of the 'SDO Upload Protocol'.

s : size indicator

0: Data set size is not indicated.

1: Data set size is indicated.

size : upload size in byes **s**=0: Size is reserved for further use, always 0.

s=1: Size contains the number of bytes to be uploaded. Byte 4 contains the LSB and byte 7 is the MSB.

blksize : number of segments per block with 0 < **blksize** < 128

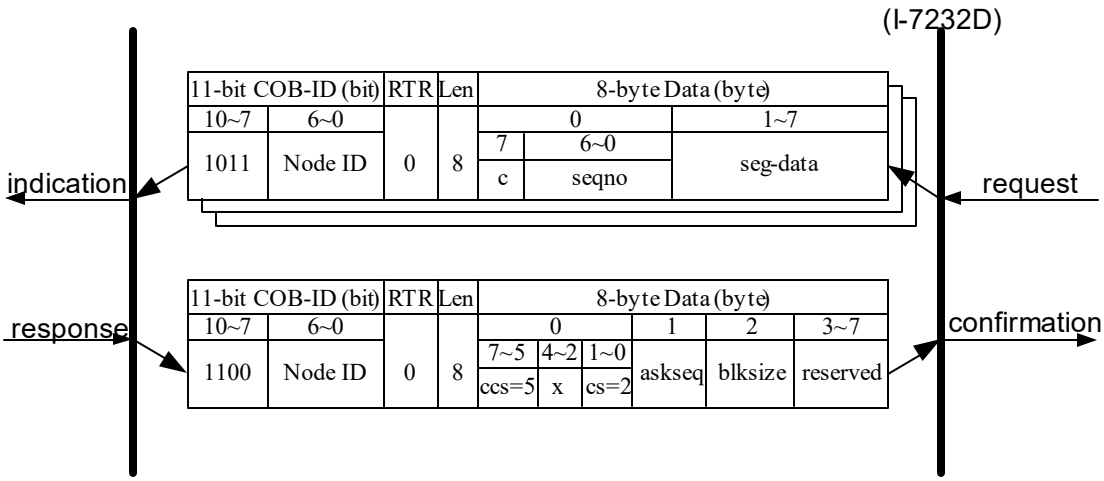
x : not used, always 0

reserved : reserved for further use , always 0

Upload SDO Block Segment Protocol

After finish the Initiate SDO Block protocol, the SDO server starts to respond to the data by using the Upload SDO Block Segment protocol. Each block contains 1 segment for minimum and 127 segments for maximum. One segment consists of 1~7 bytes. Only one block can be transmitted during an Upload SDO Block Segment protocol. The SDO server can send a maximum of 127 blocks by using 127 Upload SDO Block Segment protocols. Here is the structure of the Upload SDO Block Segment protocol.

SDO Client SDO Server



Upload SDO Block Segment Protocol

ccs : client command specifier.

5: block upload

cs : Client subcommand.

2: block upload response

c : It indicates whether there are still more segments to be uploaded.

0: more segments to be uploaded

1: no more segments to be uploaded , enter 'End block upload' phase

seqno : sequence number of segment, 0 < **seqno** < 128

seg-data : It is at most 7 bytes of segment data to be uploaded.

ackseq : sequence number of last segment that was received successfully during the last block upload

If **ackseq** is set to 0, the client indicates the server that the segment with the sequence number 1 was not received correctly and all segments have to be retransmitted by the server.

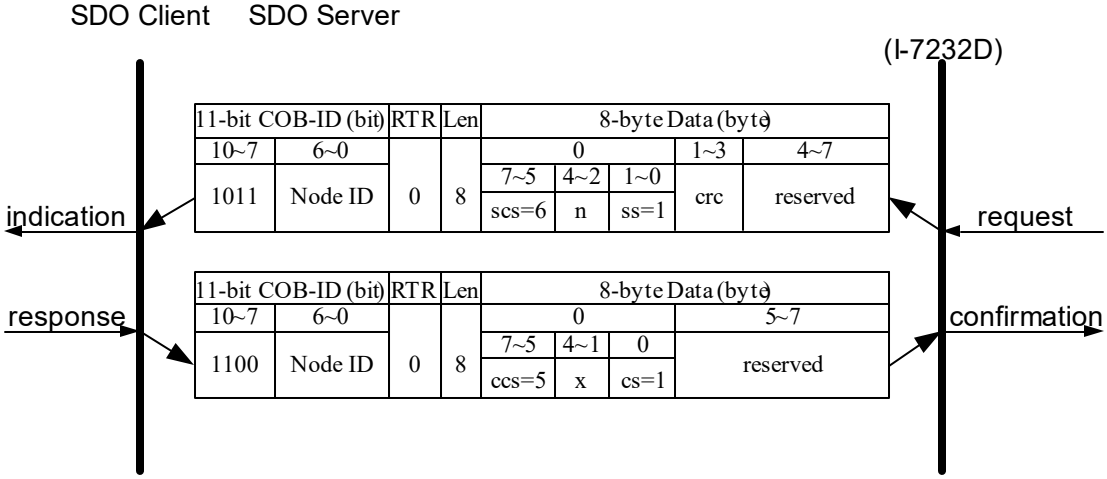
blksize : number of segments per block that has to be used by server for the following block upload with 0 < **blksize** < 128

x : not used, always 0

reserved : reserved for further use , always 0

End SDO Block Upload Protocol

The End SDO Block Upload protocol is used for finishing the SDO Block upload, and is shown in the following figure.



End SDO Block Upload Protocol

ccs : client command specifier

5: block upload

scs : server command specifier

6: block upload

cs : client subcommand

1: end block upload request

ss : server subcommand

1: end block upload response

n : It indicates the number of bytes in the last segment of the last block that do not contain data. Bytes [8-**n**,7] do not contain segment data.

crc : 16 bit Cyclic Redundancy Checksum (CRC) for the whole data set.

The algorithm for generating the CRC is as follows.

x^16+x^12+x^5+1

CRC is only valid if in Initiate Block Upload **cc** and **sc** are set to 1. Otherwise **crc** has to be set to 0. For I-7232D, it is not support CRC check mechanism.

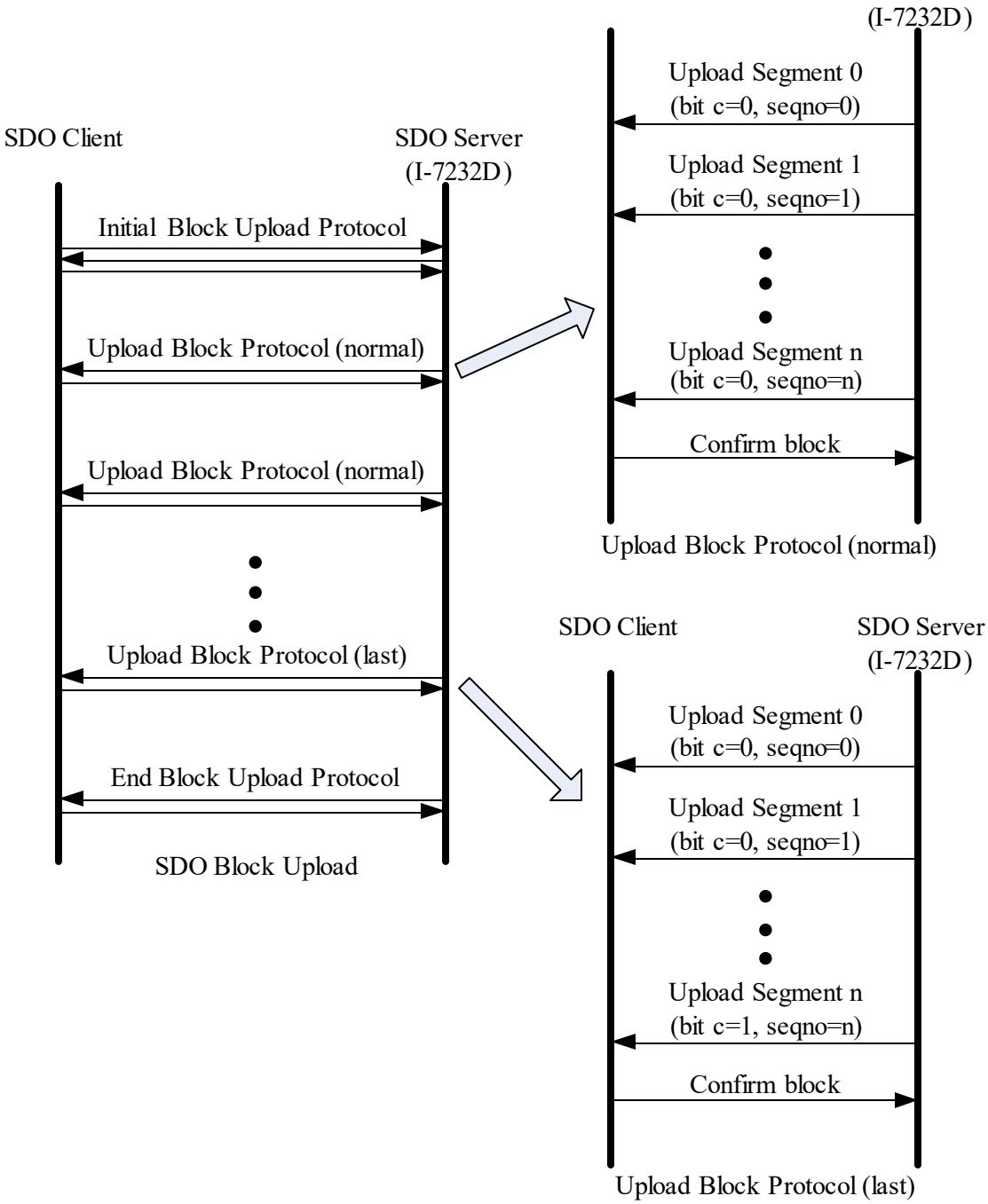
x : not used, always 0

reserved : reserved for further use , always 0

SDO Block Upload Example

The following figure indicates the general procedure for applying the SDO Block upload.

SDO Client SDO Server



By following this procedure, we provide a demo for obtaining the value of the index 0x1008 and sub-index 00 object entries.

Step 1. Request the I-7232D to transmit the data by using the SDO Block Upload method.

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	A0	08	10	00	7F	00	00	00

SDO client → **SDO server (I-7232D)**

ccs : 5
cc : 0
cs : 0
m : 08 10 00
blksize : 7F
Each block contains 127 segments.

Step 2. The I-7232D confirms the requirement with the Initiate SDO Block Upload protocol.

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	C2	08	10	00	09	00	00	00

SDO client ← **SDO server (I-7232D)**

scs : 6
sc : 0
s : 1
ss : 0
m : 08 10 00

size : 09
The I-7232D will response 9 bytes data during the SDO Block Upload.

Step 3. Send the message to finish the Initiate SDO Block Upload protocol, and inform the I-7232D 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	A3	00	00	00	00	00	00	00

SDO client → **SDO server (I-7232D)**

ccs : 5 **cs** : 3

Step 4. I-7232D responds to the first 7 bytes of data by using the Upload SDO Block Segment protocol.

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	1	43	50	53	5F	44	43	4 F

SDO client ← **SDO server (I-7232D)**

c : 0 **seqno** : 1

seg-data : 43 50 53 5F 44 43 4F

Step 5. The I-7232D transmits 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	0	1	1	0	0	0	0	0	0	1	0	8	82	4E	00	00	00	00	00	00

SDO server

SDO client ←

(I-7232D)

c : 1 **seqno**

: 2

seg-data : 4E 00

Step 6. Afterwards, users send a message to confirm the receiving data transmitted from the I-7232D.

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	A2	02	7F	00	00	00	00	00

SDO server

SDO client →

(I-7232D)

ccs : 5 **cs**

: 2 **ackseq**

: 2 **blksize**

: 7F

Step 7. When the reception confirmation is ok, the I-7232D will send a message to enter the End SDO Block Upload protocol.

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	D5	00	00	00	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 6 **n** : 5

ss : 1

Step 8. Users send a message to finish the End SDO Block Upload protocol.

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	A1	00	00	00	00	00	00	00

SDO server

SDO client ⟶

(I-7232D)

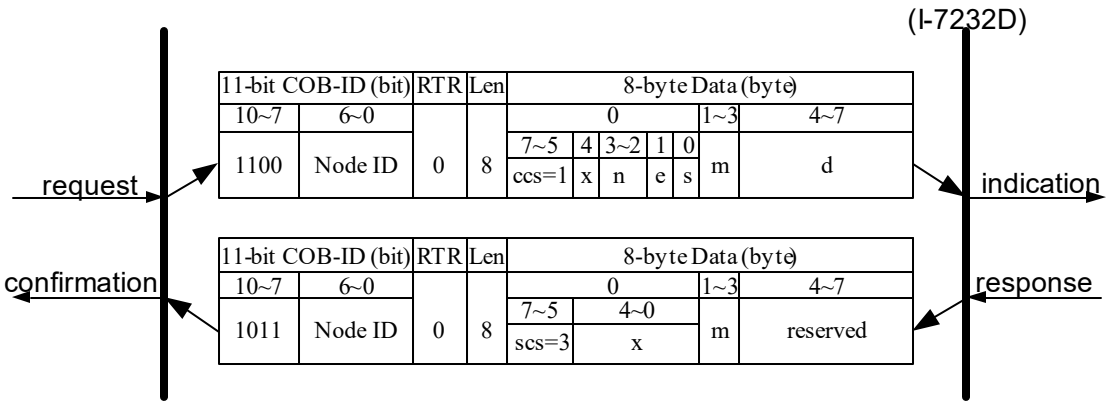
ccs : 5 **cs** : 1

5.1.3 **Download**

Initiate SDO Download Protocol

The download modes are similar to the upload modes, but different in some parameters in their 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. Or, the download segment protocol will be needed. These two protocols are shown below.

SDO Client SDO Server



Initiate SDO Download Protocol

ccs : client command specifier

1: initiate download request **scs** :

server command specifier

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.

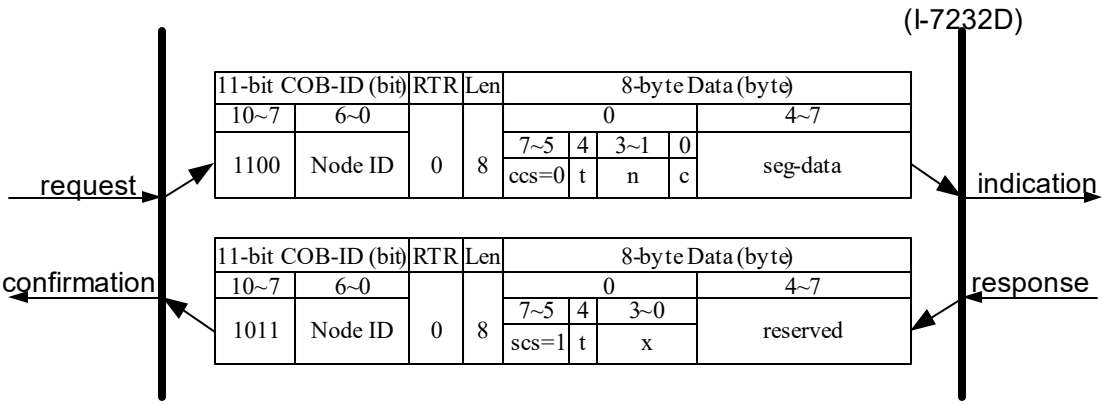
e=1,**s**=0: **d** contains unspecified number of bytes to be downloaded.

x : not used, always 0

reserved : reserved for further use , always 0

Download Segment Protocol

SDO Client SDO Server



Download SDO Segment Protocol

ccs : client command specifier

0: download segment request **scs** :

server command specifier

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

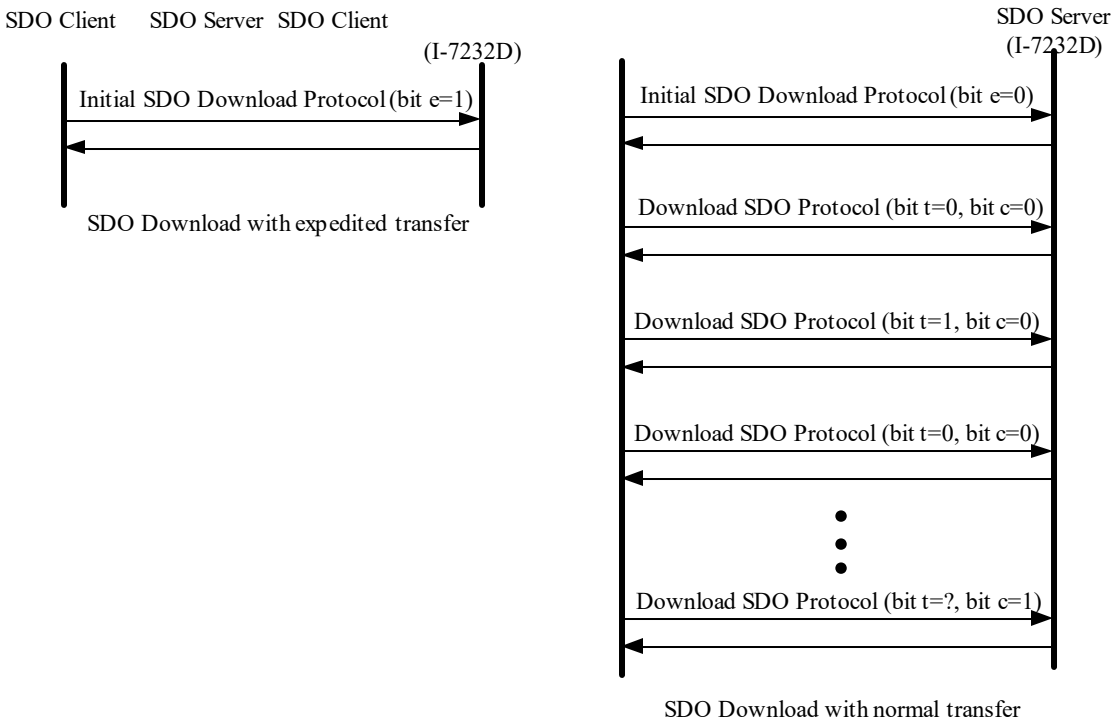
This bit must alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.

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 I-7232D are equal or less than 4 bytes, we can only provide the demo for expedited transfer.

- **Example for expedited transfer**

Step 1. Send the Rx SDO message to the I-7232D to access the object entry with index 0x1400 and sub-index 02 stored in the communication profile area. Here, change the value of this object entry to 5. Assume that the node ID for the I-7232D is set to 1.

11-bit COB-ID (bit)	RTR		8-byte Data (byte)

Func Code				Node ID								Data Length								
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 (I-7232D)**

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 00 14 02 **d**
: 05

Step 2. The I-7232D will response the message to finish the data download. Afterwards, users can use upload methods mentioned before to read back the value for confirmation.

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	60	00	14	02	00	00	00	00

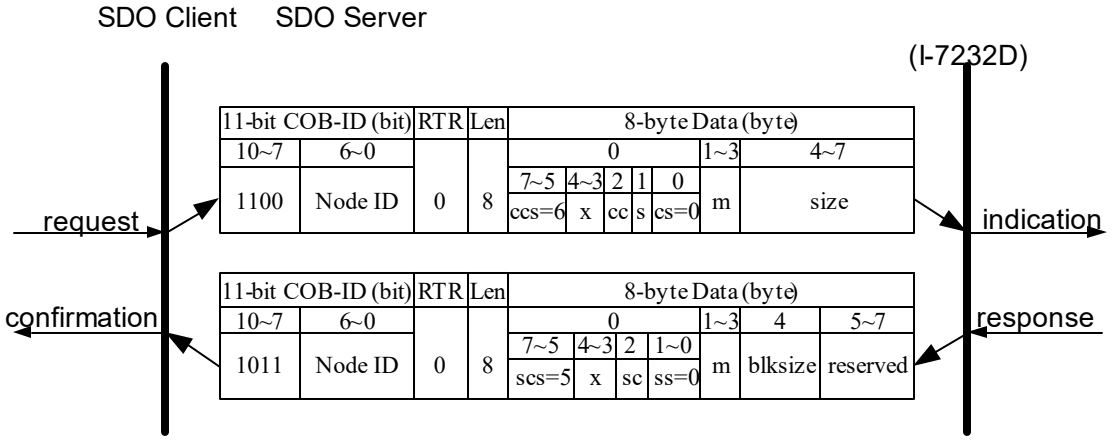
SDO client ← **SDO server (I-7232D)**

scs : 3
m : 00 14 00

5.1.4 **SDO Block Download**

The procedure of SDO Block Download is similar with the SDO Block Upload. There are three steps during the SDO Block Download. The Initiate SDO Block Download protocol is the beginning protocol for SDO Block Download. In this protocol, the SDO server and SDO client communicate each other to prepare the necessary information. Afterwards, the SDO Block Download protocol is used. And, SDO client start to send data to SDO server. After finishing the data transmission, the client and server will use the End SDO Block protocol to terminate the SDO Block Download. The following figures are the structures for the three protocols.

Initiate SDO Block Download Protocol



11-bit COB-ID (bit)		RTR	Len	8-byte Data (byte)						
10~7	6~0			0					1~3	4~7
1100	Node ID	0	8	7~5	4~3	2	1	0	m	size
				ccs=6	x	cc	s	cs=0		

11-bit COB-ID (bit)		RTR	Len	8-byte Data (byte)						
10~7	6~0			0					1~3	4	5~7
1011	Node ID	0	8	7~5	4~3	2	1~0		m	blksize	reserved
				scs=5	x	sc	ss=0				

Initiate SDO Block Download Protocol

ccs : client command specifier

6: block download

scs : server command specifier

5: block download

s : size indicator

0: Data set size is not indicated.

1: Data set size is indicated.

cs : client subcommand

0: initiate download request

ss : server subcommand

0: initiate download response

cc : client CRC support **cc**=0: Client does not support generating CRC on data. **cc**=1: Client supports generating CRC on data.

sc : server CRC support **sc**=0: Server does not support generating CRC on data. **sc**=1: Server supports generating CRC on data.

m : multiplexor

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

size : download size in byes **s**=0: Size is reserved for further use, always 0.

s=1: Size contains the number of bytes to be downloaded. Byte 4 contains the LSB and byte 7 is the MSB.

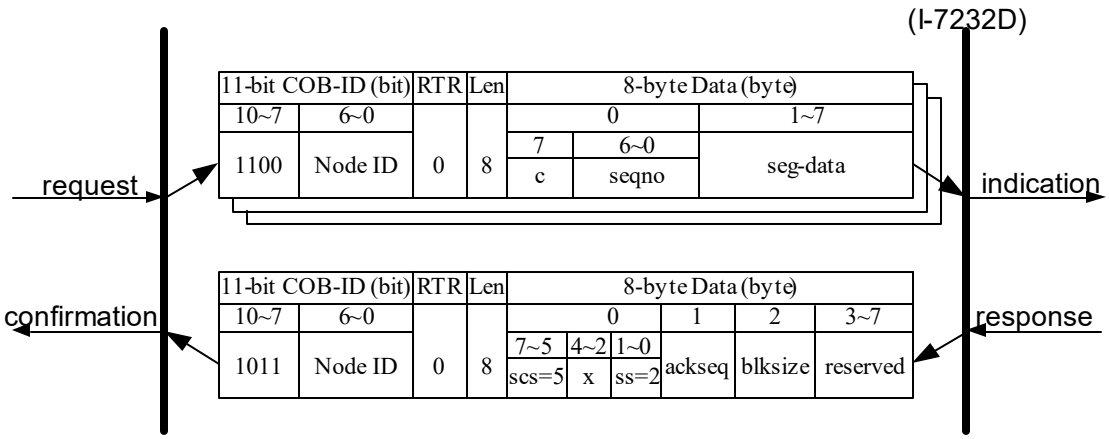
blksize : number of segments per block with 0 < **blksize** < 128

x : not used, always 0

reserved : reserved for further use , always 0

Download SDO Block Segment Protocol

SDO Client SDO Server



Download SDO Block Segment Protocol

scs : server command specifier

5: block download

ss : server subcommand

0: initiate download response **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 , enter 'End block download' phase

seqno : sequence number of segment, 0 < **seqno** < 128

seg-data : It is at most 7 bytes of segment data to be downloaded.

ackseq : sequence number of last segment that was received successfully during the last block download

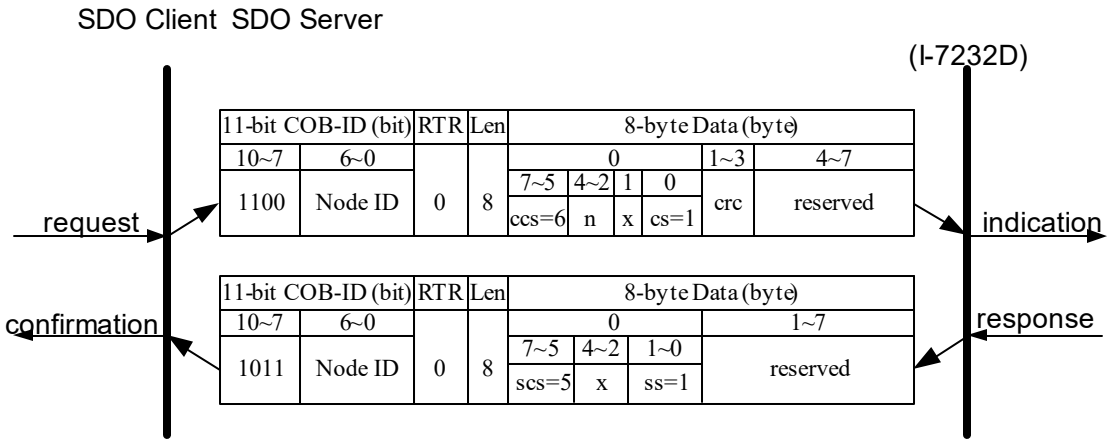
If **ackseq** is set to 0, the server indicates the client that the segment with the sequence number 1 was not received correctly and all segments have to be retransmitted by the client.

blksize : number of segments per block that has to be used by client for the following block download with 0 < **blksize** < 128

x : not used, always 0

reserved : reserved for further use , always 0

End SDO Block Download Protocol



End SDO Block Download Protocol

ccs : client command specifier.

6: block download

scs : server command specifier.

5: block download

cs : client subcommand

1: end block download request

ss : server subcommand

1: end block download response

n : It indicates the number of bytes in the last segment of the last block that do not contain data. Bytes [8-**n,**7] do not contain segment data.

crc : 16 bit Cyclic Redundancy Checksum (CRC) for the whole data set.

The algorithm for generating the CRC is as follows.

x^16+x^12+x^5+1

CRC is only valid if in Initiate Block Download cc and sc are set to 1. Otherwise, CRC has to be set to 0. For I-7232D, it is not support CRC check mechanism.

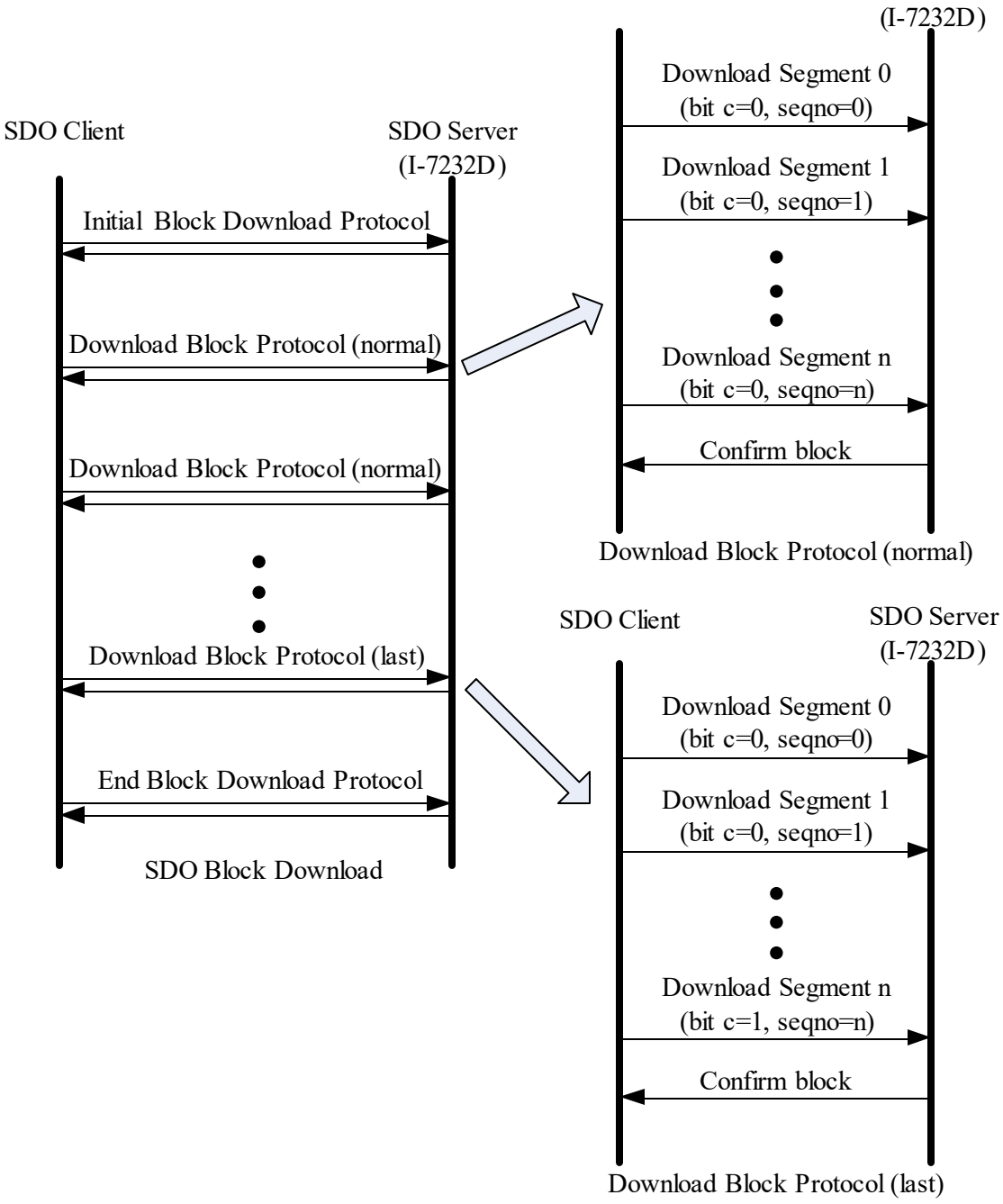
X : not used, always 0

reserved : reserved for further use , always 0

SDO Block Download Example

In this demo, the value of the object entry with index 0x1400 and sub-index 0x02 will be changed to 5 by using the SDO Block Download communication method. When the SDO Block Download is running, the procedure looks as follows.

SDO Client SDO Server



Step 1. In order to inform the I-7232D that the value of the object entry with index 0x1400 and sub-index 02 will be modified by using the SDO Block Download method, the Initiate SDO Block Download protocol is implemented.

11-bit COB-ID (bit)	RTR		8-byte Data (byte)

Func Code				Node ID								Data Length								
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	C0	00	14	02	00	00	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 6
cc : 0
s : 0
cs : 0
m : 00 14
02
size : 0
Because the value of **s** is 0, the **size** is not used.

Step 2. I-7232D responds to the message by using the Initiate SDO Block Download protocol. Afterwards, the SDO client can start to download the object's data with index 0x1400 and sub-index 02 to I-7232D.

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	A0	00	14	02	7F	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 5
sc : 0
s : 0
ss : 0
m : 00 14
02
blksize : 7F

Step 3. The SDO client starts to transmit the data of the object entry index 0x1400 and sub-index 02 by using the Download SDO Block Segment protocol. Seeing as the data length of the value is less than the maximum data length of one block, the SDO Block Segment Download protocol is only implemented once.

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	81	05	00	00	00	00	00	00

SDO client → **SDO server (I-7232D)**

c : 1 **seqno** : 1

seg-data : 05

Step 4. I-7232D responds to the message to confirm if the transmission is successful or not. If not, this block needs to be transmitted again. After finishing the data transmission, the Download SDO Block Segment protocol is terminated.

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	A2	01	7F	00	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 5 **ss** : 2

ackseq : 01 **blksize** : 7F

Step 5. The SDO client sends the ending message to finish the SDO Block 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	1	0	0	0	0	0	0	0	0	1	0	8	D5	00	00	00	00	00	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 6 **n** : 5

cs : 1

crc : 00 00

Step 6. I-7232D responds to the message to terminate the End SDO Block Download protocol.

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	A1	00	00	00	00	00	00	00

SDO server

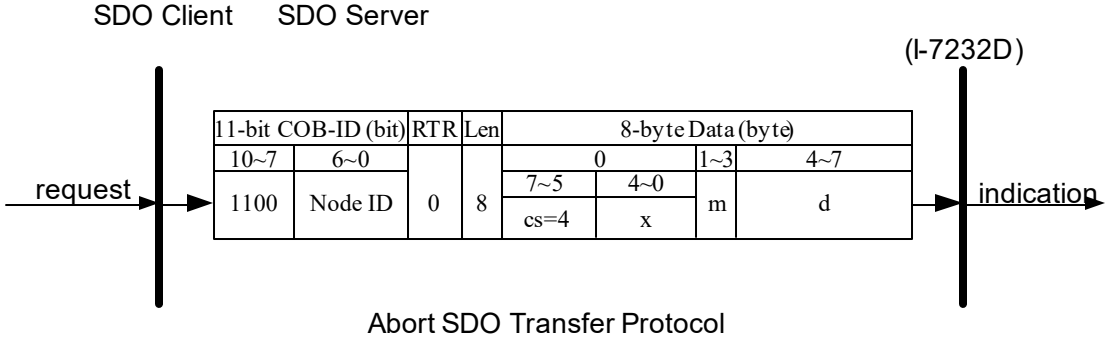
SDO client ⟵

(I-7232D)

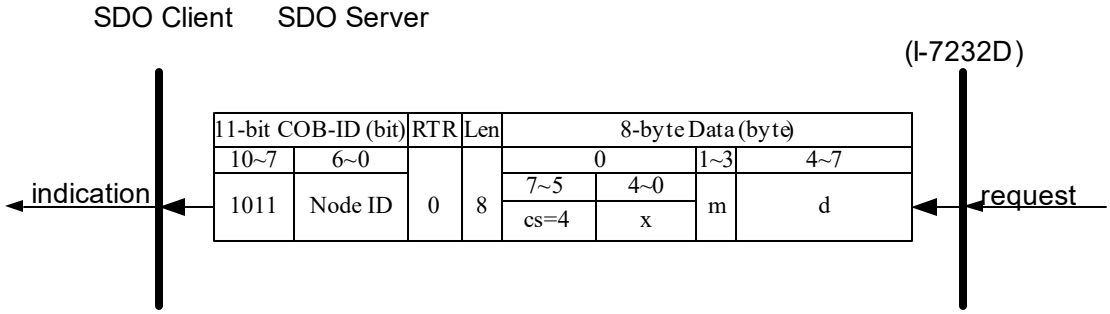
scs : 5 **ss** : 1

5.1.5 **Abort SDO Transfer Protocol**

In some situations, the SDO client or SDO server needs to terminate the SDO transmission. For example, the value of entries that users want to modify does not exist or is read-only, or users wouldn't like to continue with the uncompleted SDO protocol under some special conditions. When these situations 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 specifier.

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 specifier 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 an 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 an file error).

Abort SDO Transfer Example

The object index 0x1008 doesn't have the sub-index 01 entry. Therefore, if users read the object entry with index 0x1008 and sub-index 01, the I-7232D will response the Abort SDO Transfer message. We will also use this point as a demo to follow.

Step 1. Send the Rx SDO message to the I-7232D to obtain the object entry with index 0x1008 and sub-index 01. Assume that the node ID for the I-7232D 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 (I-7232D)**

ccs : 2

m : 08 10 01

Step 2. I-7232D will respond to the Abort SDO message as its indication.

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 server

SDO client ◄

(I-7232D)

cs : 4

m : 08 10 01

d : 11 00 09 06

Because low byte needs to transfer firstly, the data are “06 09 00 11” after converting. 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 using the PDO to transmit the real-time data, it is necessary to check the COB-ID parameter of this PDO in the PDO communication objects. This parameter determines the COB-ID of the PDO communication. It has 32 bits, and the meaning of each bit 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: I-7232D only supports CAN 2.0A.

In the following table, the default PDO COB-ID parameters are presented.

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, user can define all of the COB-ID except the reserved COB-ID described in the table in section 3.1. When users want to define the COB-ID, it is important to avoid the conflict with the 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 several parameters defined in PDO communication objects with sub-index 02. Each PDO has its own transmission type. The transmission type indicates the transmission/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 1st 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. Transmission type 1-240 indicates how many SYNC objects the TxPDO will be triggered by. The RxPDO is always triggered by the following SYNC upon reception of data independent of the transmission types 0-240.

2. Transmission type 252 and 253 are only used for TxPDO. Transmission type 252 means that the data is updated (but not sent) immediately after reception of the SYNC object. The PDO is only

transmitted on remote transmission requests for these two transmission types.

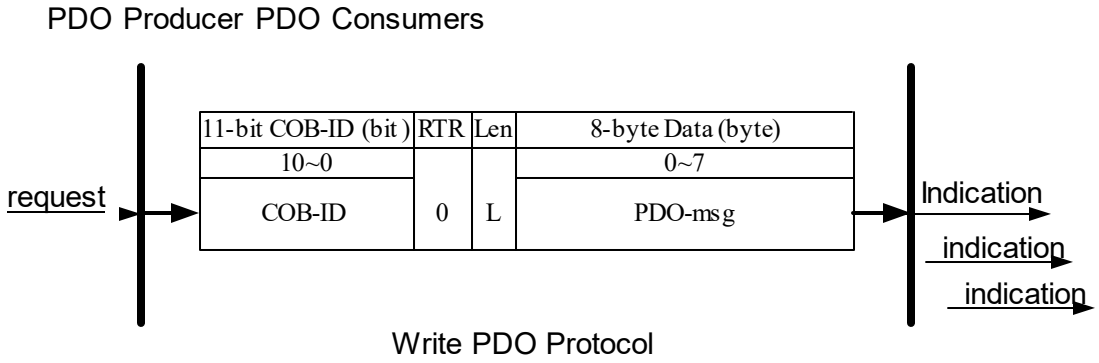
3. For the transmission types 254 and 255, the event timer can be used in the TxPDO. The PDO, which includes the DI value, will be sent when the DI value is changed. For the RxPDO, both of these two types mean that receiving the RxPDO will directly trigger an update of the mapped data.

5.2.3 **PDO Communication Rule**

The PDO related objects are indicated from index 0x1400 to 0x1BFF. For the I-7232D, RxPDO communication objects are from index 0x1400 to index 0x141F, and RxPDO mapping objects are from index 0x1600 to index 0x161F. The ranges of the TxPDO communication objects and the mapping objects are from index 0x1800 to index 0x181F and from index 0x1A00 to index 0x1A1F 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 the couple, 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 handled.

Besides, PDO communications can be only applied in the NMT operational state. Users can use the NMT module control protocol to change the NMT state of the I-7232D. It is described in section 5.3. Incidentally, 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 processed 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.

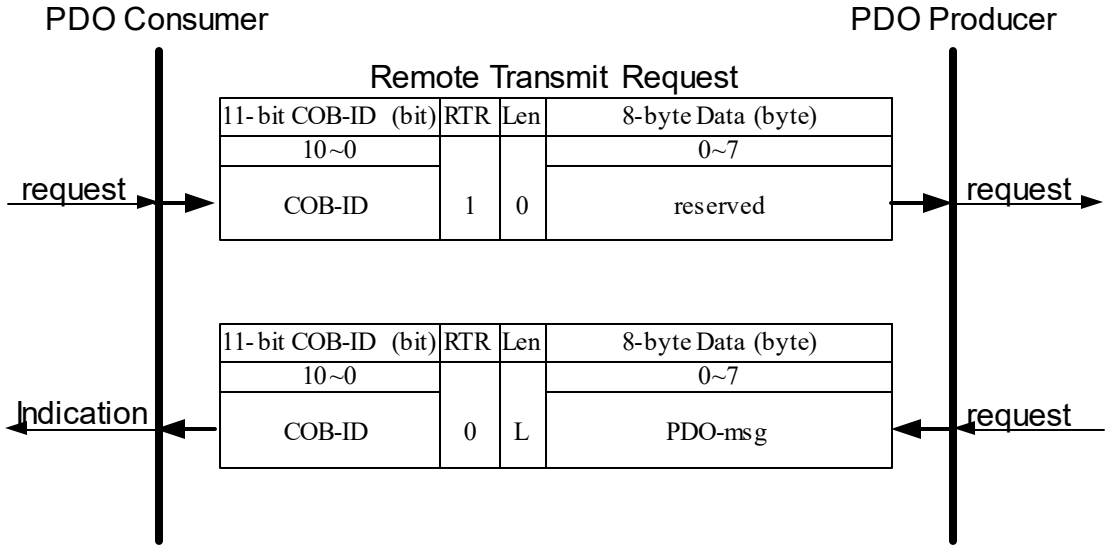


Write PDO Protocol

COB-ID : the default PDO COB-ID, or the PDO COB-ID 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 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

PDO Communication Example

In order to take a look at a PDO communication demo, some M-7000 modules may be needed. They are M-7017RD, M-7024, M-7055D and M-7052D, and their RS-485 module addresses are configured as 0x05, 0x04, 0x03, and 0x01 respectively. The output range of the M-7024 and input range of the M-7017RD modules are 0~10V and -10V~+10V respectively. The Baud Rate of these modules is set to 9600bps and the checksum is set to disable. When users want to configure the M-7000 modules, the following procedure is the best for reference.

Note: We use the ICPDAS M-7000 series Modbus RTU modules for this demo, if users want to use any other Modbus RTU device, please configure the communication parameters of this device firstly.

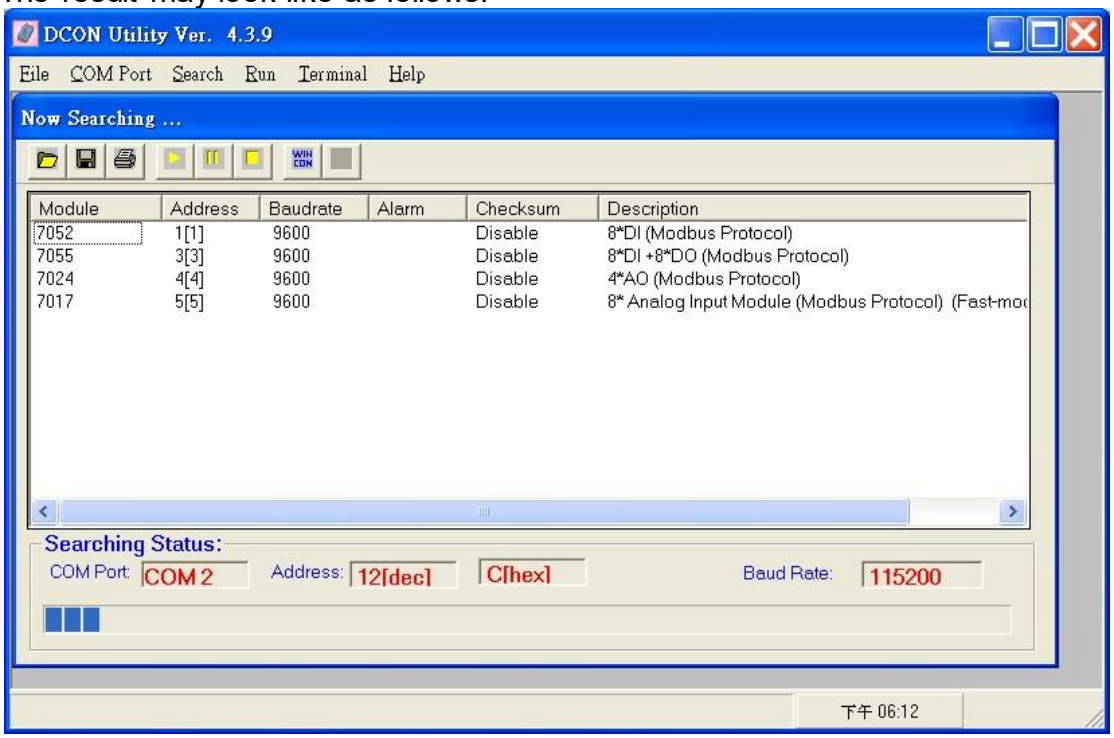
- Connect the PC RS-485 COM port with the RS-485 port on one of the Modbus modules. If there is no RS-485 COM port on your PC, the RS-232/RS-485 converter, I-7520, may be needed.
- Turn off the I-7520 and the M-7000 module. Then, connect the PC's RS232 to I-7520, and connect I-7520's RS485 port to M-7000 module, Afterward, please turn on the I-7520 and the M-7000 module, and configure the M-7000 module by using DCON Utility. For more information about DCON Utility description, please refer to the quick start on the DCON Utility on-line Help. Users can download the DCON Utility on the following web site.

  http://www.icpdas.com/download/7000/7000.htm

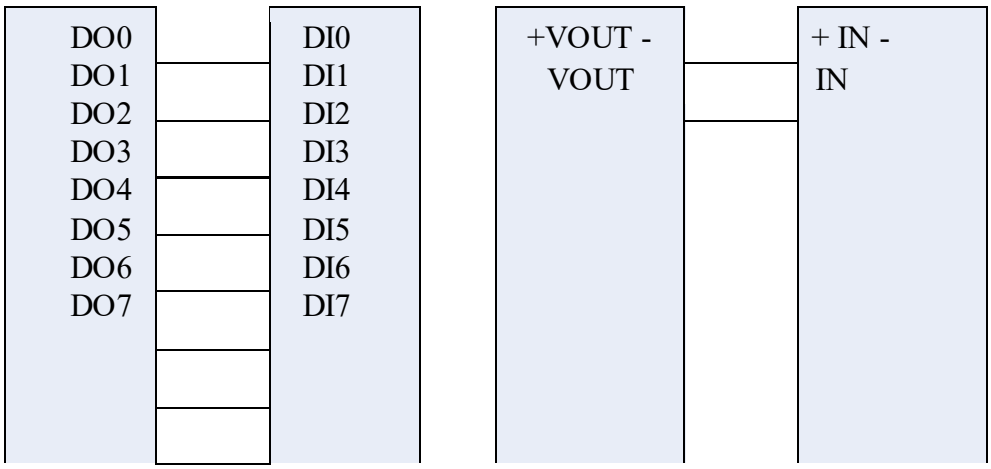
- Repeat the steps mentioned above until all of the M-7000 modules have been configured.

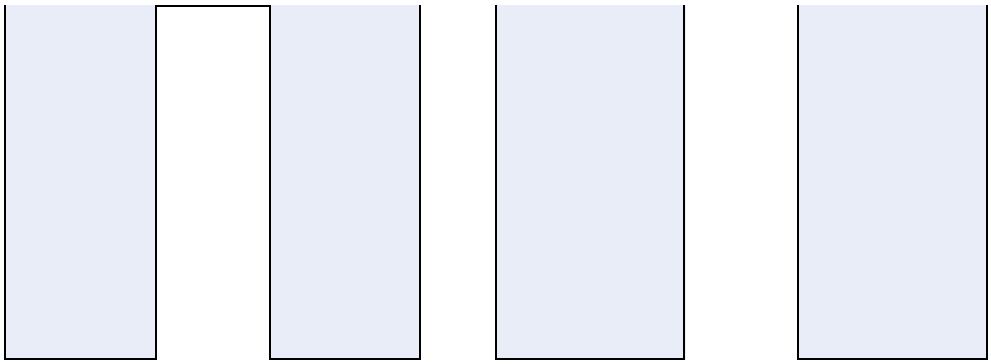
After finishing the configurations, users can connect all the M-7000 modules to their PC simultaneously, and scan them by using the DCON Utility.

The result may look like as follows.



After completing your configurations, connect these four M-7000 modules with COM2 of the I-7232D, and each I/O channels for these modules should be wired as follows.



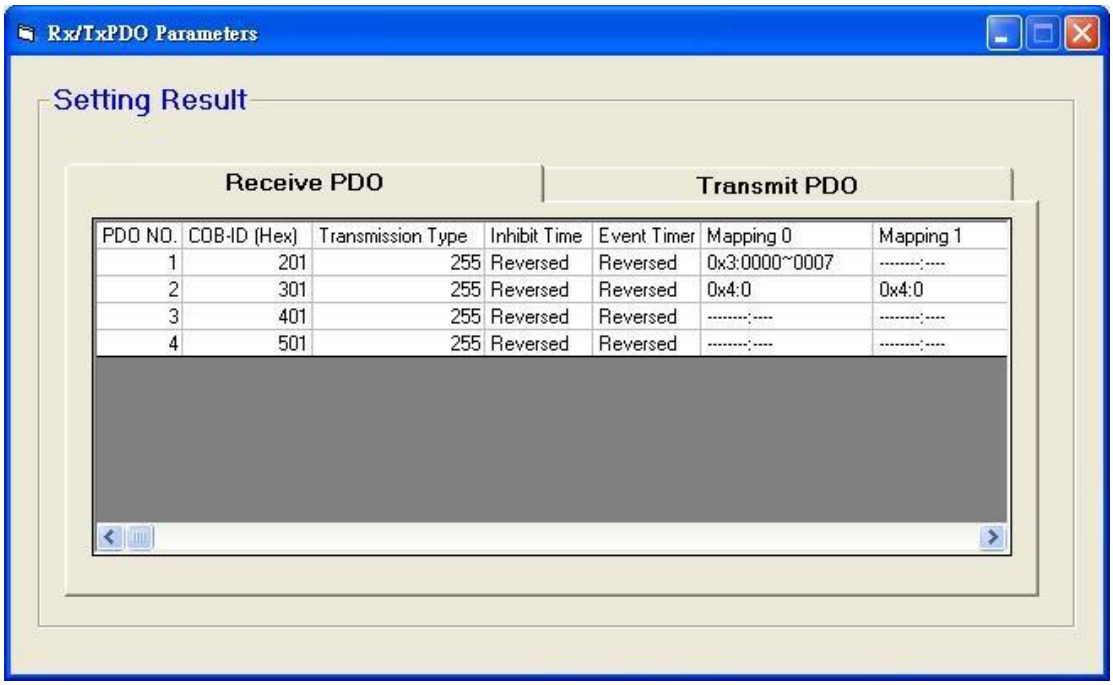


M-7055DM-7052DM-7024 M-7017R

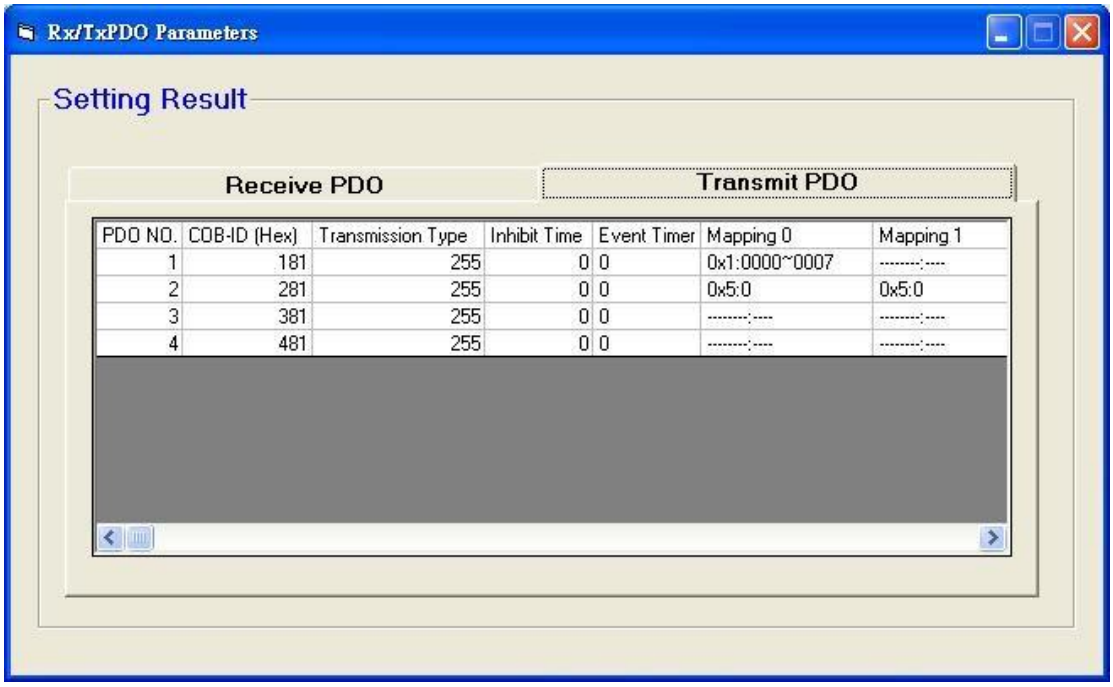
Use the CANopen/Modbus RTU Gateway Utility to set the node ID of the I-7232D, CAN bus baud rate, RS-485 baud rate, 500Kbps, 9600bps, and disable checksum respectively. The parameter information of these M-7000 series modules are shown in the following table.

No.	Device ID	IO_Type	Start_Addr	Comm_Len
1	1	Digital Input	1	8
2	3	Digital Output	1	8
3	4	Analog Output	1	1
4	5	Analog Input	1	1

Afterwards, users can get the information shown as follows.



RxPDO Information



TxPDO Information

After finishing the preparations, we will introduce several functions of PDO communication in this demo. They are shown as follows.

- Access digital I/O & analog I/O with asynchronous PDO.
- Use 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.
- Dynamic PDO mapping for DI/AI/DO/AO channels

Before starting this demo, the step0 must be checked. Assume that the default COB-ID for each communication object is being used.

Step 0: The following message must be sent for changing the NMT state of the I-7232D first, because the PDO communication can only 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 (I-7232D)**

cs : 1

Node ID : 1

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

Step 1: In order to change the DO value for the M-7055D to be 0x34 respectively, users must send the PDO message 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	34	00	00	00	00	00	00	00

PDO

PDO → **Consumer**

Producer

(I-7232D)

COB-ID : 0x201

L : 8

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

Only the one byte are useful even the **L** is set to 8, because the data in the1st RxPDO contains only one byte. According to the PDO mapping table shown above, the one byte is the DO0~DO7 channel values of the M-7055D.

Step 2: Owing to the change of the DI-channel status, the TxPDO is transmitted automatically when the transmission type is 255. It is based on the CANopen spec 401. Hence 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	0	1	0	1	34	00	00	00	00	00	00	00

PDO

PDO ← **Producer**

Consumer **(I-7232D)**

COB-ID : 0x181

L : 1

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

The DI value is 1 if the DI is OFF, because of the character of the M-7055D DI channels. Therefore, the one byte indicates that the DI2, DI4, and DI5 of the M-7052D are ON.

Step 3: In order to output 5V to the AO0 of the M-7024, 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	0	1	0	8	FF	1F	00	00	00	00	00	00

PDO **PDO**

Producer **Consumer**

(I-7232D)

COB-ID : 0x301

L : 8

PDO-msg : FF 1F 00 00 00 00 00 00

Only the first two bytes are useful, because the data in the 2nd RxPDO has only two bytes. Users need to transfer the float value to hex format, because the I-7232D only supports the hex format. The output range of the M-7024 is 0V~10V. According to the transformation table stored in the appendix table. The mapping hex-format range is from 0x000 (0) to 0xFFF (4095). Therefore, the 5V is mapped to the 0x7FF by applying following equation.

$$\frac{HexValue}{4095} = \frac{5V - 0V}{0 - 0} \quad 2047.5$$

$$2047 \quad 0x7FF$$

$\overline{10V \quad 0V}$

The first two bytes of the PDO message will be filled with “FF” and “07”. For more details about how to transfer the value between the hex and float, please refer to section 6.3.

Step 4: Even the AI input value will be changed, the RxPDO will not respond automatically in the I-7232D. 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	0	0	1	1	0	00	00	00	00	00	00	00	00

PDO Consumer → **PDO Producer (I-7232D)**

COB-ID : 0x281

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	2	EA	3F	00	00	00	00	00	00

PDO Consumer ← **PDO Producer (I-7232D)**

COB-ID : 0x281

L : 2

PDO-msg : EA 3F 00 00 00 00 00 00

The feedback AI value is 3FEA. Users also need to transfer this value to float. The M-7017R input float range is set to -10V ~ +10V and the input hex range is from 0x8000 (-32768) to

0x7FFF (32767). The value 0x3FF5 (16373) can be transferred By using the following equation.

$$FloatValue$$

$$1637332767 \qquad 3276832768 \qquad 10V$$

$$10V \qquad 10V$$

$$4.997V$$

- **Event Timer Functionality**

Step 6: Use the SDO to change the event timer of the 2nd RxPDO to 1000, which is stored in index 0x1801 with sub-index 5. The value 1000 means 1 second, because the unit in 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	0	0	1	0	8	2B	01	18	05	E8	03	00	00

SDO server

SDO client 

(I-7232D)

ccs : 1
n : 2
e : 1
s : 1
m : 01 18
05
d : E8 03
The value 0x03E8 is equal to 1000.

Step 7: I-7232D 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	0	0	1	0	8	60	00	18	05	00	00	00	00

SDO server

SDO client 

(I-7232D)

scs : 3
m : 00 18 05

Step 8: After changing the value of the event timer, the AI value will be transmitted automatically every 1 second. This is the first 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	0	0	1	0	2	EA	3F	00	00	00	00	00	00

PDO

PDO

Producer

Consumer

(I-7232D)

COB-ID : 0x281

L : 2

PDO-msg : EA 3F 00 00 00 00 00 00

Step 9: This is 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	0	0	1	0	2	F8	3F	00	00	00	00	00	00

PDO

PDO

Consumer

(I-7232D)

COB-ID : 0x281

L : 2

PDO-msg : F8 3F 00 00 00 00 00 00

Producer

The value of 0x3FF8 is equal to 4.998V. The AI value is changed because of the noise disturbance or other factors.

Step 10: This is the third 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	0	0	1	0	2	F8	3F	00	00	00	00	00	00

PDO Consumer ← **PDO Producer (I-7232D)**

COB-ID : 0x281

L : 2

PDO-msg : F8 3F 00 00 00 00 00 00

Step 11: 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	0	0	1	0	8	2B	01	18	05	00	00	00	00

SDO client → **SDO server (I-7232D)**

ccs : 1
n : 2
e : 1
s : 1
m : 01 18 05
d : 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	8	60	01	18	05	00	00	00	00

SDO server ← **SDO client**

(I-7232D)

scs : 3
m : 01 18 05

- **Transmission Type 0 for 1st RxPDO**

Step 12: Set the transmission type of the 1st 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 (I-7232D)**

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	8	60	00	14	02	00	00	00	00

SDO server

SDO client ←

(I-7232D)

scs : 3

m : 00 14 02

Step 13: Change the DO value of the M-7055D to be 0x78 respectively 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	00	00	00	00	00	00	00

PDO

PDO

Producer →

Consumer

(I-7232D)

COB-ID : 0x201

L : 8

PDO-msg : 78 00 00 00 00 00 00 00

Step 14: The DO value will not change immediately, because of the character of the transmission type 0. 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	00	00	00	00	00	00	00	00

SYNC

SYNC

Producer

Consumer

(I-7232D)

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 follows the producer/consumer relationship.

Step 15: After transmitting the SYNC object, the 1st RxPDO is triggered, and the DI value is changed. Hence, users can receive the 1st TxPDO from I-7232D.

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	0	1	0	1	78	00	00	00	00	00	00	00

PDO

PDO

Consumer

Producer

(I-7232D)

COB-ID : 0x181

L : 1

PDO-msg : 78 00 00 00 00 00 00 00

Step 16: Set the transmission type of the 1st 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 (I-7232D)**

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 00 14 02 **d**
: FF

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	60	00	14	02	00	00	00	00

SDO server → **SDO client (I-7232D)**

scs : 3
m : 00 14 02

- **Transmission Type 0 for 1st TxPDO**

Step 17: Set the transmission type of the 1st 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	0	1	0	8	2F	00	18	02	00	00	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 00 18 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	8	60	00	18	02	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3
m : 00 18 02

Step 18: Change the DO value of the M-7055D to be 0X78 respectively 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	EA	00	00	00	00	00	00	00

PDO **PDO**

Producer **Consumer**

(I-7232D)

COB-ID : 0x201

L : 8

PDO-msg : EA 00 00 00 00 00 00 00

Step 19: The 1st TxPDO will not be transmitted immediately even if the DI value is changed, because of the character of the transmission type 0. The SYNC message is needed to trigger the action of the 1st 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	00	00	00	00	00	00	00	00

SYNC **SYNC**

Producer **Consumer**

(I-7232D)

COB-ID : 0x80

Step 20: After transmitting the SYNC object, the 1st TxPDO is triggered, and users can receive the 1st TxPDO from I-7232D.

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	0	1	0	1	78	00	00	00	00	00	00	00

PDO

PDO

Producer

(I-7232D)

Consumer

COB-ID : 0x181

L : 1

PDO-msg : 78 00 00 00 00 00 00 00

Step 21: 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	00	00	00	00	00	00	00	00

SYNC

SYNC

Consumer

(I-7232D)

Producer

SYNC : 0x80

COB-ID

Step 22: Nothing has happened because the DI values don't change. This is the main difference between transmission type 0 and 1. At transmission type 1, the TxPDO is always transmitted no matter whether the DI values are changed or not, when the I-7232D receives the SYNC object.

- **Transmission Type 3 for 1st TxPDO**

Step 23: Set the transmission type of the 1st 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	0	1	0	8	2F	00	18	02	03	00	00	00

SDO client → **SDO server (I-7232D)**

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 00 18 02 **d**
: 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	0	1	1	0	0	0	0	0	0	1	0	8	60	00	18	02	00	00	00	00

SDO client ← **SDO server (I-7232D)**

scs : 3

m : 00 18 02

Step 24: Change the DO value of the M-7055D to be 0xEF respectively 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	EF	00	00	00	00	00	00	00

PDO

PDO Producer → **Consumer (I-7232D)**

COB-ID : 0x201

L : 8

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

Step 25: The SYNC message needs to be transmitted 3 times because of 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	00	00	00	00	00	00	00	00

SYNC

SYNC Consumer →

Producer (I-7232D) →

COB-ID : 0x80

Step 26: After finishing the transmission of the three SYNC objects, the 1st TxPDO is triggered, and users can receive the 1st TxPDO from I-7232D.

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	0	1	0	1	EF	00	00	00	00	00	00	00

PDO **PDO**

Consumer **Producer**

(I-7232D)

COB-ID : 0x181

L : 1

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

- **Transmission Type 252 for 1st TxPDO**

Step 27: Set the transmission type of 1st 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	0	1	0	8	2F	00	18	02	FC	00	00	00

SDO client **SDO server**

(I-7232D)

ccs : 1 **n** : 3 **e** :

1 **s** : 1

m : 00 18 02 **d**

: FC

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	60	00	18	02	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3

m : 00 18 02

Step 28: Change the DO value of the M-7055D to be 0x34 respectively 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	34	00	00	00	00	00	00	00

PDO

PDO

Producer ⟶

Consumer

(I-7232D)

COB-ID : 0x201

L : 8

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

Step 29: The 1st TxPDO will not be transmitted immediately, because of transmission type 252. Send the RTR message of the 1st 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	0	1	1	0	00	00	00	00	00	00	00	00

PDO

PDO ⟶

Producer

Consumer

(I-7232D)

COB-ID : 0x181

Step 30: The feedback DI values are the old one. (If users use the M-7055D, the LEDs on the M-7055D can indicate 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	1	0	1	0	0	0	0	0	0	1	0	2	34	00	00	00	00	00	00	00

PDO

PDO

Producer

Consumer

(I-7232D)

COB-ID : 0x181

L : 2

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

SYNC

SYNC

Consumer

Producer

(I-7232D)

COB-ID : 0x80

Step 32: Send the RTR message of the 1st 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	0	1	1	0	00	00	00	00	00	00	00	00

PDO Consumer ⟶ **PDO Producer (I-7232D)**

COB-ID : 0x181

Step 33: The feedback DI values is 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	1	0	1	0	0	0	0	0	0	1	0	1	34	00	00	00	00	00	00	00

PDO Consumer ⟵ **PDO Producer (I-7232D)**

COB-ID : 0x181

L : 2

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

- **Transmission Type 253 for 1st TxPDO**

Step 34: Set the transmission type of the 1st 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	0	1	0	8	2F	01	18	02	FD	00	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 00 18 02 **d**
: FD

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	60	00	18	02	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3
m : 00 18 02

Step 35: Change the DO value of the M-7055D to be 0x78 respectively 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	00	00	00	00	00	00	00

PDO

PDO

Producer ⟶ **Consumer**

(I-7232D)

COB-ID : 0x201

L : 8

PDO-msg : 78 00 00 00 00 00 00 00

Step 36: Because of the transmission type 253, the 1st TxPDO can only be transmitted when receiving the RTR message. So, send RTR message to get the 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	0	1	1	0	00	00	00	00	00	00	00	00

PDO

PDO

Producer

Consumer

(I-7232D)

COB-ID : 0x181

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	0	1	1	0	78	00	00	00	00	00	00	00

PDO

PDO

Producer

Consumer

(I-7232D)

COB-ID : 0x181

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	0	1	0	8	2F	00	18	02	FF	00	00	00

SDO server

SDO client
(I-7232D)

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 00 18 02 **d**
: FF

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	60	00	18	02	00	00	00	00

SDO server

SDO client

(I-7232D)

scs : 3
m : 00 18 02

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

Step 38: Use the 5th TxPDO to create a new PDO communication with PDO COB-ID 0x182, you do this because the COB-ID 0x182 is useless for the I-7232D, Before setting the COB-ID of a PDO, check bit 31 of the COB-ID first. Only the COB-ID, which has the value 0 on its bit 31, can be changed. Therefore, if users want to configure the COB-ID of a valid PDO communication (bit 31 is 1), set this PDO to an invalid state (bit 31 is 0). The COB-ID can be configured directly, because 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	0	0	1	0	8	23	05	18	01	82	01	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 0 **e** :
1 **s** : 1

m : 05 18 01

d : 82 01 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	8	60	05	18	01	00	00	00	00

SDO server

SDO client ◄——————————

(I-7232D)

scs : 3

m : 05 18 01

Step 39: Build a new PDO mapping object for the 5th TxPDO. Before starting to fill in the device objects into the index 0x1A05, 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 write the value to 0. Its value is 0 and can be configured directly, because the 0x1A05 has not been used before.

First, fill the DI0~DI7 of the M-7055D into the index 0x1A05 with sub-index 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	0	1	0	8	23	05	1A	01	08	01	00	60

SDO server

SDO client ——————————►

(I-7232D)

ccs : 1
n : 0
e : 1

s : 1

m : 05 1A 01

d : 08 01 00
60

The value “60 00 01 08” means the mapped object is stored in the index 0x6000 with sub-index 01. It is an 8-bit data unit. Users can check this object in the Standardize of object mapping table described above. It is mapped according to the DI0~DI7 of the M-7055D.

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	60	05	1A	01	00	00	00	00

SDO server

SDO client ◄——————————

(I-7232D)

scs : 3

m : 05 1A 01

Step 40: Fill the AI0 of the M-7017R into the index 0x1A05 with sub-index 03 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	0	0	1	0	8	23	05	1A	03	10	01	01	64

SDO server

SDO client ——————————►

(I-7232D)

ccs : 1

n : 0
e : 1
s : 1
m : 05 1A 03
d : 10 01 01
64
The value “64 01 01 10” means that the mapped object is stored in the index 0x6401 with sub-index 01. It is a 16-bit data unit. User can check this object in the Standardize of object mapping table described above. It is mapped according to AI0 of the M-7017R.

In I-7232D, 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	0	0	1	0	8	60	05	1A	03	00	00	00	00

SDO server

SDO client ◄

(I-7232D)

scs : 3
m : 05 1A 03

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 1. The value 1 means there are 1 objects mapped to the 5th TxPDO. They are the index 0x6000 with sub-index 01, and index 0x6401 with sub-index 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	0	1	0	8	2F	05	1A	00	01	00	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 05 1A 00 **d**
: 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	60	05	1A	00	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3
m : 05 1A 00

Step 42: Use the 5th RxPDO to create a new PDO communication with PDO COB-ID 0x202, and build the RxPDO mapping object in the index 0x1605, because the COB-ID 0x202 is useless for the I-7232D. This procedure is similar to the steps 37 to 40.

11-bit COB-ID (bit)	RTR		8-byte Data (byte)

Func Code				Node ID								Data Length								
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	23	05	14	01	02	02	00	00

SDO client ⟶ **SDO server (I-7232D)**

ccs : 1
n : 0
e : 1
s : 1
m : 05 14 01
d : 02 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	0	1	0	8	60	05	14	01	00	00	00	00

SDO client ⟵ **SDO server (I-7232D)**

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	0	1	0	8	23	05	16	01	08	01	00	62

SDO server

SDO client ⟶

(I-7232D)

ccs : 1
n : 0
e : 1
s : 1
m : 05 16 01
d : 08 01 00
62

The value “62 00 01 08” means the mapped object is stored in the index 0x6200 with sub-index 01. It is an 8-bit data unit. Users can check this object in the Standardize of object mapping table described above. It is mapped to the DO0~DO7 for M-7055D.

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	60	05	16	01	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3
m : 05 16 01

11-bit COB-ID (bit)	RTR		8-byte Data (byte)

Func Code				Node ID								Data Length								
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	23	05	16	03	10	01	11	64

SDO server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 0 **e** :
1 **s** : 1

m : 05 16 03

d : 10 01 01 64

The value “64 11 01 10” means the mapped object is stored in the index 0x6401 with sub-index 01. It is a 16-bit data unit. Users can check this object in the Standardize of object mapping table described above. It is mapped to the AO0 of the M-7024.

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	60	05	16	03	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3

m : 05 16 03

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	05	16	00	01	00	00	00

SDO server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 3
e : 1 **s** : 1
m : 05 16 00
d : 03

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	60	05	16	00	00	00	00	00

SDO server

SDO client ←

(I-7232D)

scs : 3
m : 05 16 00

Step 43: Transmit the DO0~DO7 of M-7055D and AO0 of M-7024 to be 0x54 and 0V 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	0	0	0	0	1	0	0	8	54	00	00	00	00	00	00	00

PDO

PDO

Consumer

(I-7232D)

COB-ID : 0x202
PDO-msg : 54 00 00 00

→ **Producer**

The first two bytes are the value 0xAB for the DO0~DO7 of the M-7055D. The last two bytes are the value 0x0000 for the AO0 of the M-7024. Total bytes of this PDO message are 4.

Step 44: Users will receive the 1st TxPDO and 5st TxPDO simultaneously, because the DI value has changed.

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	0	1	0	3	54	00	01	00	00	00	00	00

PDO

PDO

Producer

Consumer

(I-7232D)

COB-ID : 0x181

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	4	54	00	06	00	00	00	00	00

PDO

PDO

Consumer

(I-7232D)

COB-ID : 0x182

PDO message : 54 00 06 00

The first two bytes are for the value 0x54 for the DI0~DI7 of the

Producer

M-7055D. The last two bytes are for the value 0x0006 for the AI0 of the M-7024. After transferring, the input value of the AI0 is 0.002V.

5.3 EMCY Communication Set

5.3.1 EMCY COB-ID Parameter

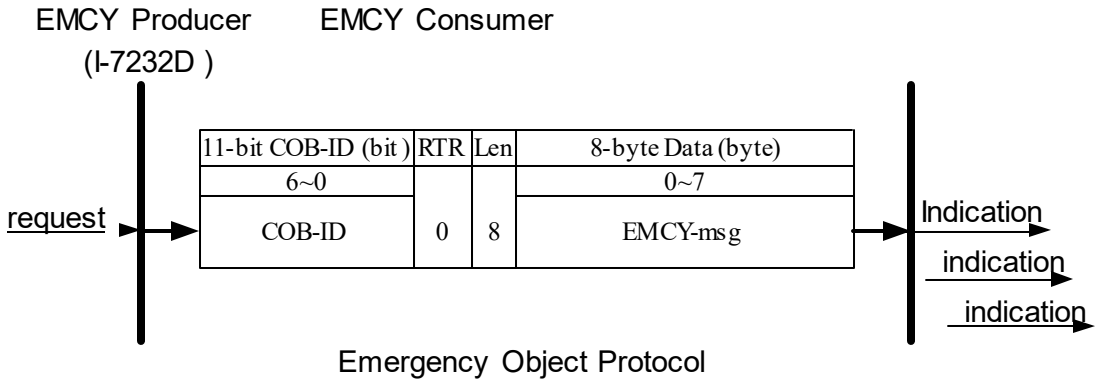
The EMCY COB-ID is similar to the PDO COB-ID. It can be a default value or be defined by users via SDO communication methods. This COB-ID is stored in the object 0x1014, and the data format is shown in the following table. Before using the EMCY mechanism, bit 31 of the EMCY COB-ID needs to be confirmed.

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 check this object to understand the history of the error's occurrences. The I-7232D supports a max of 5 records stored in the different sub-indexes of the index 0x1003 object. Sub-index 1 of

this object stores the last EMCY event, and sub-index 5 records the oldest EMCY event. The EMCY communication set is given below.



Emergency Object Protocol

COB-ID : The EMCY COB-ID

User can define the EMCY COB-ID. 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 class of the occurrence error

The data format of the emergency object data follows the structure shown bellow.

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. The I-7232D only 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
50	00	81	03	00	00 00 00	COM Port 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
FF	00	81	0B	??	00 00 00	The module with RS-485 address ?? is timeout or receive data error more than three times.

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 has happened in the I-7232D and check the error history.

EMCY Communication Example

Assume that there is a Modbus RTU module connected with COM2 of the I-7232D. This module has the module address 01, and has one digital or analog input channel at least. The node ID of the I-7232D is 5, and the I-7232D works normally with the default COB-ID.

Step 1. In order to produce the emergency event, send the data to RxPDO1 with data length 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
0	1	0	0	0	0	0	0	0	0	1	0	0	00	00	00	00	00	00	00	00

PDO → **PDO Consumer** / **PDO Producer (I-7232D)**

COB-ID : 0x201

L : 0

m 00 00 00

Step 2. Afterwards, the I-7232D will respond to an emergency message because the PDO data length of TxPDO1 doesn't match the practical 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 → **EMCY Consumer** / **EMCY (I-7232D)**

COB-ID : 0x81

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

Producer

The first two bytes “10 82” are emergency error codes. The 3rd byte “11” is the error register. It means that the I-7232D has either a manufacturer specific or generic error. The last five bytes “09 00 00 00 00” are the manufacturer specific error fields. This emergency message means that the data length of TxPDO doesn’t match the practical value defined in the PDO mapping object.

Step 3. Read the 0x1003 object with sub-index 01, users will then be able to see the emergency error code 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 server

SDO client **(I-7232D)**

ccs : 2

m 03 10 01

Step 4. I-7232D responds 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 server

SDO client

(I-7232D)

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

Step 5. Check the object 0x1001, and make sure that the manufacturer specific 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 server

SDO client ⟶

(I-7232D)

ccs : 2
m 01 10 00

Step 6. The manufacturer specific 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	8	4F	01	10	00	11	82	09	00

SDO server

SDO client ⟵

(I-7232D)

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

d : 11 82 09
00

Step 7. Send the data to RxPDO1 with data length 1. Afterwards, the EMCY message containing the error-reset information 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
0	1	0	0	0	0	0	0	0	0	1	0	1	00	00	00	00	00	00	00	00

PDO Consumer ⟶ **PDO Producer (I-7232D)**

COB-ID : 0x201

L : 1

m 00 00 00 00 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
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 (I-7232D)**

EMCY-msg : 00 00 00 00 00 00 00 00

(Note: The data “00 00 00 00 00 00 00 00” are the error reset EMCY message. It means that I-7232D has no error now.)

Step 8. Check the index 0x1003 with sub-index 01 again. The user will then see that the error reset emergency code has been recorded.

11-bit COB-ID (bit)	RTR		8-byte Data (byte)

Func Code				Node ID								Data Length								
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 server

SDO client ⟶

(I-7232D)

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	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

ccs : 1 **n** : 2 **e** :
1 **s** : 1

m : 0310 01

d : 00 00 00 00

Step 9. Check the index 0x1003 with sub-index 02, and the user will see that the emergency error code that is received previously has also 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 server

SDO client ⟶

(I-7232D)

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 server

SDO client ←

(I-7232D)

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

Step 10. Confirm the error register stored in index 0x1001. The value is 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 server

SDO client →

(I-7232D)

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	00	00	00

SDO server

SDO client 

(I-7232D)

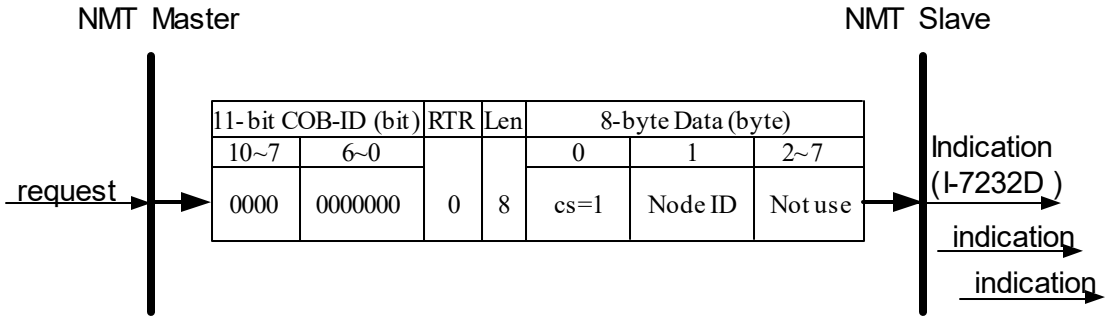
ccs : 1
n 2
e 1
s 1
m : 01 10 00
d : 00 00 00 00

5.4 NMT Communication Set

5.4.1 Module Control Protocol

The NMT communication set can be applied for changing the NMT status of the NMT slave. The following figure shows how to change the different NMT statuses for the I-7232D.

Start Remote Node Protocol



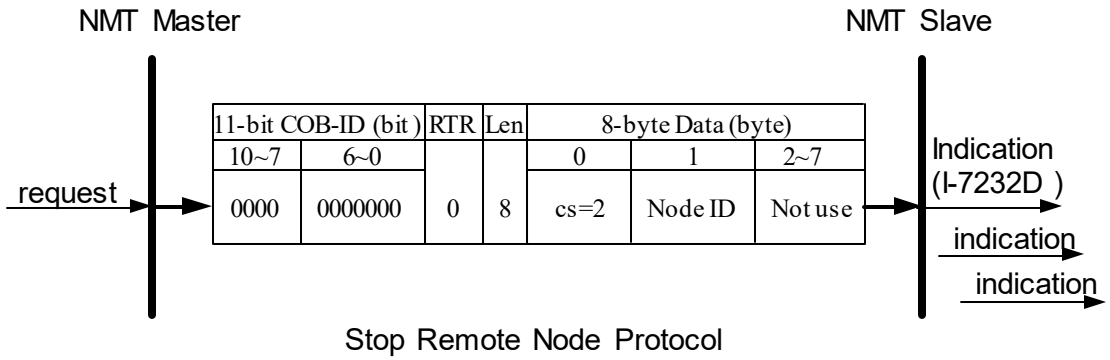
Start Remote Node Protocol

cs : NMT command specifier

1: start

Node ID : The node ID of the NMT slave device

Stop Remote Node Protocol



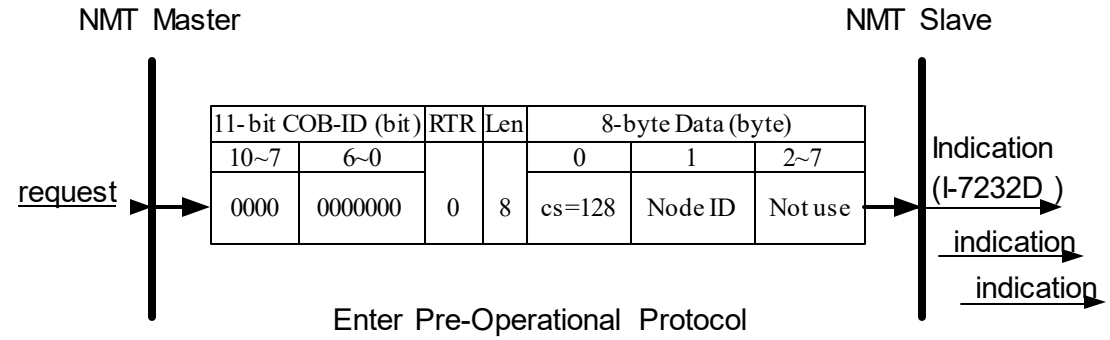
Stop Remote Node Protocol

cs : NMT command specifier

2: stop

Node ID : The node ID of the NMT slave device

Enter Pre-Operational Protocol

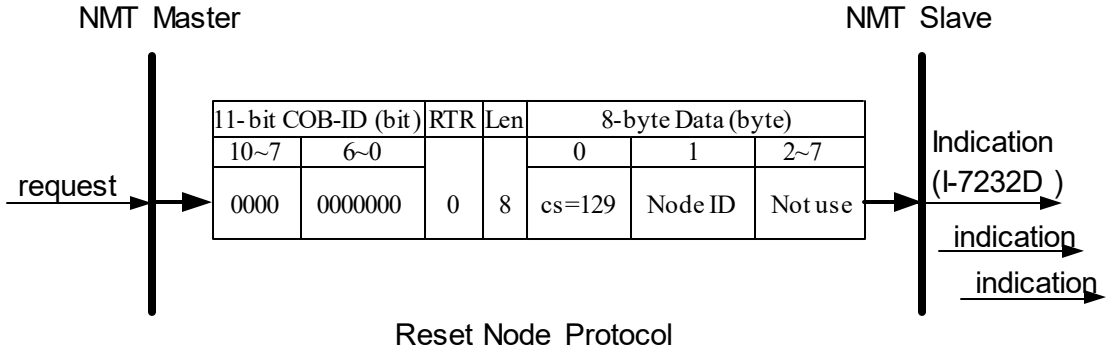


Enter Pre-Operational Protocol

cs : NMT command specifier

128: enter PRE-OPERATIONAL

Node ID : The node ID of the NMT slave device ***Reset Node Protocol***



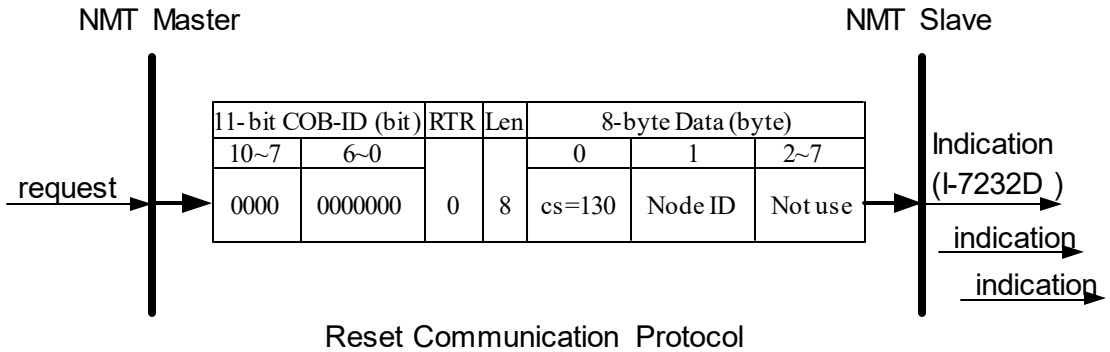
Reset Node Protocol

cs : NMT command specifier

129: Reset_Node

Node ID : The node ID of the NMT slave device

Reset Communication Protocol



Reset Communication Protocol

cs : NMT command specifier

130: Reset_Communication

Node ID : The node ID of the NMT slave device ***Module Control Protocol Example***

Assume that the I-7232D node ID is 1.

Step1. Turn off the I-7232D.

Step2. Turn on the I-7232D. After finishing the initialization, the I-7232D will enter the Pre_Operational state automatically. Then the user will see the CAN LED flashing about twice per second.

Step3. Send the NMT module control protocol to command the I-7232D to enter its 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

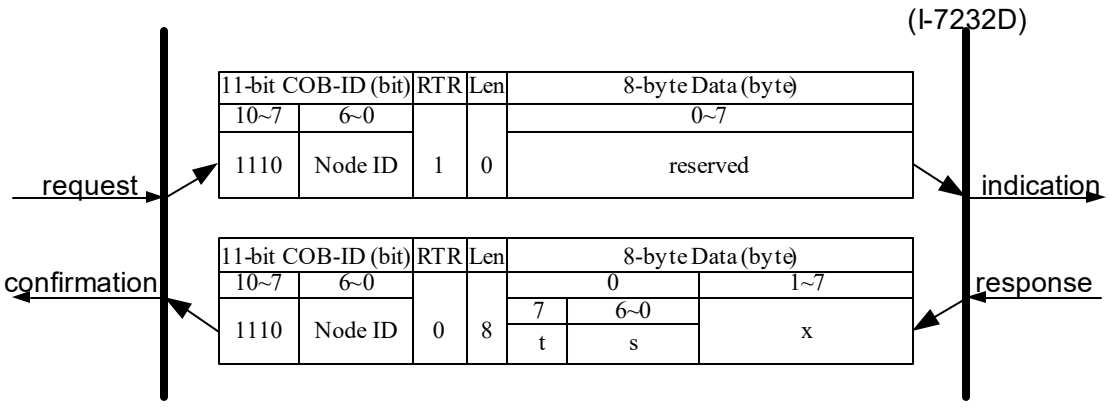
NMT master → **NMT slave (I-7232D)**

cs : 1
Node ID : 1

5.4.2 **Error Control Protocol**

Error Control Protocol is one of the ways to check if the CANopen device still lives. Related objects are indexes 0x100C and 0x100D. 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 I-7232D starts to count after receiving the first remote-transmit-request for the guarding identifier. The communication set of the Error Control protocol is displayed below.

NMT Master NMT Slaver



Node Guarding Protocol

t : Toggle bit

The value of this bit must alternate 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 is 0.

s : The state of the NMT Slave

4: STOPPED

5: OPERATIONAL

127: PRE-OPERATIONAL

Error Control Protocol Example

Assume that the default EMCY function code has been applied, and the node ID for the I-7232D is 1.

Step 1. Turn off the I-7232D. Then, turn on the I-7232D. The I-7232D will now be in the Pre_Operational state.

Step 2. Set the guard time value to 250. This value is 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 (I-7232D)**

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 0C 10 00

d : FA 00

Step 3. I-7232D will respond 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	60	0C	10	00	00	00	00	00

SDO client ⟵ **SDO server (I-7232D)**

scs : 3

m : 0C 10 00

Step 4. Set the life time factor value to 4. This value is stored in the index 0x100D with sub-index 00. Then, receive the ending message from I-7232D

11-bit COB-ID (bit)	RTR		8-byte Data (byte)

Func Code				Node ID								Data Length								
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 server

SDO client ⟶

(I-7232D)

ccs : 1 **n** : 3 **e** :
1 **s** : 1

m : 0D 10 00

d : 04

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	60	0D	10	00	00	00	00	00

SDO server

SDO client ⟵

(I-7232D)

scs : 3

m : 0D 10 00

Step 5. 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	00	00	00	00	00	00	00	00

NMT slaver

NMT master ⟶

(I-7232D)

COB-ID : 0x701

Step 5. Afterwards, users can receive the message, which records the NMT state of the I-7232D. For the reason that Life Time is equal to 1000 ms (guard time * life time factor =250*4=1000), users will need to 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	8	7F	00	00	00	00	00	00	00

NMT slaver

NMT master ⟵

(I-7232D)

COB-ID : 0x701 **t**

: 1

s 7F

The value 7F means that the I-7232D is in the NMT state Pre-Operational.

Step 6. Since Life Time is equal to 1000 ms (guard time * life time factor =250*4=1000), users will need to transmit the node guarding protocol again. If not, an error event will be triggered, and an EMCY message will be received. All values from the output channels will be changed according to index 0x6206, index 0x6207, index 0x6443, and index 0x6444.

Step 7. Afterwards, if reading the input value of this Modbus RTU module fails more than three times, the I-7232D will then respond with an emergency 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	1	0	1	0	8	30	81	11	07	00	00	00	00

EMCY producer

EMCY consumer

(I-7232D)

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 the manufacturer specific error fields.

This emergency message indicates a life-guard error.

5.5 LSS Communication Set

5.5.1 Switch mode protocols

5.5.1.1 Switch mode global

This protocol is used to implement the ‘**Switch Mode Global**’ service.

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	1	1	1	0	0	1	0	1	0	8	cs	mode	00	00	00	00	00	00

LSS

LSS Slave

Master

(I-7232D)

COB-ID 0x7E5

LSS-msg **cs**: LSS command specifier. cs = 04(0x04) for Switch Mode Global mode: The LSS mode to switch to:

0: switch to operation mode.

1: switch to configuration mode.

5.5.1.2 **Switch mode selective**

This protocol is used to implement the '**Switch Mode Selective**' service

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	Vendor-id				00	00	00

LSS

LSS Slave → **Master**

(I-7232D)

COB-ID 0x7E5

LSS-msg **cs**: LSS command specifier. cs = 64(0x40)

Vendor-id: It is one part of the LSS address, which is recorded in index 1018h, subindex 1.

Step 2:

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	1	1	1	0	0	1	0	1	0	8	cs	Product-code				00	00	00

LSS

LSS Slave

Master

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 65(0x41)

product-code: It is one part of the LSS address, which is recorded in index 1018h, subindex 2.

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	Revision-number				00	00	00

LSS

Slave

Master

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 66(0x42)

→ revision-number:
It is one part of the LSS address, which is recorded in index 1018h, subindex 3.

Step 4:

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	1	1	1	0	0	1	0	1	0	8	cs	Serial-number				00	00	00

LSS

LSS Slave

Master →

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifiers. cs = 67(0x43)
serial-number: It is one part of the LSS address, which is recorded in index 1018h, subindex 4.

Step 5:

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	1	1	1	0	0	1	0	0	0	8	cs	00	00	00	00	00	00	00

LSS

LSS Slave

Master ←

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 68(0x44)

5.5.2 **Configuration protocols**

5.5.2.1 **Configuration Node-ID protocol**

This protocol is used to implement the 'Configuration Node-ID' service for the Node-ID of I-7232D.

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	ID	00	00	00	00	00	00

LSS

LSS Slave → **Master**

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 17(0x11)
ID: Node-ID

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Err1	Err2	00	00	00	00	00

LSS

Master → **LSS**

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 17(0x11)

Slave

Err1: error code.

Err1 = 0: protocol successfully completed Err1 = 1: configuration Node-ID fail.

Err2: spec. error (reserved)

5.5.2.2 **Configuration bit timing parameters protocol**

This protocol is used to implement the '**Configuration Bit Timing Parameters**' service. The following table will be applied when users want to use this protocol.

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

Figure 5_1 Table_Index

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	Tab1	Tab2	00	00	00	00	00

LSS

LSS

Slave

Master **(I-7232D)**

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 19(0x13)

Tab1: Table_selector

Tab1 = 0: Standard CiA bit timing table.

Tab2 = 1~255: reserved.

Tab2: Table_Index. See Figure 5_1.

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Err1	Err2	00	00	00	00	00

LSS

LSS

Master **(I-7232D)**

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 19(0x13)

Slave

Err1: error code.

Err1 = 0: protocol successfully completed Err1 = 1: configuration bit timing fail.

Err2: spec error (reserved)

5.5.2.3 **Activate bit timing parameters protocol**

This protocol is used to implement the **'Activate Bit Timing Parameters'** service.

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	1	1	1	0	0	1	0	1	0	8	cs	switch_delay		00	00	00	00	00

LSS Master → **LSS Slave (I-7232D)**

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 21(0x15)

switch_delay: The duration of the two periods of time to wait after performing the switch. The first period is for waiting the finish of the bit timing parameters switch. Afterwards, I-7232D will reboot automatically. The second period is the time before transmitting any CAN message with the new bit timing parameters.

The time unit of switch delay is 1 ms.

5.5.2.4 **Store configuration protocol**

The protocol is used to implement the **'Store Configuration Parameters'** service. The protocol is store the new Node-ID and the new Bit Timing parameters. Therefore, if users do not use this protocol, the new Node-ID and baud will not be saved by I-7232D.

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	1	1	1	0	0	1	0	1	0	8	cs	00	00	00	00	00	00	00

LSS Master → **LSS Slave (I-7232D)**

COB-ID → 0x7E5

LSS-msg cs: LSS command specifier. cs =23(0x17)

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Err1	Err2	00	00	00	00	00

LSS

LSS

Master

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 23(0x17)

← **Slave**

Err1: error code.

Err1 = 0: protocol successfully completed

Err1 = 1: store configuration is not supported. Err2: spec. error (reserved)

5.5.2.5 **Restrictions on configuration Node-ID or bit Timing**

When users used the ‘configuration bit timing parameters protocol’ or ‘configuration Node-ID protocol‘, users have to send the ‘store configuration protocol’ to save the configuration parameters and send the ‘Activate bit timing parameters protocol’ to set the reboot time of I-7232D. If users do not send the ‘store configuration protocol’ and just send the ‘Activate bit timing parameters protocol’ only , it will not change the Node-ID or bit timing until the I-7232D reboots, but the configuration ‘Node-ID’ or ‘Bit timing’ is changed temporarily. Unless users send the ‘store configuration protocol’, the configuration ‘Node-ID’ or ‘Bit timing’ will be ineffective after the I-7232D reboots again.

5.5.3 **Inquire protocols**

These protocols are used to implement the **'Inquire LSS Address'** service. To implement the service, each of the following three protocols has to be executed.

5.5.3.1 **Inquire Identify Vendor-ID protocol**

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	00	00	00	00	00	00	00

LSS

LSS
Slave
Master ⟶

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs =90(0x5A)

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Vendor-id				00	00	00

LSS

LSS

Slave

Master ⟵

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 90(0x5A)

vendor-id → It is one part of the LSS address, which is recorded in index 1018h, subindex 1.

5.5.3.2 **Inquire identify product-code protocol**

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	00	00	00	00	00	00	00

LSS

LSS Slave

Master

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 91(0x5B)

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Product-code				00	00	00

LSS

LSS Slave

Master

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 91(0x5B)

product-code: It is one part of the LSS address, which is recorded in index 1018h, subindex 2.

5.5.3.3 **Inquire Identify revision-number protocol**

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	00	00	00	00	00	00	00

LSS

LSS Slave

Master ⟶

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs =92(0x 5C)

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Revision-number	00	00	00

LSS

LSS

Master

Slave

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 92(0x5C)

revision-number: It is one part of the LSS address, which is recorded in index 1018h, subindex 3.

5.5.3.4 **Inquire identity serial-number protocol**

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	00	00	00	00	00	00	00

LSS

LSS

Slave

Master 

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 93(0x5D)

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	Serial-number				00	00	00

LSS

Slave

LSS

Master 

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 93(0x5D)

serial-number: It is one part of the LSS address, which is recorded in index 1018h, subindex 4.

5.5.3.5 Inquire Node-ID protocol

The protocol is used to implement the **'Inquire Node-ID'** service.

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	00	00	00	00	00	00	00

LSS

LSS Slave

Master ⟶

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 94(0x5E)

Step 2:

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	1	1	1	0	0	1	0	0	0	8	cs	NID	00	00	00	00	00	00

LSS

LSS

Master

(I-7232D)

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 94(0x5E)

⟵ **Slave**

NID: The Node-ID of the selected module. If the Node-ID has been changed by means of previous Configure Node-ID service the original Node-ID is returned until the next power on reset.

5.5.4 **Identification protocol**

5.5.4.1 **LSS identify remote slaves**

This protocol is used to implement the **'LSS Identify Remote Slaves'** service.

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	Vendor-id				00	00	00

LSS

LSS Slave

Master

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 70(0x46)
Vender-id: The manufacturer name part of the LSS Address.

Step 2:

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	1	1	1	0	0	1	0	1	0	8	cs	Product-code				00	00	00

LSS

LSS

Slave
Master 

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 71(0x47)

Product-code: The product name part of the LSS Address.

Step 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	1	1	1	1	0	0	1	0	1	0	8	cs	Revision-number				00	00	00

LSS

LSS
Slave
Master 

(I-7232D)

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 72(0x48)

revision-number: The Revision name part of the LSS Address.

Step 4:

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	1	1	1	0	0	1	0	1	0	8	cs	Serial-number				00	00	00

LSS

LSS

Slave

Master

(I-7232D) ⟶

COB-ID 0x7E5

LSS-msg cs: LSS command specifier. cs = 74(0x4A)

Serial–number: The Serial number part of the LSS Address.

(Note: ALL LSS Slaves with matching vendor-id and product-code whose major revision-number and serial-numbers lie within the given ranges, are requested to identify themselves with the LSS Identify Slave service described in section 5.5.4.2.)

5.5.4.2 **LSS identify slave protocol**

This protocol is used to implement the **'LSS Identify Slave'** service.

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	1	1	1	0	0	1	0	0	0	8	cs	00	00	00	00	00	00	00

LSS Master ⟵ **LSS Slave (I-7232D)**

COB-ID 0x7E4

LSS-msg cs: LSS command specifier. cs = 79(0x4F)

5.6 Special Functions for Modbus RTU modules

I/O Modules R/W Error Control Entry

The I-7232D Manufacturer Specific Profile Area defines some special functions for Modbus RTU modules. The object with index 0x2000 is the I/O modules read/write error control entry. Each sub-index of this object is mapped to the corresponding Modbus RTU module except sub-index 00.

For example, there are 4 Modbus RTU modules connected with the I-7232D, and the module address for each Modbus RTU module is 1, 3, 4, and 5 respectively. Sub-index 1 is mapped to the Modbus RTU module with address 1. Sub-index 2 is mapped to the Modbus RTU module with address 3 Sub-index 3 and sub-index 5 are mapped to the Modbus RTU module with address 4 and 5 respectively, and so the sub-index 0 will be 4. This means that there are 4 Modbus RTU modules connected with the I-7232D. If accessing the I/O channels on the Modbus RTU modules fails, the value of the corresponding sub-index will count times of the errors which occur according to ether a read/write timeout or read/write failure. The counting range is from 0 to 65535, and the counting value may be returned to 0 after 65536. It can be cleared to 0 via the SDO communication method.

6 Object Dictionary of I-7232D

6.1 Communication Profile Area

The following information lists each entry into the communication profile area defined in I-7232D. In order to look these up conveniently, 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". In the table header you can see "Idx", "Sidx" and "Attr" which represent "index", "sub-index", and "attribute" respectively. The sign "---" in the default field means that the default is not defined or may be defined conditionally by the firmware built in I-7232D. 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	FEh
	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	I-8x21
1009h	0h	Manufacturer hardware version	VISIBLE_STRING	RO	---
100Ah	0h	Manufacturer software version	VISIBLE_STRING	RO	---
100Ch	0h	Guard time	UNSIGNED 16	RW	0
100Dh	0h	Life time factor	UNSIGNED 8	RW	0
1014h	0h	COB-ID of EMCY	UNSIGNED 32	RW	80 h+Node-ID
1015h	0h	Inhibit time of EMCY	UNSIGNED 16	RW	0
1018h	0h	Largest sub-index supported for "identity object"	UNSIGNED 8	RO	1

	1h	Vender ID	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 I-7232D, the specific functionality is always 0. The I/O functionality defines what kind of device the I-7232D is. Bit 16, 17, 18, 19 present the DI, DO, AI, AO respectively. For example, if bit 16 is 1, it means that the I-7232D has DI channels. If both bit 16 and 17 are 1, the I-7232D has both DI and DO channels. Bit 23 ~ bit 19 is always 0. The general information is 0x191 (0x191=401), it means that the I-7232D follows the CANopen spec DS401.

2. About the object with index 0x1001 and 0x1003, please refer to section 5.3.2.
3. The object with index 0x1005 stores the SYNC COB-ID. In the I-7232D, this 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 I-7232D doesn't support the SYNC generation, therefore 29-bit ID, bit 30 and bit 31 are always 0.

4. The object with index 0x1008, 0x1009 and 0x100A record the I-7232D product information. When interpreting these objects, the ASCII table may be needed.
5. The range of the 0x100c is 0~32767 in I-7232D. For more information of the object with index 0x100C and 0x100D, please refer to section 5.3.2.
6. For the object with index 0x1014, please refer to section 5.3.1.
7. The object with index 0x1015 store the inhibit time period between two EMCY message. The function of this object is similar to the PDO communication object with sub-index 04. It is useful for avoiding the large loading on the CAN bus because of transmitting a lot of EMCY messages. This parameter range is 0~32767 for the I-7232D, and the unit of EMCY inhibit time is ms.

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	5
	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	5
	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	5
	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	5
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	80000000h
	2h	Transmission type	UNSIGNED 8	RW	FFh
…	…	…	…	…	…
141Fh	0h	Largest sub-index supported for “receive PDO parameter”	UNSIGNED 8	RO	5
	1h	COB-ID used by PDO (Rx)	UNSIGNED 32	RW	8000 0000h
	2h	Transmission type	UNSIGNED 8	RW	FFh

RxPDO Mapping Communication Entries

Idx	Sidx	Description	Type	Attr	Default
1600h	0h	Largest sub-index supported for "receive PDO mapping"	UNSIGNED 8	RO	8
	1h	Write digital output 1h to 8h	UNSIGNED 8	RW	6200 0108h
	2h	Write digital output 9h to 10h	UNSIGNED 8	RW	6200 0208h
	3h	Write digital output 11h to 18h	UNSIGNED 8	RW	6200 0308h
	4h	Write digital output 19h to 20h	UNSIGNED 8	RW	6200 0408h
	5h	Write digital output 11h to 28h	UNSIGNED 8	RW	6200 0508h
	6h	Write digital output 19h to 30h	UNSIGNED 8	RW	6200 0608h
	7h	Write digital output 11h to 40h	UNSIGNED 8	RW	6200 0708h
	8h	Write digital output 19h to 48h	UNSIGNED 8	RW	6200 0808h
1601h	0h	Largest sub-index supported for "receive PDO mapping"	UNSIGNED 8	RO	4
	1h	Write analog output 1h	UNSIGNED 16	RW	6411 0110h
	2h	Write analog output 2h	UNSIGNED 16	RW	6411 0210h
	3h	Write analog output 3h	UNSIGNED 16	RW	6411 0310h
	4h	Write analog output 4h	UNSIGNED 16	RW	6411 0410h
1602h	0h	Largest sub-index supported for "receive PDO mapping"	UNSIGNED 8	RO	4
	1h	Write analog output 5h	UNSIGNED 16	RW	6411 0510h
	2h	Write analog output 6h	UNSIGNED 16	RW	6411 0610h
	3h	Write analog output 7h	UNSIGNED 16	RW	6411 0710h
	4h	Write analog output 8h	UNSIGNED 16	RW	6411 0810h

1603h	0h	Largest sub-index supported for “receive PDO mapping”	UNSIGNED 8	RO	4
	1h	Write analog output 9h	UNSIGNED 16	RW	6411 0910h
	2h	Write analog output Ah	UNSIGNED 16	RW	6411 0A10h
	3h	Write analog output Bh	UNSIGNED 16	RW	6411 0B10h
	4h	Write analog output Ch	UNSIGNED 16	RW	6411 0C10h
1604h	0h	Largest sub-index supported for “receive PDO mapping”	UNSIGNED 8	RO	---
	1h	---	---	RW	---
	…	…	…	…	…
	---	---	---	RW	---
…	…	…	…	…	…
161Fh	0h	Largest sub-index supported for “receive PDO mapping”	UNSIGNED 8	RO	---
	1h	---	---	RW	---
	…	…	…	…	…
	---	---	---	RW	---

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
…	…	…	…	…	…
181Fh	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

TxPDO Mapping Communication Entries

Idx	Sidx	Description	Type	Attr	Default
1A00h	0h	Largest sub-index supported for “transmit PDO mapping”	UNSIGNED 8	RO	8
	1h	Read digital input 1h to 8h	UNSIGNED 8	RW	6000 0108h
	2h	Read digital input 9h to 10h	UNSIGNED 8	RW	6000 0208h
	3h	Read digital input 11h to 18h	UNSIGNED 8	RW	6000 0308h
	4h	Read digital input 19h to 20h	UNSIGNED 8	RW	6000 0408h
	5h	Read digital input 11h to 28h	UNSIGNED 8	RW	6000 0508h
	6h	Read digital input 19h to 30h	UNSIGNED 8	RW	6000 0608h
	7h	Read digital input 11h to 40h	UNSIGNED 8	RW	6000 0708h

	8h	Read digital input 19h to 48h	UNSIGNED 8	RW	6000 0808h
1A01h	0h	Largest sub-index supported for “transmit PDO mapping”	UNSIGNED 8	RO	4
	1h	Read analog input 1h	UNSIGNED 16	RW	6401 0110h
	2h	Read analog input 2h	UNSIGNED 16	RW	6401 0210h
	3h	Read analog input 3h	UNSIGNED 16	RW	6401 0310h
	4h	Read analog input 4h	UNSIGNED 16	RW	6401 0410h
1A02h	0h	Largest sub-index supported for “transmit PDO mapping”	UNSIGNED 8	RO	4
	1h	Read analog input 5h	UNSIGNED 16	RW	6401 0510h
	2h	Read analog input 6h	UNSIGNED 16	RW	6401 0610h
	3h	Read analog input 7h	UNSIGNED 16	RW	6401 0710h
	4h	Read analog input 8h	UNSIGNED 16	RW	6401 0810h
1A03h	0h	Largest sub-index supported for “transmit PDO mapping”	UNSIGNED 8	RO	4
	1h	Read analog input 9h	UNSIGNED 16	RW	6401 0910h
	2h	Read analog input Ah	UNSIGNED 16	RW	6401 0A10h
	3h	Read analog input Bh	UNSIGNED 16	RW	6401 0B10h
	4h	Read analog input Ch	UNSIGNED 16	RW	6401 0C10h
1A04h	0h	Largest sub-index supported for “transmit PDO mapping”	UNSIGNED 8	RO	---
	1h	---	---	RW	---
	…	…	…	…	…
	---	---	---	RW	---
…	…	…	…	…	…
1A1Fh	0h	Largest sub-index supported for “transmit PDO mapping”	UNSIGNED 8	RO	---

	1h	---	---	RW	---
	…	…	…	…	…
	---	---	---	RW	---

6.2 Manufacturer Specific Profile Area

In the following table, there is information about some special functions for the Modbus RTU modules. Index 0x2000 records the access error count for the Modbus RTU module. Entries with sub-index 00 for the object indicate how many entries each object has. For more detail about these objects, please refer to section 5.5.

I/O Module Read/Write Error Counter Entries

Idx	Sidx	Description	Type	Attr	Default
2000h	0h	Largest sub-index supported for "I/O Module R/W Error Counter"	UNSIGNED 8	RO	8
	1h	Module 1 R/W Error Counter	UNSIGNED 16	RW	---
	…	…	…	…	…

6.3 Standardized Device Profile Area

When the I-7232D's power is on, These device entries will match the channel types and numbers of the Modbus RTU modules connected to the I-7232D's. In order to look them up conveniently, these entries are divided into four tables, "Digital Input Devices Entries", "Digital Output Devices Entries", "Analog Input Devices Entries" and "Analog Output Devices Entries". They are as follows.

Digital Input Devices Entries

Idx	Sidx	Description	Type	Attr	Default
6000h	0h	Largest sub-index supported for "read digital input 8-bit"	UNSIGNED 8	RO	8
	1h	Read digital input 1h to 8h	UNSIGNED 8	RO	---
	…	…	…	…	…

Digital Output Devices Entries

Idx	Sidx	Description	Type	Attr	Default
6200h	0h	Largest sub-index supported for "write digital output 8-bit"	UNSIGNED 8	RO	---
	1h	Write digital output 1h to 8h	UNSIGNED 8	RW	---
	…	…	…	…	…
6206	0h	Largest sub-index supported for "error mode digital output 8-bit"	UNSIGNED 8	RW	---
	1h	Error mode digital output 1h to 8h	UNSIGNED 8	RW	0
	…	…	…	…	---
6207	0h	Largest sub-index supported for "error value digital output 8-bit"	UNSIGNED 8	RW	---
	1h	Error value digital output 1h to 8h	UNSIGNED 8	RW	0
	…	…	…	…	---

Note: 1. When the bus-off is detected or the node guarding fails, the I-7232D will check the value of the object with index 0x6206. If the bit of this value sets to 1, the I-7232D will output the error mode digital output value to the corresponding DO channel. For example, if the sub-index 01 in the object with index 0x6206 and 0x6207 are 0x31 and 0xF8 respectively, When the error events occurs, only the DO5, DO4, DO0 will be changed to error mode output value because the bit 5, bit 4 and bit 1 of the value 0x31 is 1. And, the DO5, DO4, and DO0 will be change to 1, 1, and 0 respectively because bit 5, 4, and 1 of the value 0xF8 is 1, 1, and 0. Other channels beside DO5, DO4, and DO0 will do nothing.

Analog Input Devices Entries

Idx	Sad	Description	Type	Attr	Default

6401h	0h	Largest sub-index supported for "read analog input 16-bit"	UNSIGNED 8	RO	8
	1h	Read analog input 1h	UNSIGNED 16	RO	---
	…	…	…	…	…

Note: 1. Because the I-7232D only supports the hex format, all of the AI channels need to transfer to the hex format when storing into this object. The transformation equation is shown below.

$$FloatValue \quad HexValue \quad H\min \quad F\max \quad F\min \quad F\min$$

$$\frac{}{H\max \quad H\min}$$

The FloatValue is the result after transformation. The HexValue is the value which wants to be transferred. The Hmax and Hmin is the maximum and minimum values of the 2's complement hex range. The Fmax and Fmin is the maximum and minimum value of the float range. User can find out the Hmax, Hmin, Fmax, and Fmin, in the appendix B. For example, The input range of the module M-7017R is set to -10V ~ +10V. According to the table in the appendix B, we can find out the range for hex format is 0x8000 (+32767) ~ 0x7FFF (-32768). Therefore, if the value got from the AI channel of the M-7017R is 0x1234, the AI value with float format can be calculated as follows.

$$327674660 \quad 3276832768 \quad 10V$$

$$10V \quad 10V \quad 1.422V$$

$$\frac{}{}$$

By the way, any AI value, which is bigger then the maximum value of the input range will be set to the maximum value of the input range automatically. And, the AI value, which is small then the minimum value of the input range is also set to the minimum value of the input range automatically.

Analog Output Devices Entries

Idx	Sidx	Description	Type	Attr	Default
6411h	0h	Largest sub-index supported for "write analog output 16-bit"	UNSIGNED 8	RO	---
	1h	Write analog output 1h	UNSIGNED 16	RW	---
	…	…	…	…	…
6443	0h	Largest sub-index supported for "error mode analog output 16-bit"	UNSIGNED 8	RW	---
	1h	Error mode analog output 1h	UNSIGNED 16	RW	0
	…	…	…	…	---
6444	0h	Largest sub-index supported for "error value analog output 16-bit"	UNSIGNED 8	RW	---
	1h	Error value analog output 1h	UNSIGNED 16	RW	0
	…	…	…	…	---

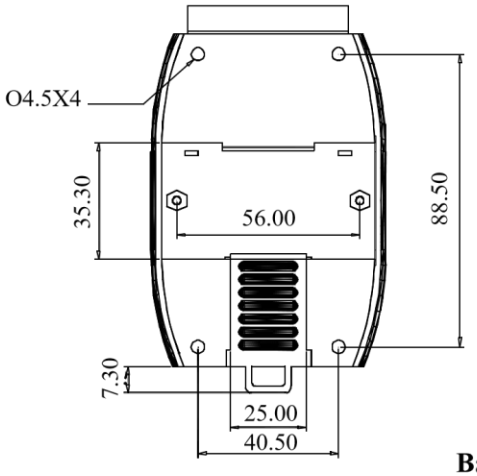
Note: 1. Because the I-7232D doesn't support float format, user need to transfer the AO value form float format to hex format. It is similar with the AI situation. The transformation equation is as follows.

$$HexValue = \frac{FloatValue - F\min}{F\max - F\min} \times (H\max - H\min) + H\min$$

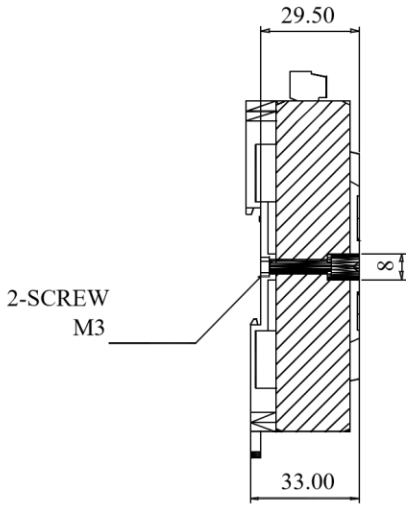
The HexValue is the result after transformation. The FloatValue is the value which wants to be transferred. The Fmax and Fmin is the maximum and minimum values of the float range. The Hmax and Hmin is the maximum and minimum value of the 2's complement hex range. User can find out the Fmax, Fmin, Hmax, and Hmin in the appendix B.

2. When the bus-off is detected or the node guarding fails, the I-7232D will check the value of the object with index 0x6443. If this value sets to 1, the I-7232D will output the error mode digital output value to the corresponding AO channel. For example, if the sub-index 01 in the object with index 0x6443 and 0x6444 are 1 and 0x0000 respectively, When the error events occurs, this AO will be output to error mode output because the value of the object with index 0x6443 and sub-index 01 is 1. The AO output value is 0 because of the value in the object with index 0x6444 and sub-index 01.

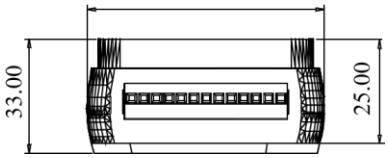
7 Appendix A: Dimensions and Mounting



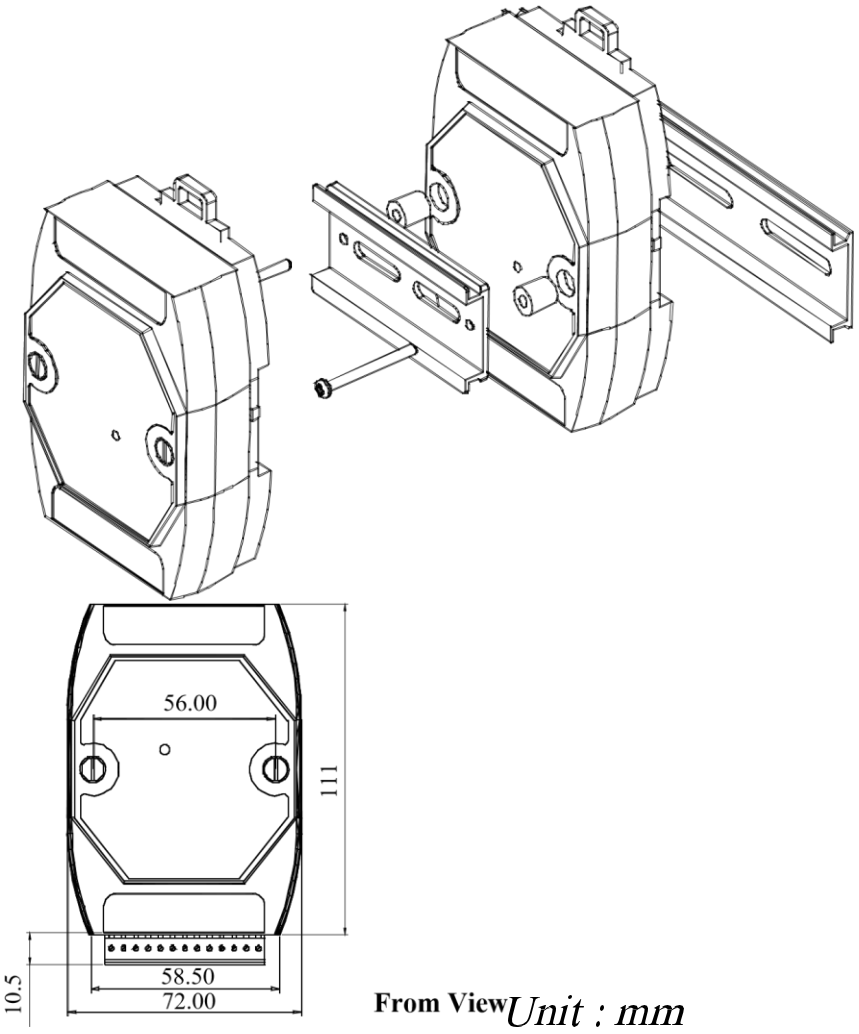
Back View

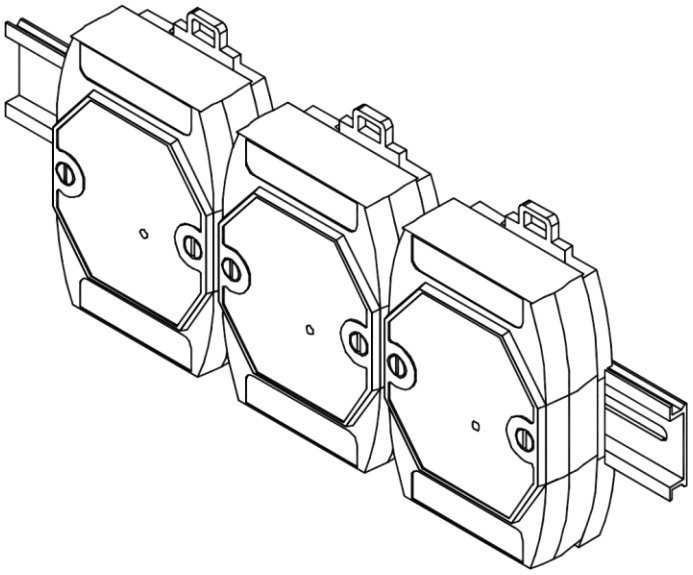


72.00 **Side View**



Top View


56.00
111
10.5
58.50
72.00
From View
Unit : mm



8 Appendix B: Analog I/O Transformation Table

In order to look up your required information, we have separated the transformation table into several parts according to the Modbus module names. They are given below.

- M-7017, M-7017R, M-7018, M-7018R, M-7019R, M-7015, M-7033 , M-7033 (D).

  (Note: The M-7018 and M-7018R doesn't have the +/- 5V).
- M-7024

M-7017, M-7017R, M-7018, M-7018R, M-7019R.

Range Code (Hex)	Data Format	Max value	Min value
08 (Default)	Input Range	+10.000V	-10.000V
	% of FSR	+100.00	-100.00
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
09	Input Range	+5.0000V	-5.0000V
	% of FSR	+100.00	-100.00
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
0A	Input Range	+1.0000V	-1.0000V
	% of FSR	+100.00	-100.00
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
0B	Input Range	+500.00mV	-500.00mV
	% of FSR	+100.00	-100.00

	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
0C	Input Range	+150.00mV	-150.00mV
	% of FSR	+100.00	-100.00
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
0D	Input Range (with 125 ohms resistor)	+20.000mA	-20.000mA
	% of FSR	+100.00	-100.00
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)

M-7015, M-7033, M-7033 (D)

Range Code (Hex)	RTD Type	Data Format	Max Value	Min Value
20 (Default)	Platinum 100 a = 0.00385	Input Range	+100.00℃	-100.00℃
		% of FSR	+100.00	-100.00
		Ohm	+138.50	+060.25
		2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
21	Platinum 100 a = 0.00385	Input Range	+100.00℃	+000.00℃
		% of FSR	+100.00	+000.00
		Ohm	+138.50	+100.00
		2's Complement HEX	0x7FFF (+32767)	0 (0)
22		Input Range	+200.00℃	+000.00℃

	Platinum 100 a = 0.00385	% of FSR	+100.00	+000.00
		Ohm	+175.84	+100.00
		2's Complement HEX	0x7FFF (+32767)	0 (0)
23	Platinum 100 a = 0.00385	Input Range	+600.00℃	+000.00℃
		% of FSR	+100.00	+000.00
		Ohm	+313.59	+100.00
		2's Complement HEX	0x7FFF (+32767)	0 (0)
24	Platinum 100 a = 0.003916	Input Range	+100.00℃	-100.00℃
		% of FSR	+100.00	-100.00
		Ohm	+139.16	+059.58
		2's Complement HEX	0x7FFF (+32767)	0x8000 (- 32768)
25	Platinum 100 a = 0.003916	Input Range	+100.00℃	+000.00℃
		% of FSR	+100.00	+000.00
		Ohm	+139.16	+100.00
		2's Complement HEX	0x7FFF (+32767)	0 (0)
26	Platinum 100 a = 0.003916	Input Range	+200.00℃	+000.00℃
		% of FSR	+100.00	+000.00
		Ohm	+177.13	+100.00
		2's Complement HEX	0x7FFF (+32767)	0 (0)
27	Platinum 100 a = 0.003916	Input Range	+600.00℃	+000.00℃
		% of FSR	+100.00	+000.00
		Ohm	+317.28	+100.00

		2's Complement HEX	0x7FFF (+32767)	0 (0)
28	Nickel 120	Input Range	+100.00℃	-80.00℃
		% of FSR	+100.00	-080.00
		Ohm	+200.64	+120.60
		2's Complement HEX	0x7FFF (+32767)	0x999A (-26214)
29	Nickel 120	Input Range	+100.00℃	+000.00℃
		% of FSR	+100.00	+000.00
		Ohm	+200.64	+120.60
		2's Complement HEX	0x7FFF (+32767)	0 (0)
2A	Platinum 1000 a = 0.00385	Input Range	+600.00℃	-200.00℃
		% of FSR	+100.00	-033.33
		Ohm	+3137.1	+0185.2
		2's Complement HEX	0x7FFF (+32767)	0xD556 (-10922)

M-7024

Range Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.000mA	+0.000mA
	2's Complement HEX	0x7FFF (+32767)	0 (0)
31	Output Range	+20.000mA	+04.000mA

	2's Complement HEX	0x7FFF (+32767)	0 (0)
32	Output Range	+10.000V	+00.000V
	2's Complement HEX	0x7FFF (+32767)	0 (0)
33 (Default)	Output Range	+10.000V	-10.000V
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)
34	Output Range	+05.000V	+00.000V
	2's Complement HEX	0x7FFF (+32767)	0 (0)
35	Output Range	+05.000V	-05.000V
	2's Complement HEX	0x7FFF (+32767)	0x8000 (-32768)