I-7242D

DeviceNet / Modbus RTU Gateway

User's Manual

Warranty

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

Warning

ICP DAS assumes 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 may be registered trademarks of their respective companies.

Table of Contents

1 Introduction

1.1 Overview

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

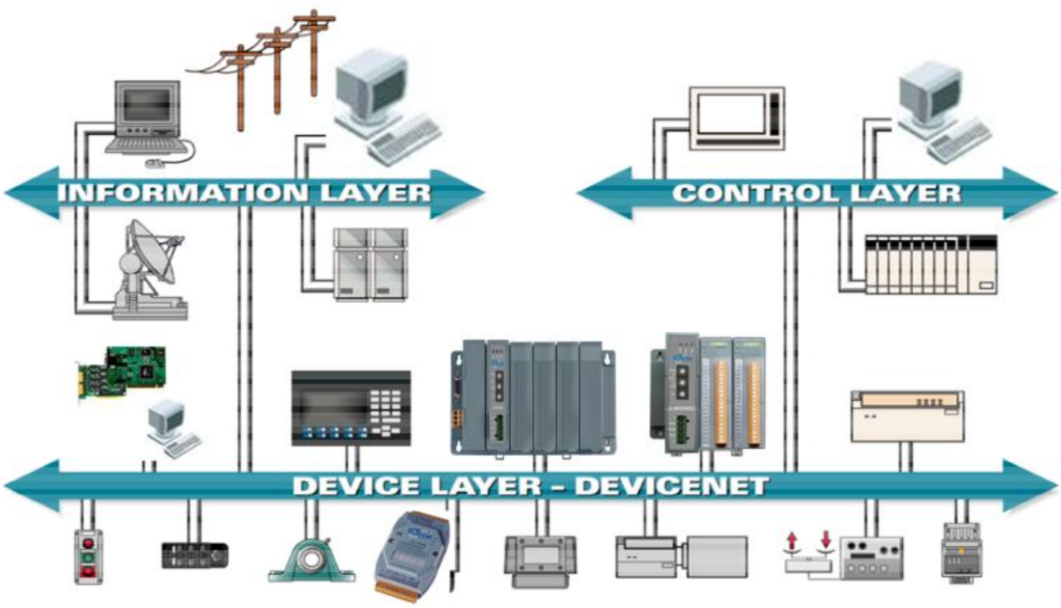


Figure 1.1 Architecture of the DeviceNet network

DeviceNet is a cost effective solution to one kind application of control area network. It reduces the connection wires between devices and provides rapid troubleshooting rejection function. The transfer rate can be up to 500Kbps within 100 meters. The transfer distance can be up to 500 meters in 125Kbps (See Table 1.1). It allows direct peer to peer data exchange between nodes in an organized and, if necessary, deterministic manner. Master/Slave connection model can be supported in the same network. Therefore, DeviceNet is able to facilitate all application communications based on a redefine a connection scheme. However, DeviceNet connection object stands as the communication path between multiple endpoints, which are application objects that is needed to share data.

Table 1.1 The Baud rate and the Bus length

Baud rate (bit/s)	Max. Bus length (m)
500 K	100
250 K	250
125 K	500

The I-7242D is one of CAN bus products in ICP DAS and stands as a DeviceNet slave/Modbus RTU master Gateway device. It allows a master located on a DeviceNet network to enter a dialogue with slave devices on the Modbus RTU network. In DeviceNet network, it functions as a Group 2 Only Slave device, and supports "Predefined Master/slave Connection Set". In Modbus RTU network, I-7242D represents the master device and responses to access the Modbus RTU slave device by DeviceNet object definition. In order to simplify the protocol converting mechanism, we also provide the DNS_MRU Utility software for users to configure the device parameters and build EDS file for the DeviceNet slave device. Users can easily apply Modbus RTU devices in DeviceNet applications through the I-7242D. The application architecture is depicted as figure 1-2. Users can connect the Modbus RTU devices to the DeviceNet network via the I-7242D.

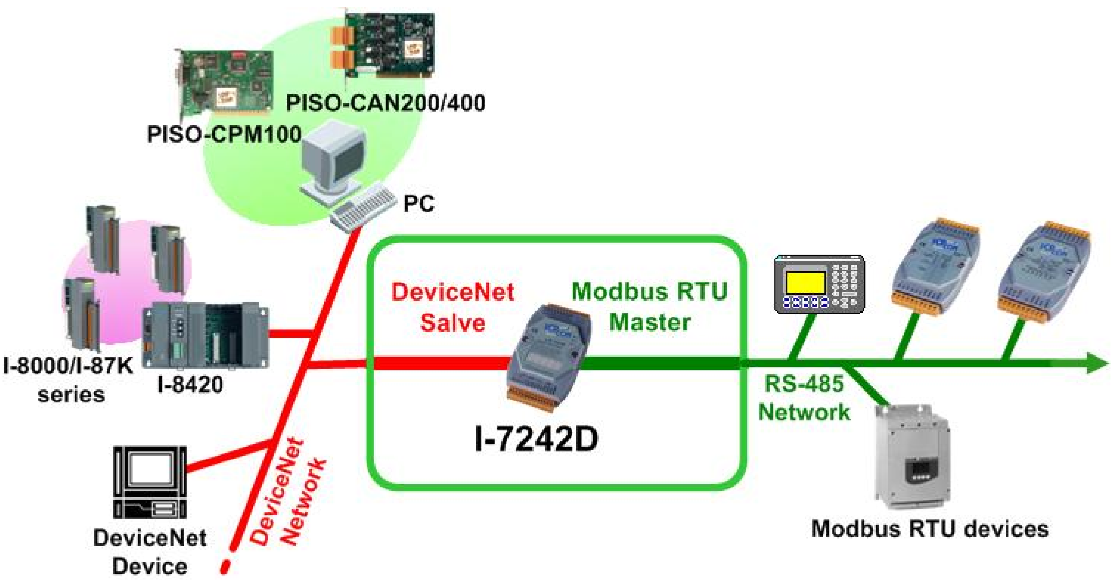


Figure 1-2 Application architecture

1.2 DeviceNet Applications

DeviceNet is the standardized network application layer optimized for factory automation. It is mainly used in low- and mid-volume automation systems. Some users have implemented DeviceNet protocol in machine control systems. The main DeviceNet application fields are demonstrated the following area. (For more information, please refer to www.odva.org):

• Production cell builds and tests CPUs	• Dinnerware production
• Beer brewery	• HVAC module production
• Equipment for food packing	• HVAC module production
• Fiberglass twist machine	• Trawler automation system
• Sponge production plant	• LCD manufacturing plant
• Sponge production plant	• Rolling steel door production
• Overhead storage bin production	• Bottling line
• Pocket-bread bakery	• Tight manufacturing

1.3 Hardware Features

System

- CPU: 80188 40MHz
- Philip SJA1000 CAN controller
- Philip 82C250 CAN transceiver
- SRAM: 512K bytes
- Flash Memory: 512K bytes
- EEPROM: 2K bytes
- Real Time Clock
- Built-in Dual-Watchdog
- 16-bit Timer
- 2500 Vrms isolation on CAN side
- Power Consumption: 2.8 W
- Unregulated +10VDC to +30VDC
- Operating Temperature: -25°C to +75°C
- Storage Temperature: -30°C to +85°C
- Humidity: 5%~95%
- NS, MS and IO Led indicator

COM1

- RS-232: TXD, RXD, RTS, CTS, GND
- Communication speed: 1200, 2400, 4800, 9600, 19200, 38400, 57600 or 115200 bits/s
- Used as configuration tool connection

COM2

- RS-485: D2+, D2-
- Communication speed: 1200, 2400, 4800, 9600, 19200, 38400, 57600 or 115200 bits/s
- Connect to Modbus RTU devices

Display

- Five 7-segmemt displays to show the information of operation mode, Node ID, CAN baud rate, RS-485 baud rate and device error code in sequence loop ways.

1.4 DeviceNet Features

- Comply with DeviceNet specification Volume I, Release 2.0 & Volume II, Release 2.0.
- "Group 2 only server" DeviceNet subscriber
- Dynamic Assembly Objects Mapping by Utility
- On-line change baud rate and MAC ID of CAN
- MS、NS and IO LED indicators
- Five 7-segmemt displays show the information of operation mode, MAC ID, baud rate and error code
- Connection supported:
  - 1 "Explicit Connection"
  - 1 "Polled Command/Response" connection
  - 1 "Bit Strobed Command/Response" connection
  - 1 "Cnange-of-State/Cyclic" connection
- Configuration facilitated by the use of specific EDS files.
- Configure user-defined Modbus RTU message by Explicit connection

1.5 Modbus RTU Features

- Maximum number of devices: 10 Modbus devices
- Communication speed: 1200、2400、4800、9600、19200、38400、57600 or 115200 bits/s, configured by using Utility.
- Data bits: 8 bits, configured by using Utility
- Parity bits: None, even or odd, configured by using Utility
- Stop bits: 1 or 2 bits, configured by using Utility
- Support Modbus devices communication error alarm.
- Support Modbus function codes: 0x01，0x02，0x03，0x04，0x0F and 0x10.

1.6 Utility Features

- Support DeviceNet node ID, baud rate setting
- Support com port communication setting
- Support Modbus RTU communication parameters setting according to specific devices
- Support DeviceNet Polling, Bit-Strobe and COS/Cyclic I/O connection path setting
- Show Modbus RTU devices configuration
- Show DeviceNet application and assembly objects configuration
- Dynamic produce EDS file

Please refer to Appendix A to know how to mount I-7242D

2 Hardware Specification

2.1 Hardware Structure

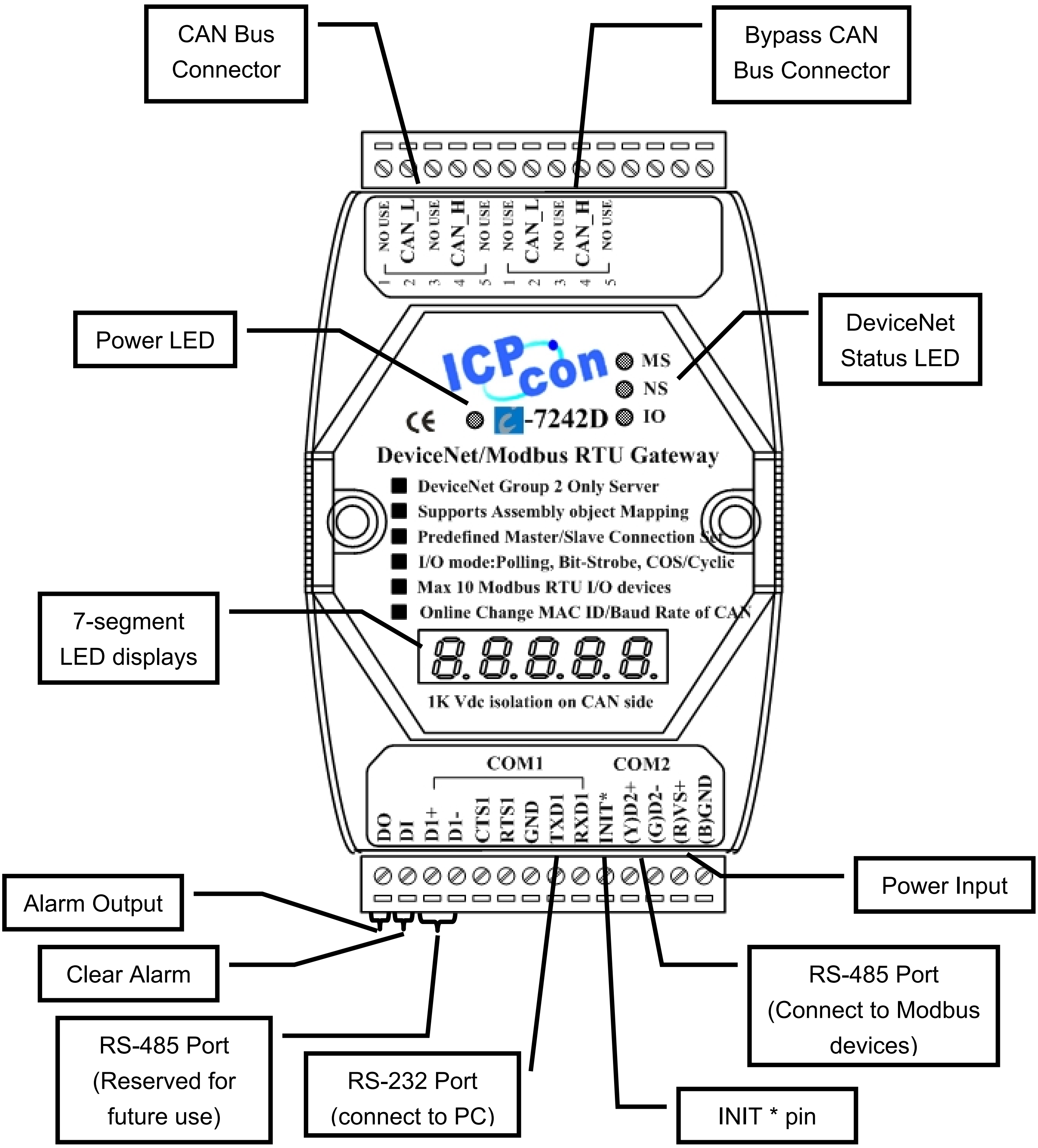


Figure 2-1 Hardware Structure of the I-7242D

2.2 Wire Connection

2.2.1 CAN bus wire connection

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

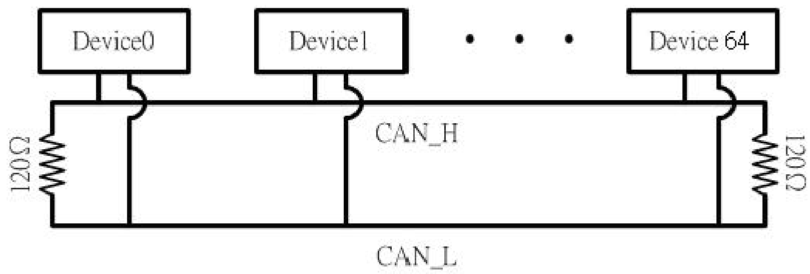


Figure 2-2 CAN Bus Wire Connections

Moreover, to minimize the voltage drop on long distance, the terminal resistance should be higher than the value defined in the ISO 11898-2. Table 2-1 may be used as a reference.

Table 2-1 Relation between bus cable and length

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

The CAN bus baud rate has the high relationship with the bus length. Table 2-2 indicates the corresponding bus length for every kind of baud rate in DeviceNet network.

Table 2-2 Baud rate and bus length for DeviceNet network

Baud rate (bit/s)	Max. Bus length (m)
500 K	100
250 K	250
125 K	500

In order to provide an easy CAN bus wiring, the I-7242D supplies one CAN port with two CAN bus connector interfaces. Each connecter built on the I-7242D looks like as figure 2-3 and table 2-3.

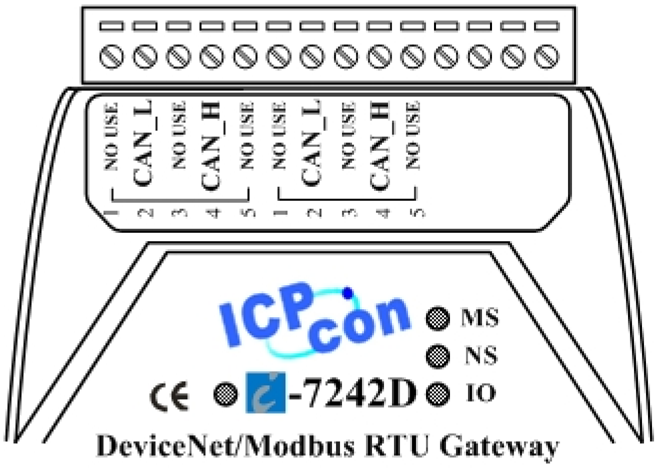


Figure 2-3 CAN bus connector of I-7242D

Table 2-3 Connector pins of I-7242D

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

Note that the bypass CAN bus connector is not another CAN channel. It is designed for connecting to another DeviceNet device conveniently. The structure of the inside electronic circuit is displayed as figure 2-4.

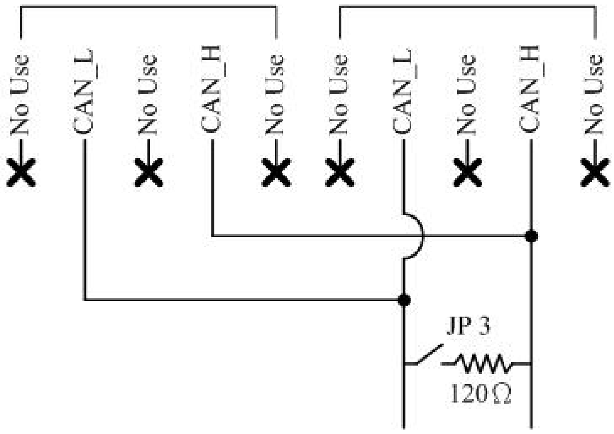


Figure 2-4 Electronic circuit of CAN bus connector

The jumper-selected termination resistor (J3) is positioned as the figure 2-5. And about the J3 jumper setting, please refer the table 2-4.

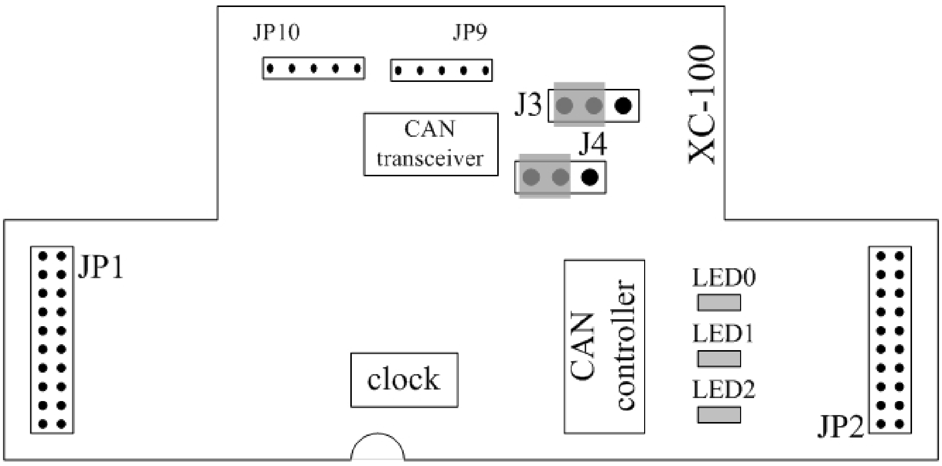
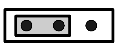
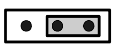


Figure2-5 XC100 I/O expansion board LAYOUT

Table 2-4 J3 Jumper Selection

Apply the termination resistor(120Ω)	**Don't apply the termination resistor**
 1 2 3	 1 2 3

2.2.2 Digital input/output wire connection

The DO and DI in the I-7242D are used for Modbus communication alarm. If the number of Modbus communication error exceeds 100, the DO value will be set as 1. And the DO status is OFF. Users can apply this to have a clear warning. Then users can clear the alarm signal by setting the DI value as 0.

After setting the DI to ON state, the data lose counter of each device would be adjusted to zero. The wire connection of digital input is as figure 2-6.

- **Digital Input level**

  Dry Contact:

  Logical level 0: closed to GND

  Logical level 1: open

  Wet contact:

  Logical level 0: +1V

  Logical level 1: +3.5V to +30V

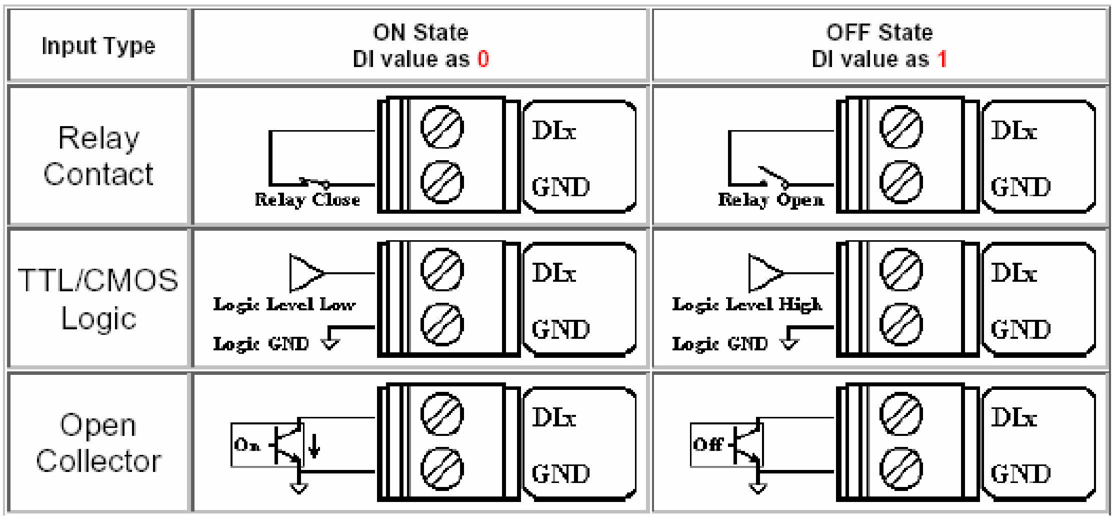


Figure 2-6 Digital Input Wire Connection

When the number of data lose counter exceeds 100, the DO would be in OFF state. Users can use the DO as the alarm of Modbus communication. The wire connection of digital output is as figure 2-7.

Digital output level

Open collector to 30V Max.

Output current: 100mA

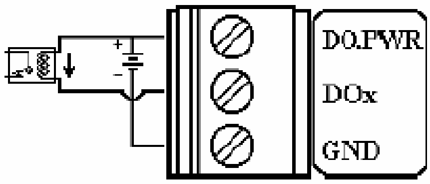
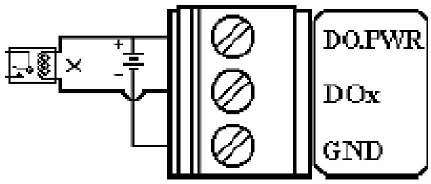
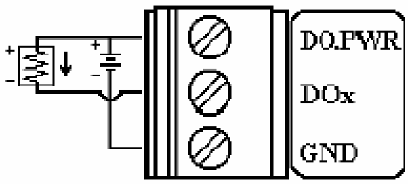
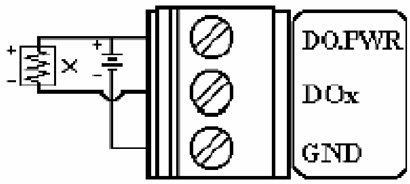
Output Type	ON State DO value as 0	OFF State DO value as 1
Drive Relay		
Resistance Load		

Figure 2-7 Digital Output Wire Connection

2.3 Power LED

After connecting the I-7242D with the electronic power (the range of input voltage is 10~30VDC). The Power LED will be turn on. If the Power LED is off after giving the proper voltage, please check the power and load of power supply firstly. If the situation is not improved, please communicate your local distributor to solve this problem. The corresponding conditions are given in table 2-5 and the location is shown in Figure 2.1.

Table 2-5 Power led conditions

Condition	**Status**	**Description**
Off	No power	No power supply
Solid red	Normal	Device is working

2.4 DeviceNet Indicator LED

The I-7242D includes three single-color LED displays to indicate the status of module, network and I/O device. They are MS LED (it is red), NS LED (it is green), and IO LED (it is red). The Indicators assist maintenance personnel in quickly identifying a problem unit. The LED test is to be performed at power–up. When the DeviceNet communication events occur, these indicators will be triggered to glitter with different conditions.

2.4.1 MS LED

This LED provides device status and indicates whether or not the device is operating properly. Table 2-6 shows the conditions of MS status. Therefore, when the device is operated normally, the MS-LED must be turned off.

Table 2-6 MS led conditions

Condition	**Status**	**Description**
Red	Critical fault	Device has unrecoverable fault
Flashing red	Non-critical fault	Device has recoverable fault; to recover: Reconfigure device Reset device Perform error recovery

2.4.2 NS LED

This LED indicates the status of the communication link for the device. Table 2-7 shows the conditions of NS status. Therefore, when the device is correctly communicating in the network, the NS-LED is normally turned on.

Table 2-7 NS led conditions

Condition	**Status**	**Description**
Off	Off line	DeviceNet is off line
Flashing green	On line	DeviceNet is on line, but not communicating
Init solid green	Link failed	The device has detected an error that has rendered it incapable of communicating on the link; for example, detected a duplicate node address or network configuration error
Solid green	On line, communicating	DeviceNet is on communication

2.4.3 IO LED

This LED provides the information of inputs or outputs access status. When Master get/set input/output data of Modbus devices via the I-7242D, the LED would be flashed. Table 2-8 shows the conditions for IO status. Therefore, when the device IO-function is working, the IO-LED should be flashed.

Table 2-8 IO led conditions

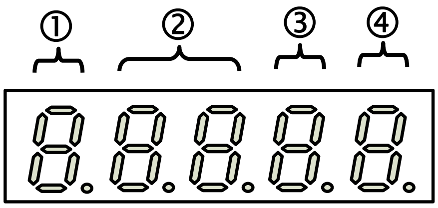
Condition	Status	Description
Off	No data	No data is being transmitted or received
Flashing red	Communicating	Data is being transmitted or received

2.5 Five 7-Segment LED Displays

The I-7242D provides five 7-Segment LED displays to show the current information of I-7242D in a sequence steps to represent the DeviceNet and Modbus network status.

Step 1. These LED displays show the string "-DEV-" in the first step.

Step 2. Then, they change to the next form, described as follows.



①: Show the operation state of the I-7242D. If it works normally, the LED display shows the character 'n'. If not, the LED display shows the error character. Table 2-9 shows the meaning of this LED.

Table 2-9

7-Segment LED Number	Error
'n'	Normal operation
'E'	I-7242D Hardware error
'd'	Default setting: Node ID=1 CAN baud 125K All I/O connection path=0

②: These two LED displays indicate the DeviceNet node ID of I-7242D in hex format. For example, if the DeviceNet node ID of I-7242D is 31, these two LEDs will show "1F".

③: This LED display shows the CAN bus baud rate of I-7242D by number 0~2. The meanings of these numbers are described in table 2-10.

Table 2-10

7-Segment LED Number	Baud rate (K bps)
0	125
1	250
2	500

④: The RS-485 baud rate of Modbus RTU in I-7242D is indicated on this LED display. The mapping table between LED number and RS-485 baud rate is displayed in table 2-11.

Table 2-11

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

Step 3. The first three LED displays show the string “ER-“ and others display the error code. The error code is described in table 2-12.

Table 2-12

Error code	Description
00	No Error
01	EEPROM data error. Use default setting
02	CAN Hardware Initial Error

If any message sent from I-7242D to Modbus devices has been lost, the LED display will be changed and display as the following form. Table 2-13 shows the meaning of these LED displays.

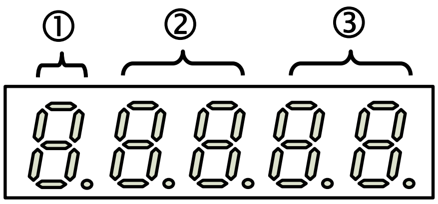
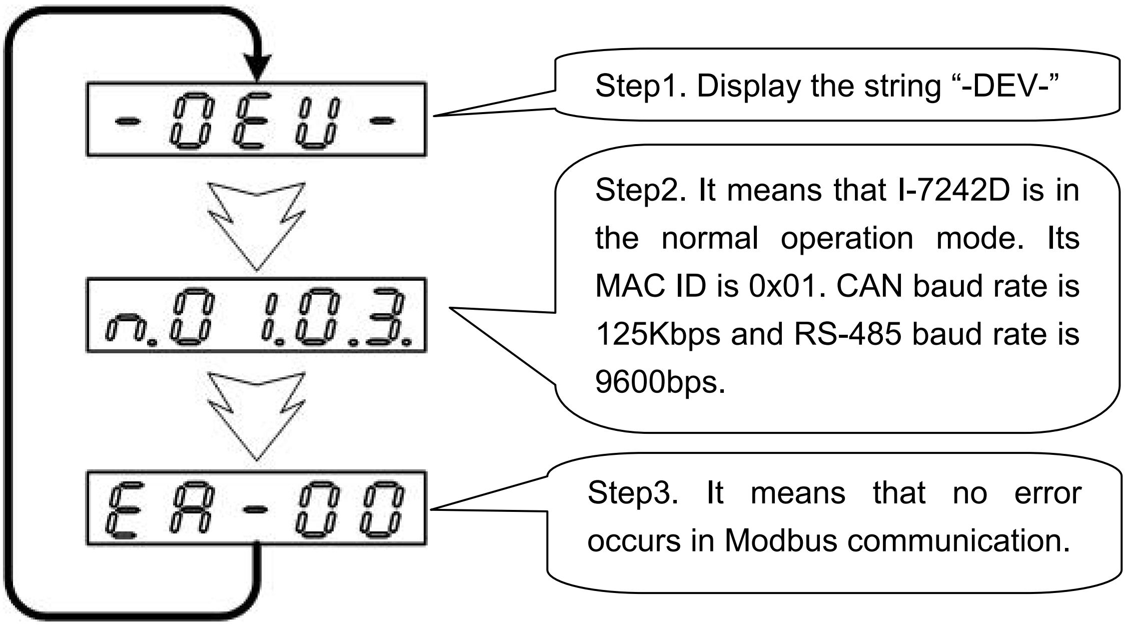


Table 2-13

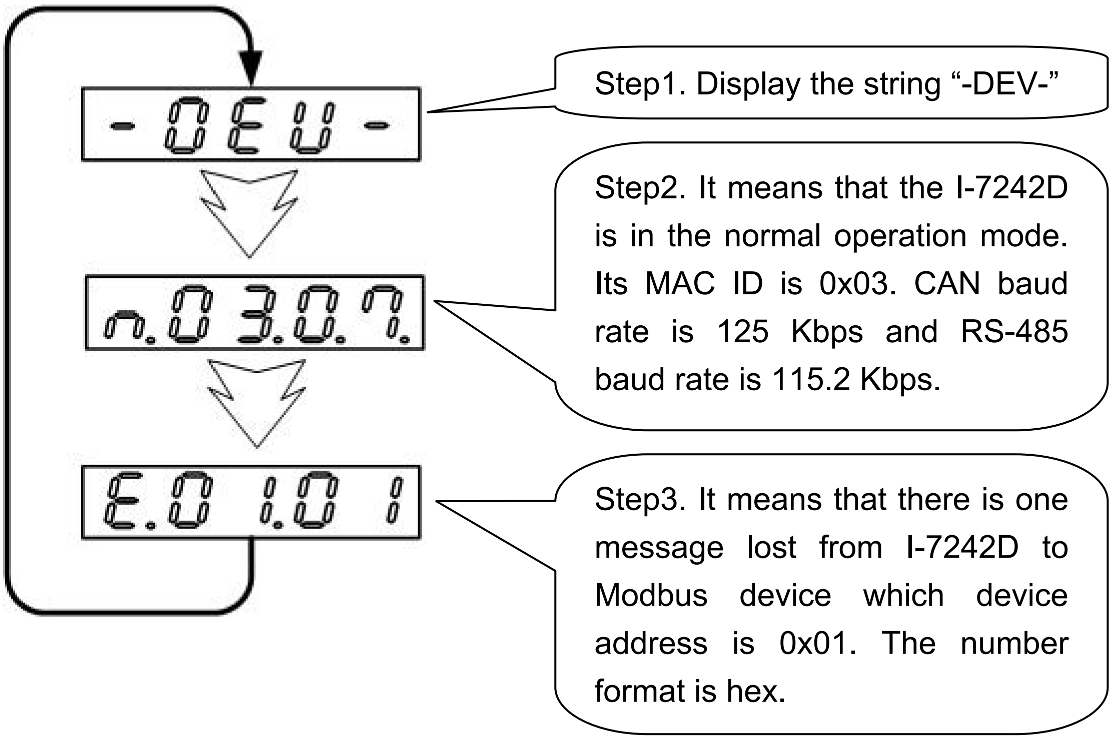
	7-segment Num	Description
①	“E”	Character ‘E’ means Error
②	Device address	Modbus device address in hex format.
③	Data-lose counter	Number of Data-lose of Modbus devices in hex format.

There are two examples of the 7-Segment LED displays in the active status.

Example1:



Example 2:



2.6 Modbus Devices Support

The I-7242D supports many kinds of Modbus commands. When users want to use Modbus RTU devices on the DeviceNet network, they must pay attention that their Modbus RTU devices can support what kind of the Modbus function codes described in chapter 8. However, I-7242D also supports special Modbus commands by the specific "User-defined Modbus Command" object (Class ID: 0x65). The Modbus functions supported in the I-7242D are as Table 2-14.

Table 2-14 Modbus functions supported in the I-7242D

Function Code	Modbus Command
0x01	Read Coil Status
0x02	Read Input Status
0x03	Read Holding Registers
0x04	Read Input Registers
0x0F	Force Multiple Coils
0x10	Preset Multiple Registers

3 DeviceNet System

3.1 DeviceNet network Introduction

DeviceNet is one of the kinds of the network protocols based on the CAN bus which are mainly used for machine control in embedded network, such as in textile machinery, printing machines, injection molding machinery, or packaging machines. DeviceNet is a low level network that provides connections between simple industrial devices (sensors, actuators) and higher-level devices (controllers). It allows direct peer to peer data exchange between nodes in an organized and, if necessary, deterministic manner. The network management functions specified in DeviceNet simplifies project design, implementation and diagnosis by providing standard mechanisms for network start-up and error management. DeviceNet defines a connection-based scheme to facilitate all application communications. A DeviceNet connection provides a communication path between multiple endpoints. The endpoints of a connection are applications that need to share data. The figure 3-11 shows the DeviceNet layer in the control and information layers.

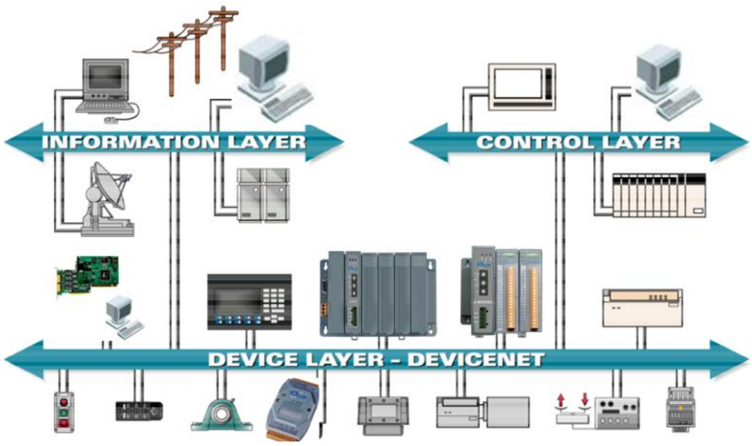


Figure 3-1 DeviceNet layer

The DeviceNet Communication Protocol is based on the concept of connections. One must establish a connection with a device in order to exclude information with that device. To establish a connection, each gateway implements Predefined Master/Slave Connection Set through the DeviceNet network. After establishing the explicit connections, the connection is then used to move information from one node to the other. Once IO connections have been established, I/O data may be moved among devices in the network.

The 11-bit CAN identifier is used to identify the connection. DeviceNet defines four separate groups of 11-bit CAN identifiers: Group 1, Group 2, Group 3, and Group 4 described in the Table 3-1. With respect to Connection Based Messages, the Connection ID is placed within the CAN Identifier Field. With this in mind, the below figure also describes the components for a DeviceNet Connection ID. Because of the arbitration scheme defined by CAN, Group 1 messages have a higher priority than group 2 messages and group 2 messages have higher priority than group 3 messages and so on. This prioritization must be taken into consideration when establishing connections.

Table 3-1 DeviceNet's Use of the CAN Identifier Field

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	9	8	7	6	5	4	3	2	1	0		
0	Group 1 Message ID				Source MAC ID						Group 1 Messages	000 – 3ff
1	0	MAC ID						Group 2 Message ID			Group 2 Messages	400 – 5ff
1	1	Group 3 Message ID			Source MAC ID						Group 3 Messages	600-7bf
1	1	1	1	1	Group 4 Message ID						Group 4 Messages	7c0–7ef

The I-7242D provides the Predefined Master/slave Connection Set for users to establish connections. The Predefined Master/Slave Connection Set is a set of Connections that facilitate communications typically seen in a Master/Slave relationship. Many of the steps involved in the creation and configuration of an Application connection have been removed within the Predefined Master/Slave Connection Set definition. This, in turn, presents the means by which a communication environment can be established using less network and device resources. The CAN Identifier Fields associated with the Predefined Master/Slave Connection Set are shown in the table 3-2. The table defines the Identifiers that are to be used with all connection based message involved in the Predefined Master/Slave Connection Set and, as such, it also illustrates the produced_connection_id and consumed_connection_id attributes associated with Predefined Master/Slave Connection Objects.

Note: The Master is the device that gathers and distributes I/O data for the process controller. Slaves are the devices from which the Master gathers I/O data and to which the Master distributes I/O data.

Table 3-2 Predefined Master/Slave Connection Set Identify Fields

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	9	8	7	6	5	4	3	2	1	0		
0	Group 1 Message ID				Source MAC ID						Group 1 Messages	000 – 3ff
0	1	1	0	0	Source MAC ID						Slave's Multicast Poll Response	
0	1	1	0	1	Source MAC ID						Slave's I/O Change of State or Cyclic Message	
0	1	1	1	0	Source MAC ID						Slave's I/O Bit–Strobe Response Message	
0	1	1	1	1	Source MAC ID						Slave's I/O Poll Response or Change of State/Cyclic Acknowledge Message	
1	0	MAC ID						Group 2 Message ID			Group 2 Messages	400 – 5ff
1	0	Source MAC ID						0	0	0	Master's I/O Bit–Strobe Command Message	
1	0	Multicast MAC ID						0	0	1	Master's I/O Multicast Poll Command Message	
1	0	Destination MAC ID						0	1	0	Master's Change of State or Cyclic Acknowledge Message	
1	0	Source MAC ID						0	1	1	Slave's Explicit/ Unconnected Response Messages/ Device Heartbeat Message/ Device Shutdown Message	
1	0	Destination MAC ID						1	0	0	Master's Explicit Request Messages	
1	0	Destination MAC ID						1	0	1	Master's I/O Poll Command/Change of State/Cyclic Message	
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages (reserved)	
1	0	Destination MAC ID						1	1	1	Duplicate MAC ID Check Messages	

A device within a DeviceNet network is represented by the below object model. The object model provides a template for organizing and implementing the Attributes (data), Services (methods or procedures) and behaviors of the components within a DeviceNet product. The figure 3-2 depicts the object model for I-7242D (Group 2 Only Server). The next section would explain these objects. The detail information about Predefined Master/Slave Connection Set is described in the next section.

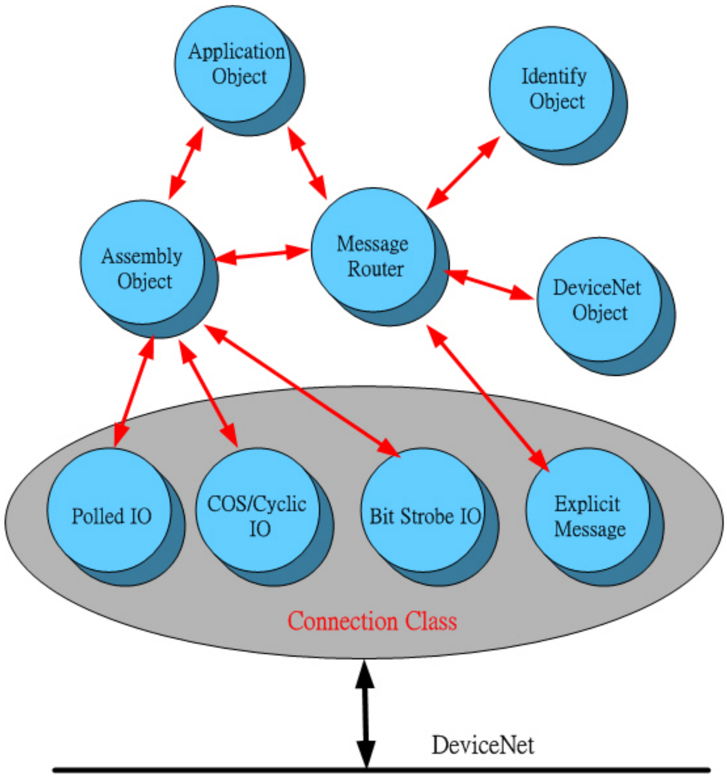


Figure 3-2 Object model of I-7242D

3.2 Predefined Master/Slave Connection Messages

The I-7242D provides "Predefined Master Slave Connection Set" device. Users must understand these connection set to know how to operate the device. The following section explains what the "Predefined Master Slave Connection Set" is.

With the Predefined Master Slave Connection Set, DeviceNet allows devices with fewer resources to take part in DeviceNet network communication. For this reason a set of identifiers has been reserved within the Message Group 2 to simplify the movement of I/O and configuration data typically seen in Master/Slave relationships. The steps, which are necessary to create and configure a connection between devices, have been removed within the Predefined Set. The Predefined Master/Slave Connection Set allows for the establishment of a DeviceNet communication environment using less network and Device resources. The Predefined Connection Set contains one Explicit Messaging Connection and allows several different I/O Connections which include a Bit Strobe Command/Response, Poll Command/Response, Change of State and Cyclic. The following type of messages are processed by a DeviceNet Slave

3.2.1 Explicit Response/Request Messages

Explicit Request Messages are used to perform operations such as reading and writing attributes. Explicit response Messages indicate the results of the slaves answer to attempt to service an Explicit Request massage. Within a Slave Explicit Request and Response messages are received/transmitted by a single Connection Object. The architecture is as figure 3-3.

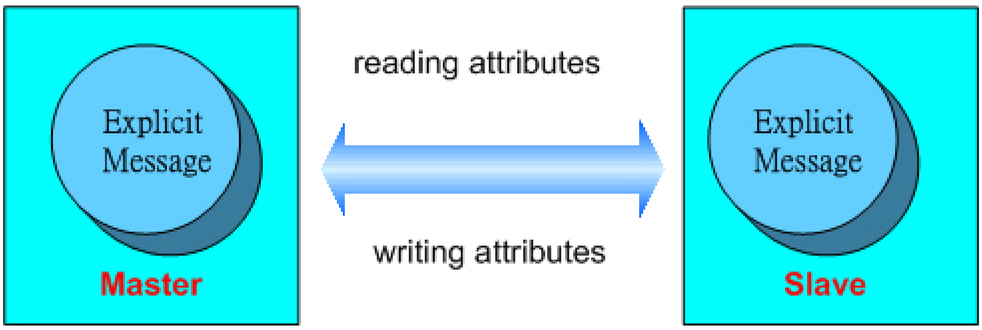


Figure 3-3 Architecture of Explicit message

3.2.2 I/O Poll Command/Response Messages

The Poll Command Message is a command that is transmitted by the Master. A Poll Command is directed towards a single, specific Slave (point-to-point connection). A Master must transmit a separate Poll command message for each one of its Slaves that will be polled. The Poll Response Message is an I/O message that the Slave transmits back to the Master when a Poll Command is received. Within a Slave the two messages are received/transmitted by a single Connection Object. The architecture is as figure 3-5.

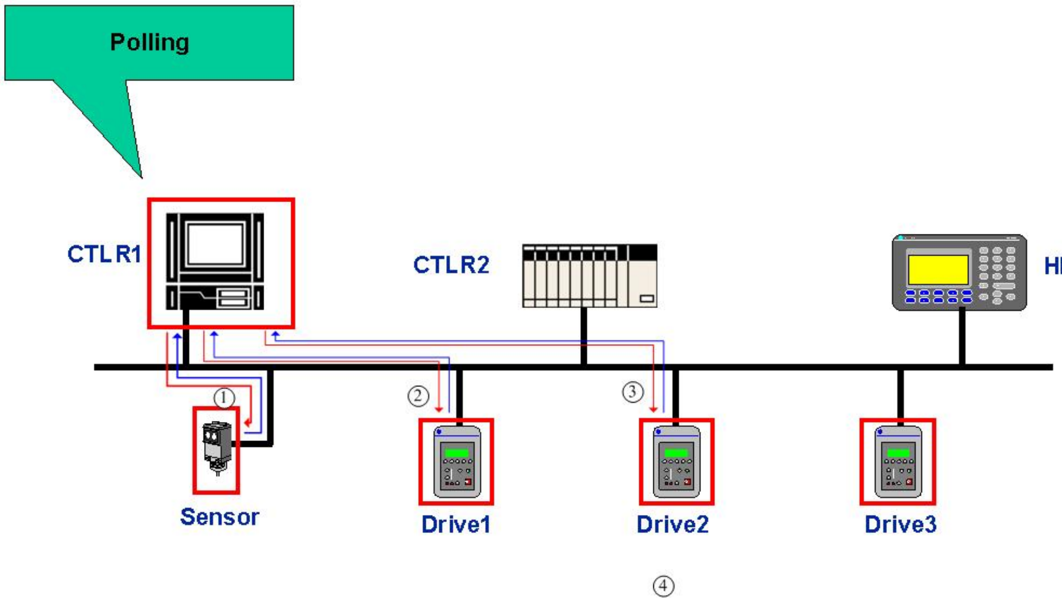


Figure 3-5 Architecture of IO poll message

3.2.3 I/O Bit-Strobe Command/Response Messages

The Bit-Strobe Command Message is an I/O message that is transmitted by the Master. A Bit-Strobe Command has multicast capabilities. Multiple Slaves can receive and react to the same Bit Strobe Command. The Bit-Strobe response is an I/O message that a Slave transmits back to the Master when the Bit-Strobe Command has been received. Within a Slave the two messages are received/ transmitted by a single Connection Object. The architecture is as figure 3-5.

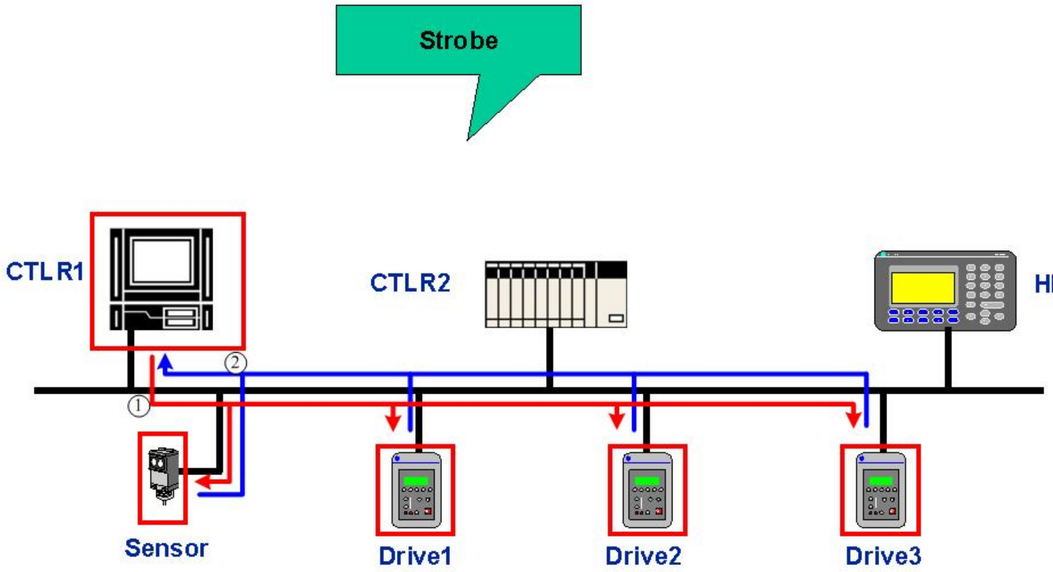


Figure 3-5 Architecture of IO bit strobe message

3.2.4 I/O Change of State/Cyclic Messages

The Change of State/Cyclic Message is transmitted by either the Master or the Slave. A Change of Sate/Cyclic is directed towards a single specific node (point-to-point). An Acknowledge Message may be returned in response to this message. Within either the Master or the Slave the producing Change of State Message and consuming Acknowledge Message are received/transmitted by one Connection Object. The consuming Change of State and producing Acknowledge Message are received/transmitted by a second Connection Object. The architecture is as figure 3-6 and figure 3-7.

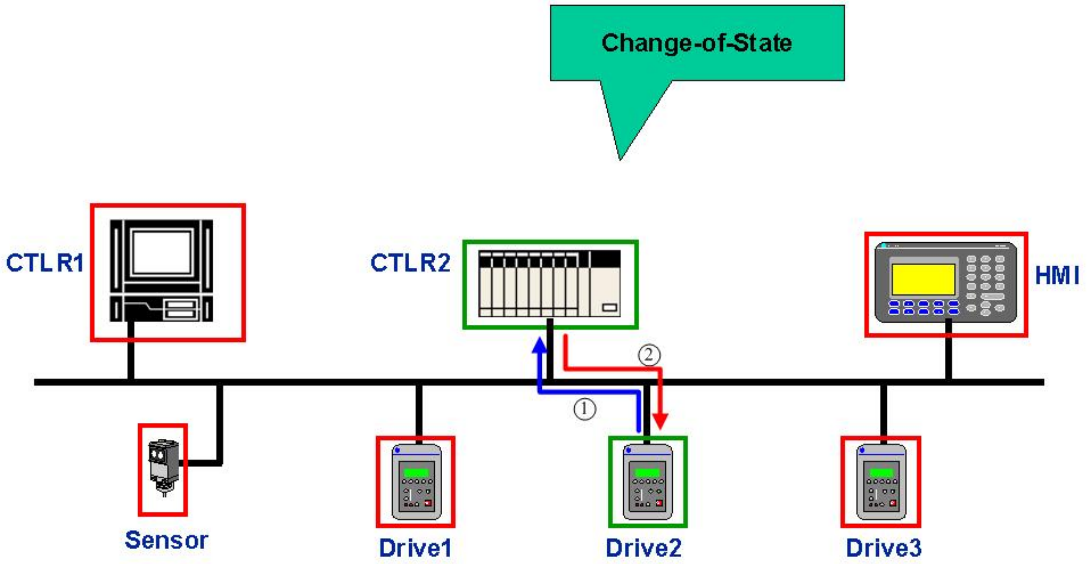


Figure 3-6 Architecture of IO COS message

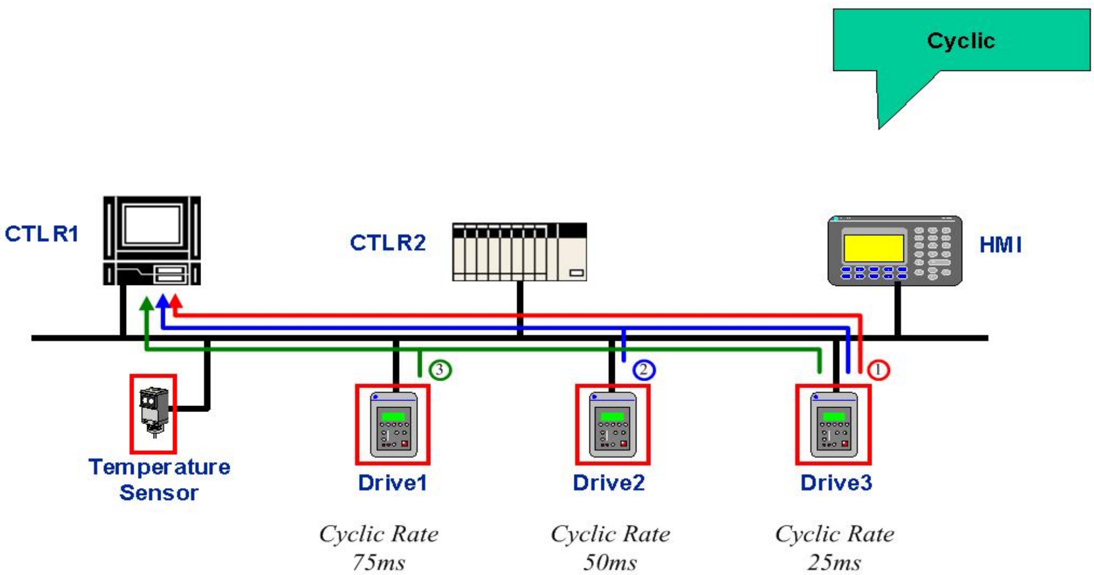


Figure 3-7 Architecture of IO Cyclic message

3.3 EDS file

An Electronic Data Sheet is an external disk that contains information about configurable attributes for devices, including the object addresses of each parameter. The following figure shows that the configuration tool uses the EDS file to configure those Modbus RTU devices. The application objects in these devices represent the destination addresses for the configuration data. These addresses are encoded in the EDS. ICP DAS provides users with DNS_MRU Utility software to create the suitable EDS file. The EDS file system architecture is as figure 3-8.

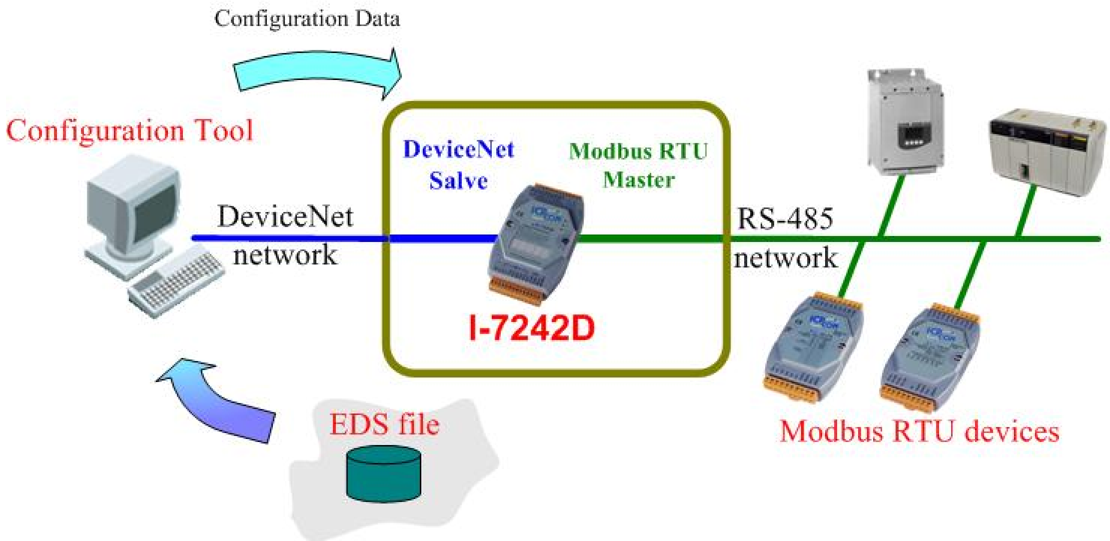


Figure 3-8 Architecture of EDS file

EDS provides information about the device's configuration data in terms of the following:

- context
- content
- format

The information in an EDS allows configuration tools to provide informative screens that guide a user through the steps necessary to configure a device. ICP DAS provides CAN utilities, so that users can setup their own EDS file. You can use the EDS file in the DeviceNet Master to access DeviceNet Slave devices. The DNS_MRU Utility is a very powerful tool for the DeviceNet network. It provides DeviceNet Slave information and supports the graph interface for users to make up the EDS file of their own system. For more detail information on this topic, please refer to the next session.

4 DeviceNet Profile Area

This chapter is for users who want to understand more detailed information related to the I-7242D device when using the DeviceNet protocol. This section documents the detailed functions for each object class that is implemented in the DeviceNet network

4.1 Introduction to the DeviceNet Objects of I-7242D

The I-7242D has been developed in accordance with the ***Object Modeling*** from the DeviceNet protocol. This model leads to a method used for addressing the I-7242D's data made up of four separate values: MAC ID、Class ID、Instance ID and Attribute ID. An address made up in this way is known as a "**Path**". The Connection by Explicit Messaging, for example, uses paths of this sort to exchange data from one node to another on a DeviceNet network. See table 4-1 to know the address field of I-7242D.

Table 4-1 Address field of I-7242D

Address	Min. - Max.	Description
Node	0-63	This field allows you to address one subscriber out of the series of subscribers on a DeviceNet network using its **MAC ID**.
Class	1-65535	All objects sharing the same characteristics belong to the same class, characterized by its **Class ID**.
Instance	0-65535	All instances from one class share the same **attributes** but each of them has its own set of values for these attributes.
Attribute	1-255	It is assigned some sort of value (byte, unsigned integer, character string, etc.) in order to supply information about the subscriber's status or to make settings on the subscriber's behaviors

4.2 DeviceNet Statement of Compliance

General Device Data

Conforms to DeviceNet Specification	Volume I, II Release 2.0
Vendor Name	ICP DAS
Device Profile Name	ICPDAS-I7242D
Production Revision	1.01

DeviceNet Physical Conformance Data

Network Power Consumption (Max)	Open-Hardwired
Isolated Physical Layer	Yes
LED Supported	Yes
MAC ID Setting	Software
Device MAC ID	Software (Default is 0x01)
Communication Rate Setting	Software (Default is 125k bits/s)
Predefined Master/Slave Connection Set	Group 2 Only Server
Connection supported	1 “Explicit Connection” 1 “Polled Command/Response” Connection 1 ”Bit Strobed Command/Response” Connection 1 ”Cnange-of-State/Cyclic” Connection

4.3 List of the I-7242D's DeviceNet Object

The I-7242D supports the following DeviceNet object classes.

Object Type	Class Code	Instances	Interfaces
Identity	01 (0x01)	1	Message Router
Message Router	02 (0x02)	1	Explicit message connection
DeviceNet	03 (0x03)	1	Message Router
Assembly	04 (0x04)	4 (3,2,1,0)	I/O connections or Message router
Connection	05 (0x05)	4 (2)	I/O connections or Explicit messages
Acknowledge handler object	43 (0x2B)	1	I/O connections or Message router
User-Defined Modbus Device	100 (0x64)	10 (max)	Message Router
User-Defined Modbus Command	101 (0x65)	3 (max)	Message Router

4.4 Identity Object (Class : 0x01)

This object provides the identification of and general information about the device. It is described in chapter 6-2 of volume II of the DeviceNet specifications.

Class Attribute

Attribute ID	Attribute name	Data Type	Method	Value
0x01	Revision	UINT	Get	1
0x02	Max Instance	UINT	Get	1

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required

Instance Attribute

Attribute ID	Description	Data Type	Method	Value
0x01	Vendor UINT	UINT	Get	803
0x02	Device type	UINT	Get	1
0x03	Product code	UINT	Get	1
0x04	Vendor Revision “major.minor”	USINT, USINT	Get	1.1
0x05	Status	WORD	Get	(16-bit register)
0x06	Serial number	UDINT	Get	(variable)
0x07	Product name	Short_String	Get	“ICPDAS-I7242D”
0x0A	Heartbeat Interval	USINT	Get/Set	0-65535

Instance Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Required
0x05	Reset	Required

4.5 Message Router Object (Class : 0x02)

The “Message Router” object is the element through which all objects of the “Explicit messages” type pass, so that they can be routed to the objects they are intended for. This object is described in chapter 6-3 of volume II of the DeviceNet specifications.

Class Attribute

Attribute ID	Attribute name	Data Type	Method	Value
0x01	Revision	UINT	Get	1

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required

Attributes of instance 0x01

This instance has no attributes.

4.6 DeviceNet Object (Class : 0x03)

The DeviceNet Object is used to provide the configuration and status of a physical attachment on the DeviceNet network. It is described in chapter 5-5 of volume I of the DeviceNet specifications. The I-7242D is a “Group two Only Server” type subscriber (see chapter 7-9 of volume I of the DeviceNet specifications).

Class Attribute

Attribute ID	Attribute name	Data Type	Method	Value
0x01	Revision	UINT	Get	2

Class Service

Service Code	Service name	Need
0x0E	Get_Attribute_Single	Required

Instance Attribute

Attribute ID	Description	Data Type	Method	Value
0x01	MAC ID	USINT	Get/Set	0~63
0x02	Baud rate	USINT	Get/Set	0~2
0x03	BOI	USINT	Get/Set	0
0x04	Bus-off counter	USINT	Get/Set	0
0x05	Allocation information	BYTE	Get/Set	(variable)

Instance Service

Service Code	Service name	Need
0x0E	Get_Attribute_Single	Optional
0x10	Set_Attribute_Single	Optional
0x4B	Allocation Master/Slave Connection Set	Optional
0x4C	Release Master/Slave Connection Set	Optional

4.7 Assembly Object (Class : 0x04)

The object from the “Assembly” class is used to group attributes belonging to different objects within a single attribute. This allows them to be accessed using a single message. With the I-7242D, this class has up to four instances (instance ID=0x64 to 0x67) and each one can be used to bind input data or output data. The terms of ”input” and ”output” are defined from the network’s point of view. An input will produce data on the network and an output will consume data from the network. This object is described in chapter 6-5 of volume II of the DeviceNet specifications.

Class Attribute

Attribute ID	Attribute name	Data Type	Method	Default Value
0x01	Revision	UINT	Get	2
0x02	Max Instance	UINT	Get	Maximum: 4

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required

Instances of assembly object are allocated in the type of Modbus devices that you selected. And they are ranked in the form of DO、AO、DI and AI types according to the definition in the DNS_MRU Utility tool. For more information, please refer to chapter 5.

Instances (0x64~0x67) Attribute

Attribute ID	Attribute name	Data Type	Method	Value
0x03	Data	USINT […]	Get/Set	(array of values)

Instance Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Optional

4.8 Connection Object (Class : 0x05)

This section presents the externally visible characteristics of the connection objects associated with the Predefined Master/Slave Connection Set within slave devices. With the I-7242D, the “Connection” object has up to four instances (Instance ID 0x01 to 0x04). Each of these instances represents one of the two ends of a virtual connection established between two nodes on the DeviceNet network. Each instance of this object belongs to one of the two following types of connection: Explicit connection, allowing *Explicit Messages* to be sent, or *I/O Connections*. This object is described in chapter 5-4 of volume I of the DeviceNet specifications.

Here is a brief description of the four instances of the I-7242D’s “Connection” object, and then details are given in the rest of this chapter:

Instance ID	Type of connection	Connection name
0x01	Explicit Messaging	Explicit Connection
0x02	I/O Connection	Polled Command/Response Connection
0x03	I/O Connection	Bit-Strobed Command/Response Connection
0x04	I/O Connection	Change-of-State / Cyclic (Acknowledged) Connection

Class Attribute

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	Revision	UINT	Get	1

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required

Instance (0x01) Attribute: Explicit Connection

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	State	USINT	Get	0x00 to 0x05
0x02	Instance type	USINT	Get	0x00
0x03	Transport class trigger	BYTE	Get/Set	0x83
0x04	Produced connection id	UINT	Get/Set	Table 3-2
0x05	Consumed connection id	UINT	Get/Set	Table 3-2
0x06	Initial comm. characteristics	BYTE	Get/Set	0x21
0x07	Produced connection size	UINT	Get/Set	0x84
0x08	Consumed connection size	UINT	Get/Set	0x84

0x09	Expected packet rate	UINT	Get/Set	0x09c4
0x0C	Watchdog timeout action	USINT	Get/Set	0x01
0x0D	Produced connection path length	UINT	Get/Set	0x00
0x0E	Produced connection path	EPATH	Get/Set	(empty path)
0x0F	Consumed connection path length	UINT	Get/Set	0x00
0x10	Consumed connection path	EPATH	Get/Set	(empty path)
0x11	Production inhibit time	UINT	Get/Set	0x00

Instance (0x02) Attribute: Polled Command/Response Connection

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	State	USINT	Get	0x00 to 0x04
0x02	Instance type	USINT	Get	0x01
0x03	Transport class trigger	BYTE	Get/Set	0x82
0x04	Produced connection id	UINT	Get/Set	Table 3-2
0x05	Consumed connection id	UINT	Get/Set	Table 3-2
0x06	Initial comm. characteristics	BYTE	Get/Set	0x01
0x07	Produced connection size	UINT	Get/Set	(size of the input data)
0x08	Consumed connection size	UINT	Get/Set	(size of the output data)
0x09	Expected packet rate	UINT	Get/Set	0x00
0x0C	Watchdog timeout action	USINT	Get/Set	0x00
0x0D	Produced connection path length	UINT	Get/Set	0x06
0x0E	Produced connection path	EPATH	Get/Set	(area path)
0x0F	Consumed connection path length	UINT	Get/Set	0x06
0x10	Consumed connection path	EPATH	Get/Set	(area path)
0x11	Production inhibit time	UINT	Get/Set	0x00

Instance (0x03) Attribute: Bit-Strobed Command/Response Connection

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	State	USINT	Get	0x00 to 0x04
0x02	Instance type	USINT	Get	0x01
0x03	Transport class trigger	BYTE	Get/Set	0x83
0x04	Produced connection id	UINT	Get/Set	Table 3-2
0x05	Consumed connection id	UINT	Get/Set	Table 3-2
0x06	Initial comm. characteristics	BYTE	Get/Set	0x02
0x07	Produced connection size	UINT	Get/Set	(size of the input data)
0x08	Consumed connection size	UINT	Get/Set	0x08
0x09	Expected packet rate	UINT	Get/Set	0x00

0x0C	Watchdog timeout action	USINT	Get/Set	0x00
0x0D	Produced connection path length	UINT	Get/Set	0x00
0x0E	Produced connection path	EPATH	Get/Set	(area path)
0x0F	Consumed connection path length	UINT	Get/Set	0x00
0x10	Consumed connection path	EPATH	Get/Set	(area path)
0x11	Production inhibit time	UINT	Get/Set	0x00

Instance (0x04) Attribute: Change-of-State/Cyclic (Acknowledged) Connection

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	State	USINT	Get	0x00 to 0x04
0x02	Instance type	USINT	Get	0x01
0x03	Transport class trigger	BYTE	Get/Set	0x12 or 0x02
0x04	Produced connection id	UINT	Get/Set	Table 3-2
0x05	Consumed connection id	UINT	Get/Set	Table 3-2
0x06	Initial comm. characteristics	BYTE	Get/Set	0x01
0x07	Produced connection size	UINT	Get/Set	(size of the input data)
0x08	Consumed connection size	UINT	Get/Set	0x00
0x09	Expected packet rate	UINT	Get/Set	0x00
0x0C	Watchdog timeout action	USINT	Get/Set	0x00
0x0D	Produced connection path length	UINT	Get/Set	0x00
0x0E	Produced connection path	EPATH	Get/Set	(area path)
0x0F	Consumed connection path length	UINT	Get/Set	0x04
0x10	Consumed connection path	EPATH	Get/Set	20h 2Bh 24h 01h
0x11	Production inhibit time	UINT	Get/Set	0x00

Instance (0x04) Attribute: Change-of-State/Cyclic (Unacknowledged) Connection

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	State	USINT	Get	0x00 to 0x04
0x02	Instance type	USINT	Get	0x01
0x03	Transport class trigger	BYTE	Get/Set	0x00
0x04	Produced connection id	UINT	Get/Set	Table 3-2
0x05	Consumed connection id	UINT	Get/Set	0xFFFF
0x06	Initial comm. characteristics	BYTE	Get/Set	0x0F
0x07	Produced connection size	UINT	Get/Set	(size of the input data)
0x08	Consumed connection size	UINT	Get/Set	0x00
0x09	Expected packet rate	UINT	Get/Set	0x00
0x0C	Watchdog timeout action	USINT	Get/Set	0x00

0x0D	Produced connection path length	UINT	Get/Set	0x00
0x0E	Produced connection path	EPATH	Get/Set	(area path)
0x0F	Consumed connection path length	UINT	Get/Set	0x00
0x10	Consumed connection path	EPATH	Get/Set	(empty path)
0x11	Production inhibit time	UINT	Get/Set	0x00

Instance Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Optional

4.9 Acknowledge Handler Object (Class 0x2B)

This object is used by connections whose producer needs to know whether its data has received by its consumers. This object is described in chapter 6-31 of volume II of the DeviceNet specifications.

Class Attribute

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	Revision	UINT	Get	1
0x02	Max instance	UINT	Get	1

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required

Instance Attribute

Attribute ID	Attribute Name	Data Type	Method	Value
0x01	Acknowledge timer	UINT	Get/Set	20 (ms)
0x02	Retry limit	USINT	Get/Set	1
0x03	COS producing connection instance	UINT	Get	4
0x04	Ack list size	BYTE	Get	1
0x05	Ack list	BYTE, UINT[…]	Get	0, (empty)
0x06	Data with ack list size	BYTE	Get	1
0x07	Data with ack path list	BYTE, (UINT, USINT, USINT[…])[…]	Get	(data with ack path list)

Instance Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Required

4.10 User-defined Modbus Device Object (Class : 0x64)

The "User-defined Modbus Device" object has maximum 10 instances and is specific to the I-7242D. Its attributes contain the application data, which is to be transmitted to the Modbus slaves via Modbus queries. The DeviceNet accesses the application data by invoking read and write functions. These functions need to be provided by the Object. So, the I-7242D provides Get_Attribute_Single and Set_Attribute_Single to read and write data to Modbus devices.

Class Attribute

Attribute ID	Attribute name	Data Type	Method	Value
0x01	Revision	UINT	Get	1
0x02	Max Instance	UINT	Get	10
0x03	Period of silence	USINT	Get/Set	40~65535 (ms)

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Optional

Instance Attribute

Attribute ID	Description	Data Type	Method	Value
0x01	Device ID	CHAR	Get	0
0x02	Device I/O Type	CHAR	Get	0
0x03	Device Start Address	WORD	Get	0
0x04	Device Length	WORD	Get	0
0x05	Data Lose Counter	WORD	Get/Set	0
0x14	DO Data	Defined by device num.	Get/Set	0
0x15	AO Data	Defined by device num.	Get/Set	0
0x16	DI Data	Defined by device num.	Get	0
0x17	AI Data	Defined by device num.	Get	0

Instance Service

Service Code	Service name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Required

4.11 User-defined Modbus Command Object (Class : 0x65)

The “User Defined Modbus Command” Object has maximum three instances and is specific to the I-7242D. Its attributes contain the application data, which can be defined by the special Modbus commands and can be transmitted to the Modbus slaves via Modbus queries. Another attribute of this object’s instances can receive data from a Modbus response

Users can only use Get_Attribute_Single and Set_Attribute_Single to read/write their commands from/to Modbus devices via this object.

Class Attribute

Attribute ID	Attribute name	Data Type	Method	Value
0x01	Revision	UINT	Get	1
0x02	Max Instance	UINT	Get	3

Class Service

Service Code	Service Name	Need
0x0E	Get_Attribute_Single	Required

Instance Attribute

Attribute ID	Description	Data Type	Method	Value
0x01	**Query Modbus Message:** Address Command Register Number of Register	Structure of USINT USINT UINT USINT	Get/Set	Determined by user defined
0x02	**Response Modbus Message:** Address Command Register	Structure of USINT USINT UINT	Get	Determined by user defined
0x03	Length of Response Modbus Message	UINT	Get/Set	0
0x04	Send User-defined Modbus Command	CHAR	Get/Set	Non-zero: Send Command
0x05	Data Lose Counter	WORD	Get/Set	0

Instance Service

Service Code	Service name	Need
0x0E	Get_Attribute_Single	Required
0x10	Set_Attribute_Single	Required

5 The components of Assembly Object

5.1 Components of Assembly Object

The Assembly Object binds the attributes of multiple objects, which allows data transfer to or from each object to be sent or received over a single connection. The I-7242D provides many assembly objects for users. The I/O type of Modbus devices is decided for the number of assembly objects. Every I/O devices represents an application object instance. The I-7242D would arrange the application instances in order by those devices that you set. Assembly object instances consist of these application object attributes. Figure 5-1 shows the architecture of the assembly object in I-7242D.

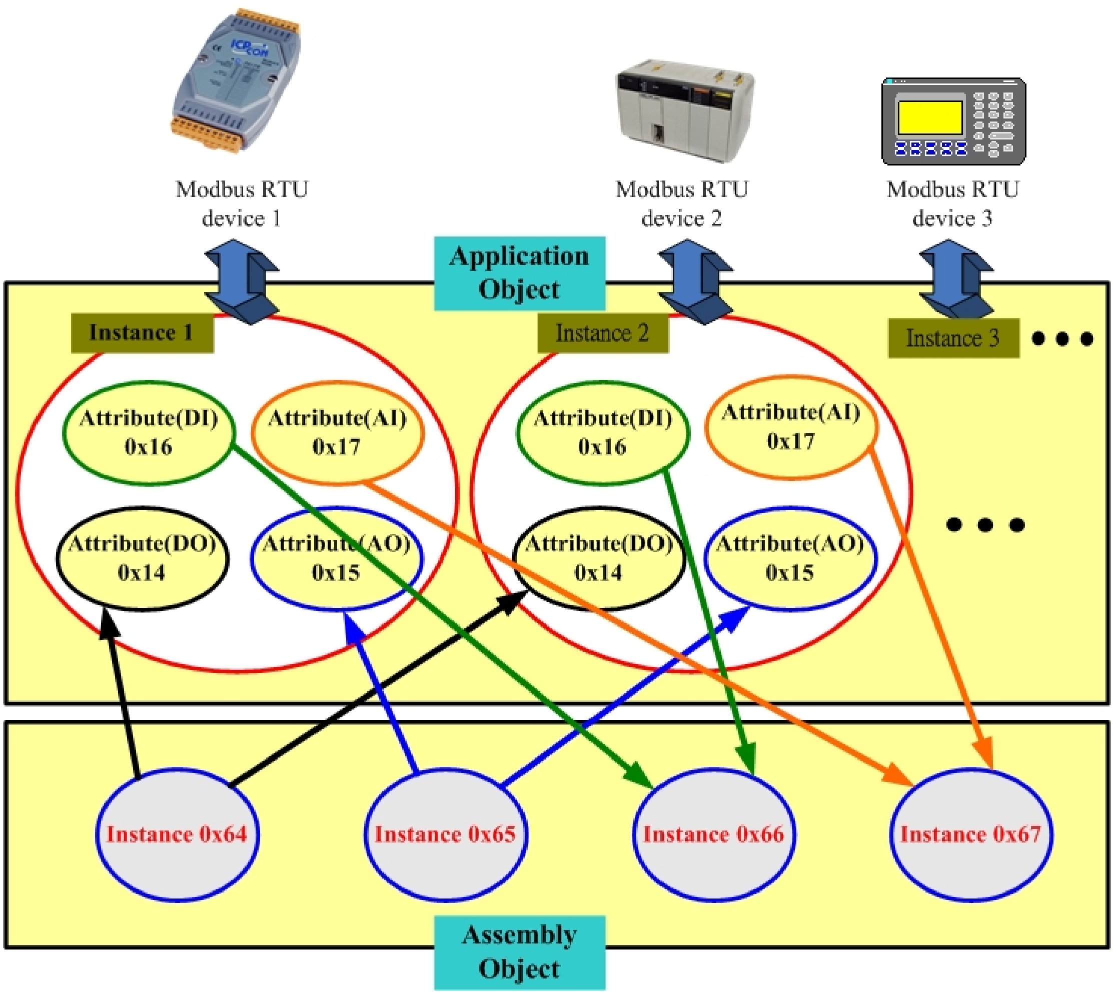


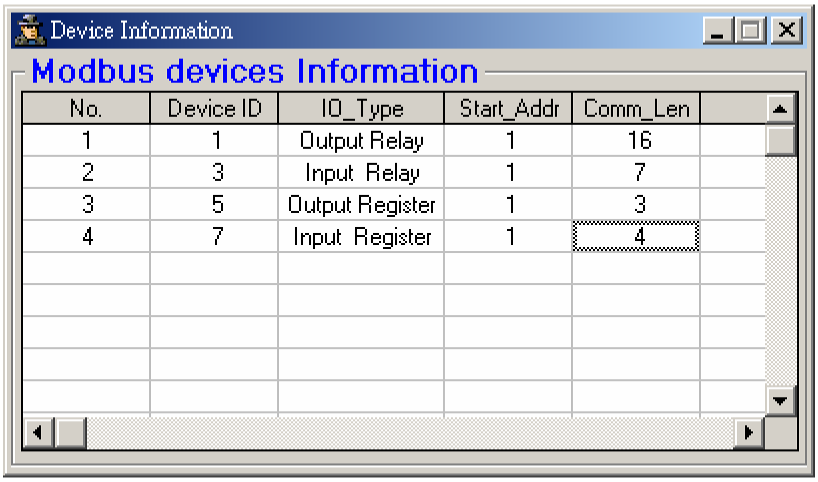
Figure 5-1 Architecture of assembly object in I-7242D

5.2 Examples of Assembly Object in I-7242D

There are many I/O examples related to the I-7242D in this section. These examples should help users more understand the usage of I-7242D.

Example 1: (one DO device, one DI device, one AO device, one AI device)

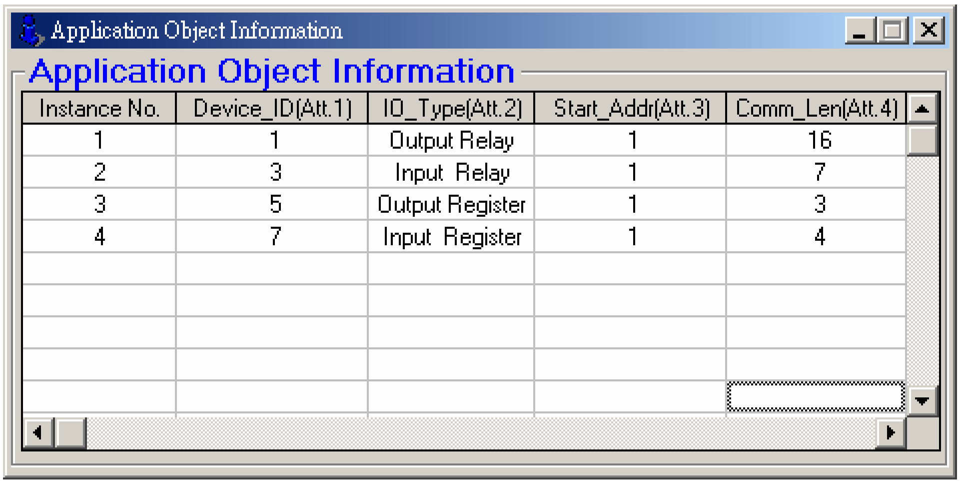
In this example, apply four Modbus devices in the system. Users can refer to the figure 6-2 to know the detail information from I-7242D to Modbus devices.



No.	Device ID	IO_Type	Start_Addr	Comm_Len
1	1	Output Relay	1	16
2	3	Input Relay	1	7
3	5	Output Register	1	3
4	7	Input Register	1	4

Figure 5-2 Communication parameters of Modbus devices

And, the application object information is showed as figure 5-3.



Instance No.	Device_ID(Att.1)	IO_Type(Att.2)	Start_Addr(Att.3)	Comm_Len(Att.4)
1	1	Output Relay	1	16
2	3	Input Relay	1	7
3	5	Output Register	1	3
4	7	Input Register	1	4

Figure 5-3 Information about application objects

Besides, the utility tool also shows the assembly object information as figure 5-4.

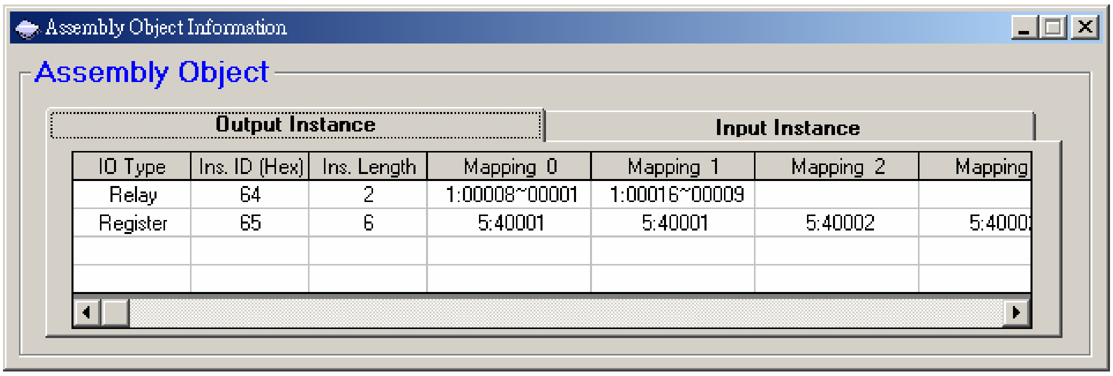


Figure 5-4 Information about Assembly instances

The I-7242D would arrange the application objects as table 5-1.

Table 5-1 Application object instances in the I-7242D

Application Instance ID	Device Address	Device I/O Type	Relay/Register Start Address	Relay/Register Data Length
0x01	1	0 (DO)	1	16
0x02	3	2 (DI)	1	7
0x03	5	1 (AO)	1	3
0x04	7	3 (AI)	1	4

According to the application object instances, I-7242D would arranges the assembly object instances as table 5-2.

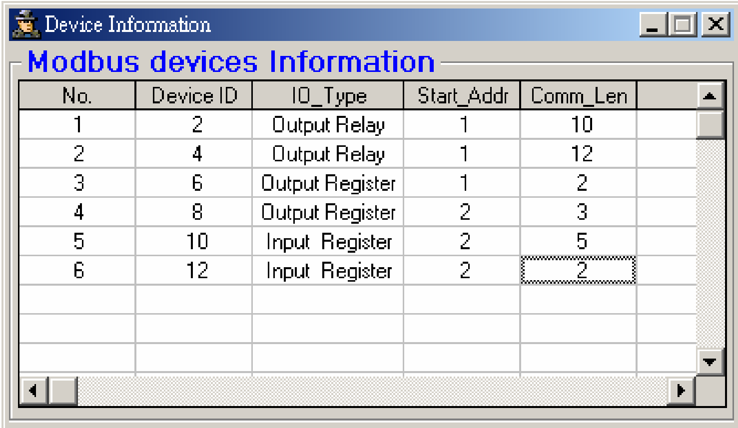
Table 5-2 Assembly object instances in the I-7242D

Assembly Object Instance ID	Data Length (Byte)	Component devices (ID, Address)
0x64	DO: 2	1(00016~00001)
0x65	AO: 6	2(40001~40003)
0x66	DI: 1	3(10007~10001)
0x67	AI: 8	4(30001~30004)

Note: The maximum number of assembly object is four. And the beginning number of assembly object instance ID is 0x64. And these instance IDs are ranked in the form of DO, AO, DI and AI types.

Example 2: (two DO devices, two AO devices, two AI devices)

In this example, apply six Modbus devices in the system (see Figure 5-5). The I-7242D would arrange these assembly and application instance Figure 5-5 and table 5-3.



No.	Device ID	IO_Type	Start_Addr	Comm_Len
1	2	Output Relay	1	10
2	4	Output Relay	1	12
3	6	Output Register	1	2
4	8	Output Register	2	3
5	10	Input Register	2	5
6	12	Input Register	2	2

Figure 5-5 Communication parameters of Modbus devices

Table 5-3 the information of the Application instances

Application Instance ID	Device Address	Device I/O Type	Relay/Register Start Address	Relay/Register Data Length
0x01	2	0 (DO)	1	10
0x02	4	0 (DO)	1	12
0x03	6	1 (AO)	1	2
0x04	8	1 (AO)	2	3
0x05	10	3 (AI)	2	5
0x06	12	3 (AI)	2	2

Table 5-4 the information of the Assembly instances

Assembly Object Instance ID	Data Length (Byte)	Component devices (ID, Address)
0x64	DO: 4	2(00010~00001), 4(00012~00001)
0x65	AO: 10	6(40001~40002), 8(40002~40004)
0x66	AI: 14	10(30002~30006). 12(30002~30003)

Note: This example lacks of DI devices. Therefore, the number of assembly object is three. If this demo has DI device, the number of assembly objects would become to four. The instance ID of DI components becomes 0x66 and the instance ID of AI components would change to 0x67.

Example 3: (two DO devices, two DI devices)

In the example, apply four Modbus devices in the system (see Figure 5-6). The I-7242D would arrange these assembly and application instances as Figure 5-6 and table 5-5.

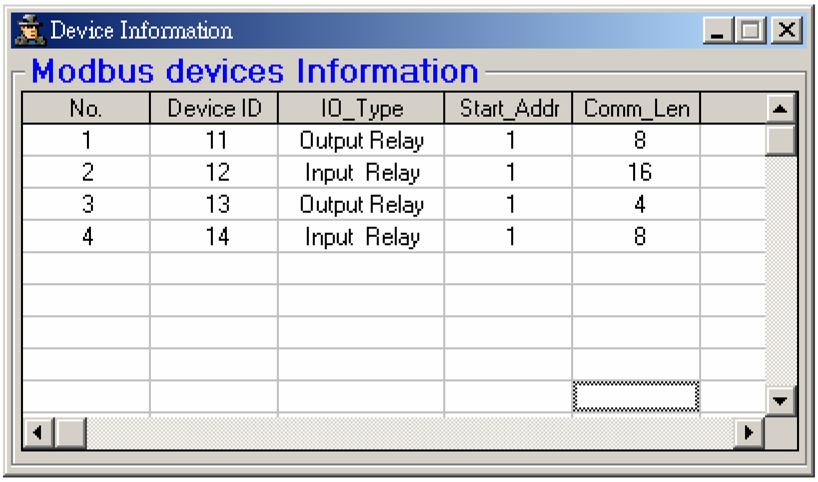


Figure 5-6 Communication parameters of Modbus devices

Table 5-5 the information of the Application instances

Application Instance ID	Device Address	Device I/O Type	Relay/Register Start Address	Relay/Register Data Length
0x01	11	0 (DO)	1	8
0x02	12	0 (DO)	1	16
0x03	13	2 (DI)	1	4
0x04	14	2 (DI)	1	8

Table 5-6 the information of the Assembly instances

Assembly Object Instance ID	Data Length (Byte)	Component devices (ID, Address)
0x64	DO: 2	11(00008~00001), 13(00004~00001)
0x65	DI: 3	12(10016~10001), 14(10008~10001)

Note: This example lacks of AO and AI devices. Therefore, the number of assembly object is two. If this example has AO device, the number of assembly objects would become to three. The instance ID of AO components becomes 0x65 and the instance ID of DI components would change to 0x66.

Example 4: (three AI devices)

In the example, apply three Modbus devices in the system (see Figure 5-7). The I-7242D would arrange these assembly and application objects as Table 5-7 and 5-8.

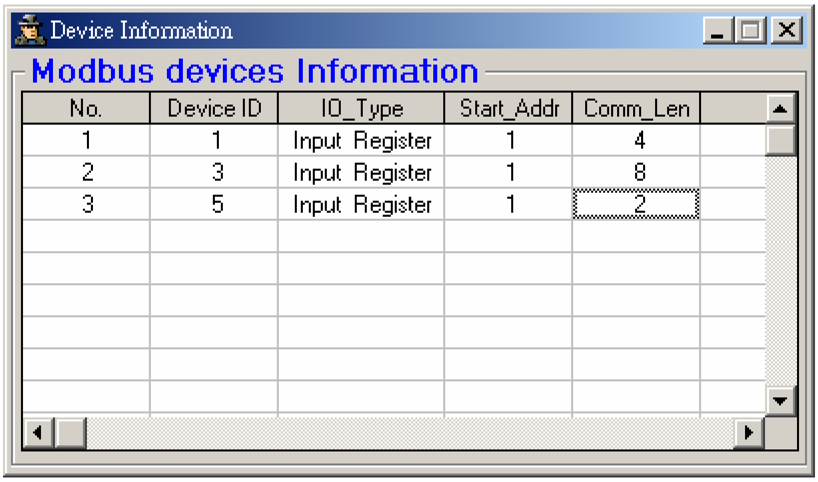


Figure 5-7 Communication parameters of Modbus devices

Table 5-7 Application object attribute

Application Instance ID	Device Address	Device I/O Type	Relay/Register Start Address	Relay/Register Data Length
0x01	1	3 (AI)	1	4
0x02	2	3 (AI)	1	8
0x03	3	3 (AI)	1	2

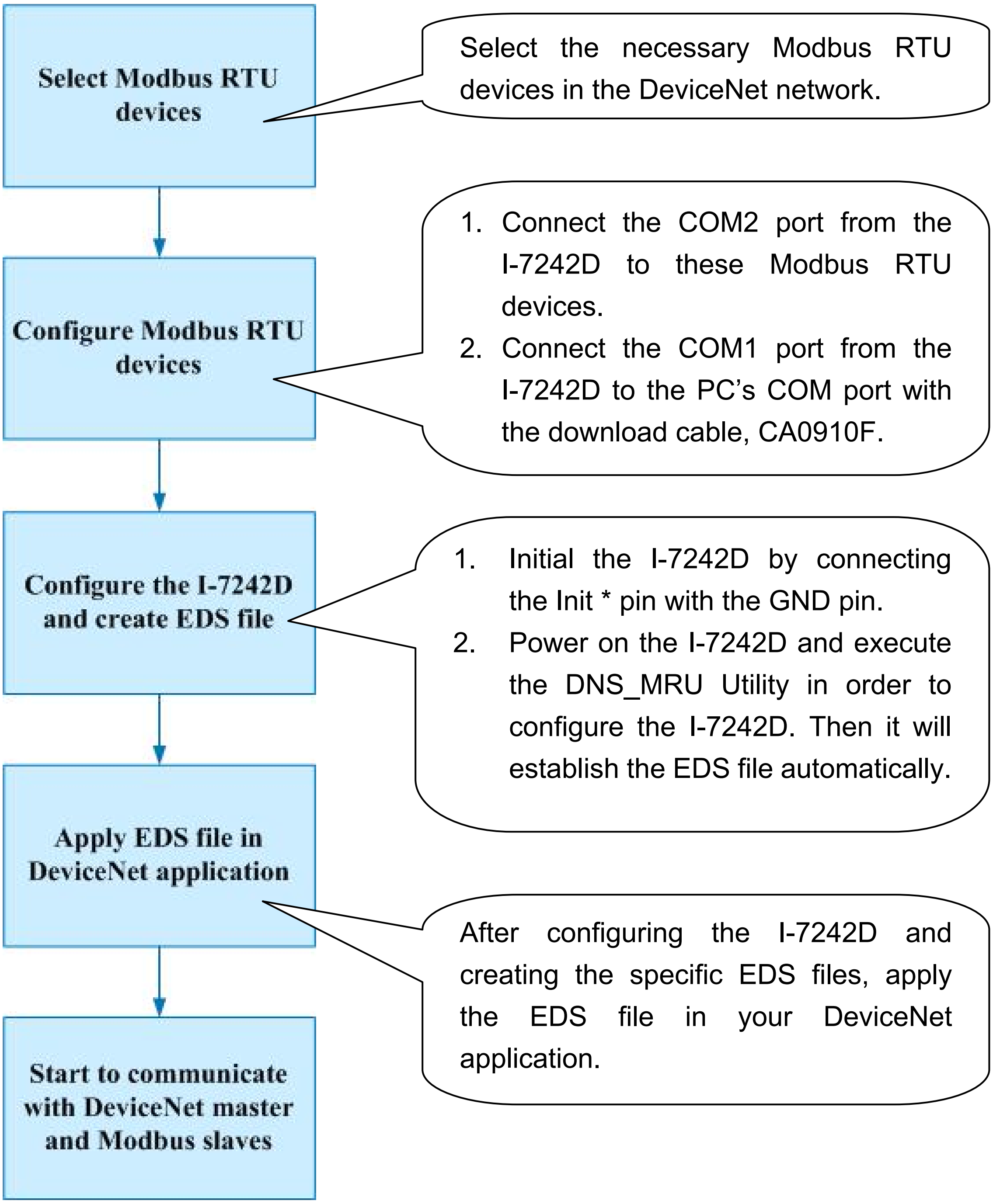
Table 5-8 Assembly object attribute

Assembly Object Instance ID	Data Length (Byte)	Component devices (ID, Address)
0x64	AI: 28	1(30001~30004), 3(30001~30008), 5(30001~30002)

Note: This example lacks of DO, AO and DI devices. Therefore, the number of assembly object is one. Therefore, the instance ID of assembly object is 0x64.

6 Configuration & Getting Started

6.1 Configuration Flowchart



6.2 The DNS_MRU Utility Overview

Before users apply the I-7242D into the DeviceNet application, they must understand the relationship between these DeviceNet application and assembly objects in the I-7242D. ICP DAS provides the DNS_MRU Utility to configure the communication parameters, I/O connection path and the EDS file for the I-7242D device. The software also provides the information of assembly objects, application objects and communication parameters that they set.

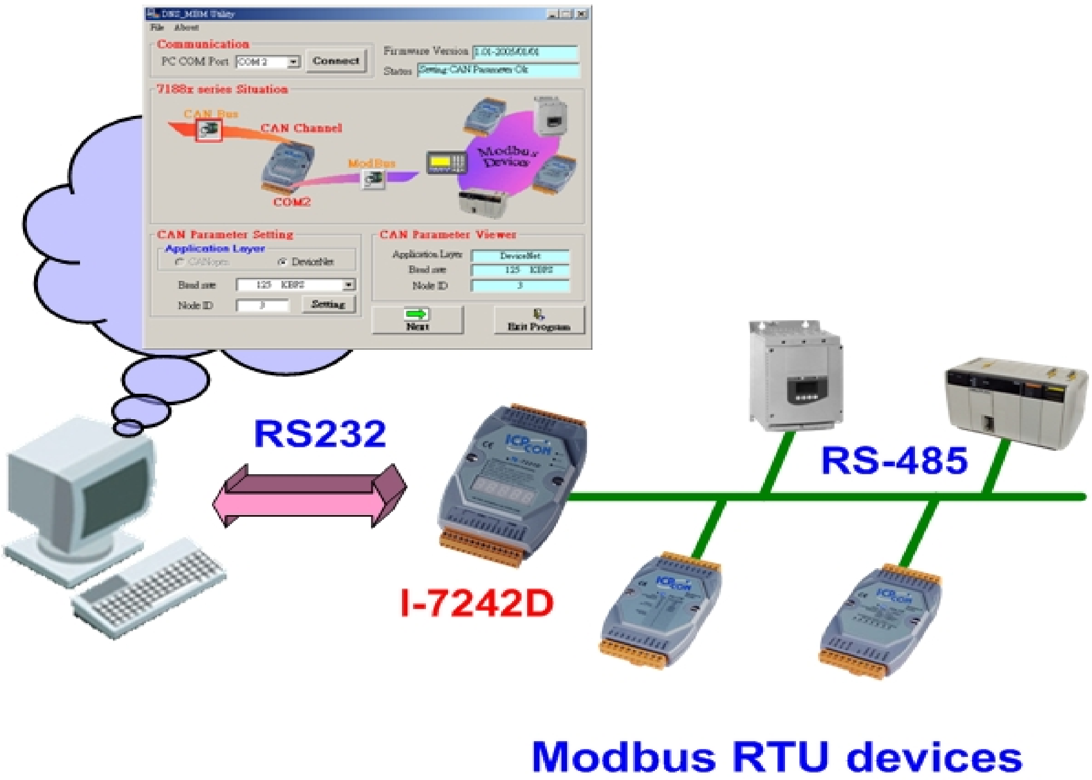


Figure 6-1 Architecture of the DNS_MRU Utility

6.3 Install & Uninstall the DNS_MRU Utility

Install DNS_MRU Utility

Step1: Download the DNS_MRU Utility setup file from the web site ftp://ftp.icpdas.com.tw/pub/cd/can_cd/devicenet/gateway/i-7242d/utility/ or the CD-ROM disk following the path of "/ CAN-CD / DeviceNet / Gateway / I-7242D / Utility /

Step 2: Execute the setup.exe file to install DNS_MRU Utility.

Step 3: A "Welcome" window pops up to prompt user to begin the installation. See figure 6-2.

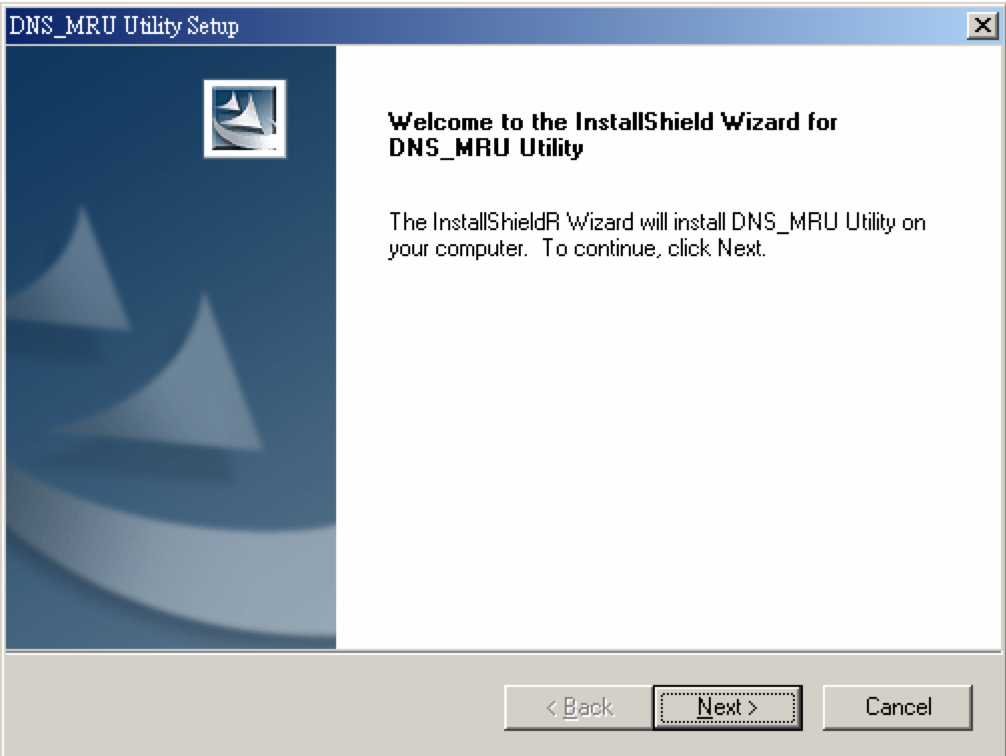


Figure 6-2. Welcome dialog

Step4: Click the "Next" button and A "Choose Destination Location" window will pop up for deciding the installation path.

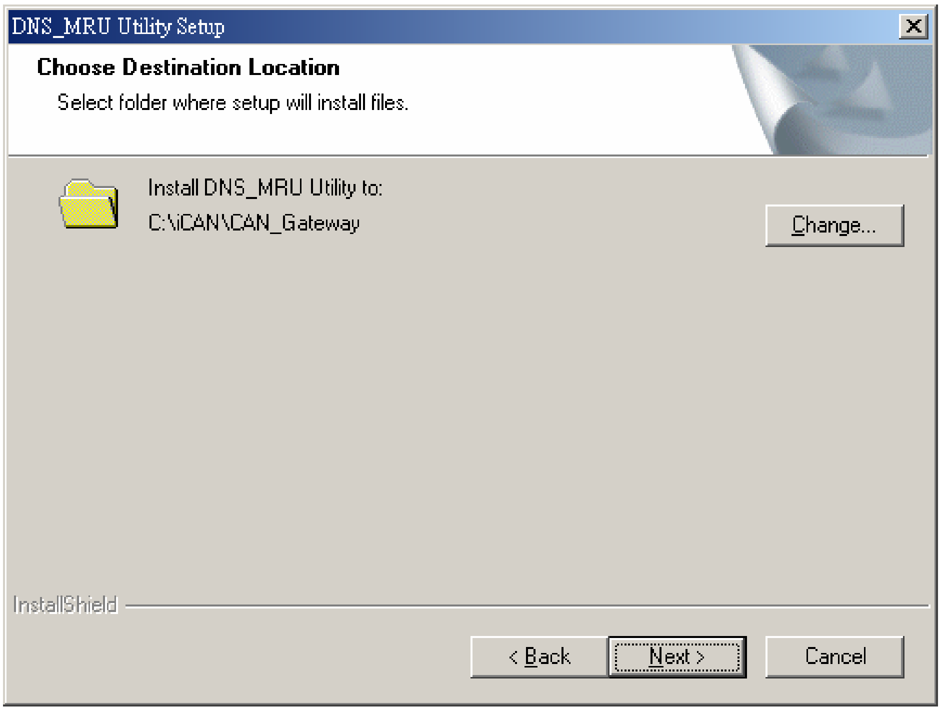


Figure 6-3. "Choose Destination Location" dialog

Step 5: Click "Next" button and a "Ready to Install the Program" window will pop up to prompt user that the wizard is ready to begin the installation See figure 6-4.

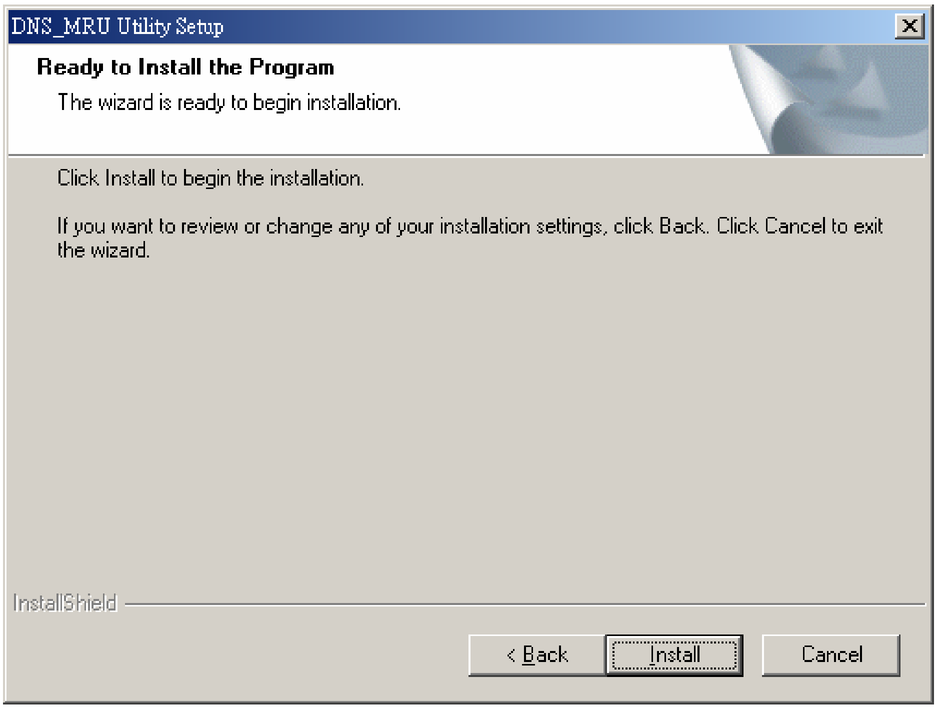


Figure 6-4. "Ready to Install the Program" dialog

Step 6: Click "Install" button and start to install the DNS_MRU Utility to the system. After finishing the process, a "Complete" window will pop up to prompt users that the successful completion of the installation. And click "Finish" button to exit. See figure 6-5.

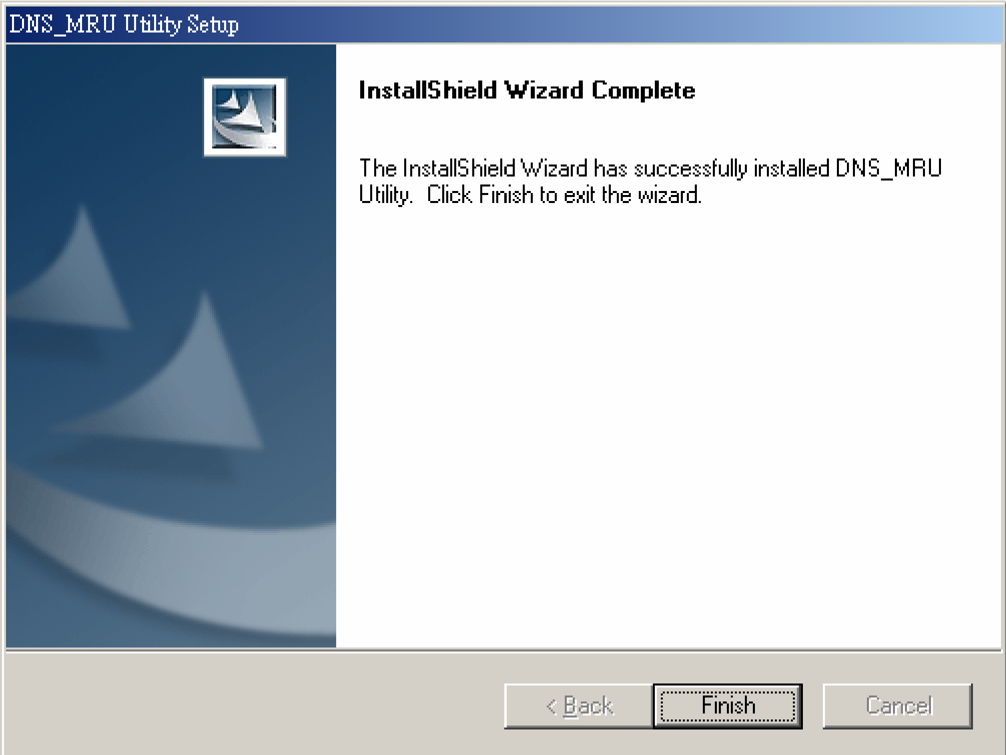


Figure 6-5. "Successful Completion of the Installation" dialog

Step 7: After finishing the installation of the DNS_MRU Utility, users can find DNS_MRU Utility as shown in the figure 6-6.

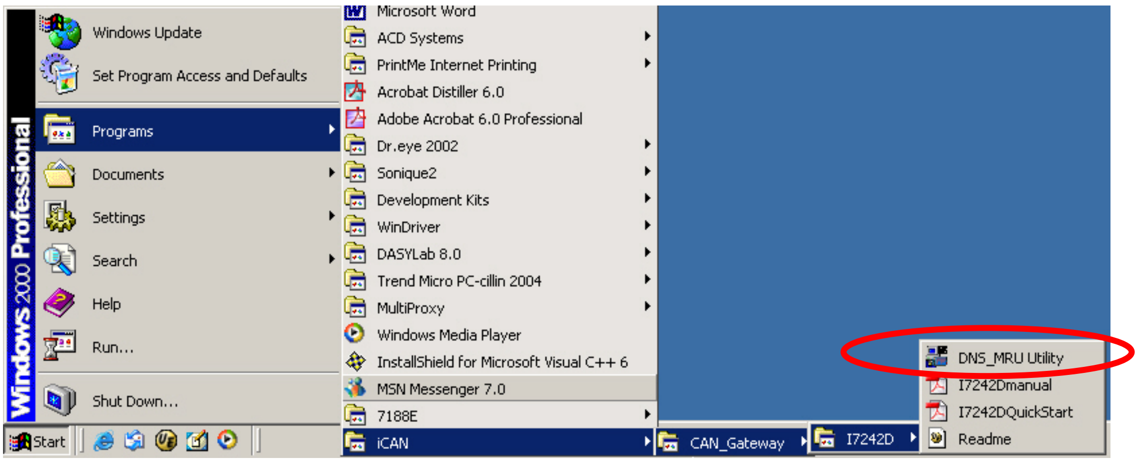


Figure 6-6. You can find "DNS_MRU Utility" at "Start" in the task bar

Uninstall DNS_MRU Utility

You can uninstall DNS_MRU Utility software by the following means described below:

Step 1: Click "Start" in the task bar, then click Settings/Control Panel as shown in figure 6-7.

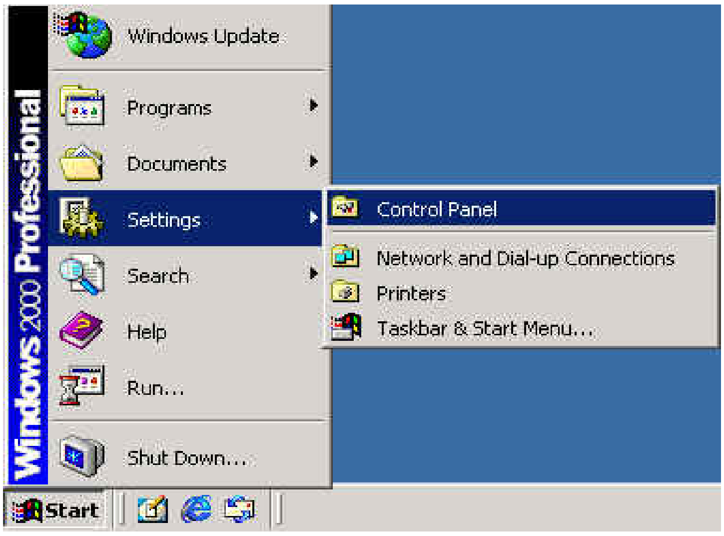


Figure 6-7. Select settings

Step 2: Click the "Add/Remove Programs" button icon to open the dialog. See figure 6-8.

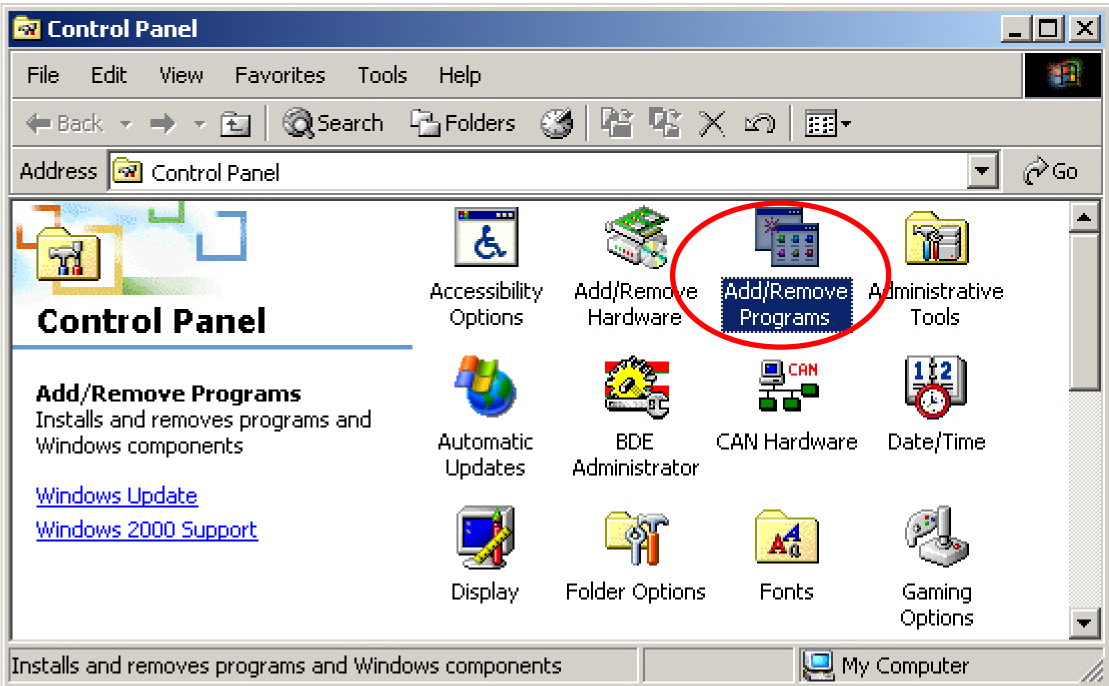


Figure 6-8 "Add/Remove Programs"

Step 3: Find out the DNS_MRU Utility, and click the Change/Remove button. See figure 6-9.

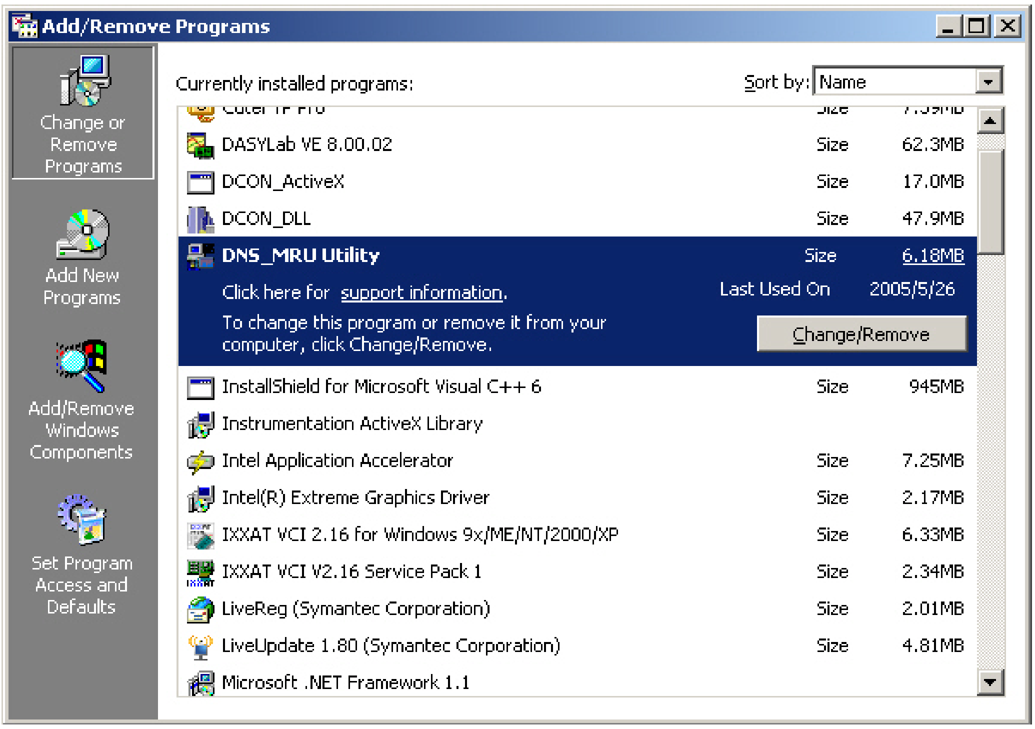


Figure 6-9. Click “Add/Remove Programs”

Step 4: Select the “Remove” option button, and press the “Next” button to remove DNS_MRU Utility. See figure 6-10.

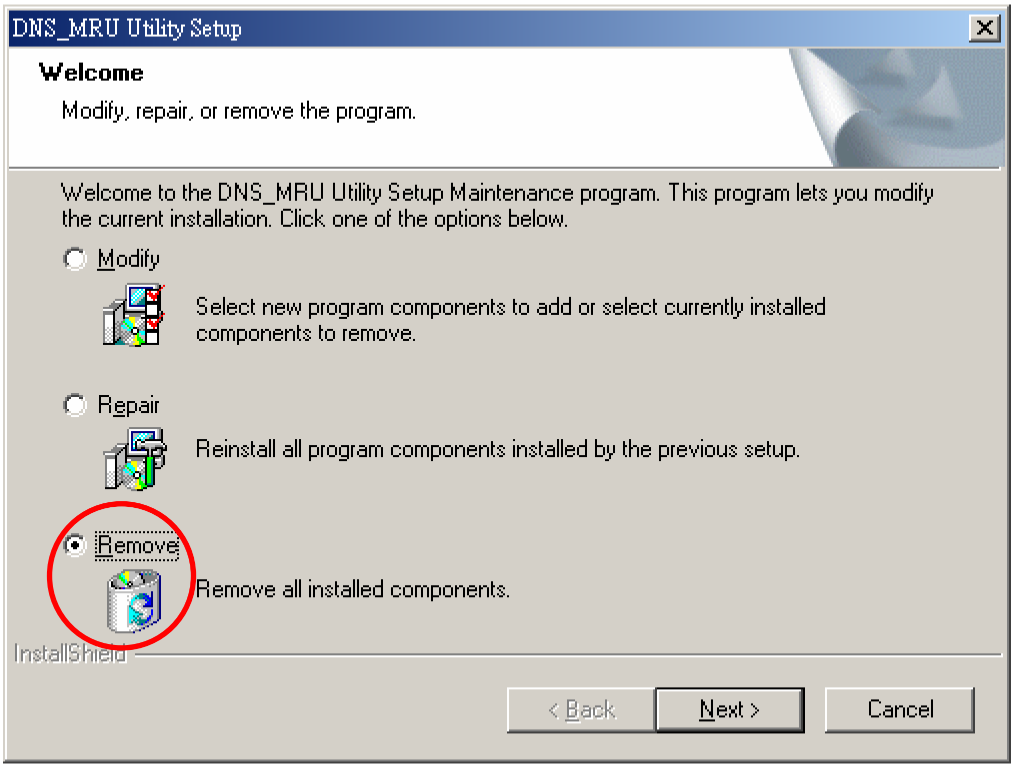


Figure 6-10 “Modify, repair, or remove the program” dialog

Step 5: Click the button “Yes” to remove the software as shown in figure 6-11.

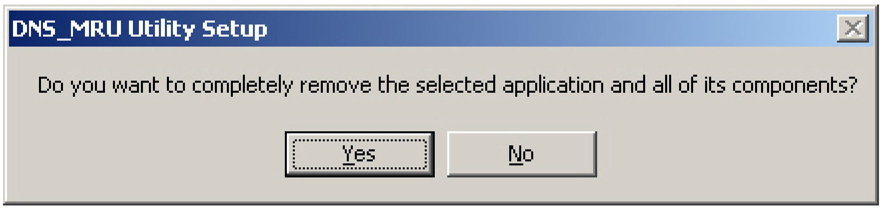


Figure 6-11. Click the button “Yes” to remove the software

Step 6: Removing DNS_MRU Utility.

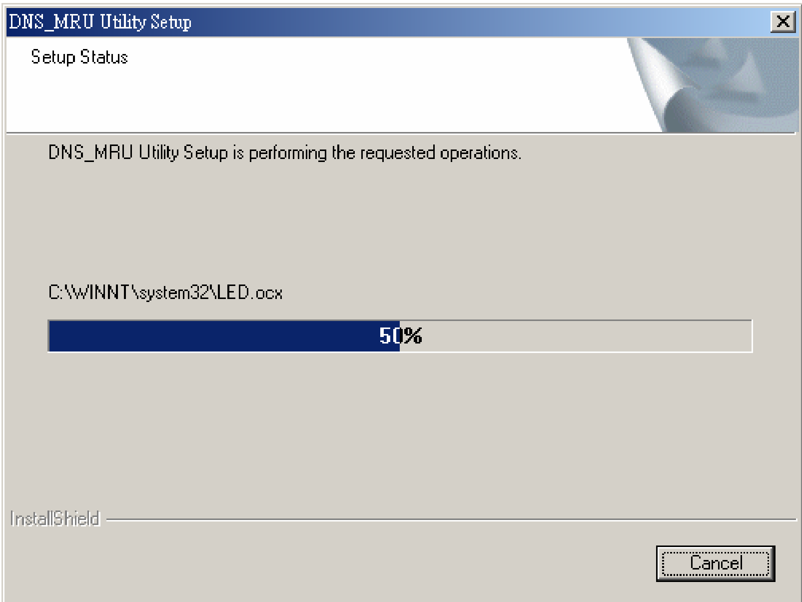


Figure 6-12. “Removing DNS_MRU Utility” dialog

Step 7: Finally, click the “Finish” button to finish the uninstall process.

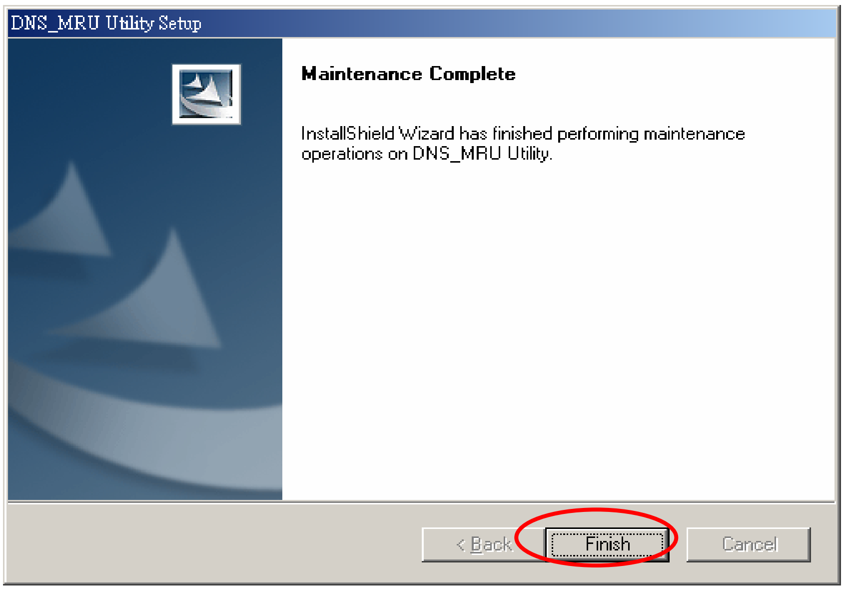


Figure 6-13. “Maintenance Complete” dialog.

6.4 Steps of the DNS_MRU Utility

Before using the DNS_MRU Utility software, please make sure that you have connected the I-7242D to your PC. The communication parameters of Modbus RTU devices are setting in offline connection mode. After setting up the I-7242D, it will start to communicate with the Modbus RTU devices that you set. The architecture is depicted in the following figure 6-14.

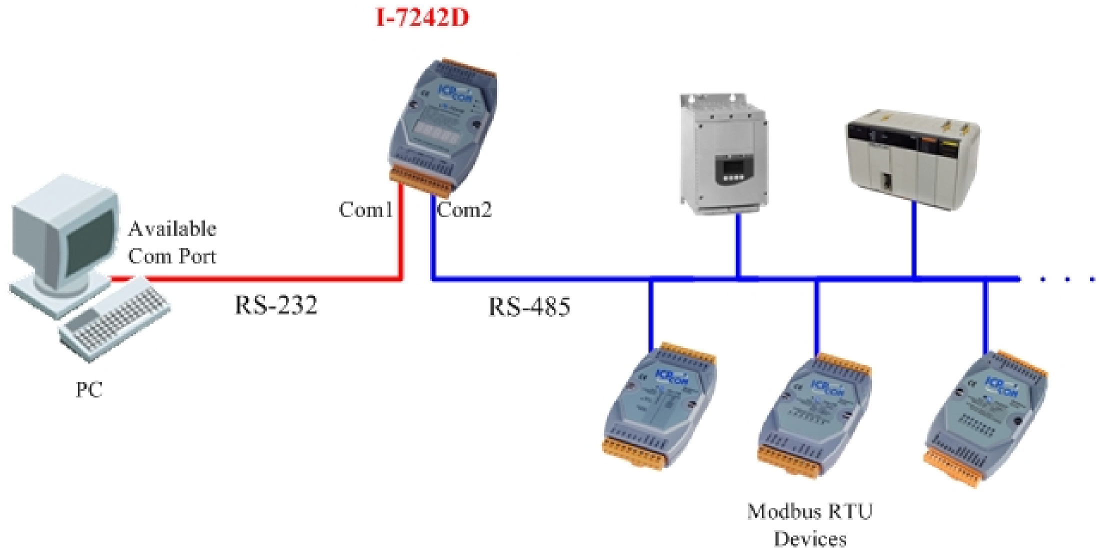


Figure 6-14. Configuration architecture of the I-7242D

Step 1: Before you use this software, turn off the I-7242D. Connect the INIT* pin with the GND pin of the I-7242D as figure 6-15.

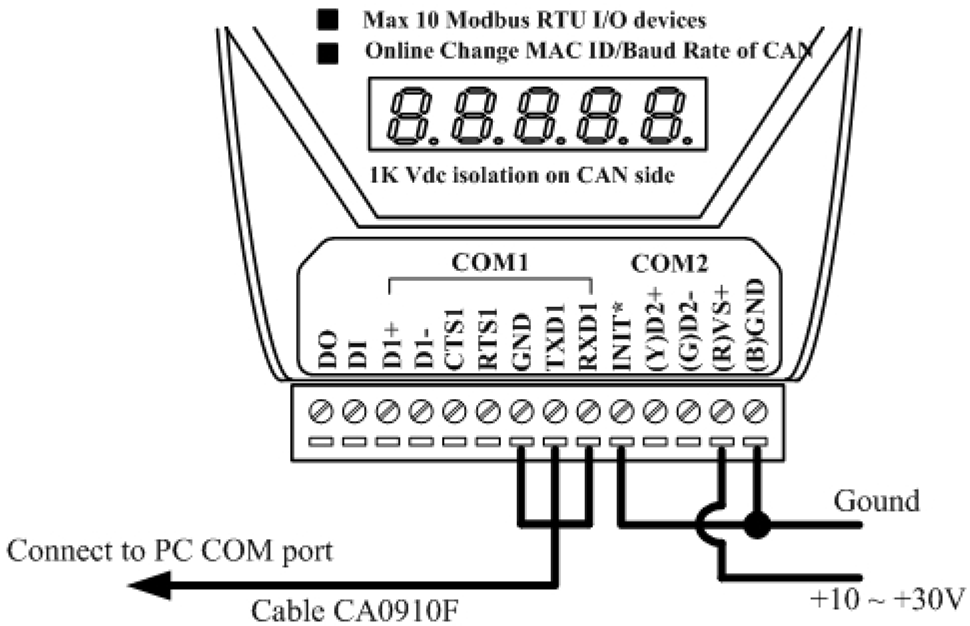


Figure 6-15. Wire Connection of the I-7242D

Step 2: Turn on the I-7242D. And then execute the DNS_MRU.exe file. The start-up figure would be displayed as figure 6-16.

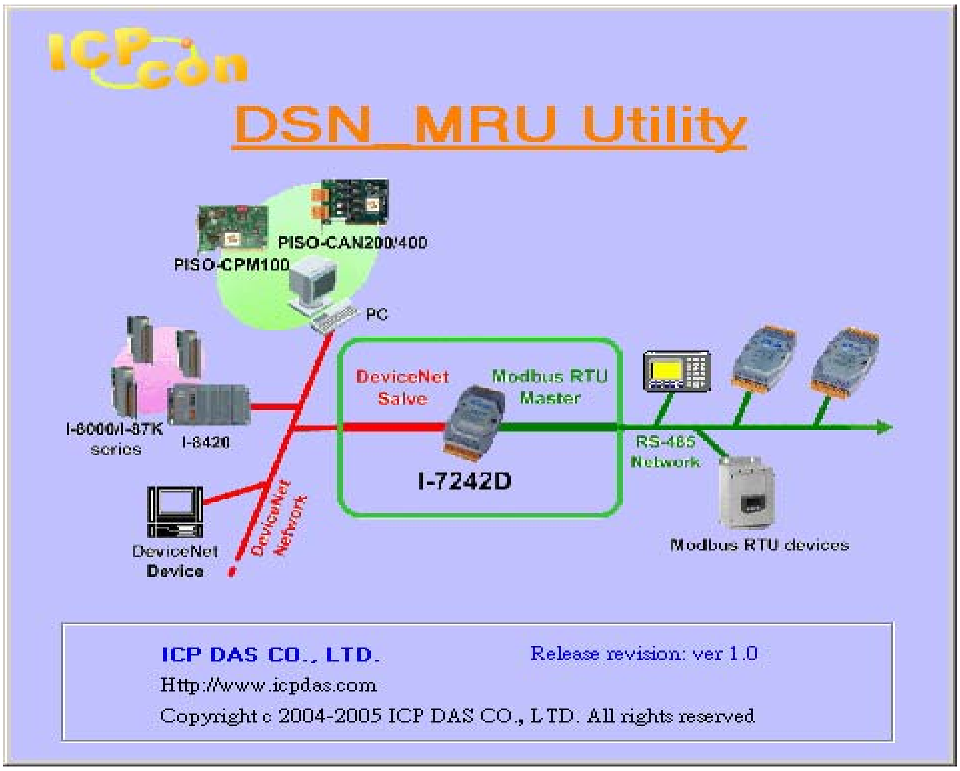


Figure 6-16. “Start-Up”

After the start-up figure, the frame would be displayed as figure 6-17.

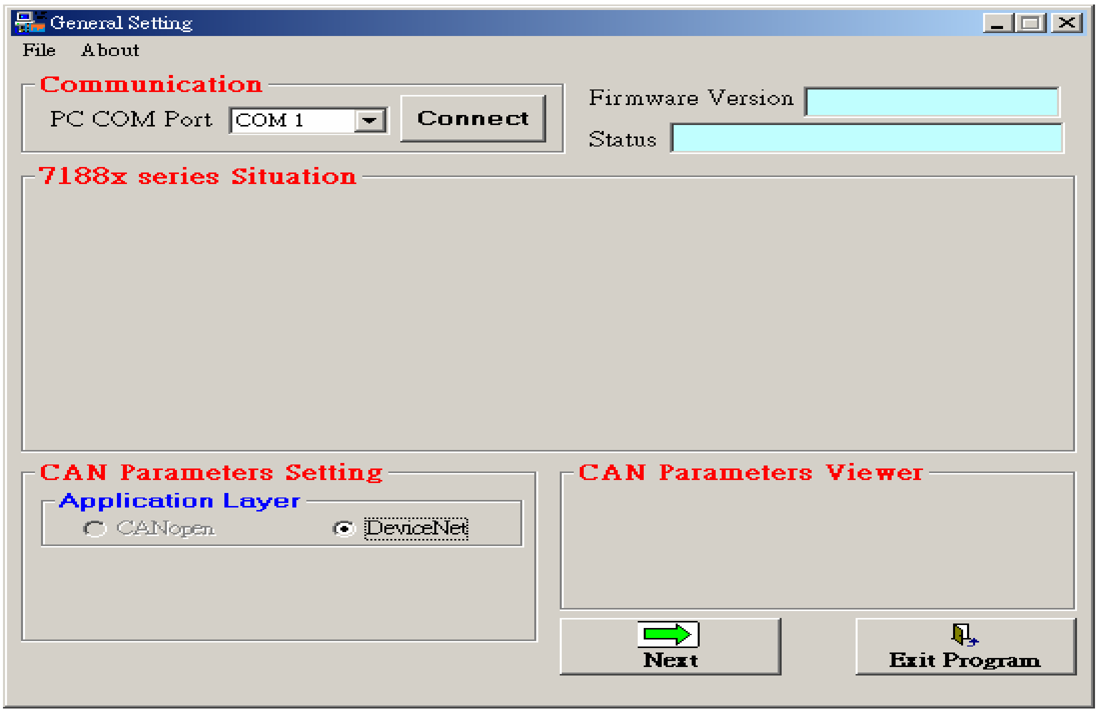


Figure 6-17. “General Setting”

Step 3: Select PC's COM port correctly. Then press the "Connect" button, 

, to connect with the I-7242D. Then it would take few seconds to read the communication parameters stored in the I-7242D's EEPROM.

Step 4: After read the parameters stored in I-7242D, these parameters will be verified that they are correct or not. If any error has been detected, the warning message will be pop-up as figure 6-18.

Figure 6-18. "EEPROM Data Error Dialog Box"

In this case, if any error has been detected, the default value will be shown on each parameter setting field. If no error or warning message occurs, the last setting value will be displayed on each parameter setting field.

Step 5: After reading parameters from EEPROM, a related information dialog box will be displayed as figure 6-19.

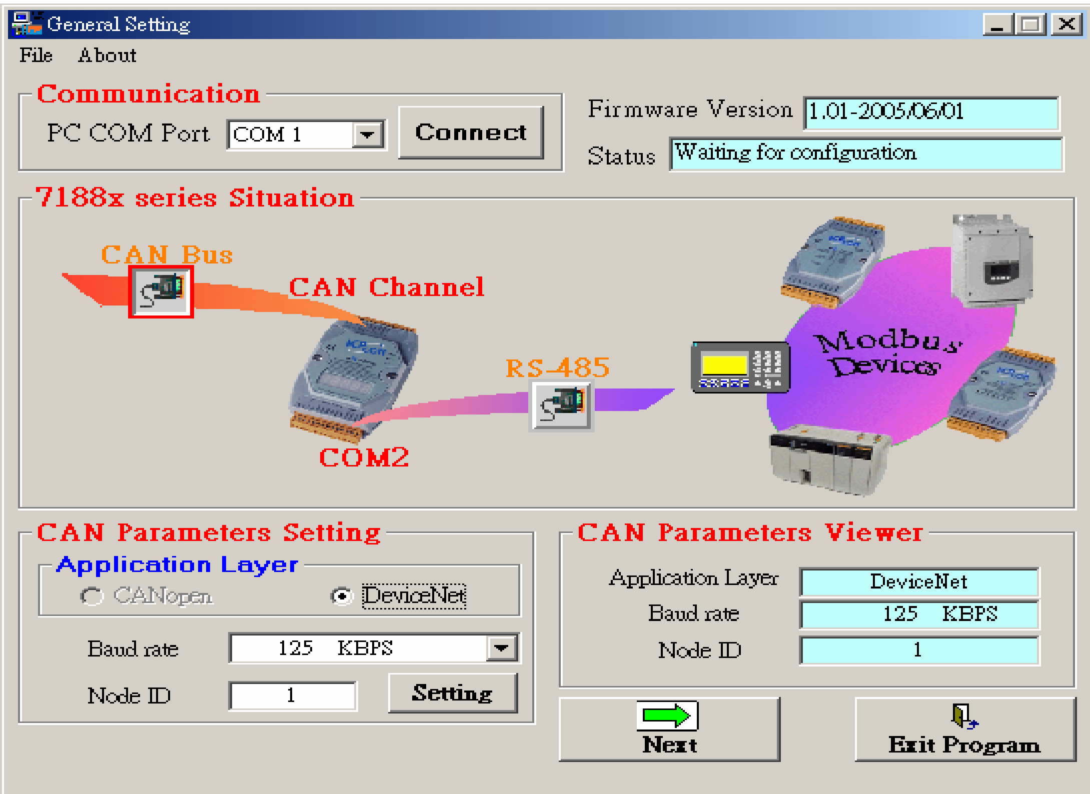
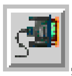


Figure 6-19. "General Setting"

Step 6: Click the "CAN Bus" button, , so that the CAN bus configuration information will be given. Then, users can set the necessary CAN bus communication information. Afterwards, click the "Setting" button to finish the CAN parameter setting. The CAN Parameter Viewer frame on the right hand side indicates the parameter setting result. After clicking the "Setting" button, users can see that the each field value of the CAN Parameter Viewer frame is changed to the value configured in the CAN Parameter Setting frame on the left hand side as figure 6-20.

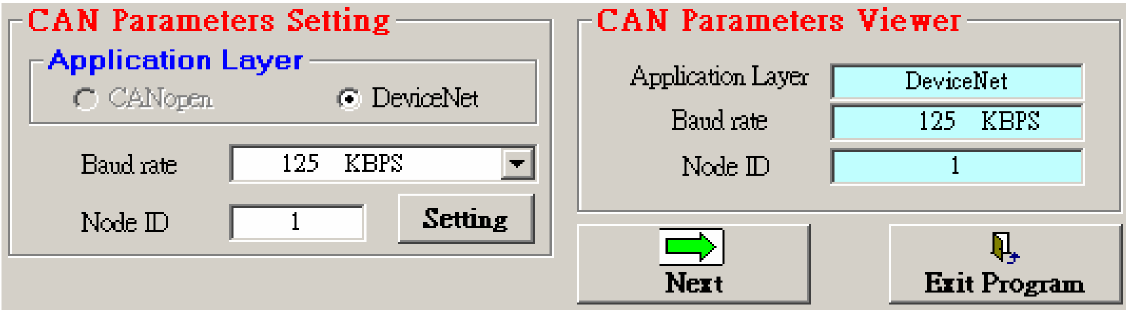
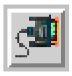


Figure 6-20. "CAN Parameter Setting & Viewer"

Step 7: Press the "RS-485" button, , to display the Com2 configuration information on the I-7242D. Press the "Setting" button to set the needed RS-485 communication information in the dialog box as figure 6-21.

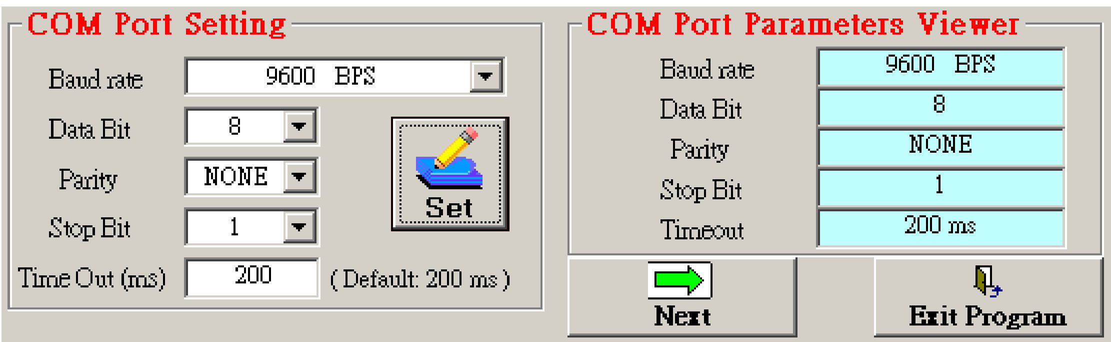


Figure 6-21. "Com Port Parameter Setting & Viewer"

Step 8: Press the “Next” button, Next, to display the “Application Object Setting” frame as figure 6-22. Users can use the add button, erase button, update button or delete button to modify their Modbus devices parameters that they want to use to communicate.

Add: Add a new Modbus device to application object.

Erase: Erase the Modbus device parameter that you set.

Update: Update the specific application object instance with the newer Modbus device parameter.

Delete: Delete the specific application object instantce.

And users can view the information of devices, application object and assembly object by clicking the Device View button, the Application Object View button and the Assembly Object View button. Then, these windows would pop-up as figure 6-23, figure 6-24 and figure 6-25.

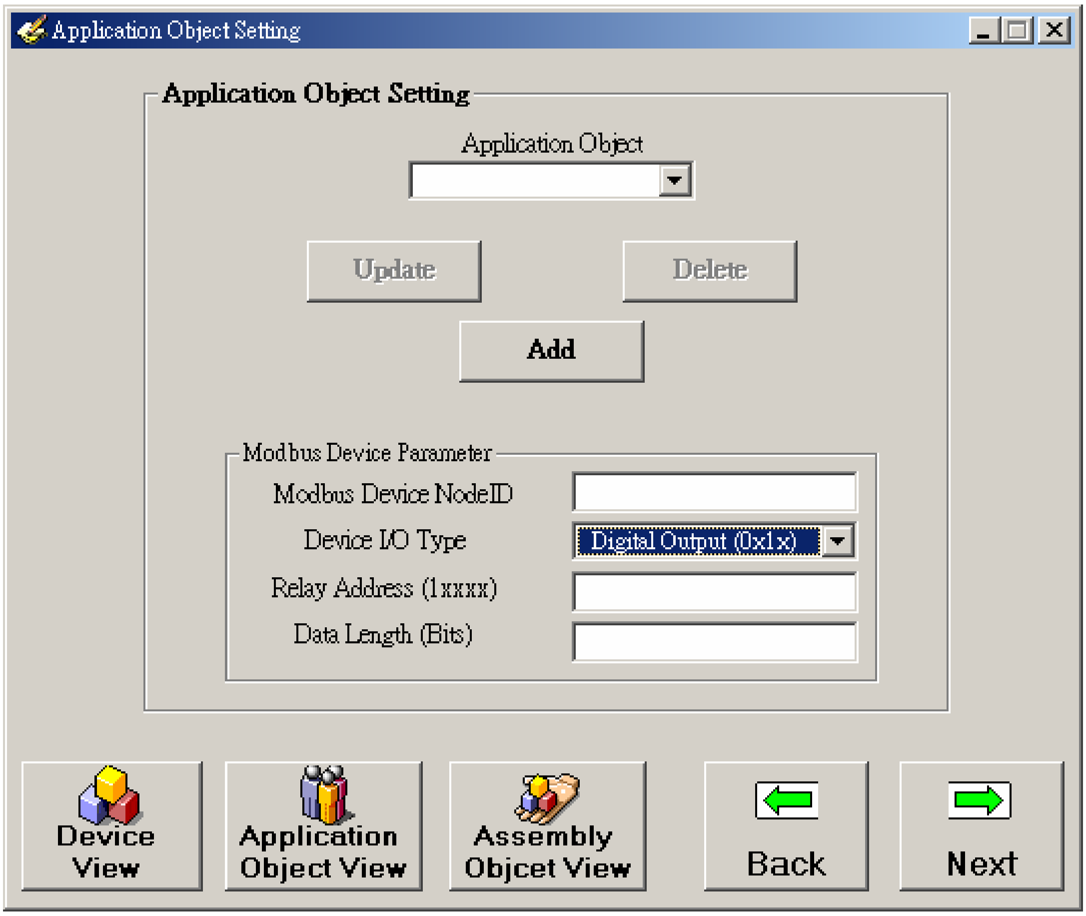


Figure 6-22. “Application instances Setting”

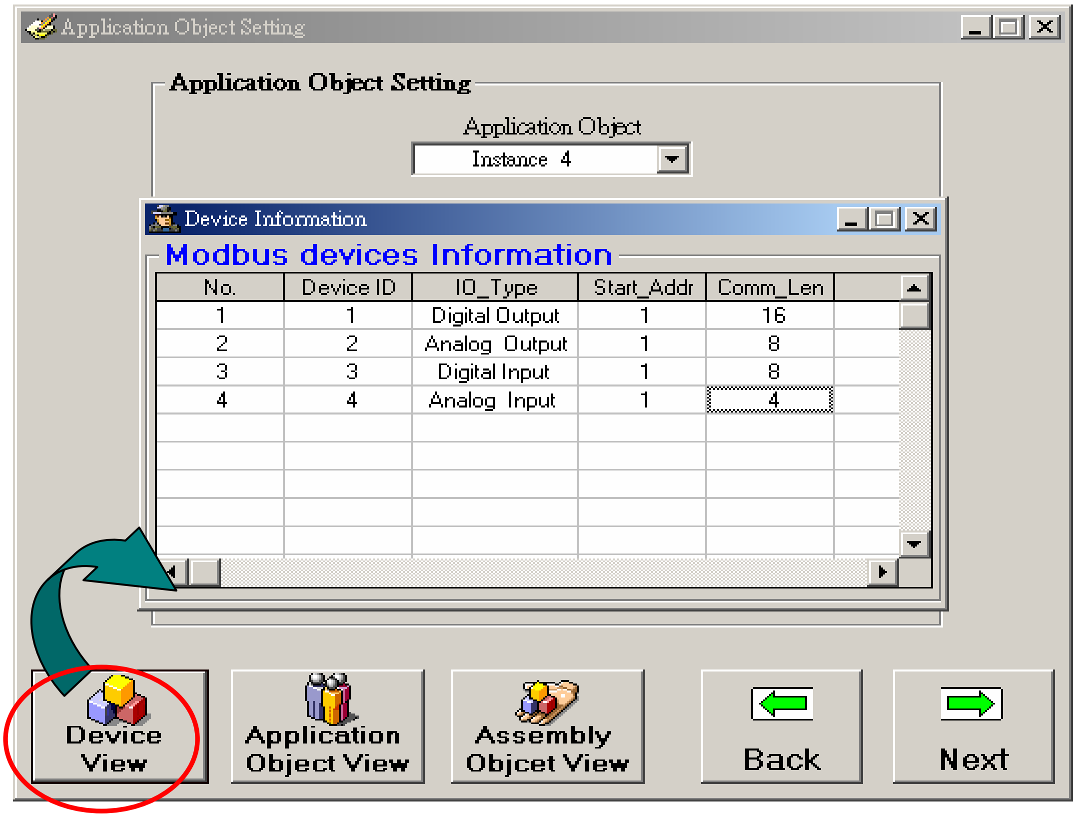


Figure 6-23. "Modbus Devices Information"

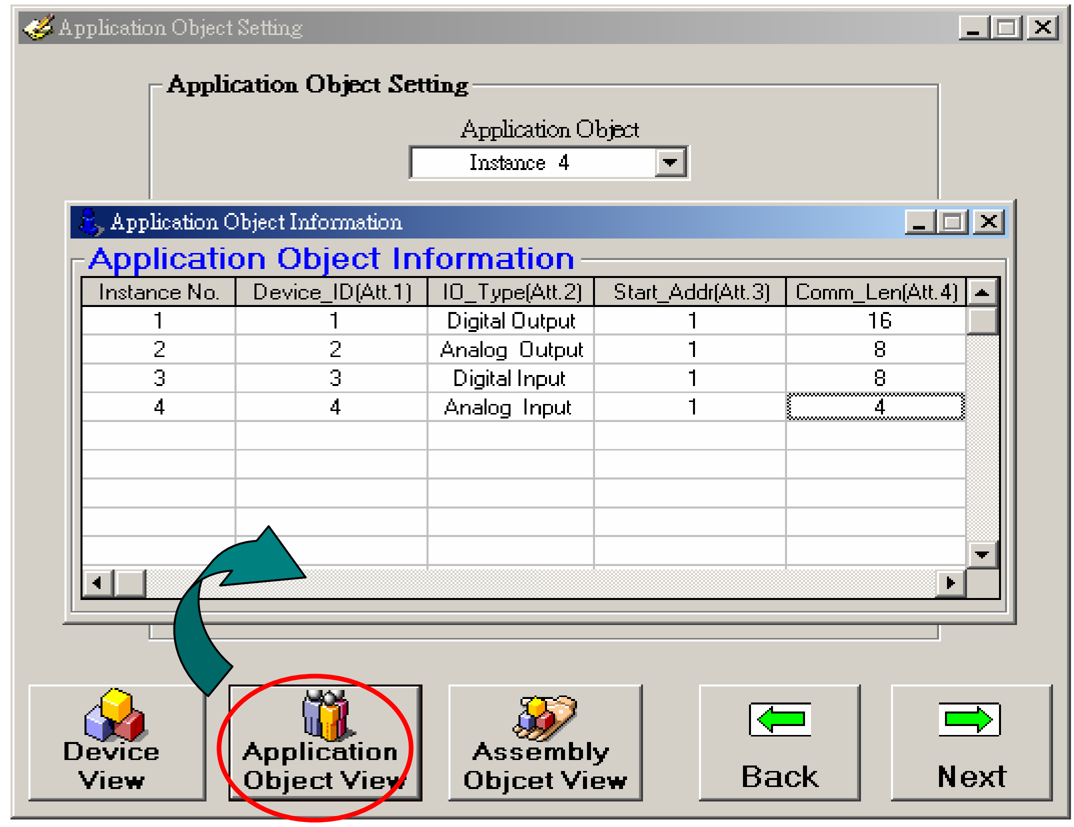


Figure 6-24. "Application Object Information"

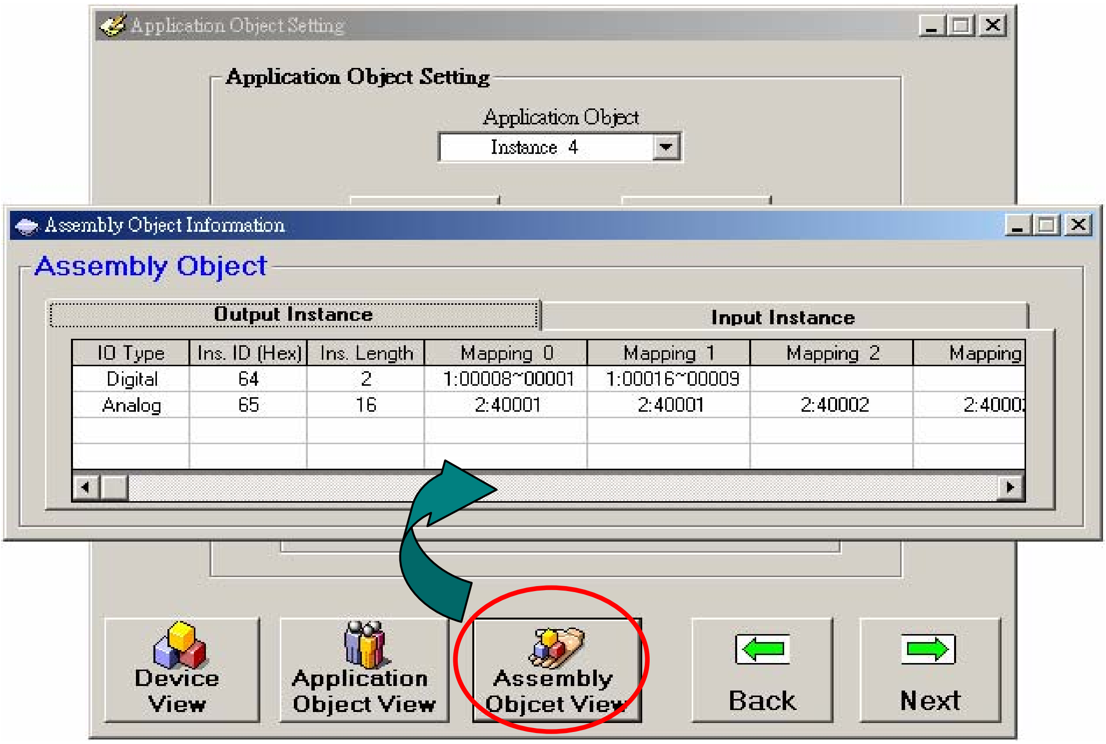
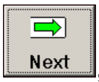


Figure 6-25. “Assembly Object Information”

Step 9: After the configuration of Modbus devices parameters, press the “Next” button, 

, and start to build the specific EDS file for the I-7242D. If the I/O connection path stored in EEPROM of I-7242D is not correct. Or you modify the parameters of Modbus devices. Then the warring dialog would be pop-up as figure 6-26.

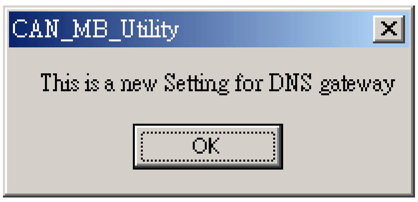


Figure 6-26. “Warning Dialog Box”

Step 10: Then the DeviceNet EDS file information is set according to the following frame. Users can configure the relative information for their EDS file by using a dialog box like figure 6-27.

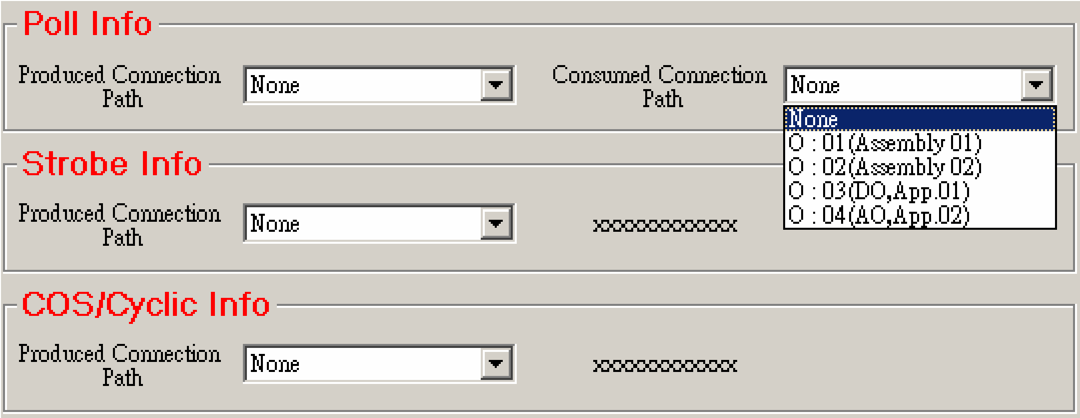


Figure 6-27. “DeviceNet EDS file Setting”

Step 11: Setting the EDS file information and describe it as figure 6-28.

Figure 6-28. “Description of EDS file”

Step 12: Set the Polling/Bit Strobe/COS/Cyclic I/O connection path for the I-7242D as figure 6-29.

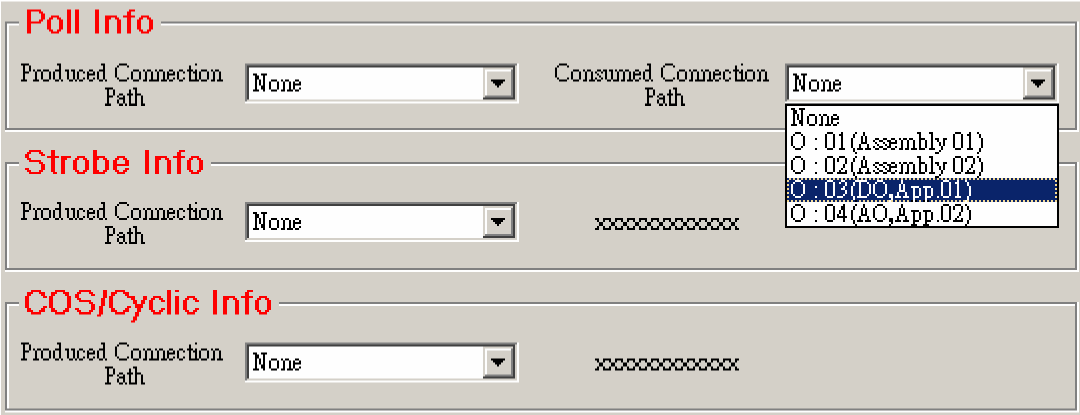


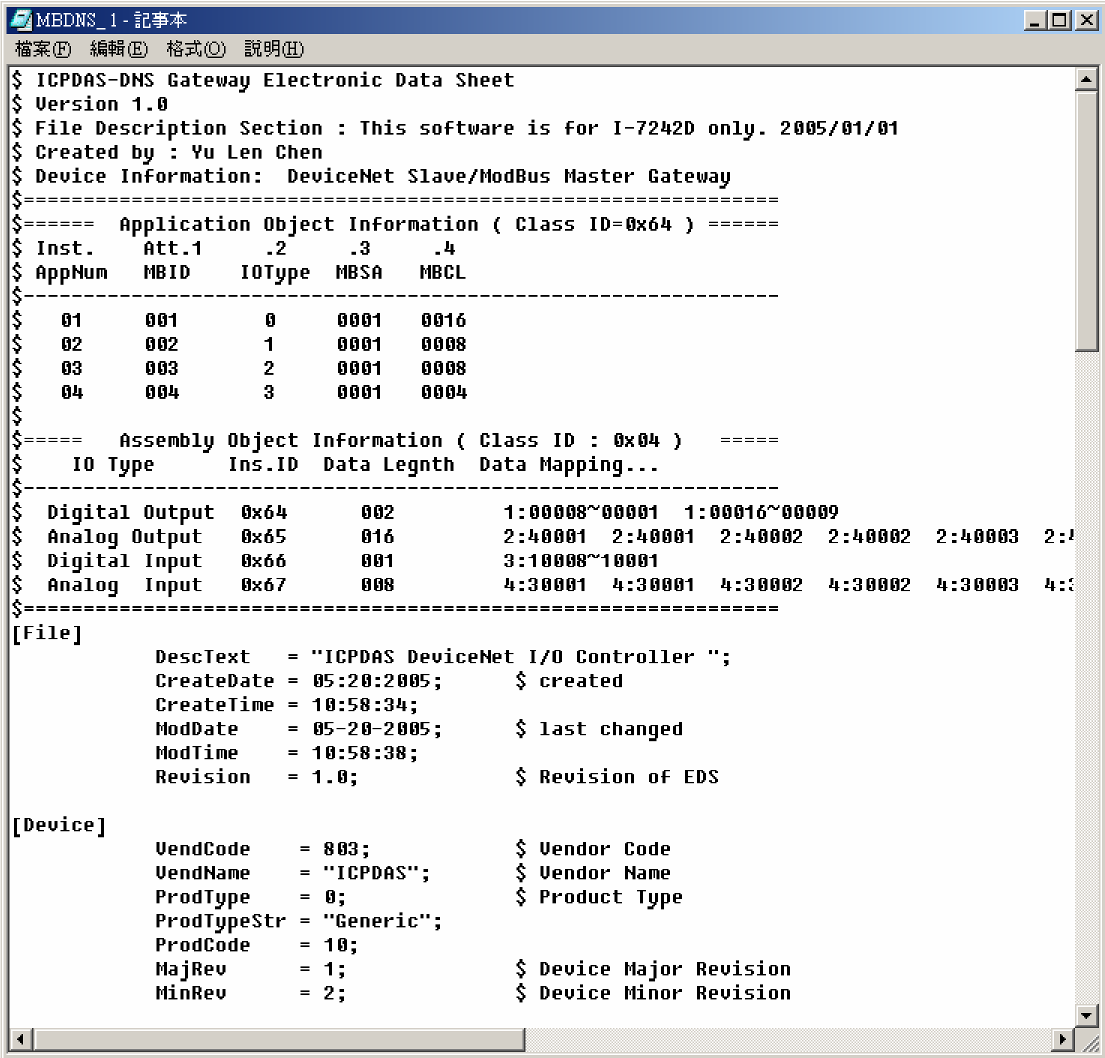
Figure 6-29. “Produced/Consumed I/O Connection Path”

Step 13: Click the “Finish” button to complete the I-7242D configuration and the DNS_MRU Utility will create the EDS file for users as figure 6-30.

Figure 6-30. “Finish and Create EDS File.”

Step 14: After click the “Finish” button, the main window would be pop-up. Then press the “Exit program” button to exit the program.

You can find the EDS file for the specific I-7242D. The file name is MBDNS_1.eds. “1” represents the Node ID that you set. Therefore, Users can apply the EDS file in the DeviceNet application as figure 6-31.




```
$ ICPDAS-DNS Gateway Electronic Data Sheet
$ Version 1.0
$ File Description Section : This software is for I-7242D only. 2005/01/01
$ Created by : Yu Len Chen
$ Device Information:  DeviceNet Slave/ModBus Master Gateway
$===============================================================
$======  Application Object Information ( Class ID=0x64 ) ======
$ Inst.    Att.1     .2     .3     .4
$ AppNum   MBID    IOType  MBSA   MBCL
$---------------------------------------------------------------
$   01      001       0      0001   0016
$   02      002       1      0001   0008
$   03      003       2      0001   0008
$   04      004       3      0001   0004
$
$=====   Assembly Object Information ( Class ID : 0x04 )   =====
$    IO Type      Ins.ID  Data Legnth  Data Mapping...
$---------------------------------------------------------------
$  Digital Output  0x64      002         1:00008~00001  1:00016~00009
$  Analog Output   0x65      016         2:40001  2:40001  2:40002  2:40002  2:40003  2:4
$  Digital Input   0x66      001         3:10008~10001
$  Analog  Input   0x67      008         4:30001  4:30001  4:30002  4:30002  4:30003  4:3
$===============================================================
[File]
            DescText   = "ICPDAS DeviceNet I/O Controller ";
            CreateDate = 05:20:2005;      $ created
            CreateTime = 10:58:34;
            ModDate    = 05-20-2005;      $ last changed
            ModTime    = 10:58:38;
            Revision   = 1.0;             $ Revision of EDS

[Device]
            VendCode    = 803;            $ Vendor Code
            VendName    = "ICPDAS";       $ Vendor Name
            ProdType    = 0;              $ Product Type
            ProdTypeStr = "Generic";
            ProdCode    = 10;
            MajRev      = 1;              $ Device Major Revision
            MinRev      = 2;              $ Device Minor Revision
```

Figure 6-31. The Part of the EDS file

Note: There is also some Modbus devices information in the EDS file. Users can see the information form the EDS file.

7 DeviceNet Communication Set

7.1 DeviceNet Communication Set Introduction

The I-7242D is a “Group 2 Only Slave” device, and supports the “Predefined Master/Slave Connection Set”. To communicate with this device, the process for establishing a connection is important. In addition, we provide some examples on how to access I/O devices.

The CAN Identifier Fields associated with the Predefined Master/Slave Connection Set for the I-7242D are given in the table 7-1. This table defines the Identifiers that are to be used with all the connection based messaging involved in the Predefined Master/Slave Connection Set for the I-7242D.

Table 7-1 CAN Identifier Fields for the I-7242D

IDENTIFIER BITS											**IDENTITYUSAGE**	**HEX RANGE**
10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
0	**Group 1 Message ID**				**Source MAC ID**						**Group 1 Messages**	**000 – 3ff**
0	1	1	0	1	Source MAC ID						Slave’s I/O Change of State or Cyclic Message	
0	1	1	1	0	Source MAC ID						Slave’s I/O Bit–Strobe Response Message	
0	1	1	1	1	Source MAC ID						Slave’s I/O Poll Response or Change of State/Cyclic Acknowledge Message	
1	**0**	**MAC ID**						**Group 2 Message ID**			**Group 2 Messages**	**400 – 5ff**
1	0	Source MAC ID						0	0	0	Master’s I/O Bit–Strobe Command Message	
1	0	Destination MAC ID						0	1	0	Master’s Change of State or Cyclic Acknowledge Message	
1	0	Source MAC ID						0	1	1	Slave’s Explicit/ Unconnected Response Messages/ Device Heartbeat Message/ Device Shutdown Message	
1	0	Destination MAC ID						1	0	0	Master’s Explicit Request Messages	
1	0	Destination MAC ID						1	0	1	Master’s I/O Poll Command/Change of State/Cyclic Message	
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages (reserved)	
1	0	Destination MAC ID						1	1	1	Duplicate MAC ID Check Messages	
1	**1**	**1**	**1**	**1**	**Group 4 Message ID**						**Group 4 Messages**	**000 – 3ff**
1	1	1	1	1	2C						Communication Faulted Response Message	
1	1	1	1	1	2D						Communication Faulted Request Message	

Table 7-2 lists the Error Codes that may be present in the General Error Code field of an Error Response message.

Table 7-2 General error codes

Error Condition	General Error code (Hex)	Additional Error Condition	Additional Error Code (Hex)
Resource unavailable	02	Invalid allocation choice	02
		Invalid Unconnected request	03
		Poll After COS_CYCLIC	04
Service not support	08	None	FF
Invalid attribute value	09	None	FF
Already in requested mode/state	0B	None	FF
Object state conflict	0C	Class specific error	01
Attribute not settable	0E	None	FF
Privilege violation	0F	None	FF
Device state conflict	10	None	FF
Reply data too large	11	None	FF
Not enough data	13	None	FF
Attribute not supported	14	None	FF
Too much data	15	None	FF
Object does not exist	16	None	FF
FRAGMENTATION EQ	17	None	FF
Invalid parameter	20	None	FF

The following steps may be useful to those users who would like to implement their DeviceNet applications by using the command set.

1. **Request the use of the Predefined Master/Slave Connection Set.**
2. **Apply the Master's Explicit Request Messages to set the expected_packet_rate attribute of the I/O connection and make the I/O Connection Object State established.**
3. **There are two ways to access I/O devices. One method is by the way of the I/O connection object. Another is by using an explicit message to set/get the IO attribute of application object.**
4. **Release the use of the Predefined Master/Slave Connection Set.**

7.2 Examples on the DeviceNet Communication Set

7.2.1 Request the use of Predefined Master/Slave Connection Set

An unconnected explicit messaging request sent by the Master node via a destination node's Group 2 Only Unconnected Explicit Request Message to requests the use of the Predefined Master/Slave Connection set. The example shows how to use these connection sets. In this demo, the Master establishes the Explicit Message, Poll IO and Bit-Strobe IO connections.

Table 7-3 shows the Group 2 Only Unconnected Explicit connection Identifier Fields.

Table 7-3 Identifier fields of the group 2 only unconnected explicit connection

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	9	8	7	6	5	4	3	2	1	0		
1	0	Source MAC ID						0	1	1	Slave's Explicit/ Unconnected Response Messages	
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages	

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

Allociation Choice : Explicit、Polled & Bit-Strobed

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 2 Message
   |      _ Destination MAC ID=0x09
   |      |      _ Message ID =6
   |      |      |           _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
   |      |      |           |    _ Service=Allocate_Master/Slave_Connection_Set Request
   |      |      |           |    |  _ Class ID=3
   |      |      |           |    |  |  _ Instance ID=1
   |      |      |           |    |  |  |  _ Allocation Choice=Explicit, Polled & Bit Strobed
   |      |      |           |    |  |  |  |    _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 07 0A
 ------------------------------------------------------------------------>
```

```
   _ Group 2 Message
   |      _ Source MAC ID=0x09
   |      |      _ Message ID =3
   |      |      |           _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
   |      |      |           |    _ Service=Allocate_Master/Slave_Connection_Set Response
   |      |      |           |    |  _Connection Message Body Format = DeviceNet (8/8)
   |      |      |           |    |  |
   |      |      |           |    |  |
   |      |      |           |    |  |
ID=10 001001 110. Data= 0A CB 00
 <--------------------------------------------------------------------
```

7.2.2 How to apply the Poll Connection

Poll Command and Responses message move any amount of I/O data between a Master and its Polled Slaves. The example explains how to apply the Poll IO connection in the DeviceNet application.

Table 7-4 shows the Poll I/O Connection Identifier Fields.

Table 7-4 Identifier Fields of the Poll I/O connection

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	9	8	7	6	5	4	3	2	1	0		
1	0	Destination MAC ID						1	0	1	Master's I/O Poll Command/Change of State/Cyclic Message	
1	0	Source MAC ID						0	1	1	Slave's Explicit/ Unconnected Response Messages	
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages	
1	0	Destination MAC ID						1	0	0	Master's Explicit Request Messages	
0	1	1	1	1	Source MAC ID						Slave's I/O Poll Response Message	

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Explicit & Polled

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
_ Group 2 Message
|       _ Destination MAC ID=0x09
|       |         _ Message ID =6
|       |         |              _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
|       |         |              |     _ Service=Allocate_Master/Slave_Connection_Set Request
|       |         |              |     |   _ Class ID=3
|       |         |              |     |   |   _ Instance ID=1
|       |         |              |     |   |   |   _ Allocation Choice=Explicit & Polled
|       |         |              |     |   |   |   |     _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 03 0A
------------------------------------------------------------------------------------->
```

```
_ Group 2 Message
|       _ Source MAC ID=0x09
|       |         _ Message ID =3
|       |         |              _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
|       |         |              |     _ Service=Allocate_Master/Slave_Connection_Set Response
|       |         |              |     |   _Connection Message Body Format = DeviceNet (8/8)
|       |         |              |     |   |
|       |         |              |     |   |
|       |         |              |     |   |
ID=10 001001 110. Data= 0A CB 00
<-------------------------------------------------------------------------------
```

2. Apply the Master's Explicit Request Messages to set the expected_packet_rate attribute of the I/O connection and make the I/O Connection Object State established.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =4
      |     |      |          _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
      |     |      |          |   _ Service=Set Attribute Request
      |     |      |          |   |  _ Class ID=5
      |     |      |          |   |  |  _ Instance ID=2   ( Poll IO connection Instance ID )
      |     |      |          |   |  |  |  _ Attribute ID=9
      |     |      |          |   |  |  |  |   _ Attribute Data= 0x0E0A
ID=10 001001 100. Data= 0A 10 05 02 09 0A 0E
```

```
      _ Group 2 Message
      |     _ Source MAC ID=0x09
      |     |      _ Message ID =3
      |     |      |          _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
      |     |      |          |   _ Service= Set Attribute Response
      |     |      |          |   |  _Response Attribute Data=0x0E10
      |     |      |          |   |  |
      |     |      |          |   |  |
      |     |      |          |   |  |
ID=10 001001 011. Data= 0A 90 10 0E
```

3. Apply the Poll I/O Connection to access the I/O devices

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =0
      |     |      |
      |     |      |            _ Poll Output Data
      |     |      |            |
ID=10 001001 101. Data= FF FF
```

```
      _ Group 1 Message
      |  _ Message ID =F
      |  |     _ Source MAC ID=0x09
      |  |     |              _ Poll Response Data
      |  |     |              |
ID= 0 1111 001001. Data= FF DF
```

7.2.3 The Bit-Strobe Connection example

Bit-Strobe Command and Response messages rapidly move small amounts of I/O data between the Master and its Bit-Strobe Slaves. Table 7-5 shows Bit-Strobe I/O Connection Identifier Fields.

Table 7-5 Identifier fields of Bit-Strobe I/O connection

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
0	1	1	1	0	Source MAC ID						Slave's I/O Bit–Strobe Response Message	
1	0	Source MAC ID						0	0	0	Master's I/O Bit–Strobe Command Message	
1	0	Source MAC ID						0	1	1	Slave's Explicit/ Unconnected Response Messages	
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages	
1	0	Destination MAC ID						1	0	0	Master's Explicit Request Messages	

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Explicit & Bit-Strobe

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 2 Message
   |     _ Destination MAC ID=0x09
   |     |     _ Message ID =6
   |     |     |        _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
   |     |     |        |   _ Service=Allocate_Master/Slave_Connection_Set Request
   |     |     |        |   |  _ Class ID=3
   |     |     |        |   |  |  _ Instance ID=1
   |     |     |        |   |  |  |  _ Allocation Choice=Explicit & Bit Strobed
   |     |     |        |   |  |  |  |   _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 05 0A
------------------------------------------------------------------------->
```

```
   _ Group 2 Message
   |     _ Source MAC ID=0x09
   |     |     _ Message ID =3
   |     |     |        _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
   |     |     |        |   _ Service=Allocate_Master/Slave_Connection_Set Response
   |     |     |        |   |  _Connection Message Body Format = DeviceNet (8/8)
   |     |     |        |   |  |
   |     |     |        |   |  |
   |     |     |        |   |  |
ID=10 001001 110. Data= 0A CB 00
<---------------------------------------------------------------------
```

2. Apply the Master's Explicit Request Messages to set the expected_packet_rate attribute of the I/O connection and make the I/O Connection Object State established.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |      _ Destination MAC ID=0x09
     |      |       _ Message ID =4
     |      |       |             _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
     |      |       |             |     _ Service=Set Attribute Request
     |      |       |             |     |   _ Class ID=5
     |      |       |             |     |   |   _ Instance ID=3    ( Bit Strobe IO connection Instance ID )
     |      |       |             |     |   |   |   _ Attribute ID=9
     |      |       |             |     |   |   |   |     _ Attribute Data= 0x0E0A
ID=10 001001 100. Data= 0A 10 05 03 09 0A 0E
```

```
     _ Group 2 Message
     |      _ Source MAC ID=0x09
     |      |       _ Message ID =3
     |      |       |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
     |      |       |             |     _ Service= Set Attribute Response
     |      |       |             |     |   _Response Attribute Data=0x0E10
     |      |       |             |     |   |
     |      |       |             |     |   |
     |      |       |             |     |   |
ID=10 001001 011. Data= 0A 90 10 0E
```

3. Apply Bit-Strobe I/O Connection to access the IO modules

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |      _ Destination MAC ID=0x09
     |      |       _ Message ID =0
     |      |       |
     |      |       |                _ Bit-Strobe 64-bit Output Data
     |      |       |                |
ID=10 001001 000. Data= FF FF FF FF FF FF FF FF
```

```
     _ Group 1 Message
     |   _ Message ID =E
     |   |       _ Source MAC ID=0x09
     |   |       |                     _ Bit-Strobe Response Data
     |   |       |                     |
ID= 0 1110 001001. Data= FF DF
```

7.2.4 Change of State/Cyclic Connection example (Acknowledged)

The Change of State/Cyclic connection sets use the connection instance 2 (the polled connection instance) for master to slave data production and slave to master acknowledgment. Connection instance 4 is used for slave to master data production and master to slave acknowledgment. If a device does not support the poll connection and has no output object, then connection instance 2 does not need to be instantiated. Table 7-6 shows COS/Cyclic I/O Connection Identifier Fields.

Table 7-6 Identifier fields of COS/Cyclic I/O connection

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	9	8	7	6	5	4	3	2	1	0		
0	1	1	0	1	Source MAC ID						Slave's I/O Change of State or Cyclic Message	
1	0	Destination MAC ID						0	1	0	Master's Change of State or Cyclic Acknowledge Message	
1	0	Source MAC ID						0	1	1	Slave's Explicit/ Unconnected Response Messages	
1	0	Destination MAC ID						1	1	0	Group 2 Only Unconnected Explicit Request Messages	
1	0	Destination MAC ID						1	0	0	Master's Explicit Request Messages	

Note: I-7242D: Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Cyclic & Explicit

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |      _ Destination MAC ID=0x09
     |      |       _ Message ID =6
     |      |       |             _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
     |      |       |             |     _ Service=Allocate_Master/Slave_Connection_Set Request
     |      |       |             |     |   _ Class ID=3
     |      |       |             |     |   |   _ Instance ID=1
     |      |       |             |     |   |   |   _ Allocation Choice=Explicit &Cyclic
     |      |       |             |     |   |   |   |     _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 21 0A
```

```
     _ Group 2 Message
     |      _ Source MAC ID=0x09
     |      |       _ Message ID =3
     |      |       |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
     |      |       |             |     _ Service=Allocate_Master/Slave_Connection_Set Response
     |      |       |             |     |   _Connection Message Body Format = DeviceNet (8/8)
     |      |       |             |     |   |
     |      |       |             |     |   |
     |      |       |             |     |   |
ID=10 001001 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to set the expected_packet_rate attribute of the I/O connection and make the I/O Connection Object State established.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |      _ Destination MAC ID=0x09
     |      |       _ Message ID =4
     |      |       |             _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
     |      |       |             |     _ Service=Set Attribute Request
     |      |       |             |     |   _ Class ID=5
     |      |       |             |     |   |   _ Instance ID=4    ( Cyclic IO connection Instance ID )
     |      |       |             |     |   |   |   _ Attribute ID=9
     |      |       |             |     |   |   |   |     _ Attribute Data= 0x0E0A
ID=10 001001 100. Data= 0A 10 05 04 09 0A 0E
```

```
     _ Group 2 Message
     |      _ Source MAC ID=0x09
     |      |       _ Message ID =3
     |      |       |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
     |      |       |             |     _ Service= Set Attribute Response
     |      |       |             |     |   _Response Attribute Data=0x0E10
     |      |       |             |     |   |
     |      |       |             |     |   |
     |      |       |             |     |   |
ID=10 001001 011. Data= 0A 90 10 0E
```

3. To start the Cyclic I/O connection and transfer IO data

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

Slave transmits I/O data to Master

```
     _ Group 1 Message
     |   _ Message ID =D
     |   |      _ Source MAC ID=0x09
     |   |      |              _ Cyclic Output Data of 2 bytes
     |   |      |              |
ID= 0 1101 001001. Data= FF DF
```

Master responses Acknowledge message

```
     _ Group 2 Message
     |     _ Destination MAC ID=0x09
     |     |      _ Message ID =2
     |     |      |
     |     |      |          _No response data
ID=10 001001 010. Data= No data
```

Slave transmits I/O data to Master after a period of time

```
     _ Group 1 Message
     |   _ Message ID =D
     |   |      _ Source MAC ID=0x09
     |   |      |              _ Cyclic Output Data of 2 bytes
     |   |      |              |
ID= 0 1101 001001. Data= FF DF
```

Master responses Acknowledge message

```
     _ Group 2 Message
     |     _ Destination MAC ID=0x09
     |     |      _ Message ID =2
     |     |      |
     |     |      |          _No response data
ID=10 001001 010. Data= No data
```

7.2.5 Change of State/Cyclic Connection example (Unacknowledged)

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Ack suppression, Cyclic & Explicit

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 2 Message
   |     _ Destination MAC ID=0x09
   |     |      _ Message ID =6
   |     |      |         _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
   |     |      |         |   _ Service=Allocate_Master/Slave_Connection_Set Request
   |     |      |         |   |  _ Class ID=3
   |     |      |         |   |  |  _ Instance ID=1
   |     |      |         |   |  |  |  _ Allocation Choice=Ack suppression, Cyclic &Explicit
   |     |      |         |   |  |  |  |    _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 61 0A
```

```
   _ Group 2 Message
   |     _ Source MAC ID=0x09
   |     |      _ Message ID =3
   |     |      |         _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
   |     |      |         |   _ Service=Allocate_Master/Slave_Connection_Set Response
   |     |      |         |   |  _Connection Message Body Format = DeviceNet (8/8)
   |     |      |         |   |  |
   |     |      |         |   |  |
   |     |      |         |   |  |
ID=10 001001 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to set the expected_packet_rate attribute of the I/O connection and make the I/O Connection Object State established.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 2 Message
   |     _ Destination MAC ID=0x09
   |     |      _ Message ID =4
   |     |      |         _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
   |     |      |         |    _ Service=Set Attribute Request
   |     |      |         |    |   _ Class ID=5
   |     |      |         |    |   |   _ Instance ID=4   ( Cyclic IO connection Instance ID )
   |     |      |         |    |   |   |   _ Attribute ID=9
   |     |      |         |    |   |   |   |    _ Attribute Data= 0x0E0A
ID=10 001001 100. Data= 0A 10 05 04 09 0A 0E
```

```
   _ Group 2 Message
   |     _ Source MAC ID=0x09
   |     |      _ Message ID =3
   |     |      |         _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
   |     |      |         |    _ Service= Set Attribute Response
   |     |      |         |    |   _Response Attribute Data=0x0E10
   |     |      |         |    |   |
   |     |      |         |    |   |
   |     |      |         |    |   |
ID=10 001001 011. Data= 0A 90 10 0E
```

3. Slave transmits data cyclically.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 1 Message
   |  _ Message ID =D
   |  |     _ Source MAC ID=0x09
   |  |     |              _ Cyclic Output Data of 2 bytes
   |  |     |              |
ID= 0 1101 001001. Data= FF DF
```

```
   _ Group 1 Message
   |  _ Message ID =D
   |  |     _ Source MAC ID=0x09
   |  |     |              _ Cyclic Output Data of 2 bytes
   |  |     |              |
ID= 0 1101 001001. Data= FF DF
```

7.2.6 Change MAC ID example

If necessarily, users can change MAC ID of I-7242D. For example, the I-7242D sends “Duplicate MAC ID Check Message” at power on or reset mode. There could be the same ID of device in the network. Therefore, the I-7242D supports the ability of on-line change the MAC ID of CAN. Please refer to the following example.

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Explicit

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 2 Message
   |     _ Destination MAC ID=0x09
   |     |      _ Message ID =6
   |     |      |          _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
   |     |      |          |    _ Service=Allocate_Master/Slave_Connection_Set Request
   |     |      |          |    |  _ Class ID=3
   |     |      |          |    |  |  _ Instance ID=1
   |     |      |          |    |  |  |  _ Allocation Choice= Explicit
   |     |      |          |    |  |  |  |    _ Allocator's MAC ID=0x0A
ID=10 000011 110. Data= 0A 4B 03 01 01 0A
```

```
   _ Group 2 Message
   |     _ Source MAC ID=0x09
   |     |      _ Message ID =3
   |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
   |     |      |          |    _ Service=Allocate_Master/Slave_Connection_Set Response
   |     |      |          |    |  _Connection Message Body Format = DeviceNet (8/8)
   |     |      |          |    |  |
   |     |      |          |    |  |
   |     |      |          |    |  |
ID=10 000011 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to change the Slave's MAC ID.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |       _ Destination MAC ID=0x09
    |       |        _ Message ID =4
    |       |        |             _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
    |       |        |             |     _ Service=Set Attribute Request
    |       |        |             |     |   _ Class ID=3
    |       |        |             |     |   |   _ Instance ID=1
    |       |        |             |     |   |   |   _ Attribute ID=1
    |       |        |             |     |   |   |   |     _ Attribute Data= 0x09 -- New MAC ID
ID=10 000011 100. Data= 0A 10 03 01 01 09
```

```
    _ Group 2 Message
    |       _ Source MAC ID=0x09
    |       |        _ Message ID =3
    |       |        |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
    |       |        |             |     _ Service= Set Attribute Response
    |       |        |             |     |
    |       |        |             |     |
ID=10 000011 011. Data= 0A 90
```

3. After changing the MAC ID, slave (I-7242D) will send out the shutdown message and reset.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |       _ Source MAC ID=0x09
    |       |        _ Message ID =3
    |       |        |             _ Frag=0. Transaction ID=0. Destination MAC ID=0x0A
    |       |        |             |     _ Service code= 0x4E, Device Shutdwon message
    |       |        |             |     |      __ Class ID=3
    |       |        |             |     |      |      _ Instance ID=1
    |       |        |             |     |      |      |          _ Shutdown Code=4
ID=10 000011 011. Data= 0A CE 03 00 01 00 04 00
```

4. After resetting, the slave (I-7242D) will send duplicate MAC ID check message twice.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |     _ Source MAC ID=0x09
     |     |      _ Message ID =7
     |     |      |          _ Physical Port Number=0
     |     |      |          |     _ Vendor ID= 803
     |     |      |          |     |      __ Serial Number=1
     |     |      |          |     |      |
     |     |      |          |     |      |
ID=10 001001 111. Data= 00 23 03 01 00 00 00
```

```
     _ Group 2 Message
     |     _ Source MAC ID=0x09
     |     |      _ Message ID =7
     |     |      |          _ Physical Port Number=0
     |     |      |          |     _ Vendor ID= 803
     |     |      |          |     |      __ Serial Number=1
     |     |      |          |     |      |
     |     |      |          |     |      |
ID=10 001001 111. Data= 00 23 03 01 00 00 00
```

7.2.7 Change CAN Baud Rate on-line example

I-7242D supports the function to change CAN baud rate on-line. But after finishing change baud rate, the new baud rate would be not effective immediately. Must to request reset service to reset the I-7242D.

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Explicit

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |    _ Destination MAC ID=0x09
    |    |     _ Message ID =6
    |    |     |        _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
    |    |     |        |   _ Service=Allocate_Master/Slave_Connection_Set Request
    |    |     |        |   |  _ Class ID=3
    |    |     |        |   |  |  _ Instance ID=1
    |    |     |        |   |  |  |  _ Allocation Choice= Explicit
    |    |     |        |   |  |  |  |   _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 01 0A
--------------------------------------------------------------->
```

```
    _ Group 2 Message
    |    _ Source MAC ID=0x09
    |    |     _ Message ID =3
    |    |     |        _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
    |    |     |        |   _ Service=Allocate_Master/Slave_Connection_Set Response
    |    |     |        |   |  _Connection Message Body Format = DeviceNet (8/8)
    |    |     |        |   |  |
    |    |     |        |   |  |
    |    |     |        |   |  |
ID=10 000011 110. Data= 0A CB 00
<-----------------------------------------------------------
```

2. Apply the Master's Explicit Request Messages to change the Slave's Baud Rate.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |      _ Destination MAC ID=0x09
    |      |        _ Message ID =4
    |      |        |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
    |      |        |             |     _ Service=Set Attribute Request
    |      |        |             |     |   _ Class ID=3
    |      |        |             |     |   |   _ Instance ID=1
    |      |        |             |     |   |   |   _ Attribute ID=2
    |      |        |             |     |   |   |   |     _ Attribute Data= 0x01 -- New Baud Rate
ID=10 001001 100. Data= 0A 10 03 01 02 01
```

```
    _ Group 2 Message
    |      _ Source MAC ID=0x09
    |      |        _ Message ID =3
    |      |        |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
    |      |        |             |     _ Service= Set Attribute Response
    |      |        |             |     |
    |      |        |             |     |
ID=10 000011 011. Data= 0A 90
```

Note: After finishing the change of the MAC ID on-line, the I-7242D will send a shutdown message and reset. However if users want to change the baud rate of I-7242D, they must send the reset service to reset the I-7242D. Then the new baud rate of the I-7242D will become effective.

7.2.8 Reset Service

This service can reset the I-7242D. If users change the baud rate of I-7242D, they must send reset service to reset I-7242D. Therefore the new baud rate of the I-7242D will become effective.

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Explicit

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |      _ Destination MAC ID=0x09
     |      |       _ Message ID =6
     |      |       |            _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
     |      |       |            |     _ Service=Allocate_Master/Slave_Connection_Set Request
     |      |       |            |     |   _ Class ID=3
     |      |       |            |     |   |   _ Instance ID=1
     |      |       |            |     |   |   |   _ Allocation Choice= Explicit
     |      |       |            |     |   |   |   |     _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 01 0A
```

```
     _ Group 2 Message
     |      _ Source MAC ID=0x09
     |      |       _ Message ID =3
     |      |       |            _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
     |      |       |            |     _ Service=Allocate_Master/Slave_Connection_Set Response
     |      |       |            |     |   _Connection Message Body Format = DeviceNet (8/8)
     |      |       |            |     |   |
     |      |       |            |     |   |
     |      |       |            |     |   |
ID=10 000011 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to set the Identify object. The service ID (0x05) is reset service.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |    _ Destination MAC ID=0x09
      |    |     _ Message ID =4
      |    |     |        _ Frag=0. Transaction ID=0. Souce MAC ID= 0x0A
      |    |     |        |   _ Service=Reset service
      |    |     |        |   |  _ Class ID=1
      |    |     |        |   |  |  _ Instance ID=1
      |    |     |        |   |  |  |
ID=10 001001 100. Data= 0A 05 01 01 00
```

```
      _ Group 2 Message
      |    _ Source MAC ID=0x09
      |    |     _ Message ID =3
      |    |     |        _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |    |     |        |   _ Service= Reset Response
      |    |     |        |   |
      |    |     |        |   |
ID=10 000011 011. Data= 0A 85
```

Slave (MAC ID =0x09) responses the shutdown message to network.

```
      _ Group 2 Message
      |    _ Source MAC ID=0x09
      |    |     _ Message ID =3
      |    |     |        _ Frag=0. Transaction ID=0. Destination MAC ID=9
      |    |     |        |   _ Service code= 0x4E, Device Shutdwon message
      |    |     |        |   |    __ Class ID=1
      |    |     |        |   |   |     _ Instance ID=1
      |    |     |        |   |  _|__ __|__ _____ Shutdown Code
ID=10 001001 011. Data= 0A CE 01 00 01 00 04 00
```

7.2.9 Device Heartbeat

This message broadcasts the current state of the I-7242D. This message is transmitted by a group 2 only server as an Unconnected Response Message (Message Group 2, Message ID 3).

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Allociation Choice : Explicit

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =6
      |     |      |          _ Frag=0. Transaction ID=0. Source MAC ID= 0x0A
      |     |      |          |    _ Service=Allocate_Master/Slave_Connection_Set Request
      |     |      |          |    |  _ Class ID=3
      |     |      |          |    |  |  _ Instance ID=1
      |     |      |          |    |  |  |  _ Allocation Choice= Explicit
      |     |      |          |    |  |  |  |    _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 01 0A
```

```
      _ Group 2 Message
      |     _ Source MAC ID=0x09
      |     |      _ Message ID =3
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |    _ Service=Allocate_Master/Slave_Connection_Set Response
      |     |      |          |    |  _Connection Message Body Format = DeviceNet (8/8)
      |     |      |          |    |  |
      |     |      |          |    |  |
      |     |      |          |    |  |
ID=10 000011 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to set the Identify object. Device Heartbeat interval Attribute ID : 0x0A

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =4
      |     |      |          _ Frag=0. Transaction ID=0. Source MAC ID= 0x0A
      |     |      |          |    _ Service=set attribute
      |     |      |          |    |  _ Class ID=1
      |     |      |          |    |  |  _ Instance ID=1
      |     |      |          |    |  |  |  _ Attribute ID= 0x0A
      |     |      |          |    |  |  |  |    _ Attribute Data= 2 seconds
ID=10 001001 100. Data= 0A 10 01 01 0A 02 00
```

Slave response Device Heartbeat Message every two seconds.

```
      _ Group 2 Message
      |     _ Source MAC ID=0x09
      |     |      _ Message ID =3
      |     |      |          _ Frag=0. Destination MAC ID= 0x09
      |     |      |          |    _ Service code=0x4D
      |     |      |          |    |     _ Identity Object Instance ID=1
      |     |      |          |    |     |    _ Device State=3
      |     |      |          |    |  __|_ _|__  ____ EV, SF, UF, DF and Consistency value
ID=10 001001 011. Data=0A CD 01 00 03 00 00 00
```

If users want to cancel the heartbeat message, they must set the heartbeat interval attribute value of the Identity object instance to zero.

7.2.10 Offline Connection Set

This section describes the Offline Connection Set Messaging Protocol and presents the details associated with the establishment of Offline Connection Set ownership. Support of the Offline Connection Set is optional for all types of devices. Table 7-7 shows Offline connection Set Identifier Fields

Table 7-7 Identifier fields of Offline connection set

IDENTIFIER BITS											IDENTITY USAGE	HEX RANGE
10	9	8	7	6	5	4	3	2	1	0		
1	1	1	1	1	Group 4 Message ID						Group 4 Messages	000 - 3ff
1	1	1	1	1	2C						Communication Faulted Response Message	
1	1	1	1	1	2D						Communication Faulted Request Message	

In this example, the I-7242D is set to an off-line state, because it has a duplicated fault. We can then apply the offline connection set to change its baud rate.

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Apply the Communication Faulted Request message to communicate with offline devices. (Group 4, message: 2D, service: 4B)

Who communication Fault Request Message:

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 4 Message
      |        _ Message ID=0x2D
      |        |
      |        |                _ match=0, value=0x3f
      |        |                |     _ Service=0x4B
      |        |                |     |   _Time Delay Byte Offset
      |        |                |     |   |
ID=11111 101101. Data=3F 4B 02
```

Who communication Fault Response Message:

```
     _ Group 4 Message
      |        _ Message ID=0x2C
      |        |                 _ Physical Port Number
      |        |                 |     _ Vendor ID
      |        |                 |     |            __ Serial Number
      |        |                 |   _|_    _____|_____
ID=11111 101100. Data= 00 23 03 01 00 00 00
```

2. *Apply the Communication Faulted Request message to change the I-7242D's MAC ID. (Group 4, message: 2D , service: 4D)*

Change MAC ID Communication Fault Request Message:

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 4 Message
    |       _ Message ID=0x2D
    |       |             _New MAC ID=7
    |       |             |    _ Service=0x4D
    |       |             |    |     _Vendor ID
    |       |             |    |   __|_    _________ Serial Number
ID=11111 101101. Data=07 4D 23 03 01 00 00 00
```

3. *After finishing the change of the MAC ID, I-7242D will send the duplicated message to the DeviceNet network.*

Slave (MAC ID =0x07)

```
   _ Group 2 Message
   |     _ Source MAC ID=0x07
   |     |      _ Message ID =7
   |     |      |          _ Physical Port Number=0
   |     |      |          |     _ Vendor ID= 803
   |     |      |          |     |      __ Serial Number=1
   |     |      |          |     |      |
   |     |      |          |   __|_ ___ |_____
ID=10 000111 111. Data= 00 23 03 01 00 00 00
```

```
   _ Group 2 Message
   |     _ Source MAC ID=0x07
   |     |      _ Message ID =7
   |     |      |          _ Physical Port Number=0
   |     |      |          |     _ Vendor ID= 803
   |     |      |          |     |      __ Serial Number=1
   |     |      |          |     |      |
   |     |      |          |   __|_ ___ |_____
ID=10 000111 111. Data= 00 23 03 01 00 00 00
```

7.2.11 Fragmentation example

7.2.11.1 Unacknowledged Fragmentation example

Fragmentation of an I/O message is performed in an unacknowledged fashion. Unacknowledged fragmentation consists of the back-back transmission of the fragments from the transmitting module. The receiving module(s) returns no acknowledgments (other than the CAN-provided Ack) on a per-fragment basis. The Connection simply invokes the Link Producer's Send service as necessary to move the message without waiting for any specific acknowledgment from the receiving module(s).

In this example, the polling consumed size is 10 bytes. Master must send fragmented messages. Data=0102030405060708090A. Assume that the I/O Connection has been established.

Unacknowledged:

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |     _ Destination MAC ID=0x09
    |     |     _ Message ID =0
    |     |     |
    |     |     |          _ Fragment Type= First Fragment, Fragment Count=0
    |     |     |         |   ---------------------- 1st portion of the I/O data
ID=10 001001 101. Data= 00 01 02 03 04 05 06 07

──────────────────────────────────────────────────────────►
```

```
    _ Group 2 Message
    |     _ Destination MAC ID=0x09
    |     |     _ Message ID =0
    |     |     |
    |     |     |          _ Fragment Type= Final Fragment, Fragment Count=1
    |     |     |         |   ---------------------- final portion of the I/O data
ID=10 001001 101. Data= 81 08 09 0A

──────────────────────────────────────────────────────────►
```

7.2.11.2 Acknowledged Fragmentation example

Fragmentation of an Explicit Message is performed in an Acknowledged fashion. Acknowledged fragmentation consists of the transmission of a fragment from the transmitting module followed by the transmission of an acknowledgment by the receiving module. The receiving module acknowledges the reception of each fragment. This provides a degree of flow control. The assumption is that larger bodies of information may be moved across Explicit Messaging Connections (e.g. Upload/Download functions) and, as such, a degree of flow control is necessary.

In this example, assume the attribute data=0102030405060708090A. The assembly instance ID=0x64, attribute=0x03.

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |    _ Destination MAC ID=0x09
    |    |     _ Message ID =6
    |    |     |         _ Frag=0. Transaction ID=0. Source MAC ID= 0x0A
    |    |     |         |    _ Service=Allocate_Master/Slave_Connection_Set Request
    |    |     |         |    |  _ Class ID=3
    |    |     |         |    |  |  _ Instance ID=1
    |    |     |         |    |  |  |  _ Allocation Choice= Explicit
    |    |     |         |    |  |  |  |    _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 01 0A
```

```
    _ Group 2 Message
    |    _ Source MAC ID=0x09
    |    |     _ Message ID =3
    |    |     |         _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
    |    |     |         |    _ Service=Allocate_Master/Slave_Connection_Set Response
    |    |     |         |    |  _Connection Message Body Format = DeviceNet (8/8)
    |    |     |         |    |  |
    |    |     |         |    |  |
    |    |     |         |    |  |
ID=10 000011 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to set the Assembly object instance attributes.

Service (0x10)=Set attribute service, Data=0102030405060708090A.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |       _ Destination MAC ID=0x09
     |       |        _ Message ID =4
     |       |        |              _ Frag=1. Transaction ID=0. Source MAC ID= 0x0A
     |       |        |              |     _ Fragment Type= First Fragment, Fragment Count=0
     |       |        |              |     |   _ Service=set attribute
     |       |        |              |     |   |   _ Class ID=4
     |       |        |              |     |   |   |   _ Instance ID=2
     |       |        |              |     |   |   |   |   _ Attribute ID= 0x03
     |       |        |              |     |   |   |   |   |
     |       |        |              |     |   |   |   |   |     _ Attribute Data= 0103
     |       |        |              |     |   |   |   |   |    _|_
ID=10 001001 100. Data= 8a 00 10 04 02 03 01 02
--------------------------------------------------------------------->
```

```
     _ Group 2 Message
     |       _ Source MAC ID=0x09
     |       |        _ Message ID =3
     |       |        |              _ Frag=1. Destination MAC ID= 0x0A
     |       |        |              |     _ Fragment Type= Acknowledge, Fragment Count=0
     |       |        |              |     |
     |       |        |              |     |   _Ack State=Success
ID=10 001001 011. Data=8A C0 00

<---------------------------------------------------------------
```

```
     _ Group 2 Message
     |       _ Destination MAC ID=0x09
     |       |        _ Message ID =4
     |       |        |              _ Frag=1. Transaction ID=0. Source MAC ID= 0x0A
     |       |        |              |     _ Fragment Type= Middle Fragment, Fragment Count=1
     |       |        |              |     |     _ Attribute Data
     |       |        |              |     |   __|___________
ID=10 001001 100. Data= 8a 41 03 04 05 06 07 08

--------------------------------------------------------------------->
```

```
     _ Group 2 Message
     |       _ Source MAC ID=0x09
     |       |        _ Message ID =3
     |       |        |              _ Frag=1. Destination MAC ID= 0x0A
     |       |        |              |     _ Fragment Type= Acknowledge, Fragment Count=1
     |       |        |              |     |
     |       |        |              |     |   _Ack State=Success
ID=10 001001 011. Data=8A C1 00

<---------------------------------------------------------------
```

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
     _ Group 2 Message
     |      _ Destination MAC ID=0x09
     |      |      _ Message ID =4
     |      |      |          _ Frag=1. Transaction ID=0. Source MAC ID= 0x0A
     |      |      |          |    _ Fragment Type= Final Fragment, Fragment Count=2
     |      |      |          |    |     _ Attribute Data
     |      |      |          |    |   __|_
ID=10 001001 100. Data= 8a 82 09 0A
```

```
     _ Group 2 Message
     |      _ Source MAC ID=0x09
     |      |      _ Message ID =3
     |      |      |          _ Frag=1. Destination MAC ID= 0x0A
     |      |      |          |    _ Fragment Type= Acknowledge, Fragment Count=2
     |      |      |          |    |
     |      |      |          |    |  _Ack State=Success
ID=10 001001 011. Data=8A C2 00
```

7.2.12 User-defined Modbus commands example

In “User-defined Modbus Command” class, the I-7242D supports three instances for users who want to define their own Modbus messages. Users can use the Master’s Explicit Message to set the 0x01 attribute and 0x03 attribute of this class. Please see the following steps.

Step 1: Set 0x01 attribute value as the needed query Modbus command and set 0x03 attribute value as the response Modbus command length.

Step 2: Then set the 0x04 attribute value as a non-zero value. Thus the I-7242D will send out the message according to 0x01 attribute to Modbus devices.

Step 3: The I-7242D will receive the response message from Modbus devices and store the response message in 0x02 attribute.

Step 4: After sending out the user-defined Modbus command, users can use the Master’s Explicit Message to get the 0x02 attribute. And then the response Modbus message will be returned to the Master.

In this example, we will use user-defined Modbus message to set Modbus device (M-7017), see table 7-8. The request and response Modbus commands are in the table 7-9 and table 7-10.

Table 7-8 User-defined function code (0x46): Read/Write Module Setting

Sub-function Code	**Description**
00 (0x00)	Read the module name

Table 7-8 Request Modbus command message

00	Address	1 Byte	1 to 247
01	Function code	1 Byte	0x46
02	Sub function code	1 Byte	0x00

Table 7-9 Response Modbus command message

00	Address	1 Byte	1 to 247
01	Function code	1 Byte	0x46
02	Sub function code	1 Byte	0x00
03~06	Module name	4 Byte	0x00 0x70 0x17 0x00 for M-7017 serious modules

Now, start to communication with I-7242D.

Note: Slave (I-7242D): Node ID=0x09, Master: Node ID=0x0A

1. Request the use of the Predefined Master/Slave Connection set

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
   _ Group 2 Message
   |       _ Destination MAC ID=0x09
   |       |        _ Message ID =6
   |       |        |             _ Frag=0. Transaction ID=0. Source MAC ID= 0x0A
   |       |        |             |      _ Service=Allocate_Master/Slave_Connection_Set Request
   |       |        |             |      |   _ Class ID=3
   |       |        |             |      |   |   _ Instance ID=1
   |       |        |             |      |   |   |   _ Allocation Choice= Explicit
   |       |        |             |      |   |   |   |      _ Allocator's MAC ID=0x0A
ID=10 001001 110. Data= 0A 4B 03 01 01 0A
```

```
   _ Group 2 Message
   |       _ Source MAC ID=0x09
   |       |        _ Message ID =3
   |       |        |             _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
   |       |        |             |      _ Service=Allocate_Master/Slave_Connection_Set Response
   |       |        |             |      |   _Connection Message Body Format = DeviceNet (8/8)
   |       |        |             |      |   |
   |       |        |             |      |   |
   |       |        |             |      |   |
ID=10 000011 110. Data= 0A CB 00
```

2. Apply the Master's Explicit Request Messages to set the User-defined Modbus command object instance attribute 0x01.

Service (0x10)=Set attribute service, Data = 01 46 00.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =4
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |   _ Service=Set Attribute Request
      |     |      |          |   |  _ Class ID=0x65
      |     |      |          |   |  |  _ Instance ID=0x01
      |     |      |          |   |  |  |  _ Attribute ID=0x01
      |     |      |          |   |  |  |  |   _ Attribute Data= 0x01 0x46 0x00 -- Request Modbus
ID=10 001001 100. Data= 0A 10 65 01 01 01 46 00                                  Command
```

```
      _ Group 2 Message
      |     _ Source MAC ID=0x09
      |     |      _ Message ID =3
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |   _ Service= Set Attribute Response
      |     |      |          |   |
      |     |      |          |   |
ID=10 000011 011. Data= 0A 90
```

3. Apply the Master's Explicit Request Messages to set the User-defined Modbus command object instance attribute 0x03.

Set the response Modbus message length = 0x07 bytes

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =4
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |    _ Service=Set Attribute Request
      |     |      |          |    |  _ Class ID=0x65
      |     |      |          |    |  |  _ Instance ID=0x01
      |     |      |          |    |  |  |  _ Attribute ID=0x03
      |     |      |          |    |  |  |  |   _ Attribute Data= 0x07 – Response Modbus Command
ID=10 001001 100. Data= 0A 10 65 01 03 07                                        length
```

```
      _ Group 2 Message
      |     _ Source MAC ID=0x09
      |     |      _ Message ID =3
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |    _ Service= Set Attribute Response
      |     |      |          |    |
      |     |      |          |    |
ID=10 000011 011. Data= 0A 90
```

4. Apply the Master's Explicit Request Messages to set the User-defined Modbus command object instance attribute 0x04.

Set 0x04 attribute with a non-zero value.

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =4
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |    _ Service=Set Attribute Request
      |     |      |          |    |  _ Class ID=0x65
      |     |      |          |    |  |  _ Instance ID=0x01
      |     |      |          |    |  |  |  _ Attribute ID=0x04
      |     |      |          |    |  |  |  |    _ Attribute Data= 0x01 – Non-zero value
ID=10 001001 100. Data= 0A 10 65 01 04 01
```

```
      _ Group 2 Message
      |     _ Source MAC ID=0x09
      |     |      _ Message ID =3
      |     |      |          _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
      |     |      |          |    _ Service= Set Attribute Response
      |     |      |          |    |
      |     |      |          |    |
ID=10 000011 011. Data= 0A 90
```

After setting the 0x04 attribute with a non-zero value, this user-defined message will be sent to the Modbus device.

5. Then apply the Master's Explicit Request Messages to get the User-defined Modbus command object instance attribute 0x02.

Service (0x0E)=Get attribute service,

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
    _ Group 2 Message
    |    _ Destination MAC ID=0x09
    |    |     _ Message ID =4
    |    |     |         _ Frag=0. Transaction ID=0. Destination MAC ID= 0x0A
    |    |     |         |    _ Service=Get Attribute Request
    |    |     |         |    |  _ Class ID=0x65
    |    |     |         |    |  |  _ Instance ID=0x01
    |    |     |         |    |  |  |  _ Attribute ID=0x02
    |    |     |         |    |  |  |  |
ID=10 001001 100. Data= 0A 0E 65 01 02
```

```
    _ Group 2 Message
    |    _ Source MAC ID=0x09
    |    |     _ Message ID =3
    |    |     |         _ Frag=1. Transaction ID=0. Destination MAC ID= 0x0A
    |    |     |         |    _ Fragment Type= Acknowledge, Fragment Count=0
    |    |     |         |    |
    |    |     |         |    |
ID=10 000011 011. Data= 8A 00 8E 01 46 00 00 70
```

```
    _ Group 2 Message
    |    _ Destination MAC ID=0x09
    |    |     _ Message ID =4
    |    |     |         _ Frag=1. Destination MAC ID= 0x09
    |    |     |         |    _ Fragment Type= Acknowledge, Fragment Count=0
    |    |     |         |    |
    |    |     |         |    |  _Ack State=Success
ID=10 001001 100. Data=8A C0 00
```

```
    _ Group 2 Message
    |    _ Source MAC ID=0x09
    |    |     _ Message ID =3
    |    |     |         _ Frag=1. Transaction ID=0. Destination MAC ID= 0x0A
    |    |     |         |    _ Fragment Type= Acknowledge, Fragment Count=1
    |    |     |         |    |
    |    |     |         |    |
ID=10 000011 011. Data= 8A 81 17 00
```

Master (MAC ID =0x0A) Slave (MAC ID =0x09)

```
      _ Group 2 Message
      |     _ Destination MAC ID=0x09
      |     |      _ Message ID =4
      |     |      |          _ Frag=1. Destination MAC ID= 0x09
      |     |      |          |    _ Fragment Type= Acknowledge, Fragment Count=1
      |     |      |          |    |
      |     |      |          |    |   _Ack State=Success
ID=10 001001 100. Data=8A C1 00
```

After using the Master's Explicit Message to get 0x02 attribute, the slave would return the Modbus response message. Then users can get the specific Modbus response message that Modbus device returned.

8 Modbus Commands

Only the Modbus commands shown in the table 8-1 are supported by the I-7242D. The structure of the query and response frames for each of these commands is then described in the following section.

Table 8-1 Support commands of I-7242D

Function Code	**Modbus Command**
0x01	Read Coil Status
0x02	Read Input Status
0x03	Read Holding Registers
0x04	Read Input Registers
0x0F	Force Multiple Coils
0x10	Preset Multiple Registers

In the table 8-2, each byte of the query and response frames of the Modbus command are described with the excepted of the field shown opposite. These are always present in the queries and responses of all Modbus command.

Table 8-2 Frames of a Modbus command

<table>
<tr><td>Device address</td><td>Value cannot be changed (Valid Modbus device address: 1 to 247.)</td></tr>
<tr><td>Function code</td><td>Value can not be changed (code of the Modbus command)</td></tr>
<tr><td>… Other fields …</td><td>… Specific features of Modbus commands …</td></tr>
<tr><td>CRC (LSB)</td><td rowspan="2">Cyclic Redundancy Check, containing 16-bit binary value</td></tr>
<tr><td>CRC (MSB)</td></tr>
</table>

It’s a better idea to get hold of a standard Modbus document, such as the guide entitled Modicon Modbus Protocol Reference Guide, so that you can see the correspondence between the elements displayed in Utility and the content of the corresponding Modbus frames

Here is an example of correspondences for a full frame, based on the "Read Holding Registers" Command (0x03).

	Element under Utility		**Modbus frame fields**	**Size (byte)**
Modbus query	Modbus Devices Address		Device no.	1
	Device I/O Type		Function no.	1
	Register Address		No. 1st word (MSB/LSB)	2
	Communication Words		No. of words (MSB/LSB)	2

	Element under Gateway		**Modbus frame fields**	**Size (byte)**
Modbus response	Modbus Devices Address		Device no.	1
	Device I/O Type		Function no.	1
	Byte count		No. of bytes read	1
	Data		Value of 1st word (MSB/LSB)	2
			…………………………	…
			Value of last word (MSB/LSB)	2

8.1 “Read Coil Status” Command (0x01)

Read the On/Off status of discrete output in the slave. The query message specifies the starting coil and quantity of coils to be read. And the coil status in the response message is packed as one coil per bit of the data field.

Frame	Field	Description
Query	Starting Address (Hi)	Address of 1st output coil
	Starting Address (Lo)	
	No. of Points (Hi)	Number of output coils
	No. of Points (Lo)	
Response	Byte Count	Number of data bytes=(number of output coils + 7) / 8
	Data (Coils: H-L order)(L)	Byte swap=”Swap 1 byte” (or “No swapping”)
	…..	Data length=Value of the “Byte Count” field
	Data (Coils: H-L order)(H)	Data location=Address in the gateway’s “DO” memory

8.2 “Read Input Status” Command (0x02)

Read the On/Off status of discrete input in the slave. The query message specifies the starting input and quantity of inputs to be read. And the input status in the response message is packed as one input per bit of the data field.

Frame	Field	Description
Query	Starting Address (Hi)	Address of 1st input coil
	Starting Address (Lo)	
	No. of Points (Hi)	Number of input coils
	No. of Points (Lo)	
Response	Byte Count	Number of data bytes=(number of input coils + 7) / 8
	Data (Inputs: H-L order)(L)	Byte swap=”Swap 1 byte” (or “No swapping”)
	…..	Data length=Value of the “Byte Count” field
	Data (Inputs: H-L order)(H)	Data location=Address in the gateway’s “DI” memory

8.3 "Read Holding Registers" Command (0x03)

Read the binary content of holding registers in the slave. The query message specifies the starting register and quantity of registers to be read. And the register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte.

Frame	Field	Description
Query	Starting Address (Hi)	Address of 1st output register
	Starting Address (Lo)	
	No. of Points (Hi)	Number of output registers
	No. of Points (Lo)	
Response	Byte Count	Number of data bytes=number of output registers x 2
	Data (first register/MSB)	Byte swap="Swap 2 bytes" (or "No swapping") Data length=Value of the "Byte Count" field Data location=Address in the gateway's "AO" memory
	Data (first register/LSB)	
	…..	
	Data (last register/MSB)	
	Data (last register/LSB)	

8.4 "Read Input Registers" Command (0x04)

Read the binary content of input registers in the slave. The query message specifies the starting register and quantity of registers to be read. And the register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte.

Frame	Field	Description
Query	Starting Address (Hi)	Address of 1st input register
	Starting Address (Lo)	
	No. of Points (Hi)	Number of input registers
	No. of Points (Lo)	
Response	Byte Count	Number of data bytes=number of input registers x 2
	Data (first register/MSB)	Byte swap="Swap 2 bytes" (or "No swapping") Data length=Value of the "Byte Count" field Data location=Address in the gateway's "AI" memory
	Data (first register/LSB)	
	…..	
	Data (last register/MSB)	
	Data (last register/LSB)	

8.5 “Force Multiple Coils” Command (0x0F)

Forces each coil in a sequence of coils to either ON or OFF. The query message specifies the coil references to be forced. The normal response returns the slave address, function code, starting address, and quantity of registers preset.

<table>
<tr><th>Frame</th><th>Field</th><th>Description</th></tr>
<tr><td rowspan="8">Query</td><td>Coil Address (Hi)</td><td rowspan="2">Address of 1st output coil</td></tr>
<tr><td>Coil Address (Lo)</td></tr>
<tr><td>Quantity of Coils (Hi)</td><td rowspan="2">Number of output coils</td></tr>
<tr><td>Quantity of Coils (Lo)</td></tr>
<tr><td>Byte Count</td><td>Number of data bytes=(number of output coils + 7) / 8</td></tr>
<tr><td>Force Data (Coils: H-L)(L)</td><td rowspan="3">Byte swap=”Swap 1 byte”
Data length=Value of the “Byte Count” field
Data location=Address in the gateway’s “DO” memory</td></tr>
<tr><td>…..</td></tr>
<tr><td>Force Data (Coils: H-L)(H)</td></tr>
<tr><td rowspan="4">Response</td><td>Coil Address (MSB)</td><td rowspan="2">Address of 1st output coil</td></tr>
<tr><td>Coil Address (LSB)</td></tr>
<tr><td>Quantity of Coils (MSB)</td><td rowspan="2">Number of output coils</td></tr>
<tr><td>Quantity of Coils (LSB)</td></tr>
</table>

8.6 “Preset Multiple Registers” Command (0x10)

Preset values into sequence of holding registers. The query message specifies the register reference to be preset. The normal response returns the slave address, function code, starting address, and quantity of register preset.

<table>
<tr><th>Frame</th><th>Field</th><th>Description</th></tr>
<tr><td rowspan="10">Query</td><td>Starting Address (Hi)</td><td rowspan="2">Address of 1st output register</td></tr>
<tr><td>Starting Address (Lo)</td></tr>
<tr><td>No. of Registers (Hi)</td><td rowspan="2">Number of output registers</td></tr>
<tr><td>No. of Registers (Lo)</td></tr>
<tr><td>Byte Count</td><td>Number of data bytes=number of output registers x 2</td></tr>
<tr><td>Data (first register/MSB)</td><td rowspan="5">Byte swap=”Swap 2 bytes”
Data length=Value of the “Byte Count” field
Data location=Address in the gateway’s “AO” memory</td></tr>
<tr><td>Data (first register/LSB)</td></tr>
<tr><td>…..</td></tr>
<tr><td>Data (last register/MSB)</td></tr>
<tr><td>Data (last register/LSB)</td></tr>
<tr><td rowspan="4">Response</td><td>Starting Address (Hi)</td><td rowspan="2">Address of 1st output register</td></tr>
<tr><td>Starting Address (Lo)</td></tr>
<tr><td>No. of Registers (Hi)</td><td rowspan="2">Number of output registers</td></tr>
<tr><td>No. of Registers (Lo)</td></tr>
</table>

8.7 Exception Responses

When a slave receives the query without a communication error, but cannot handle it, the slave will return an exception response informing the master of the nature of the error. The structure of an exception response is independent of the Modbus command associated with the "Function" field of the query involved. The whole frame of an exception response is shown below.

Slave Address	Modbus address (1 to 247). The value of this field is identical to that of the "Slave Address" field of the query involved.
Function	The value of this field is set to 0x80 + the value of the "Function" field of the query involved.
Exception Code	Code indicating the nature of the error which has caused the exception response
Checksum (Lo)	Error check
Checksum (Hi)	

Code	Name	Meaning
0x01	Illegal Function	The function code received in the query is not an allowable action for the slave.
0x02	Illegal Data Address	The data address received in the query is not an allowable address for the slave.
0x03	Illegal Data Value	A value contained in the query data field is not an allowable value for the slave.
0x04	Slave Device Failure	An unrecoverable error occurred while the slave was attempting to perform the requested action.
0x05	Acknowledge	The slave has accepted the request and is processing it, but a long duration of time will be required to do so. The gateway should transmit subsequent queries in order to determine whether the request has finished or not.
0x06	Slave Device Busy	The slave is engaged in processing a long–duration program command. So the gateway should re-transmit the query subsequently.
0x07	Negative Acknowledge	The slave cannot perform the program function received in the query. This exception only affects commands 0x0D and 0x0E.
0x08	Memory Parity Error	The slave attempted to read extended memory, but detected a parity error in the memory. This exception only affects standard commands 0x14 and 0x15.

9 Application with PISO-CAN 200/400-T

In this chapter, we describe the I-7242D application in DeviceNet network with two demo programs. ICP DAS DeviceNet Master Library (DLL functions) provides users to establish DeviceNet network rapidly by Master/Slave connection model. Using the library, users don't need to take care of the detail of the DeviceNet protocol. The library will implement the DeviceNet protocol automatically.

In these demo programs, the master device is the PISO-CAN200/400-T of ICP DAS and the I-7242D is a slave device in the DeviceNet network. These demo programs implemented by applying DeviceNet lib of PISO-CAN200/400-T PCI card. Thus, the following diagram shows how to apply the I-7242D in easy way. The architecture is depicted as figure 9-1.

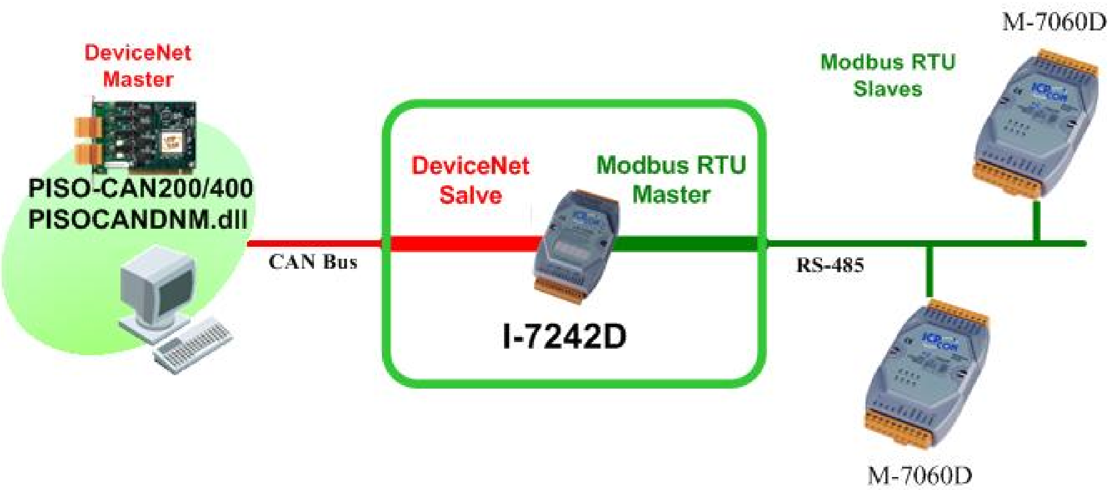


Figure 9-1 Architecture of the demos in PISO-CAN200/400 PCI card

The information of devices and software in these applications is below:

- **Hardware:**

DeviceNet master: PISO-CAN 200/400-T
DeviceNet slave: I-7242D
Modbus RTU devices: M-7060D, M-7060D

- **Software:**

The demos in PISO-CAN200/400 PCI card

- **The DeviceNet information in the I-7242D**

Device Information

Application Instance ID	Device Address	Device I/O Type	Relay/Register Start Address	Relay/Register Data Length
0x01	0x04	0 (DO)	1	4
0x02	0x05	2 (DI)	1	4

"User-defined Modbus Device Object" Instance 1

Attribute ID	Description	Data Type	Method	Value
0x01	Device ID	CHAR	Get	4
0x02	Device I/O Type	CHAR	Get	0
0x03	Device Start Address	WORD	Get	1
0x04	Device Length	WORD	Get	4
0x05	Data Lose Counter	WORD	Get/Set	0
0x14	DO Data	Defined by device num.	Get/Set	0
0x15	AO Data	Defined by device num.	Get/Set	0
0x16	DI Data	Defined by device num.	Get	0
0x17	AI Data	Defined by device num.	Get	0

"User-defined Modbus Device Object" Instance 2

Attribute ID	Description	Data Type	Method	Value
0x01	Device ID	CHAR	Get	5
0x02	Device I/O Type	CHAR	Get	2
0x03	Device Start Address	WORD	Get	1
0x04	Device Length	WORD	Get	4
0x05	Data Lose Counter	WORD	Get/Set	0
0x14	DO Data	Defined by device num.	Get/Set	0
0x15	AO Data	Defined by device num.	Get/Set	0
0x16	DI Data	Defined by device num.	Get	0
0x17	AI Data	Defined by device num.	Get	0

Refer to the application object instances. The I-7242D will define the default assembly object instances according to the following table.

Assembly Object Instance ID	Data Length (Byte)	Component devices (ID, Address)
0x64	DO: 1	1(00004~00001)
0x65	DI: 1	3(10004~10001)

Please do the following two steps to setup the system before you execute these application programs.

Step 1:

Setup the I-7242D and these Modbus devices parameters by using the DNS_MRU Utility. Here, the MAC ID and CAN baud rate of I-7242D is set as 3 and 125Kbps. The COM port parameters is set as 9600,N,8,1. The Modbus devices parameters and DeviceNet I/O connections are set as figure 9-3 and 9-4 and 9-5.

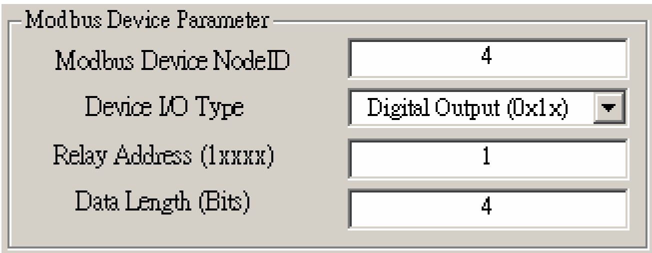


Figure 9-3 Device_1 parameters

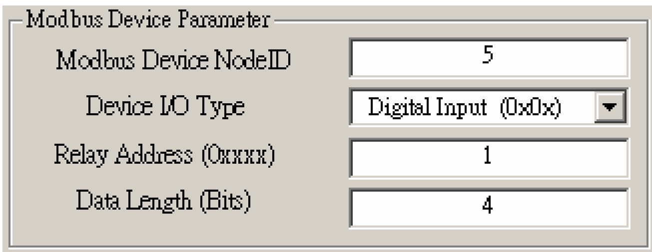


Figure 9-4 Device_2 parameters

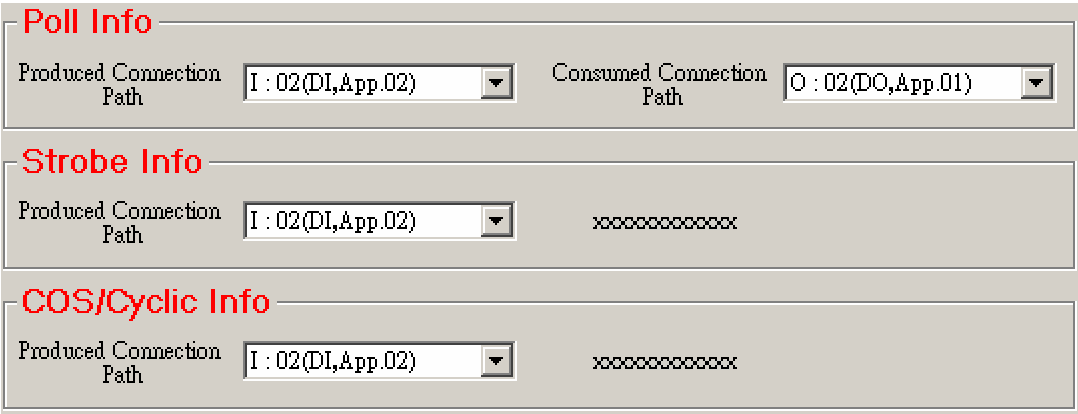


Figure 9-5 DeviceNet I/O connections

Step 2:

Connect the CAN port of PISO-CAN card with the I-7242D. And then connect Com2 port of the I-7242D with these Modbus RTU devices.

9.1 Application 1

The demo1 program shows users how to set/get the attribute value in the I-7242D. The I/O data of the Modbus devices also can be driven in the same way. The frame of demo1 program is shown as figure 9-2. Please do the following steps to apply the I-7242D in DeviceNet network.

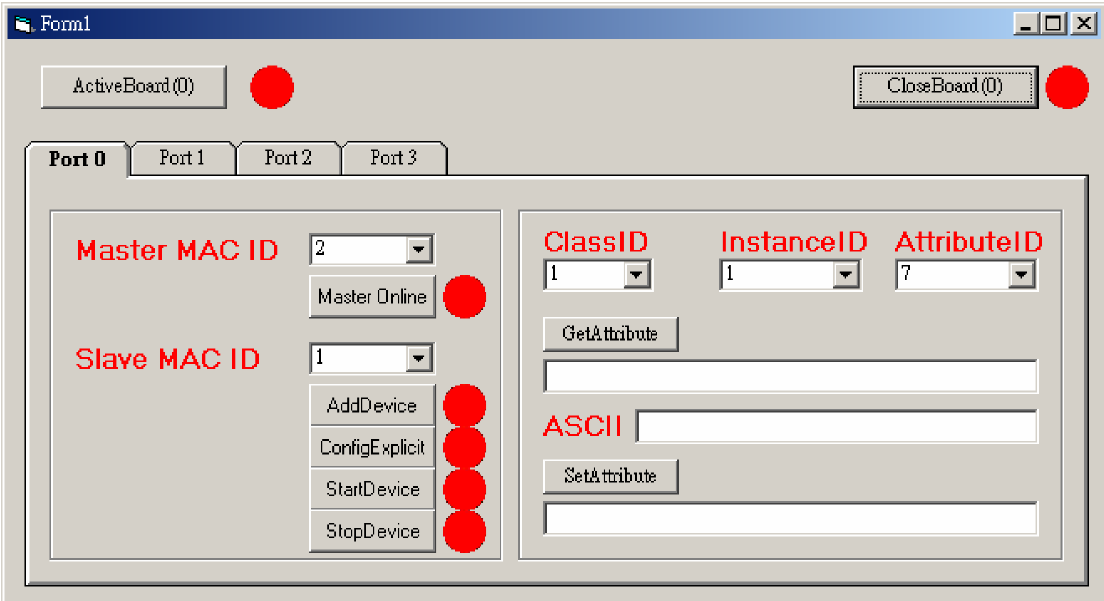


Figure 9-2 The frame of Demo1 program

Step 1:

Start to run the demo program. And first, you must to active the PISO-CAN board by clicking “Active board” button.

Step 2:

Select “Master MAC ID” as the Master ID and “Slave MAC ID” according to the I-7242D’s MAC ID in the DeviceNet network. The master device need to be in on-line mode by clicking the “Master Online” button. The next, add the I-7242D to the list in PISO-CAN200/400 by clicking the “AddDevice” button. Then, configure the explicit connection by clicking the “ConfigExplicit” button. Finally, start to communicate with I-7242D by “StartDevice” button.

Step 3:

When completing the above steps, users can get/set the attribute value supported in I-7242D in the program. Please refer to chapter 4 to understand what attributes provided by the I-7242D.

Example:

In the application, we should set Master MAC ID as 0x05 and Slave MAC ID as 0x03. Then, we try to set/get the data of attribute 0x14 of instance 0x01 of class 0x64.

Press the “GetAttribute” button to get the data of attribute 20 of Instance 1 of class 100. The result is shown as figure 9-3.

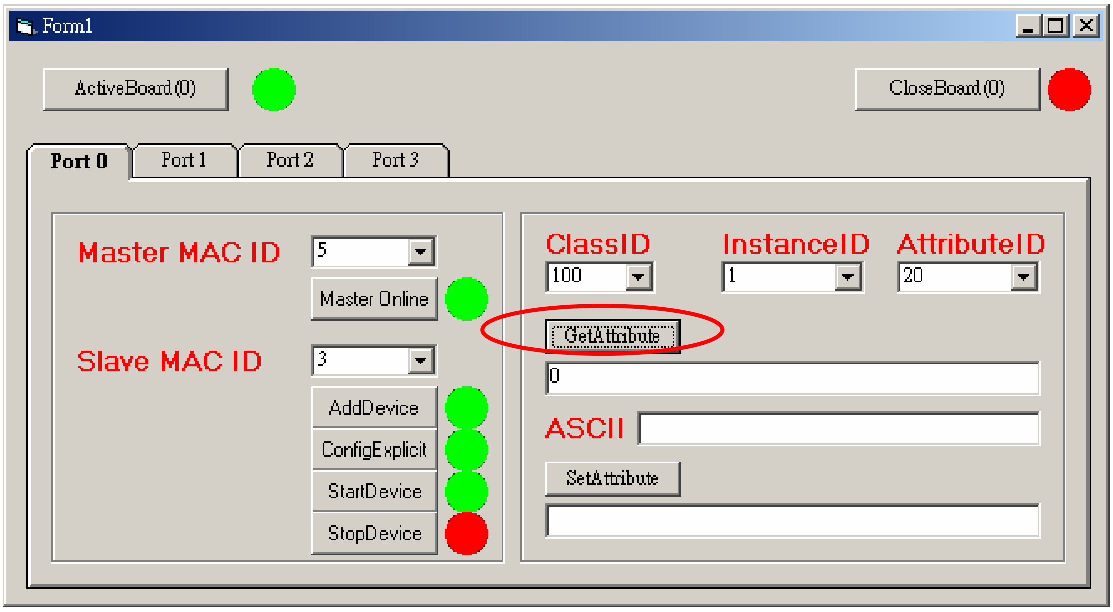


Figure 9-3 “GetAttribute”

Press the “SetAttribute” button to set the needed data to attribute 20 of Instance 1 of class 100. The result is shown as figure 9-4, .

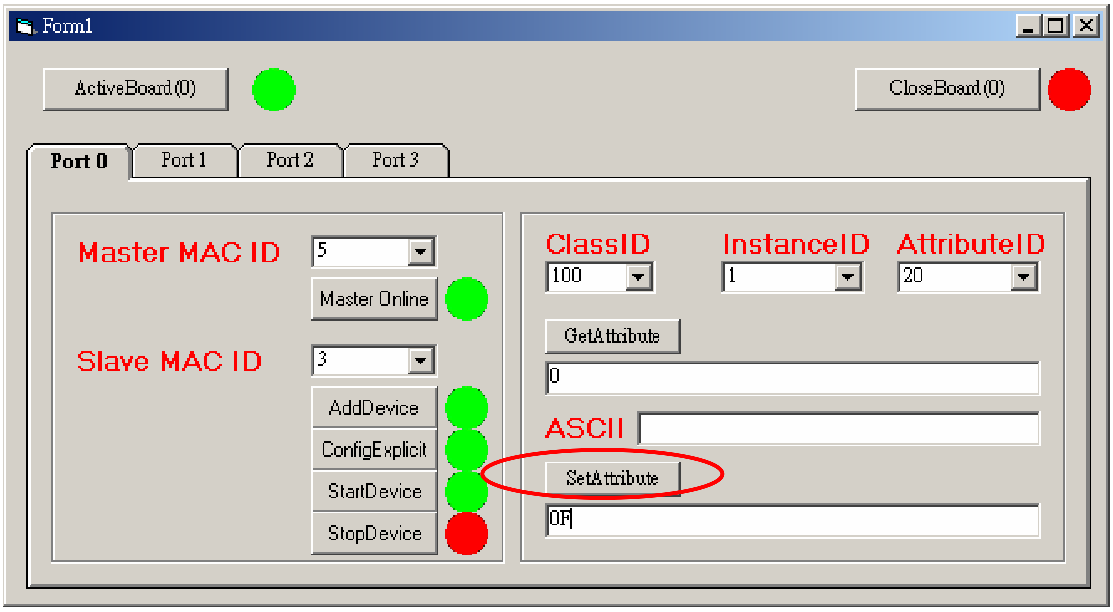


Figure 9-3 “SetAttribute”

If users want to stop communication with I-7242D, they can click the "StopDevice" button or "CloseBoard(0)" button. See figure 9-5 and figure 9-6.

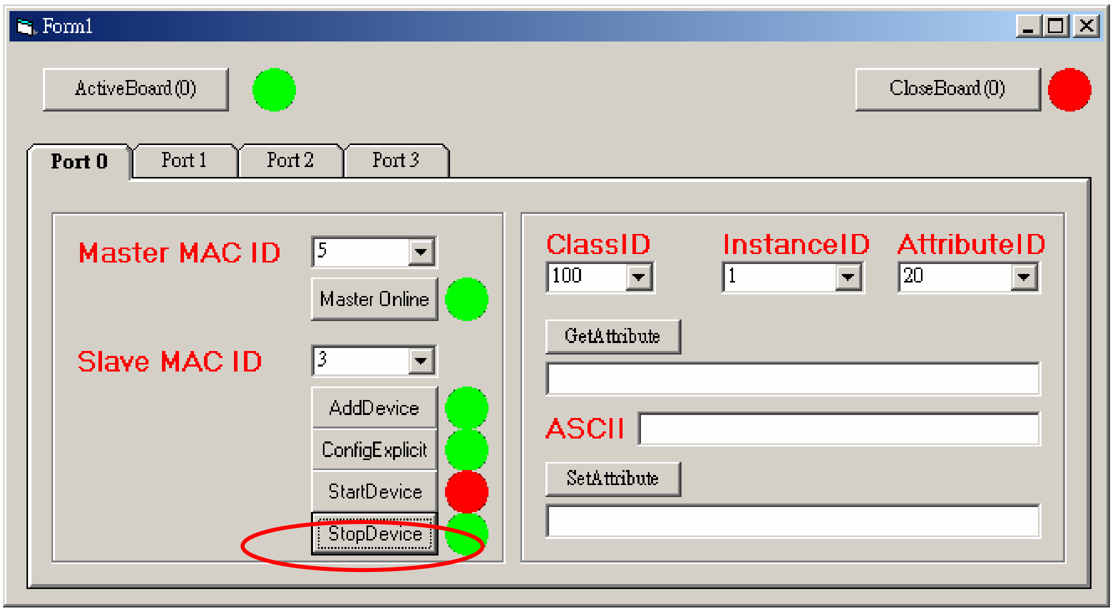


Figure 9-5 "StopDevice"

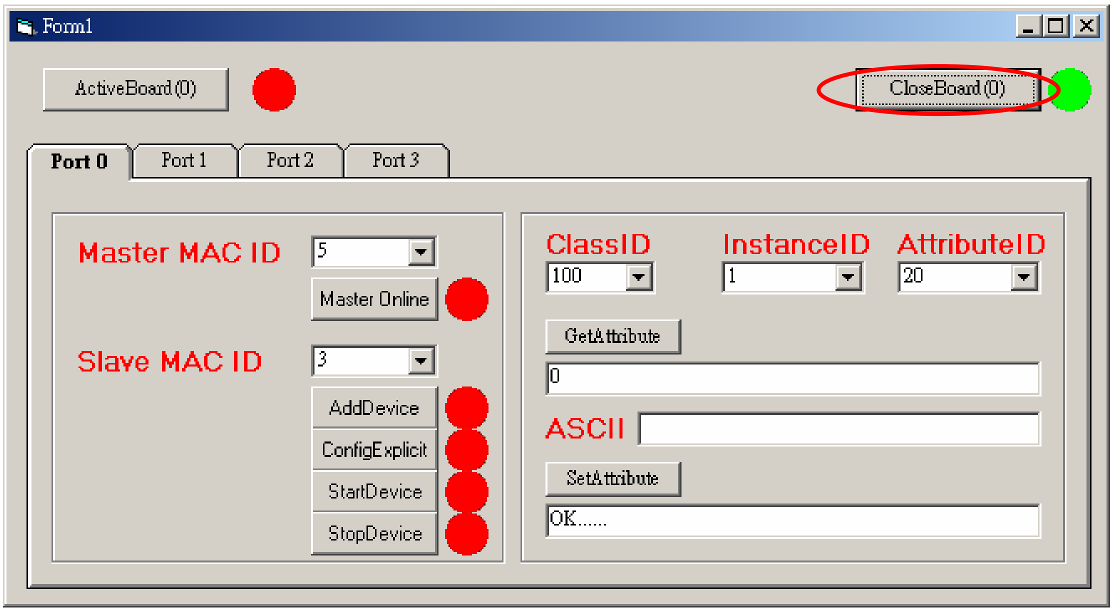


Figure 9-6 "CloseBoard(0)"

9.2 Application 2

The demo2 program can show the I/O communication with I-7242D by the polling, Bit-strobe and Cyclic/COS connections. Users can apply this demo in a real DeviceNet network to the system. The frame of program is shown as figure 9-6. Please do the following steps to apply the I-7242D in DeviceNet network.

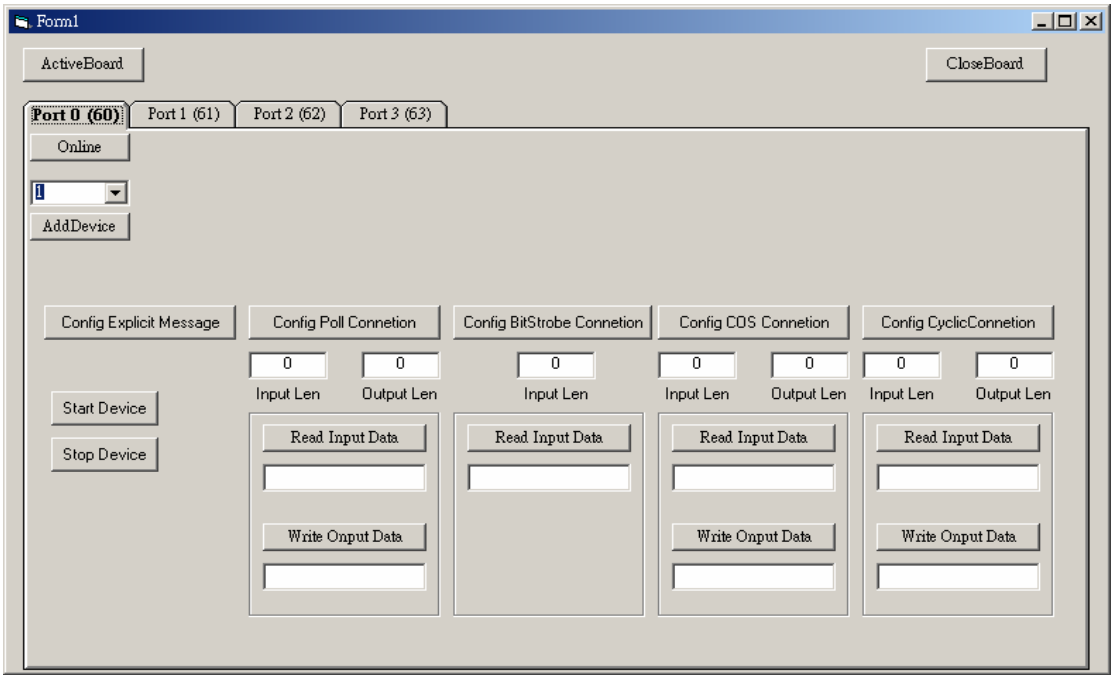


Figure 9-6 The frame of Demo2 program

Step 1:

Start to run the demo program. Firstly, you must to active board by clicking the "Active board" button.

Step 2:

Select the necessary port number and the slave's MAC ID according to the I-7242D in the DeviceNet work. The master device must be on-line by clicking the "Master Online" button. And add the I-7242D to the list in PISO-CAN200/400 by clicking "AddDevice" button. The users should check what the IO connection supported in the slave device. And set the input and output data length according to the IO connections path defined in DNS_MRU Utility. By pressing the "Config XXX Connection" button to finish the configuration. Then press the "start device" button to start to communicate with I-7242D via the I/O connection.

Step 3:

When completing the above steps, the polling, Bit-strobe and Cyclic/COS connections can be used to communicate with I-7242D. The result is shown as figure 9-7. Therefore, the DO LED displays in the M-7060 will change, if the “write Output data” filed is given by different value as figure 9-7.

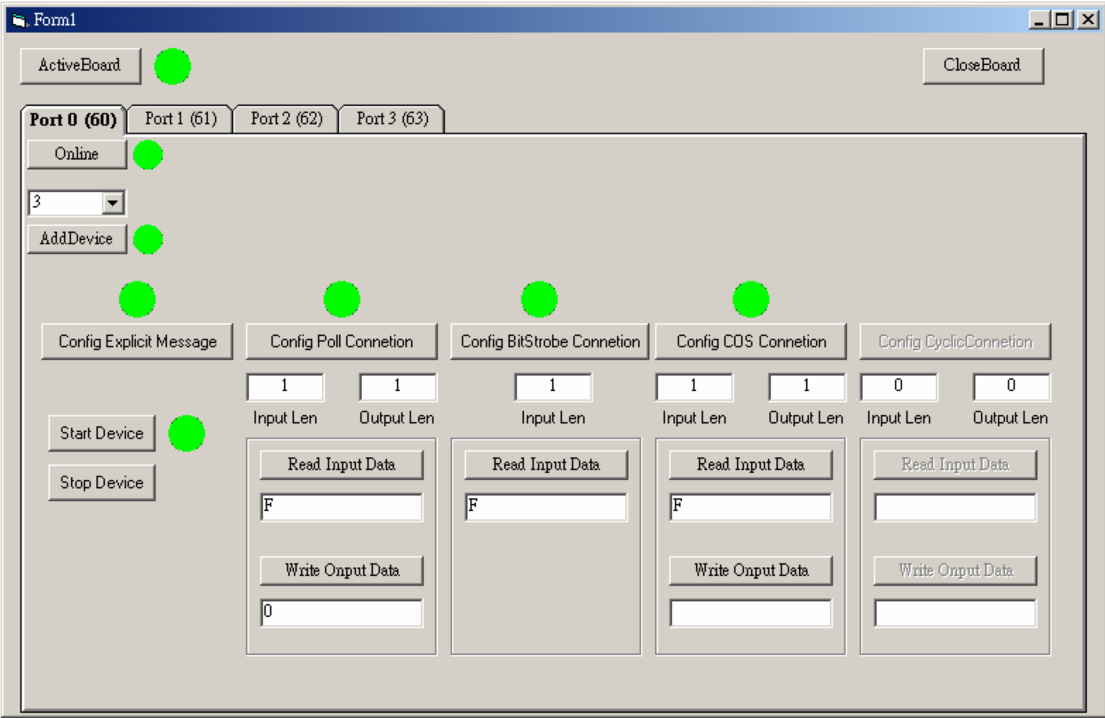


Figure 9-7 Using I/O Connection to communicate with I-7242D

If users want to stop communication with I-7242D, they can click the "Stop Device" button or "CloseBoard" button. See figure 9-8 and figure 9-9

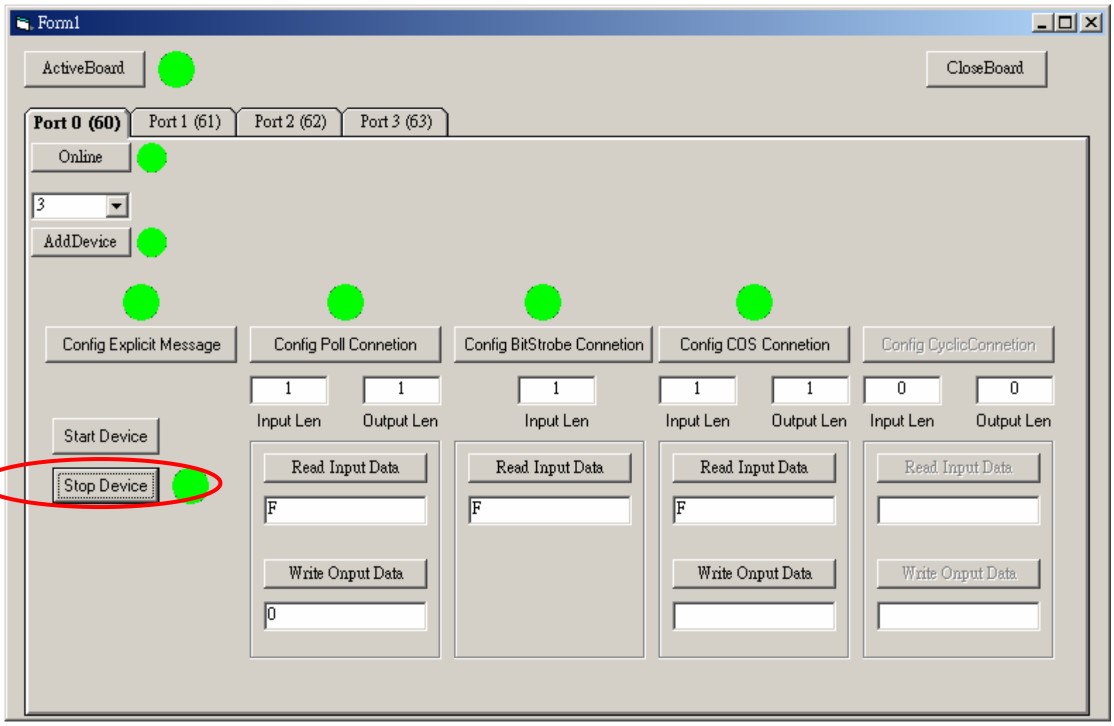


Figure 9-8 "Stop Device"

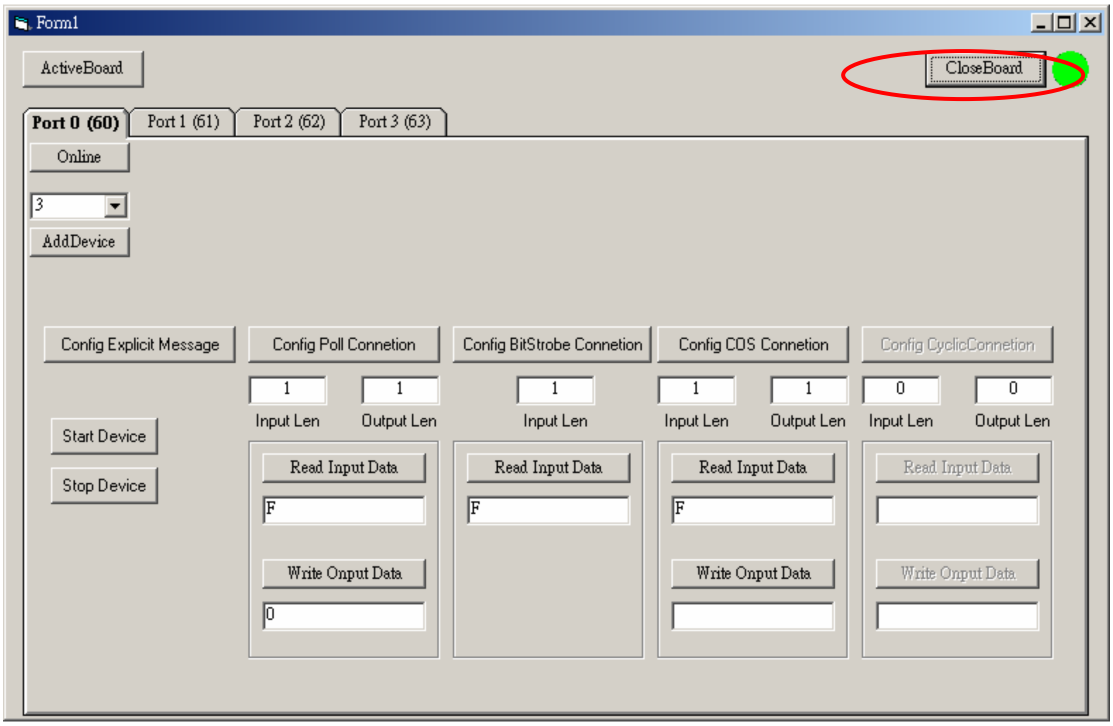
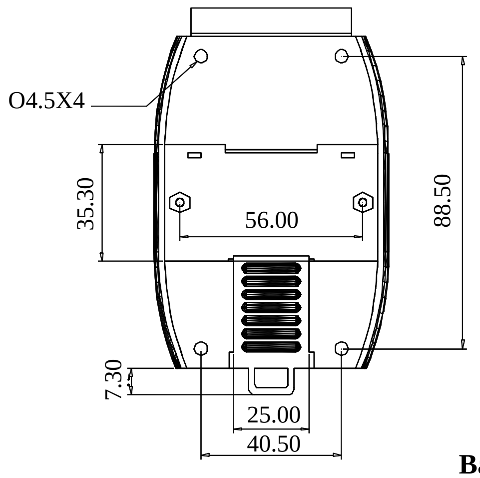
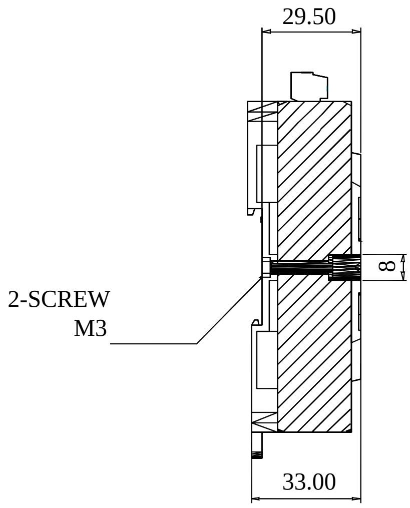


Figure 9-9 "CloseBoard"

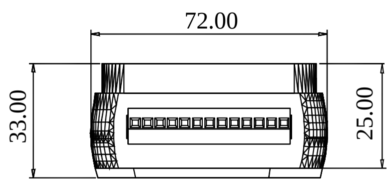
Appendix A: Dimension and Mounting



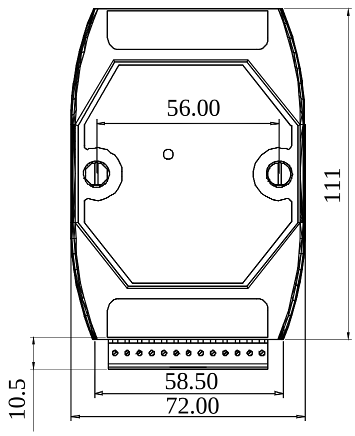
Back View



Side View



Top View



From View

Unit : mm