EDAM-4200 Digital I/O series

Data Acquisition Modules
User's Manual

Web site: **www.inlog.com.tw**

Table of Contents

Chapter 1 Product Overview

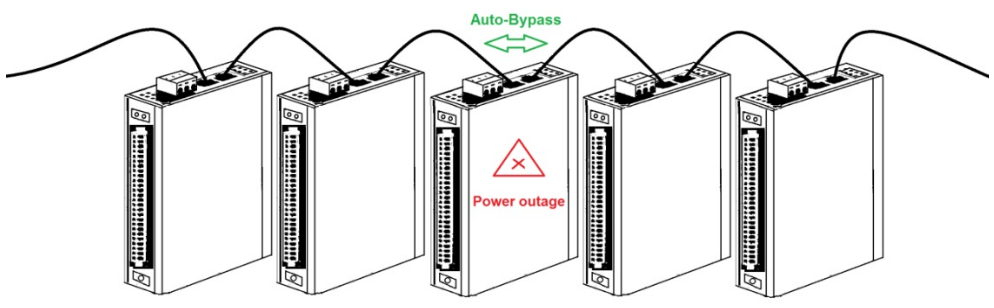
1.1 Introduction

EDAM-4200 is based on the popular Ethernet networking standards used today in most business environments. Users can easily add EDAM-4200 I/O modules to existing Ethernet networks or use EDAM-4200 modules in new Ethernet-enabled Manufacturing networks. EDAM-4200 module features a 10/100 Mbps Ethernet switching chip to allow *Daisy Chain connections* in an Ethernet network, making it easier to deploy, and supports industrial popular Modus/TCP protocol over TCP/IP for data connection. EDAM-4200 also supports UDP protocol over Ethernet networking. With UDP/IP, EDAM-4200 I/O modules can actively send I/O data stream to 8 Ethernet nodes. Through Ethernet networking HMI/SCADA system and controller can access or gather real-time data from EDAM-4200 Ethernet enabled DA&C modules. And, these real-time data can be integrated with business system to create valuable, competitive business information immediately.

1.2 Daisy Chain connection with auto-bypass protection

EDAM-4200 module has built in two-port Ethernet switches to allow daisy chain connections in an Ethernet network, making it easier to deploy, and helping improve scalability. The two Ethernet ports are fully compliant with IEEE 802.3u 10/100Mbpst through standard RJ-45 connectors Although daisy chain topology brings attractive benefits for users, it still comes with the risk that once any device in the daisy-chain network suffers power outage, it will cause the disconnection of all devices downstream, to prevent this critical issue from happening, inLog especially refined the hardware design of EDAM-4200 so that it(Auto-bypass protection) can rapidly recover the network connection in about 1 seconds, Therefore, the damage will be greatly minimized.

EDAM-4200 Series



Daisy-Chain Ethernet Cabling

Note: ***Auto-Bypass Protection*** feature guarantees the Ethernet communication. It will automatically active to continue the network traffic when the EDAM-4200 modules loses its power after 2 second.

1.3 Mixed I/O in One Module to fit all applications

EDAM-4200 mixed I/O module design concept provides the most cost-effective I/O usage for application system. The most common used I/O type for single function unit are collected in ONE module. This design concept not only save I/O usage and spare modules cost but also speed up I/O relative operations. For small DA&C system or standalone control unit in a middle or large scale, EDAM-4200 mixed I/O design can easily fit application needs by one or two modules only. With additional embedded control modules, EDAM-4200 can easily create a localized, less complex, and more distributed I/O architecture.

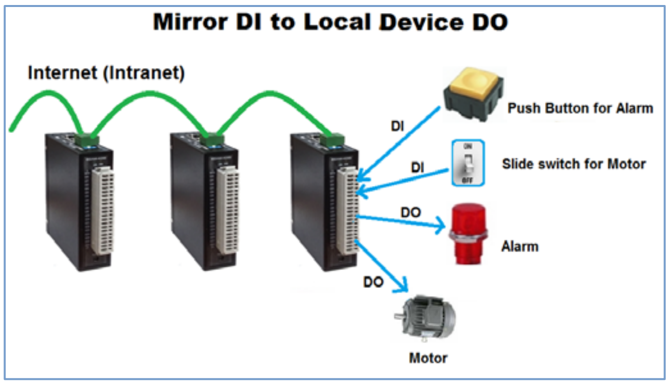
1.4 Industrial standard Modbus/TCP protocol supported for open connectivity

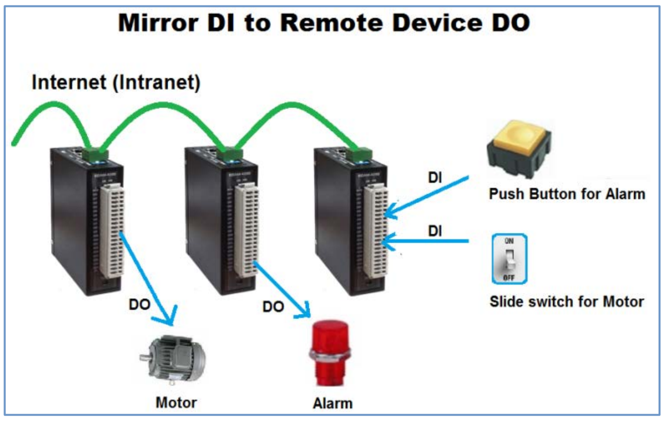
EDAM-4200 modules support the popular industrial standard, Modbus/TCP protocol, to connect with Ethernet Controller or HMI/SCADA software built with Modbus/TCP driver. Inlog also provides OPC server for Modbus/TCP to integrate EDAM-4200 I/O real-time data value with OPC client enabled software. Users don't need to take care of special driver's development.

1.5 Features

1.5.1 DIO Synchronization (Mirror DI to Local /Remote Device DO)

EDAM-4200 series modules also provide a *DIO Synchronization* function. A Local/Remote device single digital output channel can be activated (1 or 0) dependent on the digital input channels value. When the specific DI channels value changed from "match" to "mismatch" (or "mismatch" to "match")DI mask pattern, the Local/Remote device corresponding DO will be set to active state(1 or 0) dependent on the DO setting (For more detailed information refer to "Appendix").

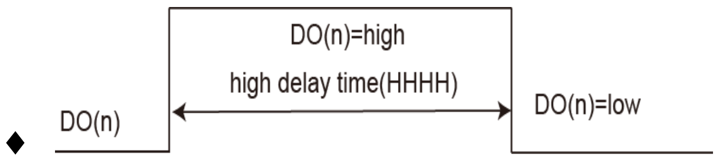




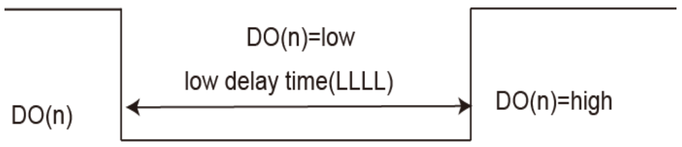
1.5.2 DO Auto-Off Time Mode

This function is used to force the specific DO channel to work as a monostable operation. After a certain period of time, the DO returns to the stable state until another triggering command is applied.
(For more detailed information refer to "Appendix").

♦ **Low to High DO(n) pulse output**



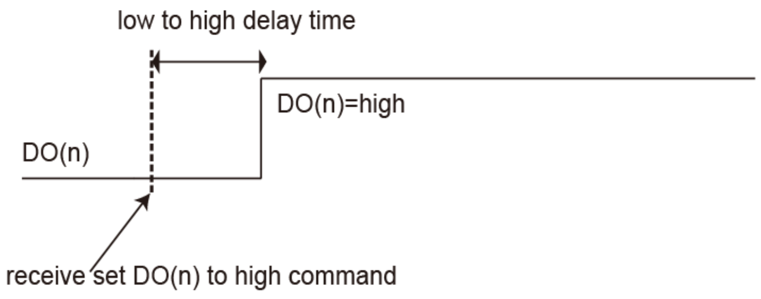
♦ **High to Low DO(n) pulse output**



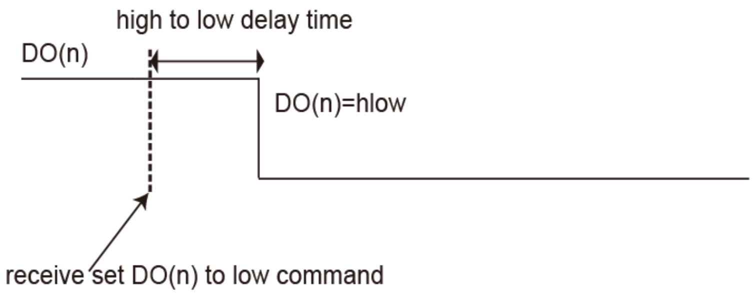
1.5.3 High/Low delay output mode

EDAM-4200 series modules supports high-to-low and low-to-high delay output function.
(For more detailed information refer to "Appendix").

- Low to High Delay output:



- High to Low Delay output:



1.5.4 Power-on & Safe valu

- Power-on value:

Power-on value is used to set the module default output value when the module is turned-on or watch dog timeout reset. This function is especially importance in some application where the specific initial output states are required User can set power on value by sending Set power-on/safe value command

- Safe value:

Safe value are used to set the module outputs into the specific values when Host watchdog timeout If The host watchdog timer is enabled by sending Set host watchdog timeout value, the host should send Host OK command periodically within Timeout value to refresh the timer, otherwise the module will be forced to safety state.

1.6 Common technical specification of EDAM-4200

- Ethernet: 10 BASE-T IEEE 802.3 100 BASE-TX IEEE 802.3u
- Wiring: UTP, category 5 or greater
- Bus Connection: Two-port RJ45 modular jack(Auto-bypass protection)
- Comm. Protocol: Modbus/TCP on TCP/IP and UDP
- Data Transfer Rate: Up to 100 Mbps
- Unregulated 10 to 30VDC
- Protection: Over-voltage and power reversal
- Status Indicator: Power, CPU, Communication (Link, Collide, 10/100 Mbps, Tx, Rx)
- Mounting: DIN rail or wall
- Wiring: I/O cable 14 to #28 AWG wire for terminal block.
- Operating Temperature: - 10 to 70º C (14 to 158º F)
- Storage Temperature: - 25 to 85º C (-13 to 185º F)
- Humidity: 5 to 95%, non-condensing
- Atmosphere: No corrosive gases

NOTE:
Equipment will operate below 30% humidity. However, static electricity problems occur much more frequently at lower humidity levels. Make sure you take adequate precautions when you touch the equipment. Consider using ground straps, anti-static floor coverings, etc. if you use the equipment in low humidity environments.

1.7 Software Support

Based on the Modbus/TCP standard, the EDAM-4200 firmware is a built-in Modbus/TCP server. Therefore, Inlog provides the necessary DLL drivers, and Windows Utility for users for client data for the EDAM-4200. Users can configure this DA&C system via Windows Utility; integrate with HMI software package via Modbus/TCP driver or Modbus/TCP OPC Server. Even more, you can use the DLL driver and ActiveX to develop your own applications.

1.8 Package Information

The package of EDAM-4200 series module will contain the following items. Please check and feel free to contact us if any part missing or damaged after purchasing EDAM-4200 product.

- EDAM-4200 module (assembled with DIN Rail)
- Product CD
- Panel mounting bracket
- Start-up manual

1.9 Product Warranty (1 years)

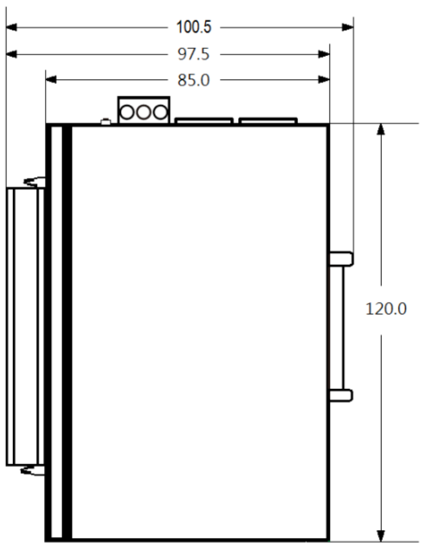
Inlog warrants to you, the original purchaser, that each of its products will be free from defects in materials and workmanship for one year from the date of purchase. This warranty does not apply to any products which have been repaired or altered by persons other than repair personnel authorized by Inlog, or which have been subject to misuse, abuse, accident or improper installation. Inlog assumes no liability under the terms of this warranty as a consequence of such events.

Because of Inlog's high quality-control standards and rigorous testing, most of our customers never need to use our repair service. If an Inlog product is defective, it will be repaired or replaced at no charge during the warranty period. For out-of-warranty repairs, you will be billed according to the cost of replacement materials, service time and freight. Please consult your dealer for more details.

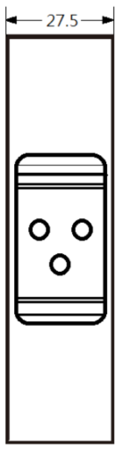
1.10 Dimensions

The following diagrams show the dimensions of the EDAM-4200 I/O module in millimeters.

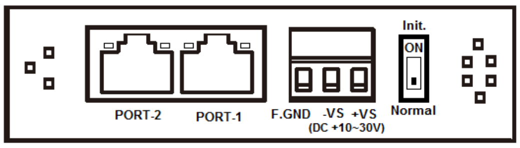
♦ **EDAM-4200 series**



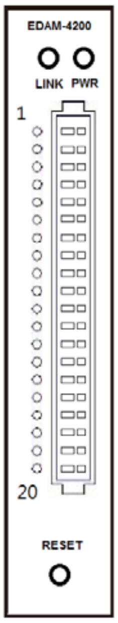
Side View



Rear View



Top View



Front View

1.11 Summary of DIO modules

The EDAM-4200 provides a series of digital input or output modules to sense the digital signal or to control the remote devices.

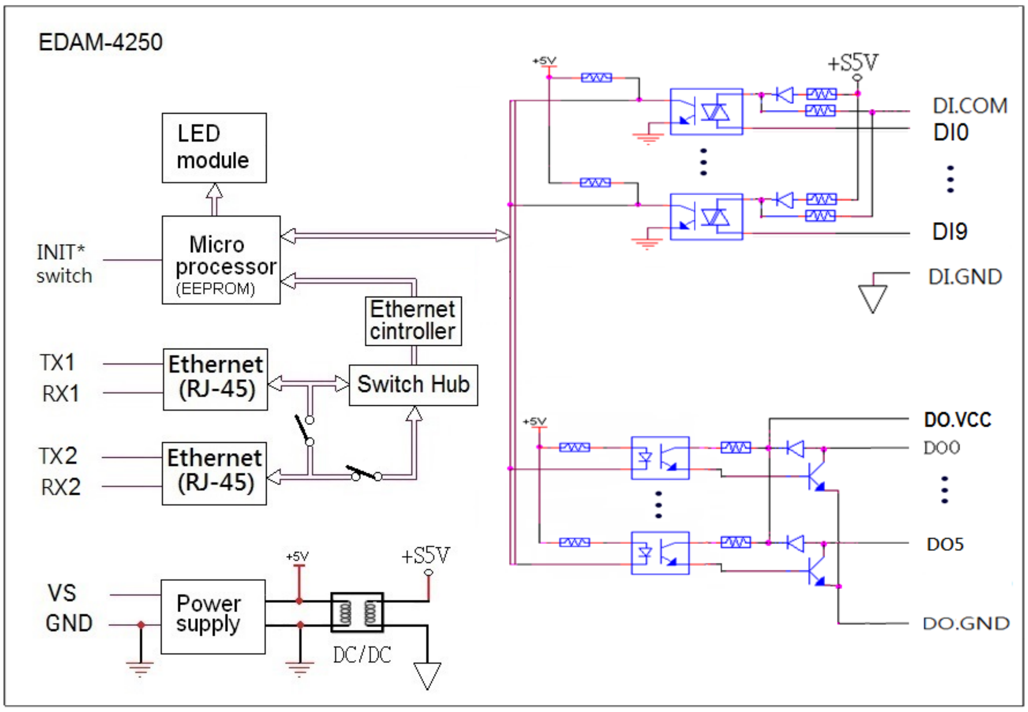
DC Input and DC Output modules

Module	DI ch.	Input type	DO ch.	Output type	Pg.
4250	10	Isolated single ended with Dry/Wet Contact(common source or ground)	6	Isolation with Open collector (NPN)	15
4251	10	Isolated single ended with Dry Contact (common ground)	4	Isolation with Open collector (NPN)	16
	2	Iso. with differential counter input			
4251A	10	Isolated single ended with Wet Contact(common source)	4	Isolation with Open collector (NPN)	17
	2	Iso. with differential counter input			
4254	5	differential digital input, (common source or ground)	8	Isolated with open drain (P-MOSFET), 750mA/channel.	18
4255	8	Isolated single ended with Dry/Wet Contact(common source or ground)	8	Isolated with open drain (P-MOSFET), 750mA/channel.	19
4260	7	Isolated single ended with Dry Contact (common source).	4	Relay output,0.6A@125VAC/2A@30VDC, RL0,RL1,RL2 Form C and RL3 Form A.	20
4264	4	differential digital input, (common source or ground)	4	Relay output,0.6A@125VAC/2A@30VDC, RL0,RL1,RL2 Form C and RL3 Form A.	21

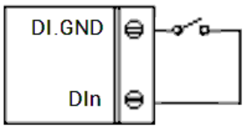
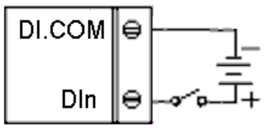
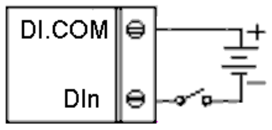
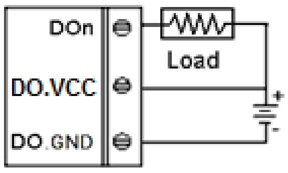
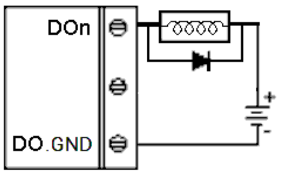
Chapter 2 Block diagram of DIO modules

2.1 EDAM-4250

2.1.1 Block diagram

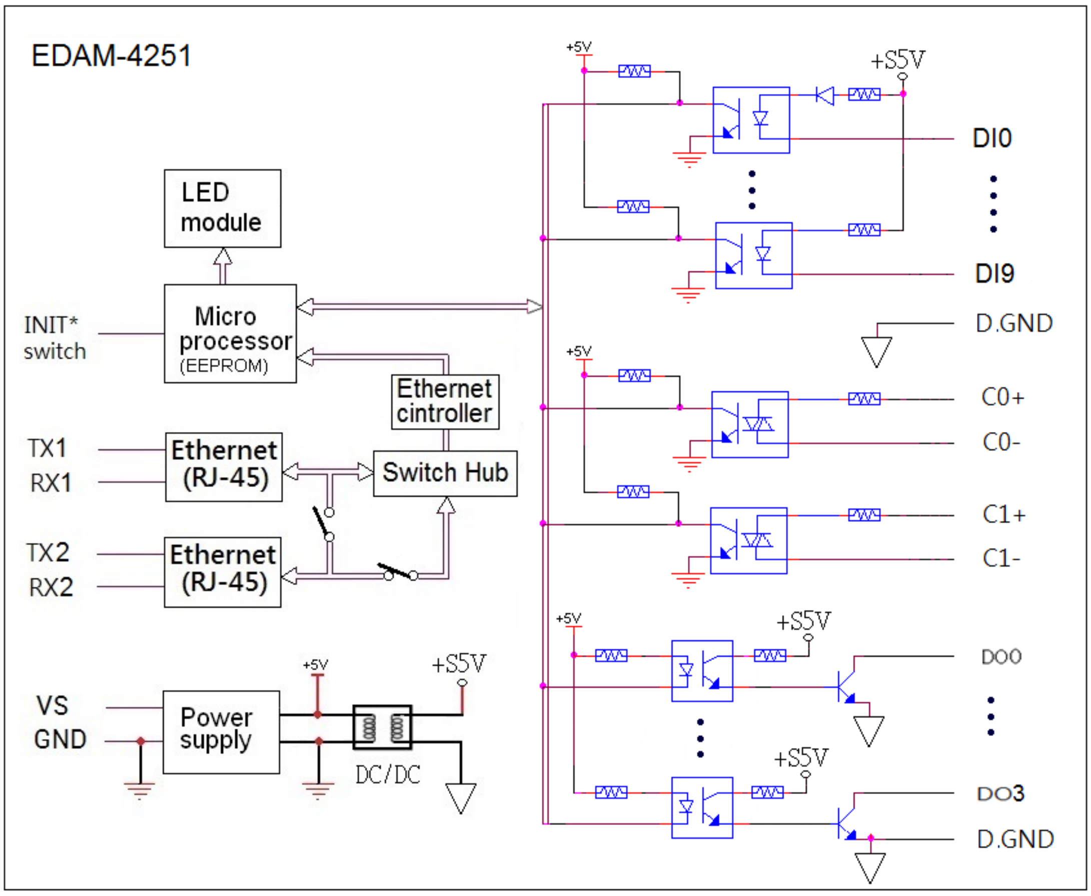


2.1.2 Wire connection

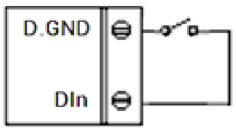
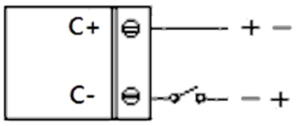
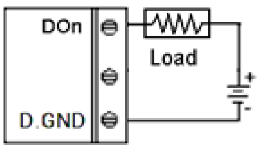
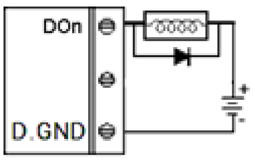
Dry Contact	
	
Wet Contact	
	
Note: To use wet contact , the DI.GND pin must be opened.	
Open collector output	
	

2.2 EDAM-4251

2.2.1 Block diagram

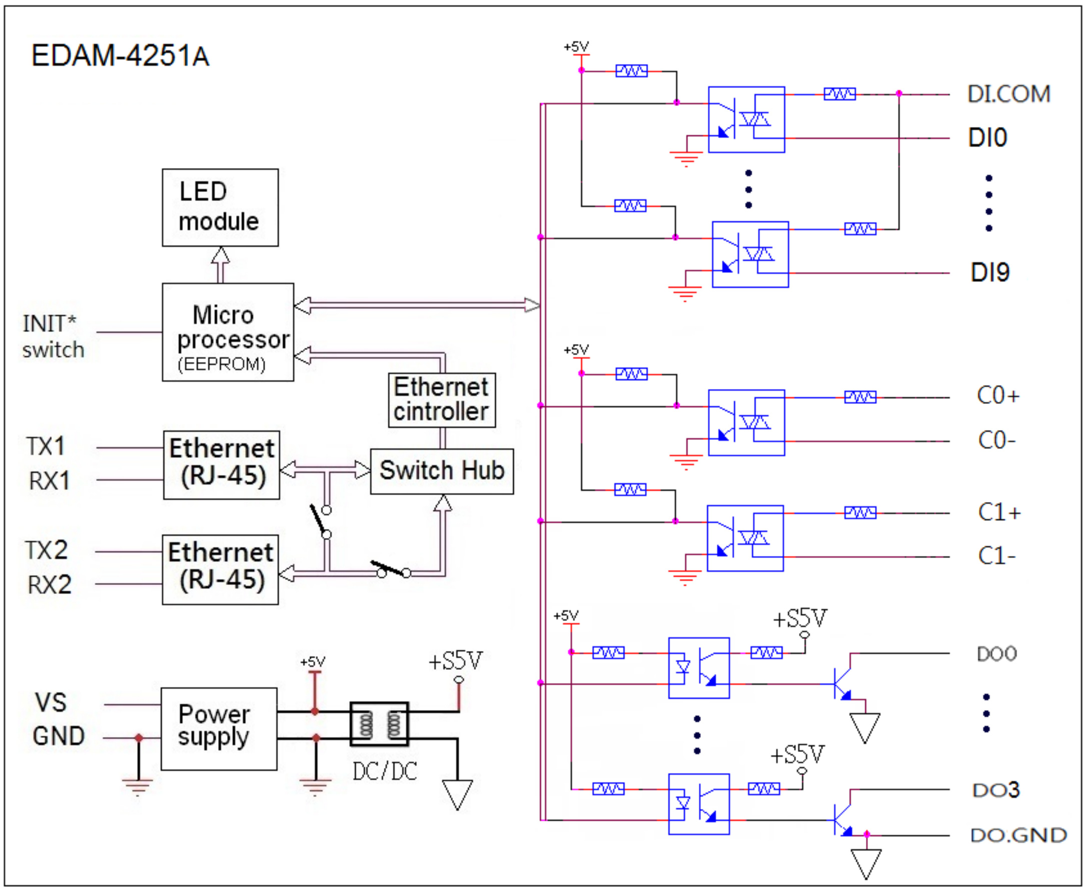


2.2.2 Wire connection

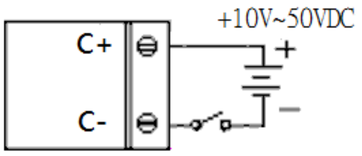
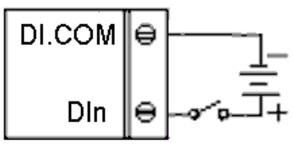
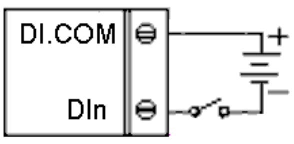
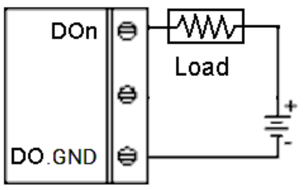
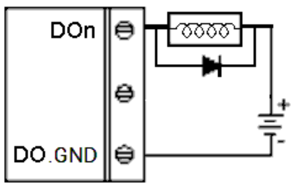
Dry Contact	Counter Input
	
Open collector output	
	

2.3 EDAM-4251A

2.3.1 Block diagram

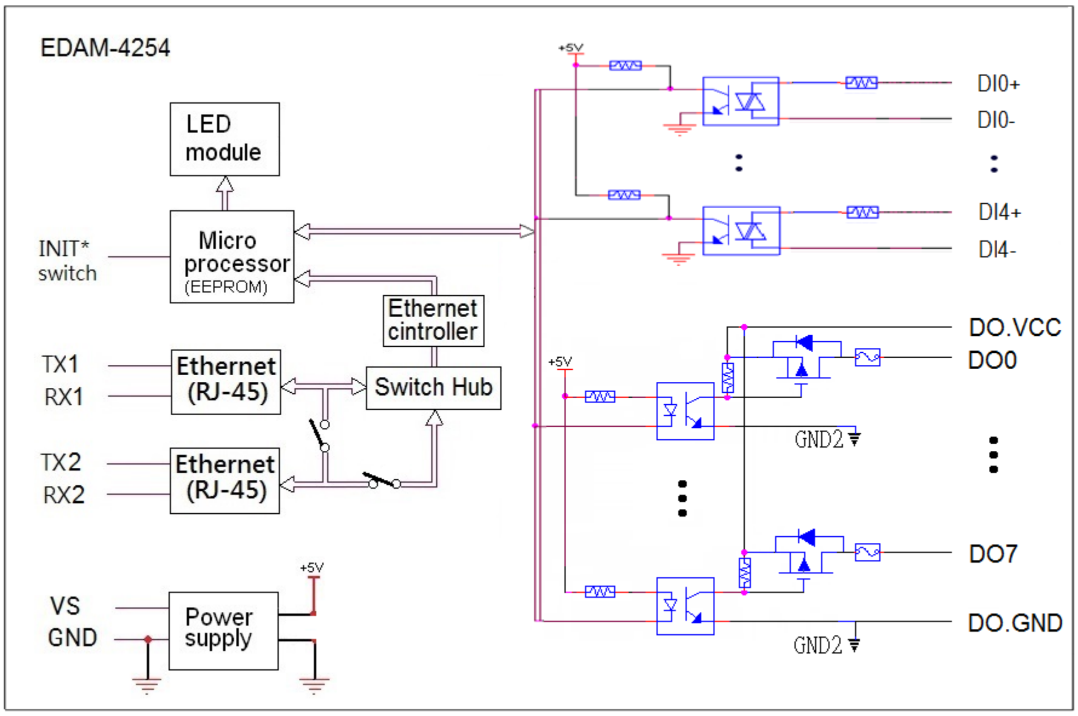


2.3.2 Wire connection

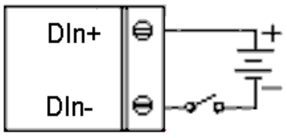
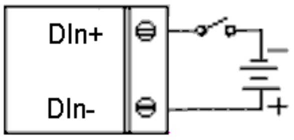
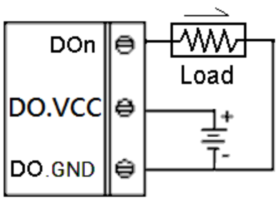
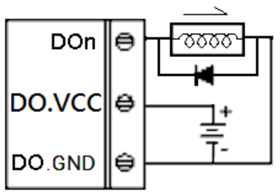
Counter Input	
	
Wet Contact	
	
Open collector output	
	

2.4 EDAM-4254

2.4.1 Block diagram

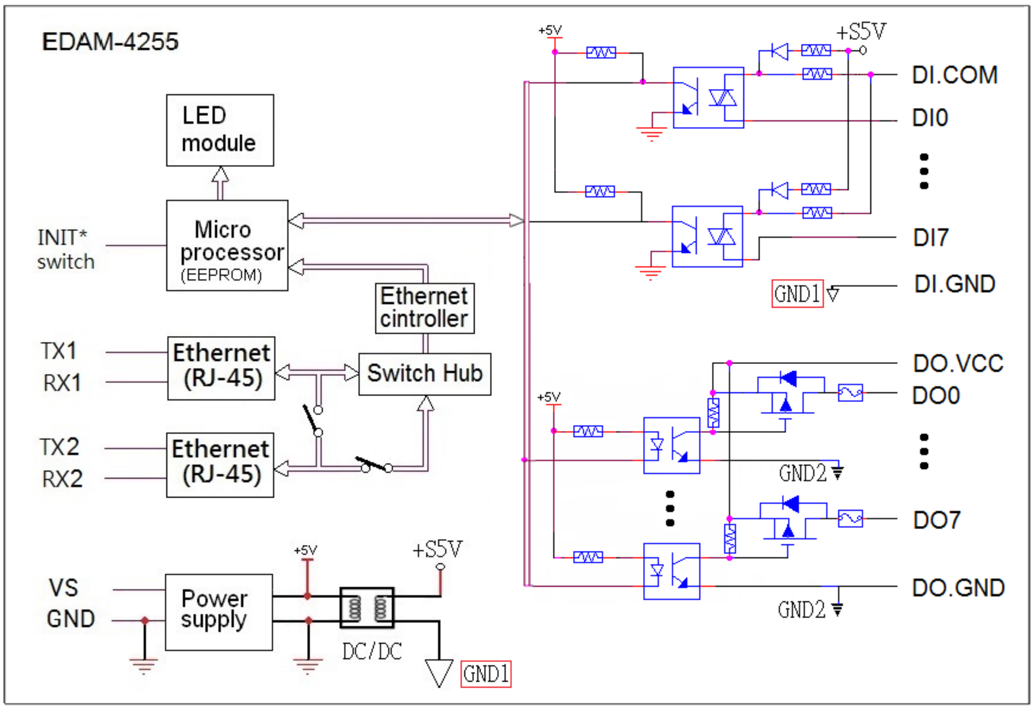


2.4.2 Wire connection

Wet Contact	
	
Digital output	
	

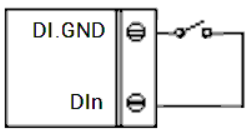
2.5 EDAM-4255

2.5.1 Block diagram

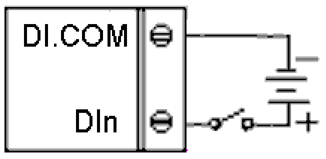


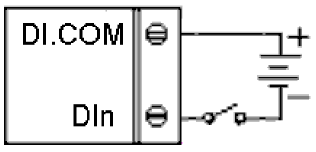
2.5.2 Wire connection

Dry Contact



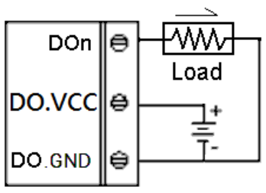
Wet Contact

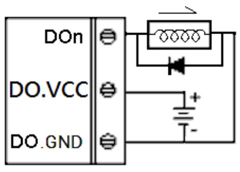




Note: To use wet contact , the DI.GND pin must be opened.

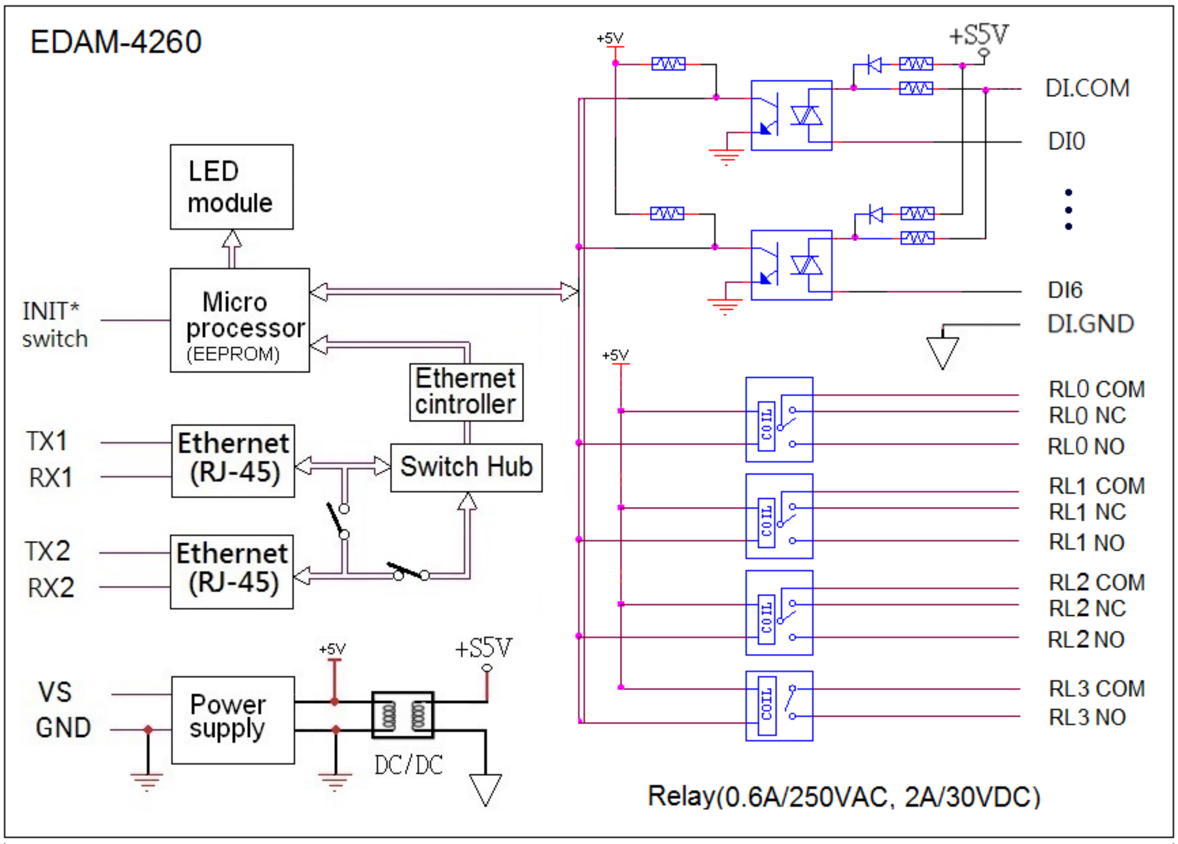
Open collector output



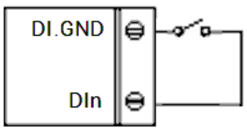
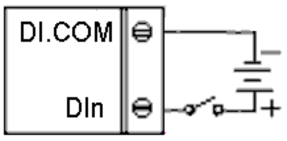
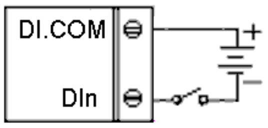
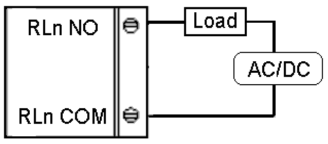
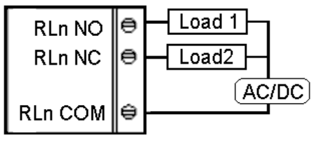


2.6 EDAM-4260

2.6.1 Block diagram

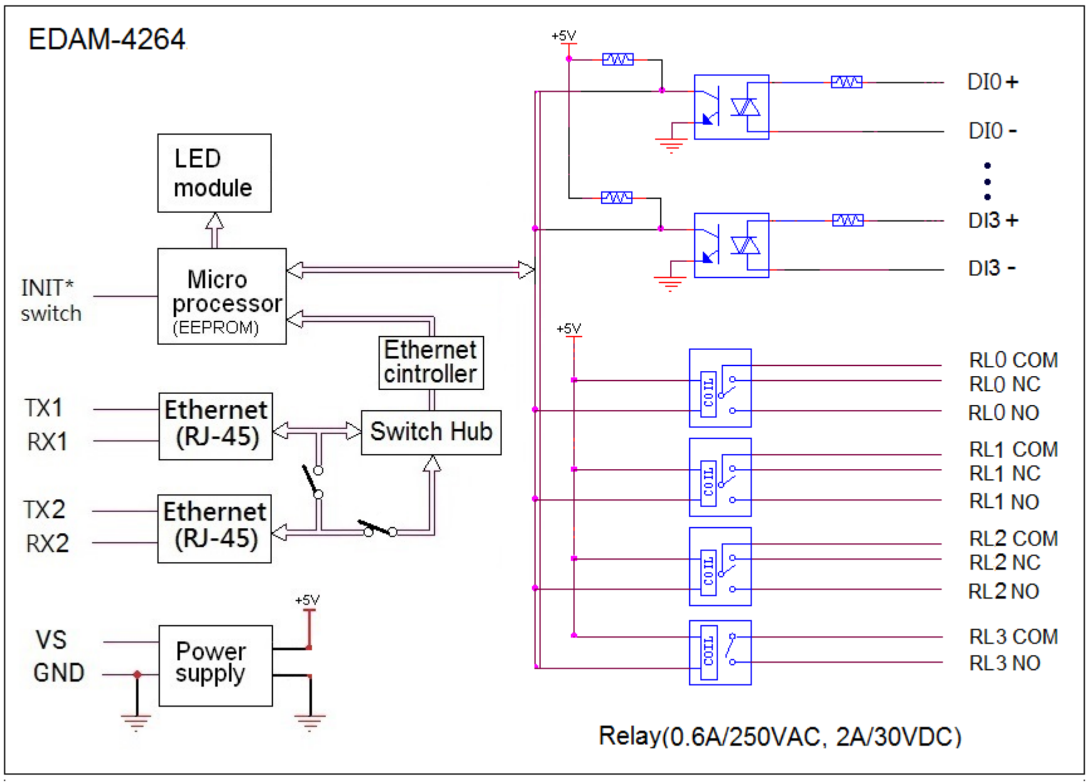


2.6.2 Wire connection

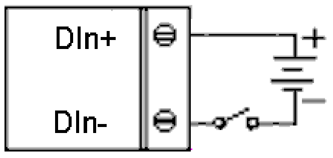
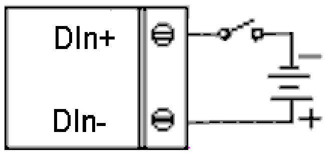
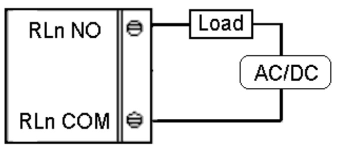
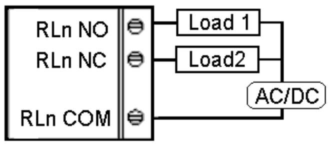
Dry Contact	
	
Wet Contact	
	
Note: To use wet contact , the DI.GND pin must be opened.	
Relay output	
	

2.7 EDAM-4264

2.7.1 Block diagram



2.7.2 Wire connection

Wet Contact	
	
Relay output	
	

Chapter 3 System Requirements

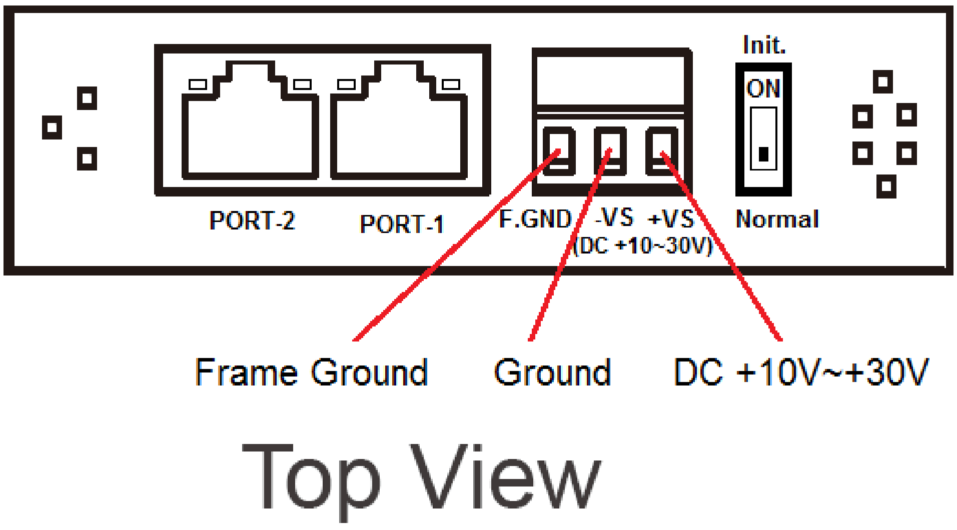
- IBM PC compatible computer with 486 CPU (Pentium is recommended)
- Microsoft 95/98/2000/NT 4.0 (SP3 or SP4)/XP/Win 7,8,10 or higher versions
- At least 32 MB RAM
- 20 MB of hard disk space available
- VGA color monitor
- 2x or higher speed CD-ROM
- Mouse or other pointing devices
- 10 or 100 Mbps Ethernet Card
- 10 or 100 Mbps Ethernet Hub (at least 2 ports)
- Two Ethernet Cable with RJ-45 connector
- Power supply for EDAM-4200 (+10 to +30 V unregulated)

3.1 Wiring and Connections

This section provides basic information on wiring the power supply, I/O units, and network connection.

3.2 Power supply wiring

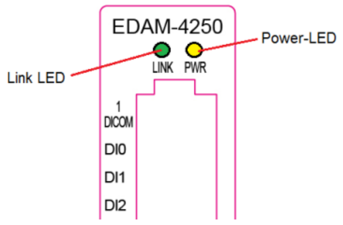
Although the EDAM-4200/TCP systems are designed for a standard industrial unregulated 24 V DC power supply, they accept any power unit that supplies within the range of +10 to +30 VDC. The power supply ripple must be limited to 200 mV peak-to-peak, and the immediate ripple voltage should be maintained between +10 and +30 VDC. Screw terminals +Vs and GND are for power supply wiring.



Note: The wires used should be sized at least 2 mm.

3.3 Status LED indicator for EDAM-4200 I/O modules

There are Tow flash types of the Status LED indicator on the front panel of EDAM-4200 series.



No.	Color	LED Status	Definition
1	Yellow (Power-LED)	On	Power-LED, Always ON.
2	Green (LINK-LED)	On	(LINK) This LED is normal on whenever the EDAM-4200 module's Ethernet wiring is connected
3	Green (LINK-LED)	Blinking	(COM) Blinks whenever EDAM-4200 module is transmitting or receiving data(I/O command) via Ethernet.

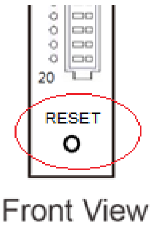
3.4 Ethernet LED indicator of EDAM-4200

There are two ports of RJ-45 ethernet connector. Each RJ-45 port LEDs built with two indicators to represent the EDAM-4200 ethernet status as explained below:

- ♦ Yellow indicator (Speed): This LED is always ON.
- ♦ Green indicator(Link): This LED is normal on whenever the EDAM-4200 module's Ethernet wiring is connected.

3.5 Reset

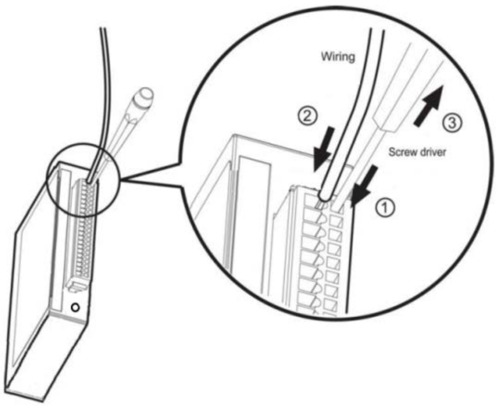
There is a RESET push button on the front panel. After press the reset button the module will be rebooted.



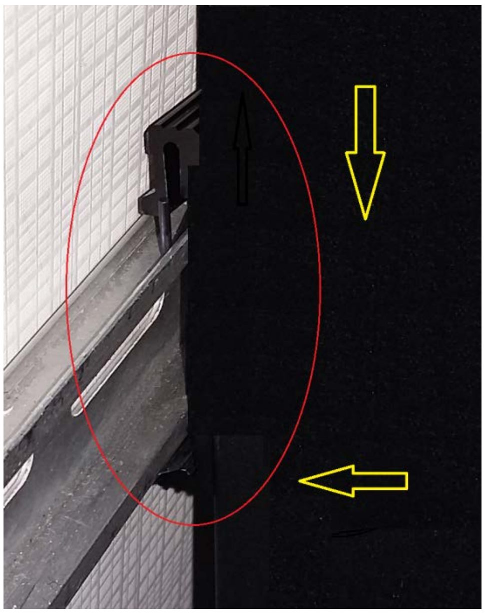
3.6 I/O modules wiring

The system uses a plug-in screw terminal block for the interface between I/O modules and field devices. The following information must be considered when connecting electrical devices to I/O modules. The terminal block accepts wires from # 14 AWG ~ 28 AWG. Always use a continuous length of wire. Do not combine wires to make them longer.

1. Insert the screw driver into the left hole of the terminal.
2. Insert the wiring into the left hole of the terminal.

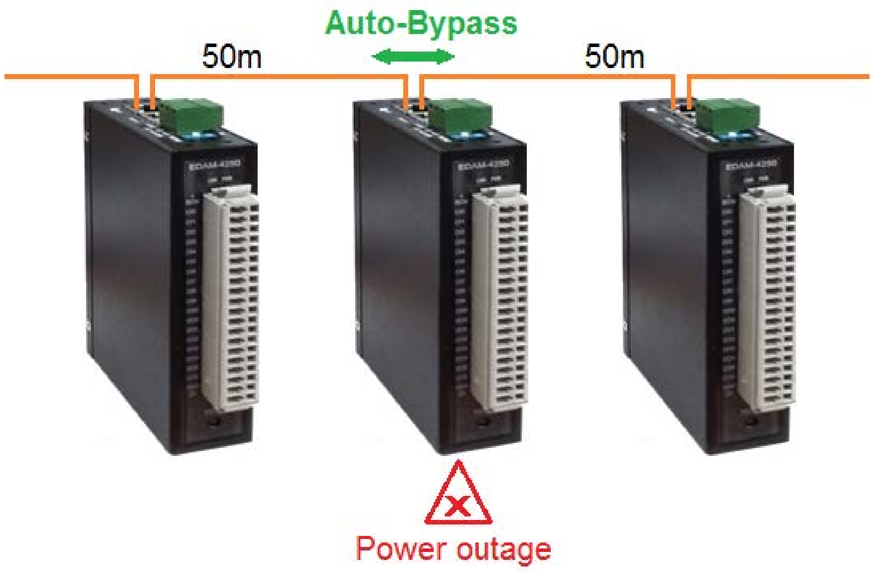


3.7 Rear side installation



3.8 Daisy Chain Limitations

In general, an ethernet cable distance of each 100BASE-TX network segment is can be run 100 meters (about 328 feet). However, cables can pick up electrical noise on long runs. Based on this limitation, the maximum total connection length in daisy chain wiring should also be 100m as if auto-bypass protection active. For example, the distance from first to second module is 50m, so as second to third. When the power fails on second (middle) module, the auto-bypass will activate to bridge the network connection. The total distance from first to the 3rd will become 100m, that means the total network segment is close to limitation.



3.9 Initializing a Module

All EDAM modules in an Ethernet network must have a *unique IP* address. Therefore, to configure the brand-new EDAM before using is necessary.

3.9.1 Factory default settings:

♦ IP Address :	10.0.0.1
♦ Subnet Mask:	255.255.255.0
♦ Gateway:	10.0.0.1
♦ DHCP:	Disabled
♦ Web Server:	Disabled
♦ Module ID:	01
♦ Password:	00000000

3.9.2 INIT* State settings:

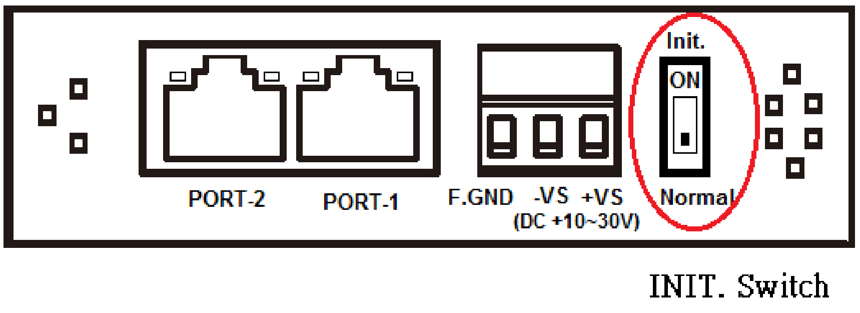
All EDAM-4200 I/O modules have a special slide-switch as INIT-SWITCH (Ref. Appendix). The I/O modules must be set at ***"INIT" State*** when you want to change the default settings, such as the *IP address,* Subnet Mask, Gateway, Password etc. If the "INIT" switch set to "INIT-ON" mode when power ON, Under this state the default configuration is set as following :

♦ IP Address :	10.0.0.1
♦ Subnet Mask:	255.255.255.0
♦ Gateway:	10.0.0.1
♦ Password:	00000000

Press the "**Default**" button on the EDAM-4200 utility(Network), the module will be set to factory default state as following :

♦ IP Address :	10.0.0.1
♦ Subnet Mask:	255.255.255.0
♦ Gateway:	10.0.0.1
♦ DHCP:	Disabled
♦ Web Server:	Disabled
♦ Module ID:	01
♦ Password:	00000000
♦ I/O:	factory default Mode

Note: Each module must has a unique ID number to be identified when the DHCP enabled, because you would not know the module IP address when DHCP enabled, but if with the different ID number. You can call provided function call(TCP_GetIPFromID() in TCPDAQ.dll) to get correct IP address for each ID number



Top View

Chapter 4 Specification and wiring

4.1 **EDAM-4250** 10 D*I* and 6 DO channels Digital I/O Module

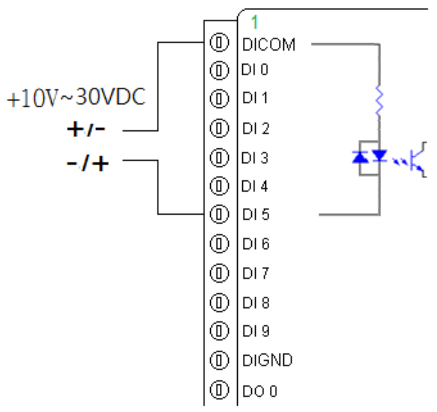
The EDAM-4250 is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It provides ***10 digital input and 6 digital output channels*** with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 300Hz counter. Opposite to the intelligent DI functions, the EDAM-4250 Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function.

4.1.1 Specification

- ◆ Digital input : Isolated single ended with common source/ground
  - ✓ Channel : 10 channels (DI0~DI9).
  - ✓ Input Level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Dry Contact : Single ended with common source.
    - ➢ Logic level 0 (active): Close to GND.
    - ➢ Logic level 1 (inactive): Open.
  - ✓ Wet Contact : To use Wet Contact , the DI.GND pin must be opened.
    - ➢ Logic level 0 (active): *+5V to +30VDC max.*
    - ➢ Logic level 1 (inactive): +2VDC max.
  - ✓ Input Impedance : 2K ohm(Wet Contact)
  - ✓ Counter mode : Supports 300Hz counter(by software,32-bit + 1-bit overflow)
  - ✓ Optical Isolation Voltage : 3750Vrms
- ◆ Digital Output : Isolated Open collector (NPN) output channels.
  - ✓ Channel : 6 channels (DO0~DO5) .
  - ✓ Logical level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Open Collector : +5V~30V/500 mA max. load
  - ✓ Pulse Output : Each channel supports 300Hz pulse output
  - ✓ Optical Isolation Voltage : 3750Vrms
- ◆ Display : 10 digital inputs & 6 digital output status LED
- ◆ Power requirements : Unregulated +10 ~ +30 VDC
- ◆ Power Consumption : 3.5 W (Typical)

4.1.2 Application Wiring

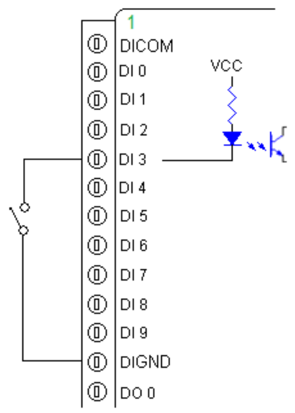
Wet Contact



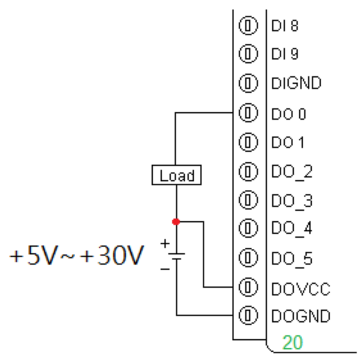
To use Wet Contact , the DIGND pin must be opened.

Digital Input

Dry Contact



Digital Input



Digital Output

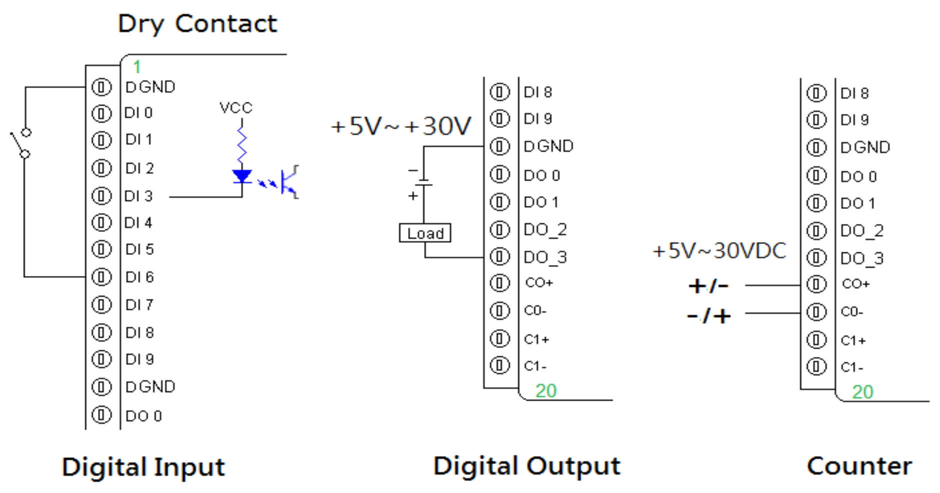
4.2 **EDAM-4251** 10 DI , 4 DO and 2 Counter chs DIO Module

The EDAM-4251 is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It provides *10 digital input(Dry contact), 4 digital output, and 2 counter* channels with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 300Hz(DI0~9)/4.5KHz(C0~C1) counter. Opposite to the intelligent DI functions, the EDAM-4251 Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function.

4.2.1 Specification

- ◆ Digital Input : Isolated single ended with common source (Dry Contact).
  - ✓ Channel : 10 channels (DI0~DI9)
  - ✓ Input level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Dry contact : Single ended with common ground.
    - ➢ Logic level 0 (active) : Close to GND.
    - ➢ Logic level 1 (inactive): Open.
  - ✓ Counter : 300Hz software counter(32-bit + 1-bit overflow)
  - ✓ Optical Isolation Voltage : 3750Vrms
- ◆ Digital Output : Isolated Open collector (NPN) output channels.
  - ✓ Channel : 4 channels (DO0~DO3) .
  - ✓ Logical level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Open Collector : +5V~30V / 500 mA max. load
  - ✓ Pulse Output : Each channel supports 300Hz pulse output
  - ✓ Optical Isolation Voltage: 3750Vrms
- ◆ Counter : 2 channels input hardware counter
  - ✓ Channel : 2 (C0=DI10, C1=DI11)
  - ✓ Input logic level : 30VDC max.
    - ➢ Logic level 1 (active): +5V to 30VDC max.
    - ➢ Logic level 0 (inactive): +2 Vac max.
  - ✓ Maximum Count : 4,294,967,285 (32-bit + 1-bit overflow)
  - ✓ Input Impedance : 2K ohm(Wet Contact)
  - ✓ Input frequency : 4500 Hz max.
  - ✓ Optical Isolation Voltage: 3750Vrms
- ◆ Display : 10 digital inputs, 2 Counter & 4 digital output status LED
- ◆ Power requirements : Unregulated +10 ~ +30 VDC
- ◆ Power Consumption : 3.5 W (Typical)

4.2.2 Application Wiring



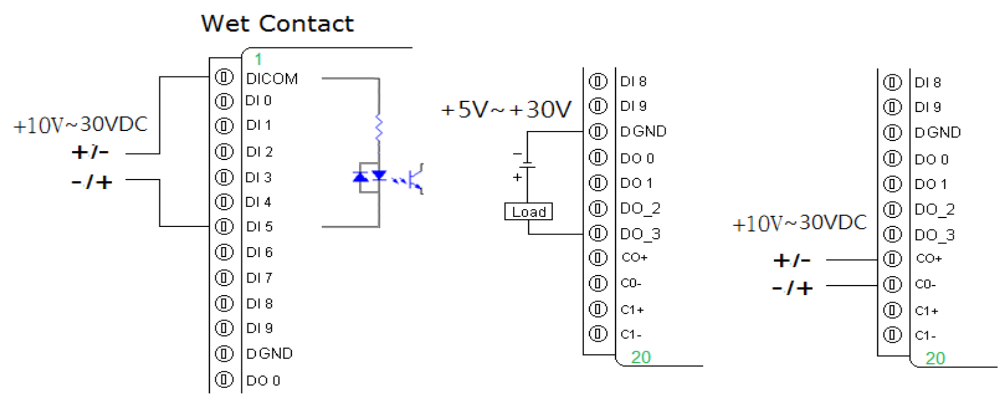
4.3 **EDAM-4251A** 10 DI , 4 DO and 2 Counter chs DIO Module

The EDAM-4251A is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It provides *10 digital input(Wet contact), 4 digital output, and 2 counter* channels with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 300Hz(DI0~9)/4.5KHz(C0~C1) counter. Opposite to the intelligent DI functions, the EDAM-4251A Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function.

4.3.1 Specification

- Digital Input : Isolated single ended with common source/ground (Wet Contact).
  - ✓ Channel : 10 channels (DI0~DI9)
  - ✓ Input level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Wet Contact : Single ended digital input with common source/ground.
    - ➢ Logic level 0 (active) : +5V to +30VDC max.
    - ➢ Logic level 1 (inactive): +2 Vdc max.
  - ✓ Impedance : 2K ohm.
  - ✓ Counter : 300Hz software counter(32-bit + 1-bit overflow)
  - ✓ Optical Isolation Voltage : 3750Vrms
- Digital Output : Isolated Open collector (NPN) output channels.
  - ✓ Channel : 4 channels (DO0~DO3) .
  - ✓ Logical level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Open Collector : +5V~30V / 500 mA max. load
  - ✓ Pulse Output : Each channel supports 300Hz pulse output
  - ✓ Optical Isolation Voltage: 3750Vrms
- Counter : 2 channels input *hardware* counter
  - ✓ Channel : 2 (C0=DI10, C1=DI11)
  - ✓ Input logic level : 30VDC max.
    - ➢ Logic level 1 (active),+5V to 30VDC max.
    - ➢ Logic level 0 (inactive) ,+2 Vac max.
  - ✓ Maximum Count : 4,294,967,285 (32-bit + 1-bit overflow)
  - ✓ Input Impedance : 2K ohm(Wet Contact)
  - ✓ Input frequency : 4500 Hz max.
  - ✓ Optical Isolation Voltage: 3750Vrms
- Display : 10 digital inputs, 2 Counter & 4 digital output status LED
- Power requirements : Unregulated +10 ~ +30 VDC
- Power Consumption : 3.5 W (Typical)

4.3.2 Application Wiring



Digital Input **Digital Output** **Counter**

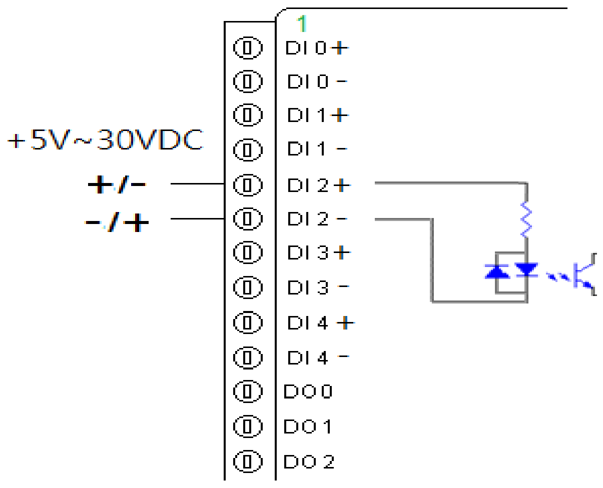
4.4 EDAM-4254 5 differential DI and 8 DO channels Digital I/O Module

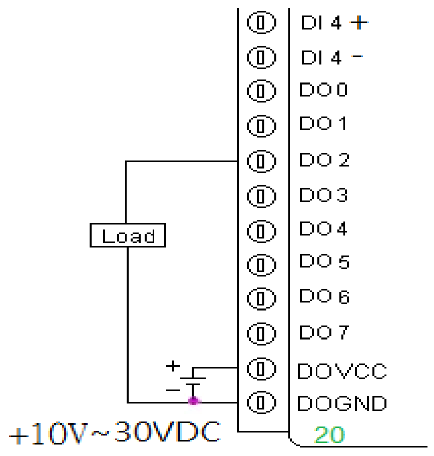
The EDAM-4260 is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It is a high-density I/O module. It provides 5 differential digital input and 8 digital output channels with 3750VRMS Isolating protection. The EDAM-4254 Digital Input channels support 5 isolated differential digital input (sink/source) channels and All of the Digital Input channels support input latch function for important signal handling. Meanwhile, these DI channels allow to be used as 300Hz counter. Opposite to the intelligent DI functions, the EDAM-4254 Digital Output channels also support pulse output function, Auto-Off Time of digital output and DIO Synchronization function.

4.4.1 Specification

- **Digital input** : Isolated differential digital inputs.
  - ✓ Channel : 5 (DI0~DI4) isolated differential input channels (sink/source).
  - ✓ Input Level : Logic level status can be inversed via ASCII/Modbus command.
    - ➢ Logical level 1 (inactive): +0V ~ +1Vdc Max.
    - ➢ Logical level 0 (active): +5V ~ +30Vdc
  - ✓ Input Impedance : 2K ohm
  - ✓ Counter mode : Supports 300Hz counter(by software,32-bit + 1-bit overflow)
  - ✓ Optical Isolation Voltage : 3750Vrms
- **Digital Output** : Isolated open drain (P-MOSFET) output channels.
  - ✓ Channel : 8 channels (DO0~DO7) .
  - ✓ Logical level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Load voltage : +10V ~ +30Vdc
  - ✓ Load current : 750mA/ channel Max. (with short-circuit protection)
  - ✓ Pulse Output : Each channel supports 300Hz pulse output
  - ✓ Optical Isolation Voltage: 3750Vrms
- **Display** : 5 digital inputs & 8 digital outputs status LED
- **Power requirements** : Unregulated +10 ~ +30 VDC
- **Power Consumption** : 3.6 W (Typical)

4.4.2 Application Wiring





Digital Input Digital Output

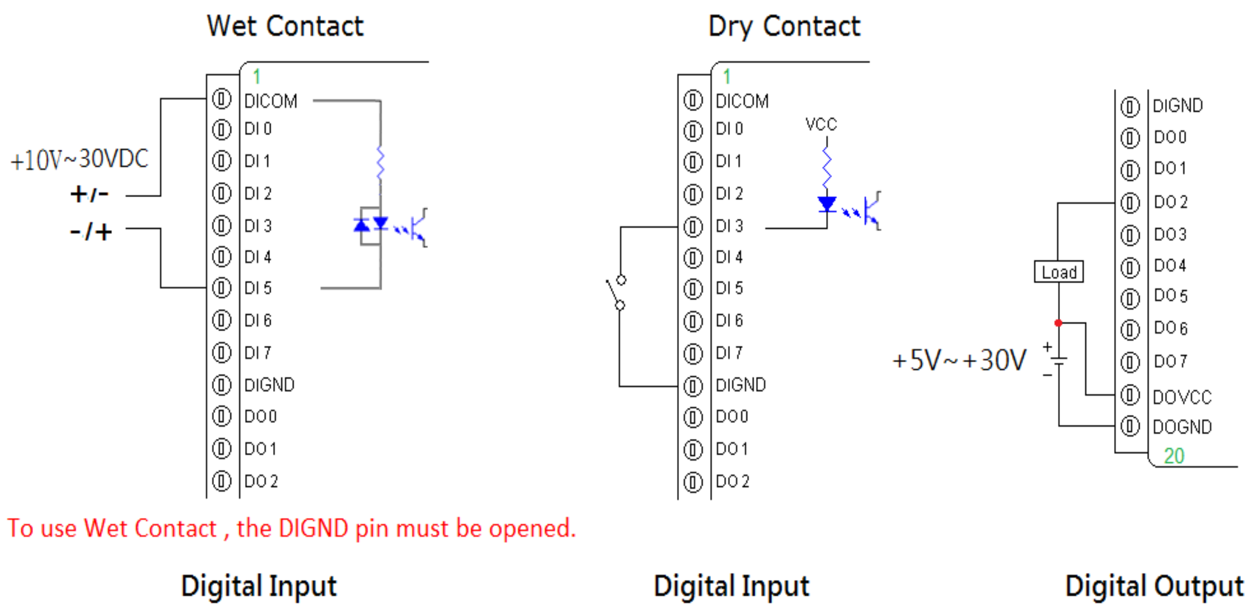
4.5 **EDAM-4255** 8 DI and 8 DO channels Digital I/O Module

The EDAM-4255 is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It provides *8 digital input channels, and 8 digital output* channels**.** All of the digital input channels support the input latch function for important signal handling. The digital output channels support source type output. Opposite to the intelligent DI functions, the EDAM-4255 Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function.

4.5.1 Specification

- **Digital input** : Isolated single ended with common source/ground.
  - ✓ Channel : 8 channels (DI0~DI7).
  - ✓ Input Level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Dry Contact : Single ended with common Ground.
    - ➢ Logic level 0 (active) : Close to DI.GND
    - ➢ Logic level 1 (inactive): Open
  - ✓ Wet Contact : To use Wet Contact , the DI.GND pin must be opened.
    - ➢ Logic level 1 (active) : *+5V to +30VDC max.*
    - ➢ Logic level 0 (inactive) : +2VDC max.
  - ✓ Input Impedance : 2K ohm(Wet Contact)
  - ✓ Counter : 300Hz software counter(32-bit + 1-bit overflow)
  - ✓ Optical Isolation Voltage: 3750Vrms
- **Digital Output** : Isolated open drain (P-MOSFET) output channels.
  - ✓ Channel : 8 channels (DO0~DO7) .
  - ✓ Logical level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Load voltage : +10V ~ +30Vdc
  - ✓ Load current : 750mA/ channel Max. (with short-circuit protection)
  - ✓ Pulse Output : Each channel supports 300Hz pulse output
  - ✓ Optical Isolation Voltage: 3750Vrms
- **Display** : 8 digital inputs & 8 digital outputs status LED
- **Power requirements** : Unregulated +10 ~ +30 VDC
- **Power Consumption** : 3.6 W (Typical)

4.5.2 Application Wiring



4.7 EDAM-4260 7-channel Digital Input and 4 RELAY output Module

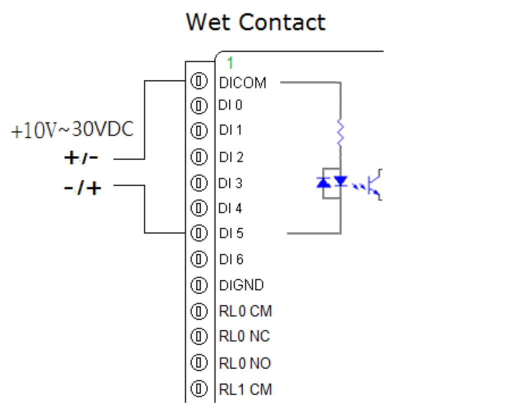
The EDAM-4260 is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It provides 7 isolated digital input channels and 4 relay output channels(0.6A/125Vac, 2A/30Vdc). All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 300Hz counter. All relay output channels are differential with individually common . Opposite to the intelligent DI functions, the EDAM-4260 Digital Output channels also support *Auto-Off Time of digital output* and *DIO Synchronization* function.

4.7.1 Specification

- **Digital input** : Isolated single ended with common source.
  - ✓ Channel : 7 channels (DI0~DI6).
  - ✓ Input Level : Logic level status can be inversed via ASCII/Modbus command.
  - ✓ Dry Contact : Single ended with common Ground.
    - ➢ Logic level 0 (active) : Close to DI.GND
    - ➢ Logic level 1 (inactive): Open
  - ✓ Wet Contact : To use Wet Contact , the DI.GND pin must be opened.
    - ➢ Logic level 0 (active) : *+5V to +30VDC max.*
    - ➢ Logic level 1 (inactive) : +2VDC max.
  - ✓ Counter mode : Supports up to 300Hz counter(by software, 32-bit + 1-bit overflow)
  - ✓ Optical Isolation Voltage : 3750Vrms
- **Relay Output** :
  - ✓ Output channels : 4 relay output channels (RL0,RL1,RL2 Form C(SPDT) and RL3 Form A(SPST NO).
  - ✓ Surge strength : 500V
  - ✓ Relay contact rating : 0.6A/125Vac, 2A/30Vdc
  - ✓ Operate Time : 3mS max.
  - ✓ Release Time : 2mS max.
  - ✓ Min Life : 5*105 ops
  - ✓ Pulse Output : Each channel supports 300Hz pulse output
- Display : 7 digital input & 4 Relay output status LED
- Power requirements : Unregulated, +10V ~ +30 VDC
- Power Consumption : 2.5 W (Typical)

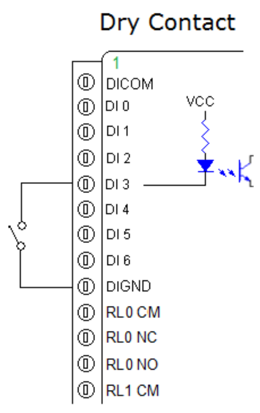
4.7.2 Application Wiring

- **Digital Input & Relay output:**

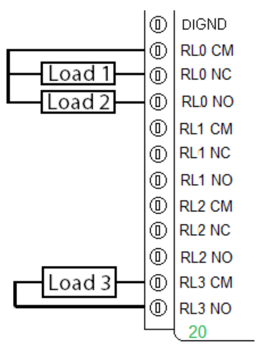


To use Wet Contact , the DIGND pin must be opened.

Digital Input



Digital Input



Relay Contact

Note: *Relay(x): RLx CM=Common, RLx NO=Normal open, RLx NC=Normal Close*

4.8 **EDAM-4264** 4-channel differential Digital Input and 4 RELAY output Module

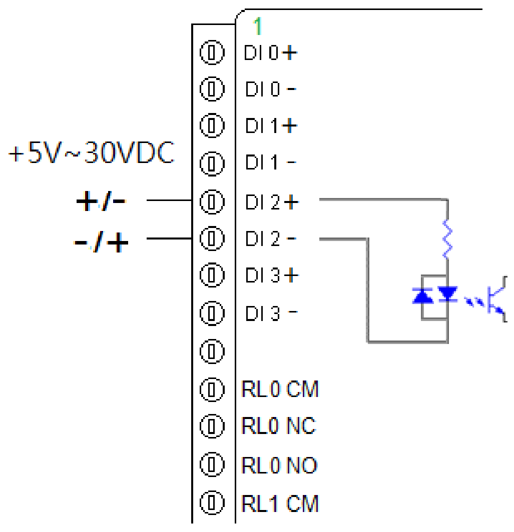
The EDAM-4264 is a high-density I/O module built-in two port(RJ-45) 10/100 based-T interface for seamless Ethernet connectivity. It provides 4 isolated differential digital input channels and 4 relay output channels(0.6A/125Vac, 2A/30Vdc). All input channels are sdifferential digital input (sink/source) and support input latch function for important signal handling. Mean while, these DI channels allow to be used as 300Hz counter. All relay output channels are differential with individually common . Opposite to the intelligent DI functions, the EDAM-4264 Relay Output channels also support *Auto-Off Time of Relay output* and *DIO Synchronization* function.

4.8.1 Specification

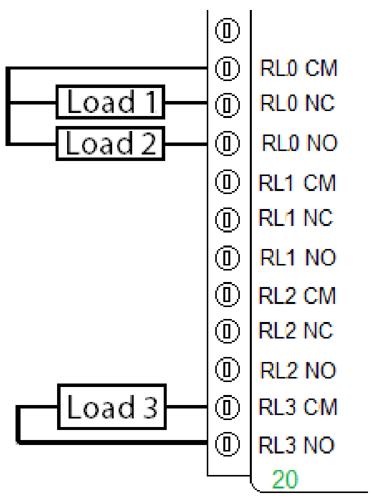
◆ **Digital input**	:	Isolated differential digital inputs (sink/source).
✓ Channel	:	4 (DI0~DI3) isolated differential input channels (sink/source).
✓ Input Level	:	Logic level status can be inversed via ASCII/Modbus command.
		➢ Logical level 0: +0V ~ +1Vdc Max.
		➢ Logical level 1: +10V ~ +30Vdc
✓ Input Impedance	:	2K ohm
✓ Counter mode	:	Supports 300Hz counter(by software,32-bit + 1-bit overflow)
✓ Optical Isolation Voltage	:	3750Vrms
◆ **Relay Output**	:	
✓ Output channels	:	4 relay output channels (RL0,RL1,RL2 Form C(SPDT) and RL3 Form A(SPST NO).
✓ Surge strength	:	500V
✓ Relay contact rating	:	0.6A/125Vac, 2A/30Vdc
✓ Operate Time	:	3mS max.
✓ Release Time	:	2mS max.
✓ Min Life	:	5*105 ops
✓ Pulse Output	:	Each channel supports 300Hz pulse output
◆ Display	:	4 digital input & 4 Relay output status LED
◆ Power requirements	:	Unregulated, +10V ~ +30 VDC
◆ Power Consumption	:	2.5 W (Typical)

4.8.2 Application Wiring

- **Digital Input & Relay output:**



Digital Input



Relay Contact

Note: *Relay(x): RLx CM=Common, RLx NO=Normal Open, RLx NC=Normal Close*

Chapter 5 EDAM-4200 Utility Guide

In order to properly configure EDAM series. You will need following items to complete your system hardware configuration.

5.1 System Requirement

Host computer

- ♦ IBM PC compatible computer with 486 CPU (Pentium is recommended)
- ♦ Microsoft 95/98/2000/NT 4.0 (SP3 or SP4)/Win 7,8,10 or higher versions
- ♦ At least 32 MB RAM
- ♦ 20 MB of hard disk space available
- ♦ VGA color monitor
- ♦ 2x or higher speed CD-ROM
- ♦ Mouse or other pointing devices
- ♦ 10 or 100 Mbps Ethernet Card
- ♦ 10 or 100 Mbps Ethernet Hub (at least 2 ports)
- ♦ Two Ethernet Cable with RJ-45 connector
- ♦ Power supply for EDAM-4200 (+10 to +30 V unregulated), (for 94xx: option).
- ♦ Make sure to prepare all of the items above, then connect the power and network wiring as Figure 5-1

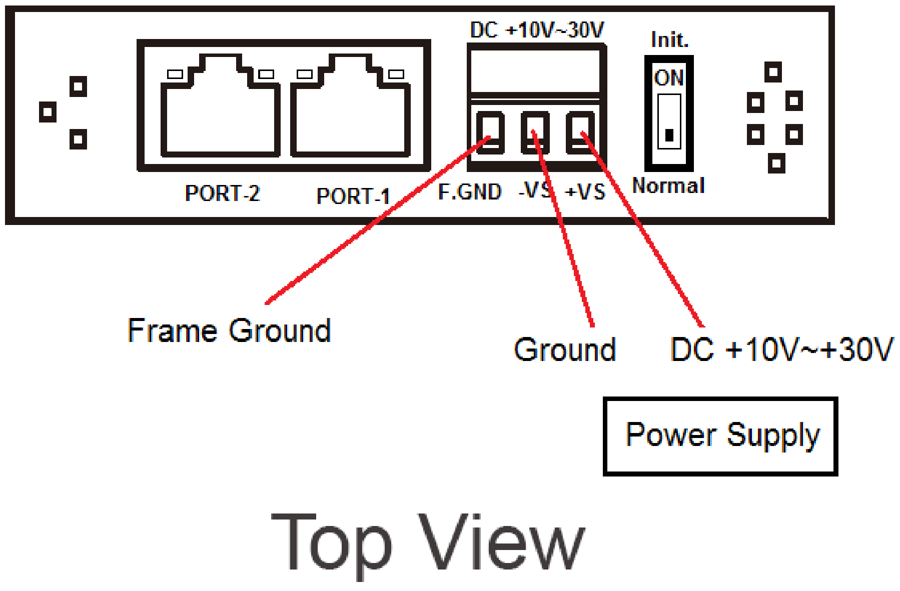


Figure 5-1 Power wiring

5.2 Install Utility Software on Host PC

Inlog provide free download Manual and Utility software for EDAM-4200 modules' operation and configuration. Link to the web site: www.inlog.com.tw and click into the "Download Area" to get the latest version EDAM-4200 manual and Ethernet I/O Utility. Once you download and setup the Utility software, there will be a shortcut of the Utility executive program on Windows' desktop after completing the installation.

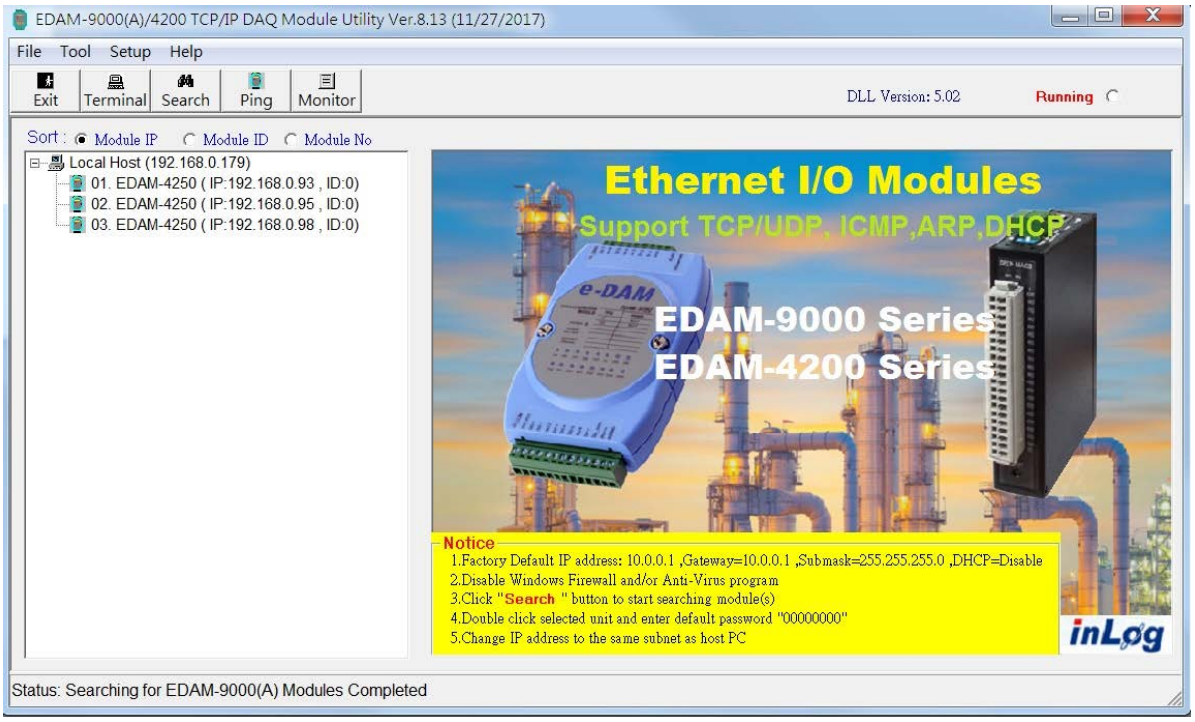
5.3 EDAM Ethernet I/O Utility Overview

The Utility software offers a graphical interface that helps you configure the EDAM-4200 modules. It is also very convenient to test and monitor your remote DAQ system. The following guidelines will give you some brief instructions on how to use this Utility.

- ♦ Main Menu
- ♦ Network Setting
- ♦ Adding Remote Station
- ♦ Security setting
- ♦ I/O Module Configuration
- ♦ Alarm Setting
- ♦ I/O Module Calibration
- ♦ Security Setting
- ♦ Terminal emulation
- ♦ Data/Event Stream

5.4 Main Menu

Double Click the icon of EDAM Ethernet I/O Utility shortcut, the Operation screen will pop up as Figure 5-2 main window.



EDAM-9000(A)/4200 TCP/IP DAQ Module Utility Ver.8.13 (11/27/2017)

DLL Version: 5.02 Running

Local Host (192.168.0.179)
01. EDAM-4250 (IP:192.168.0.93 , ID:0)
02. EDAM-4250 (IP:192.168.0.95 , ID:0)
03. EDAM-4250 (IP:192.168.0.98 , ID:0)

Notice
1.Factory Default IP address: 10.0.0.1 ,Gateway=10.0.0.1 ,Submask=255.255.255.0 ,DHCP=Disable
2.Disable Windows Firewall and/or Anti-Virus program
3.Click "Search " button to start searching module(s)
4.Double click selected unit and enter default password "00000000"
5.Change IP address to the same subnet as host PC

Status: Searching for EDAM-9000(A) Modules Completed

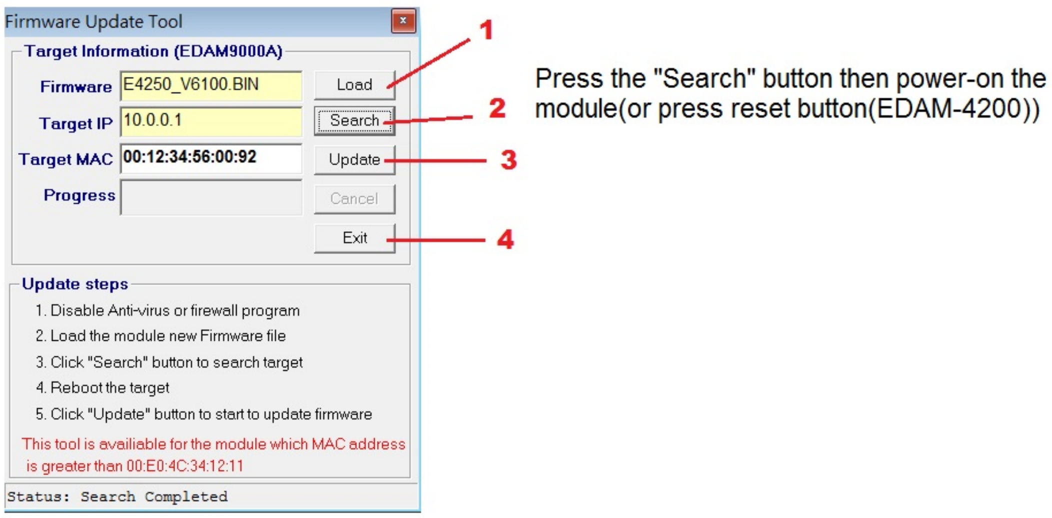
Figure 5-2 main window

The top of the operation screen consists of a function menu and a tool bar for user's commonly operating functions.

5.4.1 Function Menu

- ◆ File : using to exit this Utility program.
- ◆ Tool : contents functions as below:
  - ➢ Search modules
  - ➢ Ping remote Ethernet device.
  - ➢ Monitor Stream/Event Data
  - ➢ Terminal

- Firmware update:



Press the "Search" button then power-on the module(or press reset button(EDAM-4200))

Firmware	E4250_V6100.BIN
Target IP	10.0.0.1
Target MAC	00:12:34:56:00:92

Update steps
1. Disable Anti-virus or firewall program
2. Load the module new Firmware file
3. Click "Search" button to search target
4. Reboot the target
5. Click "Update" button to start to update firmware

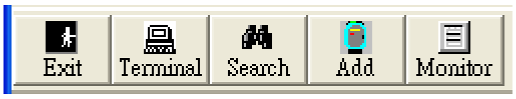
This tool is availiable for the module which MAC address is greater than 00:E0:4C:34:12:11

Status: Search Completed

- ◆ Setup : Contents Timeout and Scan Rate setting functions. Please be aware of the time setting for other Ethernet domination usually longer than local network.
- ◆ Help : Contents on-line help function as user's operation guide; the item "About" contents information about software version, released date, and support modules.

5.4.2 Tool Bar

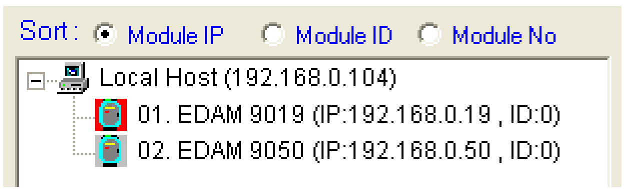
There are five push buttons in the tool bar.



♦ Exit	:	Exit utility program
♦ Terminal	:	Terminal Call up the operation screen of Terminal emulation to do the request / response command execution.
♦ Search	:	Search all the EDAM modules you connected in local Ethernet
♦ Add	:	Add remote EDAM I/O module
♦ Monitor	:	Monitor the Stream/Event Data

5.4.3 List Sort

The searched units will be listed in the tree-structure display area in order by "**Sort**" selection


Sort :
Module IP
Module ID
Module No
Local Host (192.168.0.104)
01. EDAM 9019 (IP:192.168.0.19 , ID:0)
02. EDAM 9050 (IP:192.168.0.50 , ID:0)

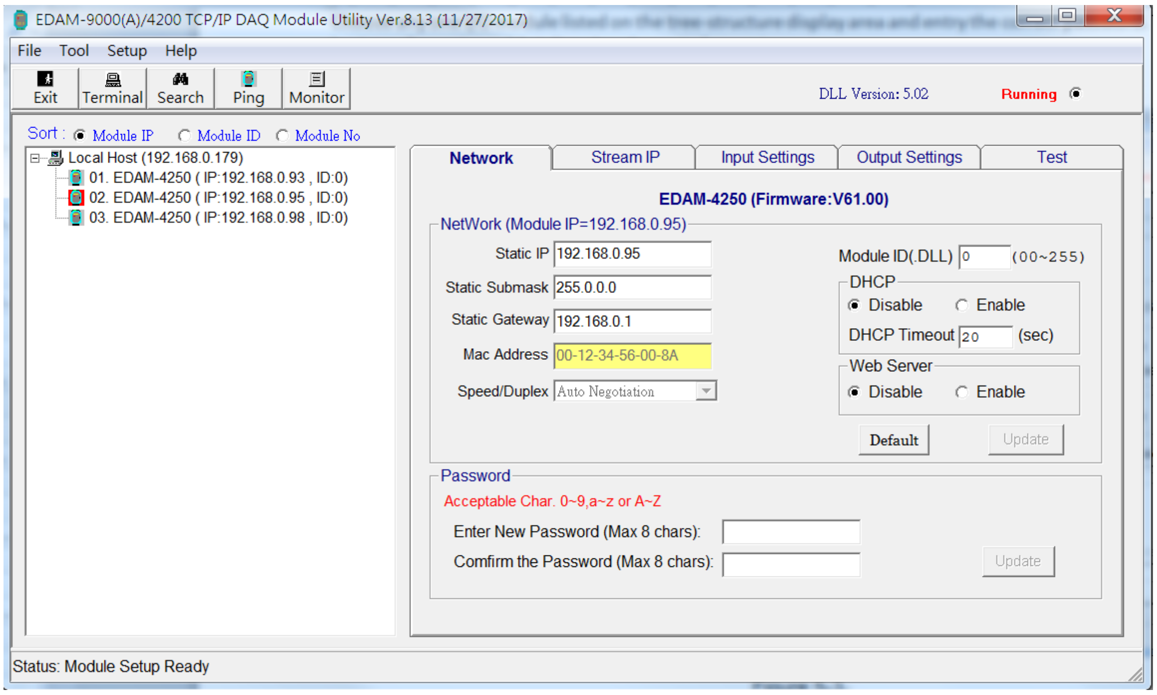
♦ Module IP	: Sort by module IP
♦ Module ID	: Sort by module ID
♦ Module No	: Sort by module name

5.5 Network Setting

As the moment you start up this Windows Utility, it will search all EDAM-4200 I/O modules on the host PC's domination Ethernet network automatically. Then the tree-structure display area will appeal with the searched units and the relative IP address.

Since Utility software detects the EDAM-4200 on the network, user can begin to setup each unit.

Choose any one I/O module listed on the tree-structure display area and entry the correct password. The module basic configuration table is listed as shown in for setting



EDAM-9000(A)/4200 TCP/IP DAQ Module Utility Ver.8.13 (11/27/2017)

File Tool Setup Help

Exit | Terminal | Search | Ping | Monitor

DLL Version: 5.02 Running

Sort : [✓ Module IP] Module ID Module No

Local Host (192.168.0.179)
- 01. EDAM-4250 (IP:192.168.0.93 , ID:0)
- 02. EDAM-4250 (IP:192.168.0.95 , ID:0)
- 03. EDAM-4250 (IP:192.168.0.98 , ID:0)

Network | Stream IP | Input Settings | Output Settings | Test

EDAM-4250 (Firmware:V61.00)

NetWork (Module IP=192.168.0.95)

Static IP	192.168.0.95
Static Submask	255.0.0.0
Static Gateway	192.168.0.1
Mac Address	00-12-34-56-00-8A
Speed/Duplex	Auto Negotiation
Module ID(.DLL)	0 (00~255)
DHCP	[✓ Disable] Enable
DHCP Timeout	20 (sec)
Web Server	[✓ Disable] Enable

Default Update

Password

Acceptable Char. 0~9,a~z or A~Z

Enter New Password (Max 8 chars):

Comfirm the Password (Max 8 chars):

Update

Status: Module Setup Ready

Figure 5-3

5.5.1 Module IP

MAC Address **:** This is also called Ethernet address and needs no further configuration.

IP Address, Subnet Mask, and Default Gateway: (default 10.0.0.1, 255.255.255.0 and 10.0.0.1)

The IP address identifies your EDAM-4200 devices on the global network. Each EDAM-4200 has same default IP address 10.0.0.1. Therefore, *please do not initial many EDAM-4200 at the same time to avoid the Ethernet collision*. If you want to configure the EDAM-4200 in the host PC's dominating network, only the IP address and Subnet Mask will need to set (The host PC and EDAM Ethernet I/O must belong to same subnet Mask).

If you want to configure the EDAM-4200 via Internet or other network domination, you have to ask your network administrator to obtain a specific IP and Gateway addresses, and then configure each EDAM-4200 with the individual setting.

DHCP **:** (default Disabled)

Allow you to get IP address from the DHCP server without setting IP address by manual.

DHCP timeout **:** (default 20 sec)

Allow you to set timeout to search for the DHCP servo. If there is no DHCP servo exist, the module will reboot and use static IP address assigned by E9KUtility.exe

Web Server **:** (default Disabled)

Allow you monitor and control I/O status on EDAM-4200 modules remotely through web browser.

Module ID **:** (default 01)

Each module must has a unique ID number to be identified when the DHCP enabled, because you would not know the module IP address when DHCP enabled, but if with the different ID number. You can call provided function call(TCP_GetIPFromID() in TCPDAQ.DLL) to get correct IP address for each ID number

Password : (default 00000000)

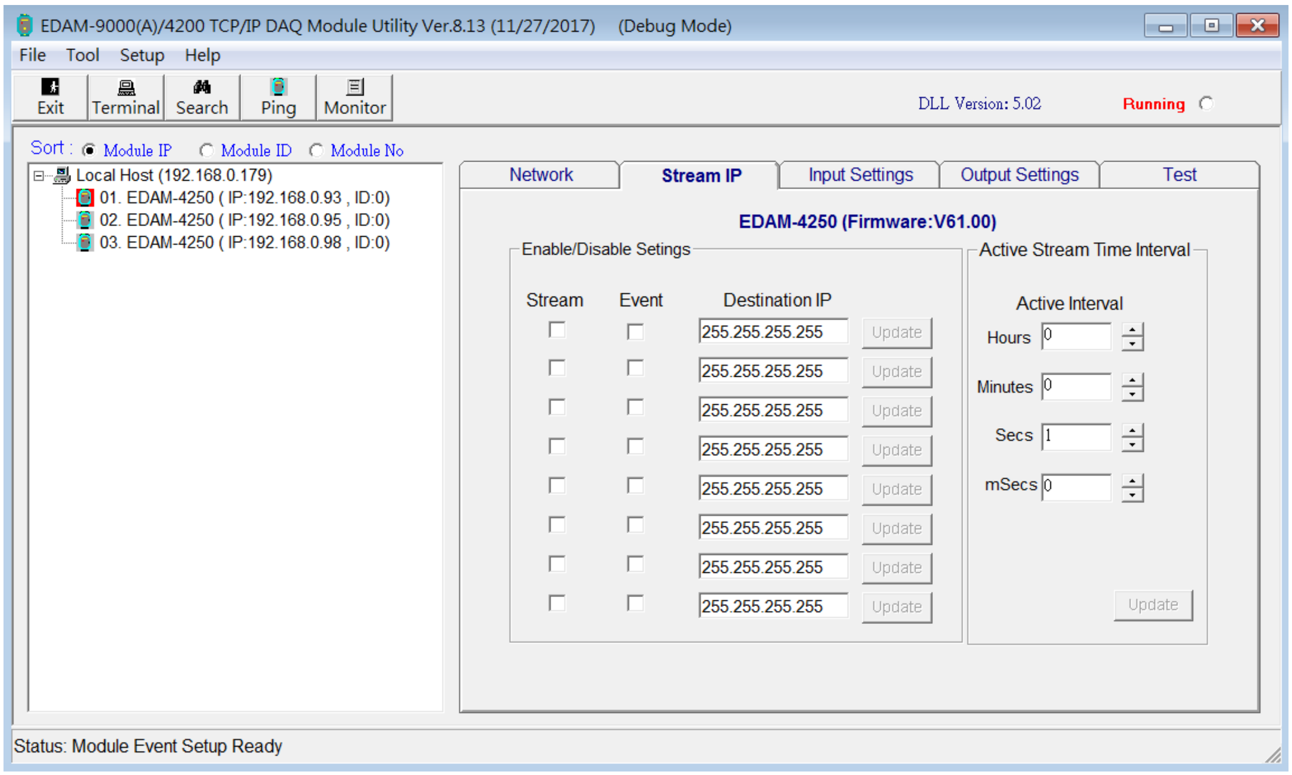
Allow you to change the password of the module

5.5.2 TCP/IP port:

EDAM-4200 series use four ports to communication with Host as shown below table

Protocol	Port (dec)	Description
TCP	502	MODBUS/TCP
UDP	1025	ASCII Command
UDP	5168	Event/Stream trigger
TCP	80	HTTP (web)

5.5.3 Stream/Alarm(for DI latch Mode) IP


EDAM-9000(A)/4200 TCP/IP DAQ Module Utility Ver.8.13 (11/27/2017) (Debug Mode)
File Tool Setup Help
Exit Terminal Search Ping Monitor
DLL Version: 5.02
Running
Sort : Module IP Module ID Module No
Local Host (192.168.0.179)
01. EDAM-4250 (IP:192.168.0.93 , ID:0)
02. EDAM-4250 (IP:192.168.0.95 , ID:0)
03. EDAM-4250 (IP:192.168.0.98 , ID:0)
Network
Stream IP
Input Settings
Output Settings
Test
EDAM-4250 (Firmware:V61.00)
Enable/Disable Setings
Stream Event Destination IP
255.255.255.255 Update
Active Stream Time Interval
Active Interval
Hours 0
Minutes 0
Secs 1
mSecs 0
Update
Status: Module Event Setup Ready

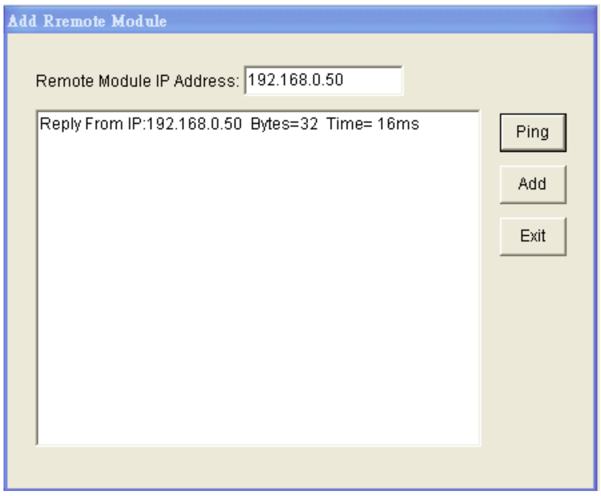
Stream/Alarm event Enable Setting : Set Stream /Event data Destination IP (default all disabled),

Active Stream time period : Set time interval for sending stream data (for DI latch Mode, default 1 sec)

5.6 Add Remote Stations

To meet the remote monitoring and maintenance requirements, The EDAM-4200 system does not only available to operate in local LAN, but also allowed to access from Internet or Intranet. Thus users would able to configure an EDAM-4200 easily no matter how far it is.

Select item **Tool**\Add Remote Ethernet I/O in function menu or click the button, the adding station screen will pop up as Figure1 6 Add remote module. Then key-in the specific IP address and click the **"Ping"** button. If the communication success, click **"Add"** to add EDAM Ethernet I/O unit into the tree-structure display area.



Remote Module IP Address: 192.168.0.50

Reply From IP:192.168.0.50 Bytes=32 Time= 16ms

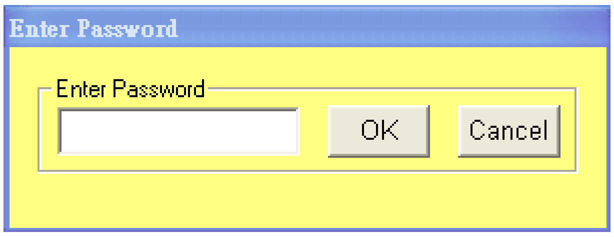
Figure5-4 Add remote module

Note:

- ♦ There is several conditions need to be sure before adding a remote EDAM-4200 system in the Window Utility.
- ♦ Be sure the specific IP is existed and available.
- ♦ Be sure to complete the network linkage for both sides.
- ♦ Be sure to adjust the best timing of timeout setting.
- ♦ Even you are not sure whether the communication is workable or not, there is also a **"Ping"** function for testing the network connection.

5.7 Security Setting

Though the technology of Ethernet discovered with great benefits in speed and integration, there also exist risk about network invading form anywhere. For the reason, the security protection design has built-in EDAM-4200 I/O modules. Once user setting the password into the EDAM-4200 firmware, the important system configurations (Network, Firmware, Password) are only allowed to be changed by password verification.



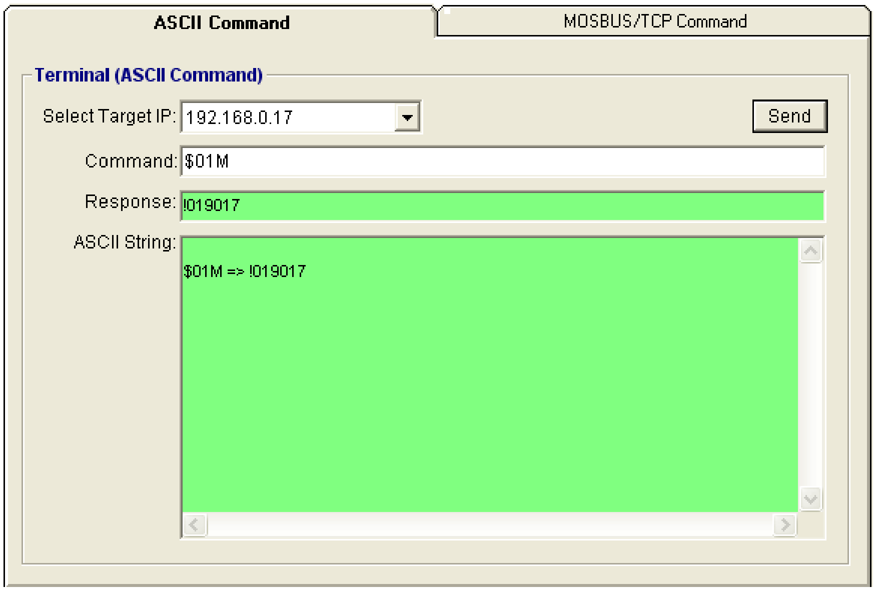
Note:

The default password of EDAM-4200 is "**00000000**". Please make sure to keep the correct password by yourself. If you lose it, please contact to Inlog's technical support center for help.

5.8 Terminal Emulations

You can issue commands and receive response by clicking the Terminal button on the tool bar. There are two kinds of command format supported by this emulating function. Users can choose ASCII or ModBus Hexadecimal mode as their communication base. If the ASCII mode has been selected, the Windows Utility will translate the request and response string in ASCII format.

ASCII Command mode: Shown as ASCII Command Terminal

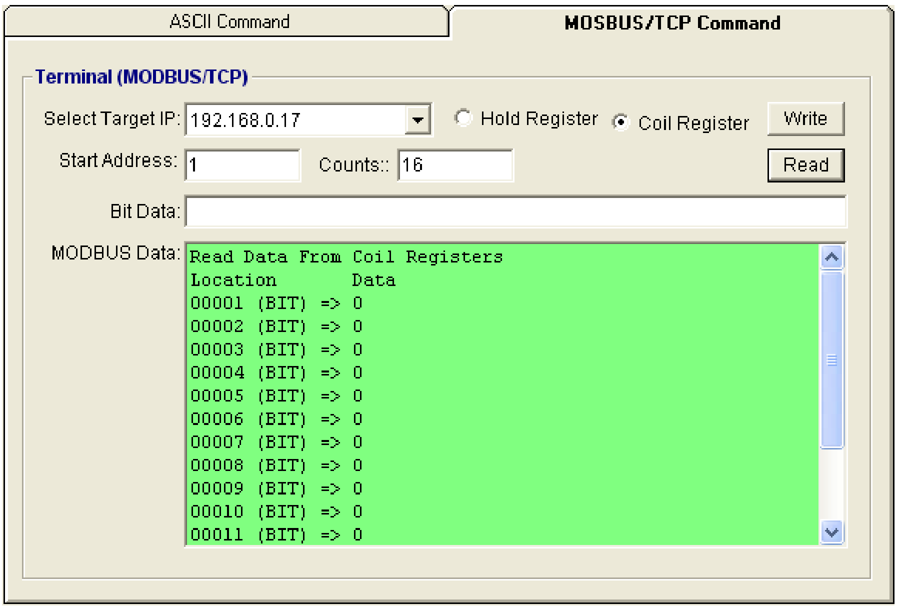


Select Target IP: 192.168.0.17
Command: $01M
Response: !019017
ASCII String:

```
$01M => !019017
```

Figure 5-5 ASCII Command Terminal

ModBus Hexadecimal mode: shown as Chapter 9



Select Target IP: 192.168.0.17 ( ) Hold Register (● Coil Register)
Start Address: 1 Counts:: 16
Bit Data:
MODBUS Data:

```
Read Data From Coil Registers
Location        Data
00001  (BIT)  =>  0
00002  (BIT)  =>  0
00003  (BIT)  =>  0
00004  (BIT)  =>  0
00005  (BIT)  =>  0
00006  (BIT)  =>  0
00007  (BIT)  =>  0
00008  (BIT)  =>  0
00009  (BIT)  =>  0
00010  (BIT)  =>  0
00011  (BIT)  =>  0
```

Figure 5-6 ModBus Terminal

5.9 Data /Event Stream

Data Stream Configuration:

In addition to TCP/IP communication protocol, EDAM-4200 supports UDP communication protocol to regularly broadcast data to specific host PCs. Click the tab of Data Stream, then configure the broadcasting interval and the specific IP addresses which need to receive data from the specific EDAM-4200 I/O module. This UDP Data Stream function broadcasts up to 8 host PCs simultaneously, and the interval is user-defined from 50ms to 7 Days.

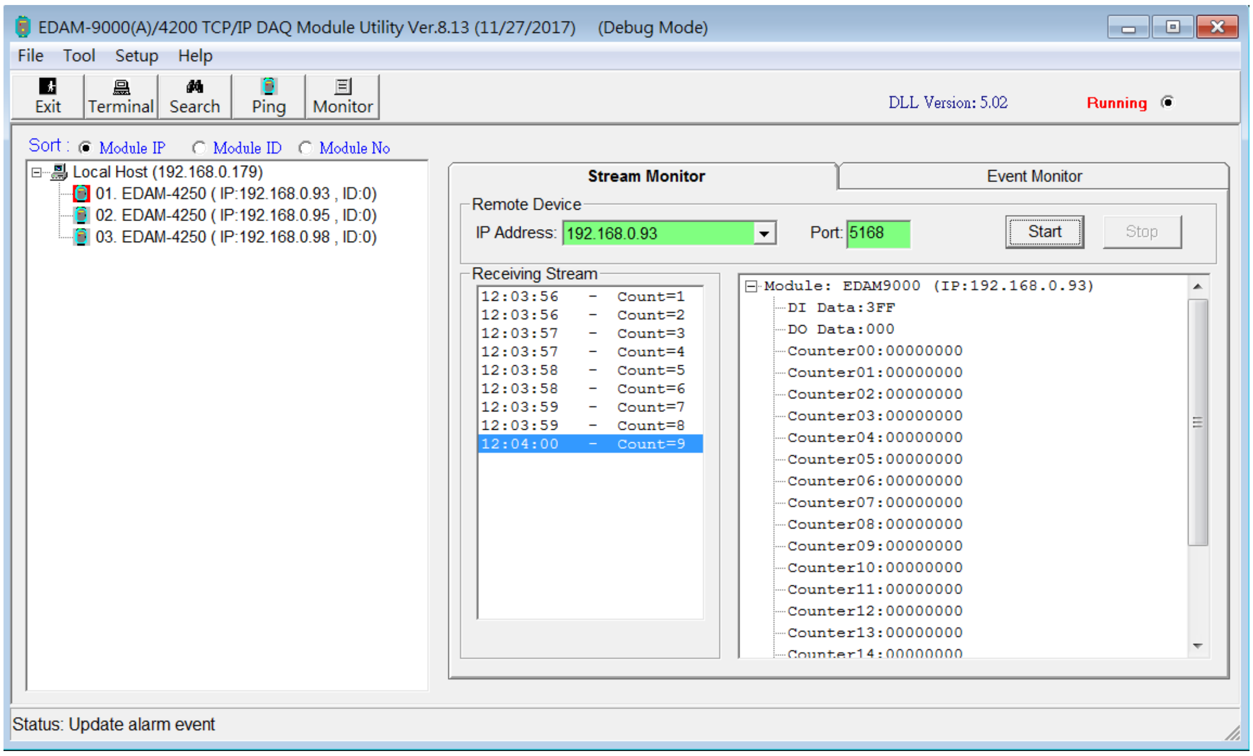
Event Stream Configuration:

In addition to TCP/IP communication protocol, EDAM-4200 supports UDP communication protocol to regularly broadcast data to specific host PCs. Click the tab of Data Stream, then configure the broadcasting interval and the specific IP addresses which need to receive data from the specific EDAM-4200 I/O module. This UDP Data Stream function broadcasts up to 8 host PCs simultaneously, and the interval is user-defined from 50ms to 7 Days.

Data Stream Monitoring:

After finishing the configuration of Data Stream, you can select the tab "Stream Monitor" in the function bar or click icon to call up operation display as Figure 1 7 Stream display.

Select the IP address of the EDAM-4200 you want to read data, then click "**Start** " button. The Utility software will begin to receive the stream data on this operation display.


EDAM-9000(A)/4200 TCP/IP DAQ Module Utility Ver.8.13 (11/27/2017) (Debug Mode)
File Tool Setup Help
Exit Terminal Search Ping Monitor
DLL Version: 5.02
Running
Sort : Module IP Module ID Module No
Stream Monitor
Event Monitor
Remote Device
Start
Stop
Receiving Stream

Sort : [✓ Module IP] Module ID Module No

```
Local Host (192.168.0.179)
    01. EDAM-4250 ( IP:192.168.0.93 , ID:0)
    02. EDAM-4250 ( IP:192.168.0.95 , ID:0)
    03. EDAM-4250 ( IP:192.168.0.98 , ID:0)
```

IP Address: 192.168.0.93 Port: 5168

```
12:03:56   -   Count=1
12:03:56   -   Count=2
12:03:57   -   Count=3
12:03:57   -   Count=4
12:03:58   -   Count=5
12:03:58   -   Count=6
12:03:59   -   Count=7
12:03:59   -   Count=8
12:04:00   -   Count=9
```

```
Module: EDAM9000 (IP:192.168.0.93)
   DI Data:3FF
   DO Data:000
   Counter00:00000000
   Counter01:00000000
   Counter02:00000000
   Counter03:00000000
   Counter04:00000000
   Counter05:00000000
   Counter06:00000000
   Counter07:00000000
   Counter08:00000000
   Counter09:00000000
   Counter10:00000000
   Counter11:00000000
   Counter12:00000000
   Counter13:00000000
   Counter14:00000000
```

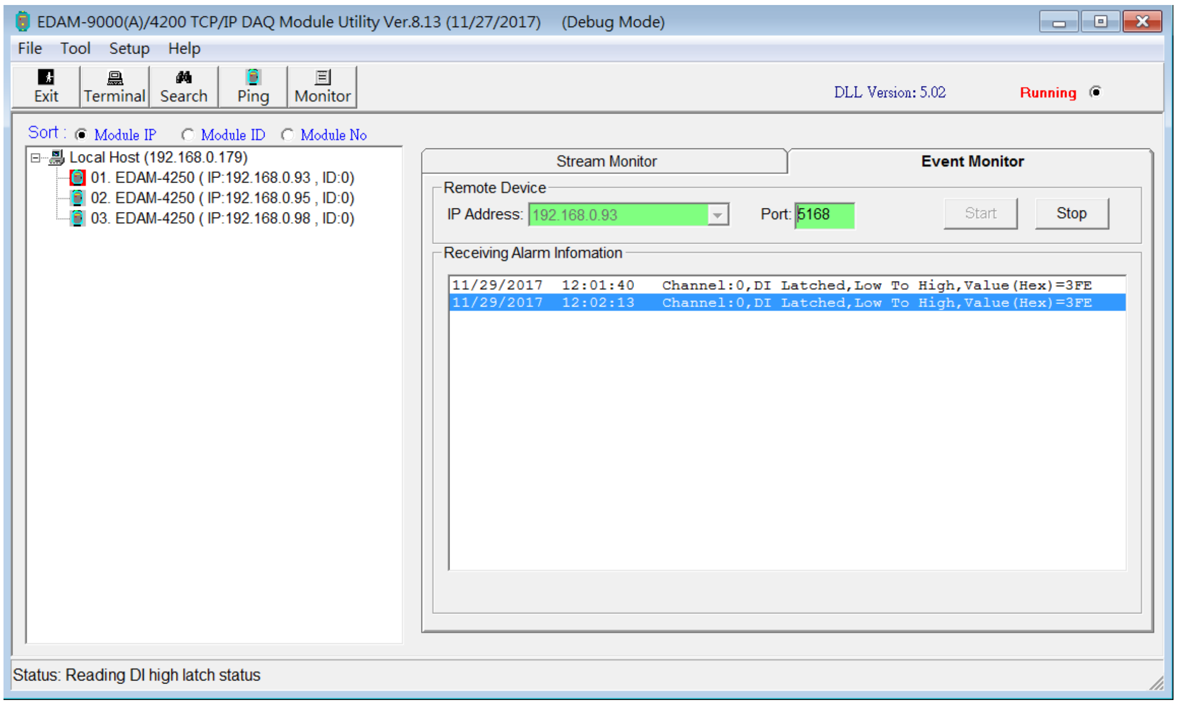
Status: Update alarm event

Figure 5-7 Stream display

Data Event Monitoring:

After finishing the configuration of Data Event(for DI latch Mode), you can select the tab "Event Monitor" in the function bar or click icon to call up operation display as Figure Event display.

Select the IP address of the EDAM-4200 you want to read DI data, then click "**Start**" button. The Utility software will begin to receive the stream DI data(DI changed) on this operation display.


EDAM-9000(A)/4200 TCP/IP DAQ Module Utility Ver.8.13 (11/27/2017) (Debug Mode)
File Tool Setup Help
Exit Terminal Search Ping Monitor
DLL Version: 5.02
Running
Sort : Module IP Module ID Module No
Stream Monitor
Event Monitor
Start
Stop

Local Host (192.168.0.179)
01. EDAM-4250 (IP:192.168.0.93 , ID:0)
02. EDAM-4250 (IP:192.168.0.95 , ID:0)
03. EDAM-4250 (IP:192.168.0.98 , ID:0)

Remote Device
IP Address: 192.168.0.93 Port: 5168

Receiving Alarm Infomation

```
11/29/2017  12:01:40   Channel:0,DI Latched,Low To High,Value(Hex)=3FE
11/29/2017  12:02:13   Channel:0,DI Latched,Low To High,Value(Hex)=3FE
```

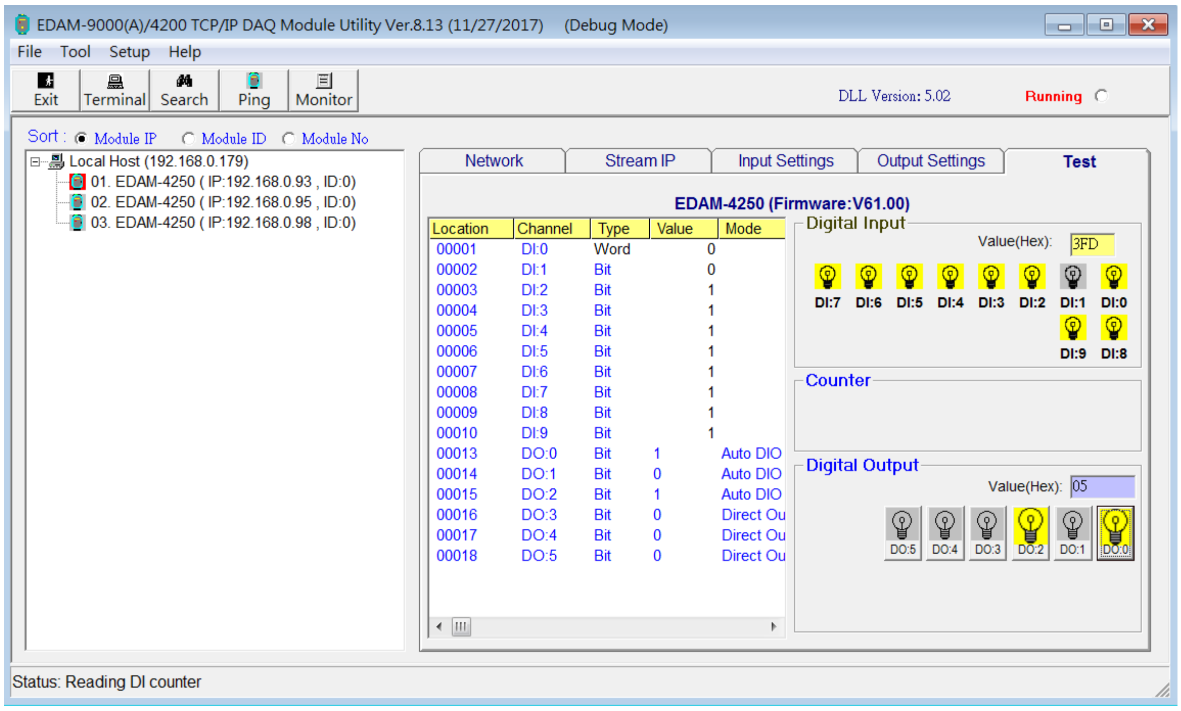
Status: Reading DI high latch status

Figure 5-8 Event display

5.10 Digital I/O Module Settings

Selecting EDAM-4200 Digital Modules and select "**Test"** tab, user can read following information from the Utility.

5.10.1 Digital Test Tab


EDAM-9000(A)/4200 TCP/IP DAQ Module Utility Ver.8.13 (11/27/2017) (Debug Mode)
File Tool Setup Help
Exit Terminal Search Ping Monitor
DLL Version: 5.02
Running
Sort : Module IP Module ID Module No
Local Host (192.168.0.179)
01. EDAM-4250 (IP:192.168.0.93 , ID:0)
02. EDAM-4250 (IP:192.168.0.95 , ID:0)
03. EDAM-4250 (IP:192.168.0.98 , ID:0)
Network Stream IP Input Settings Output Settings Test
Digital Input
DI:7 DI:6 DI:5 DI:4 DI:3 DI:2 DI:1 DI:0 DI:9 DI:8
Counter
Digital Output
DO:5 DO:4 DO:3 DO:2 DO:1 DO:0

EDAM-4250 (Firmware:V61.00)

Location	Channel	Type	Value	Mode
00001	DI:0	Word	0	
00002	DI:1	Bit	0	
00003	DI:2	Bit	1	
00004	DI:3	Bit	1	
00005	DI:4	Bit	1	
00006	DI:5	Bit	1	
00007	DI:6	Bit	1	
00008	DI:7	Bit	1	
00009	DI:8	Bit	1	
00010	DI:9	Bit	1	
00013	DO:0	Bit	1	Auto DIO
00014	DO:1	Bit	0	Auto DIO
00015	DO:2	Bit	1	Auto DIO
00016	DO:3	Bit	0	Direct Ou
00017	DO:4	Bit	0	Direct Ou
00018	DO:5	Bit	0	Direct Ou

Digital Input Value(Hex): 3FD

Digital Output Value(Hex): 05

Status: Reading DI counter

Figure 5-9 ModBus location and I/O status

Digital I/O Module Test tab"

Location : Standard Modbus address. EDAM Ethernet I/O Utility shows the Modbus mapping address of each I/O channel. And the addresses will be the indexes for applying into the database of HMI or OPC Server.

Channel : Indicate the channel number of digital I/O module.

Type : Data Type of the I/O channel. The data type of Digital I/O modules is always "Bit".

Value : The current status on each channel of I/O Module. The value of digital I/O modules could be "0" (OFF) or "1" (ON).

Mode :Describes the I/O types of the specific module. In addition to monitor the current DI/DO status, the Windows Utility offers a graphical operating interface as Figure1 12 DI/O status display. You can read the Digital input status through the change of the indicator icons. Oppositely, you can write the digital output status through clicking the indicator icons.

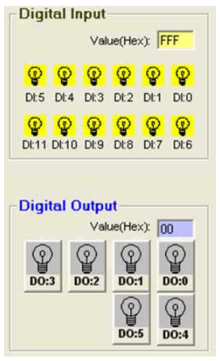
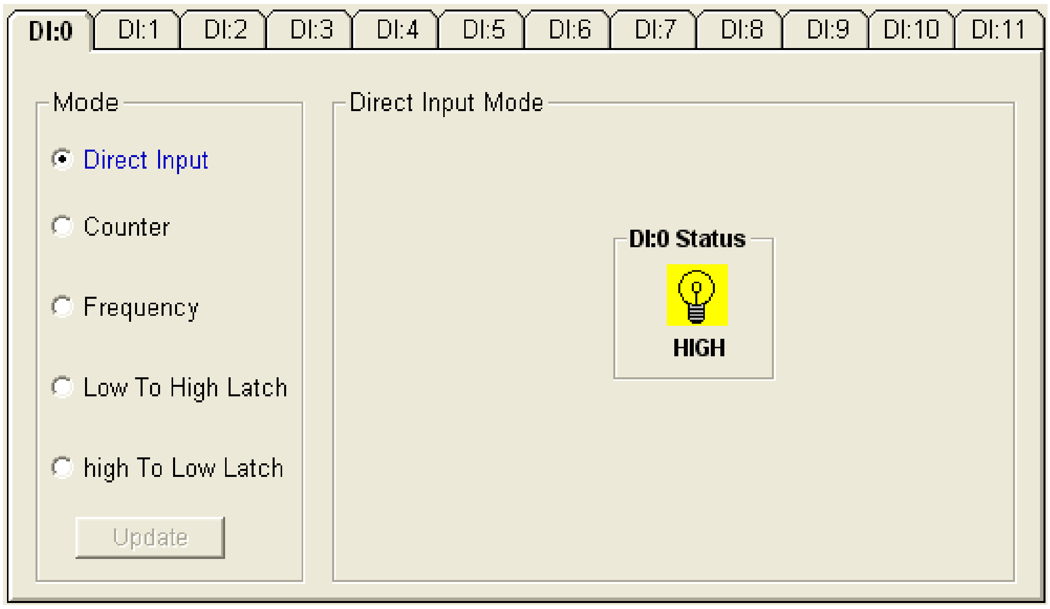


Figure 5-10 DI/O status display

5.10.2 Digital Input Settings Tab

The digital input channels support counter and signal latch functions. Click the specific channel, there will be five working modes for choosing.


DI:0 | DI:1 | DI:2 | DI:3 | DI:4 | DI:5 | DI:6 | DI:7 | DI:8 | DI:9 | DI:10 | DI:11

Mode
- (●) Direct Input
- ( ) Counter
- ( ) Frequency
- ( ) Low To High Latch
- ( ) high To Low Latch

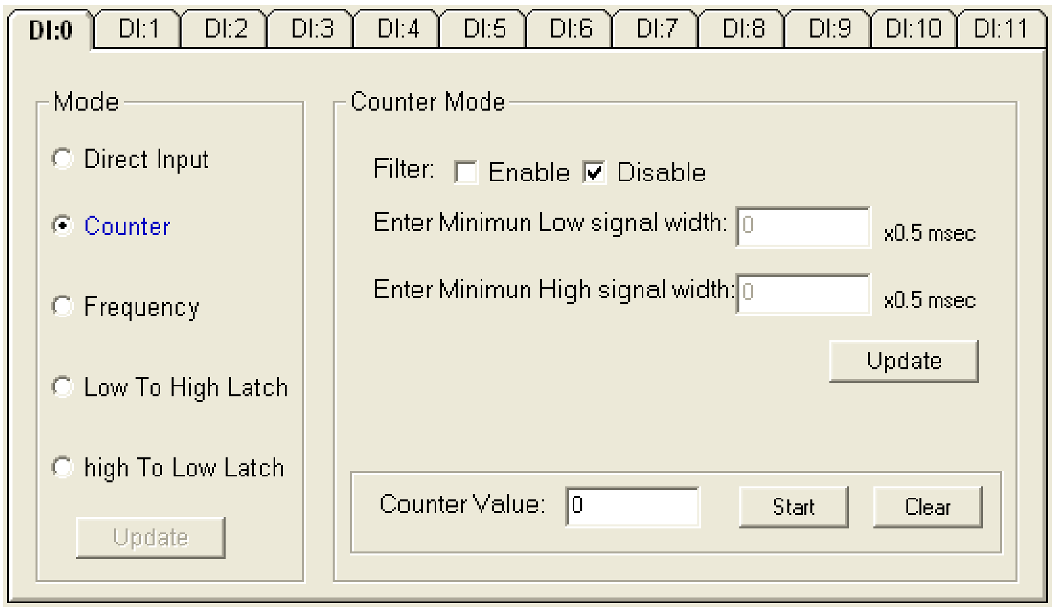
Update

Direct Input Mode

DI:0 Status

HIGH

Figure 5-11 Direct input mode


DI:0 | DI:1 | DI:2 | DI:3 | DI:4 | DI:5 | DI:6 | DI:7 | DI:8 | DI:9 | DI:10 | DI:11

Mode
- ( ) Direct Input
- (●) Counter
- ( ) Frequency
- ( ) Low To High Latch
- ( ) high To Low Latch

Update

Counter Mode

Filter: ☐ Enable ☑ Disable

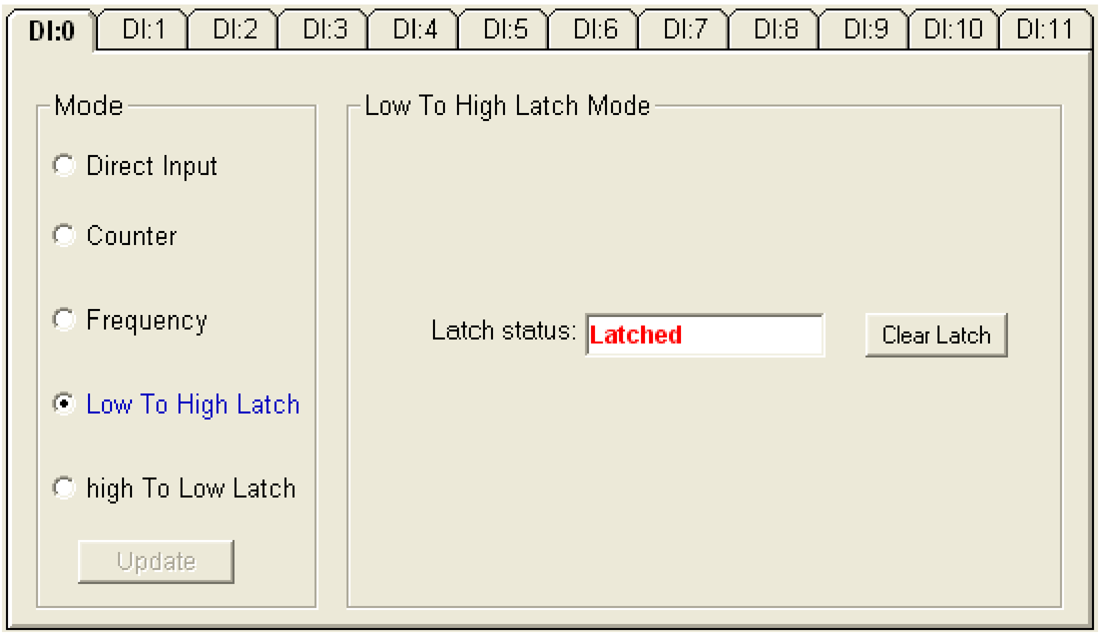
Enter Minimun Low signal width: 0 x0.5 msec

Enter Minimun High signal width: 0 x0.5 msec

Update

Counter Value: 0 Start Clear

Figure 5-12 Counter mode



Mode: [✓ Low To High Latch]

Low To High Latch Mode

Latch status: Latched

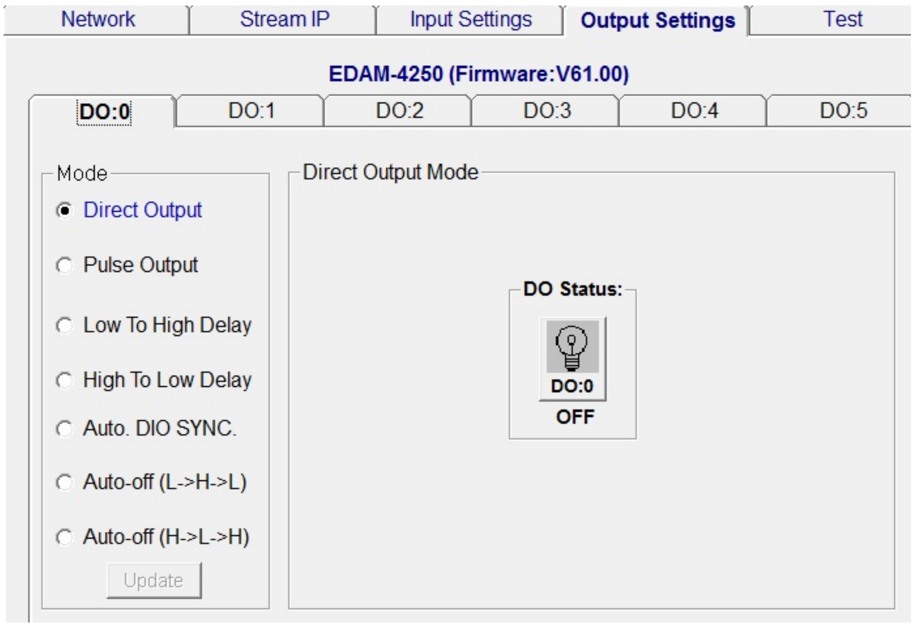
Figure 5-13 Input latch mode

Note:

1. The new working mode setting will take effective after click the "Update" button.
2. If necessary, users could invert the original single for flexible operation needs.
3. Supported Data Event Monitoring.

5.10.3 Digital Output Settings Tab

The digital output channels support pulse output and delay output functions. Click the specific channel, there will be four working modes for choosing.



Network | Stream IP | Input Settings | **Output Settings** | Test

EDAM-4250 (Firmware:V61.00)

DO:0 | DO:1 | DO:2 | DO:3 | DO:4 | DO:5

Mode
- (●) Direct Output
- ( ) Pulse Output
- ( ) Low To High Delay
- ( ) High To Low Delay
- ( ) Auto. DIO SYNC.
- ( ) Auto-off (L->H->L)
- ( ) Auto-off (H->L->H)

Update

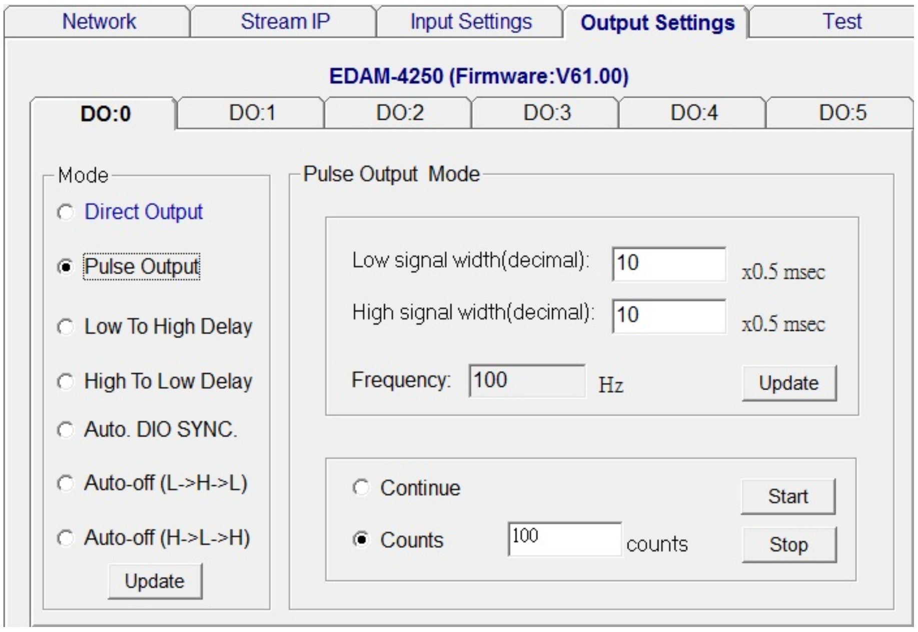
Direct Output Mode

DO Status:

DO:0

OFF

Figure 5-14 **Direct output mode**



Network | Stream IP | Input Settings | **Output Settings** | Test

EDAM-4250 (Firmware:V61.00)

DO:0 | DO:1 | DO:2 | DO:3 | DO:4 | DO:5

Mode
- ( ) Direct Output
- (●) Pulse Output
- ( ) Low To High Delay
- ( ) High To Low Delay
- ( ) Auto. DIO SYNC.
- ( ) Auto-off (L->H->L)
- ( ) Auto-off (H->L->H)

Update

Pulse Output Mode

Low signal width(decimal): 10 x0.5 msec

High signal width(decimal): 10 x0.5 msec

Frequency: 100 Hz

Update

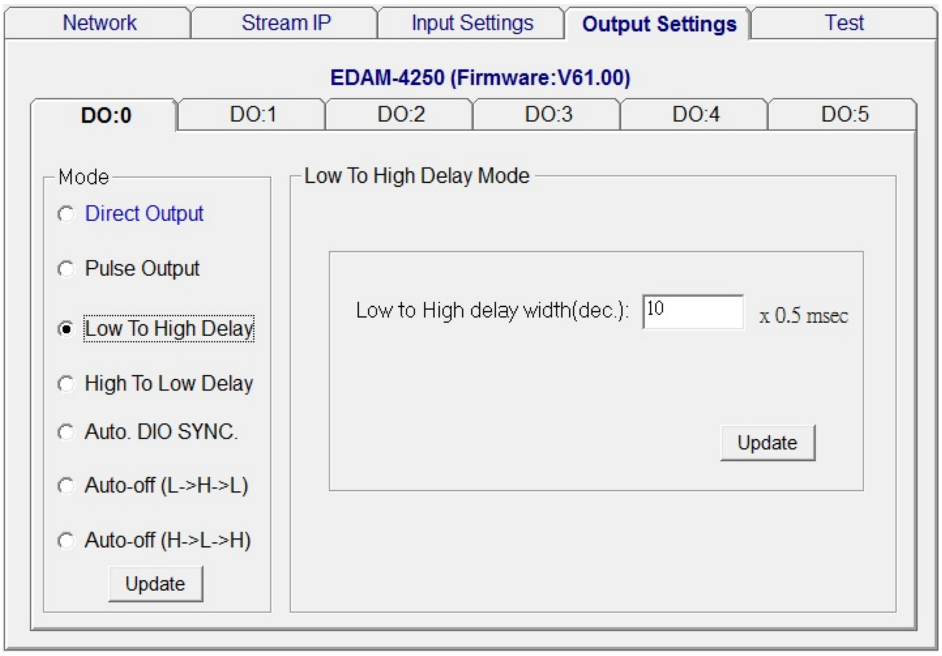
( ) Continue

(●) Counts 100 counts

Start

Stop

Figure 5-15 **Pulse output mode**



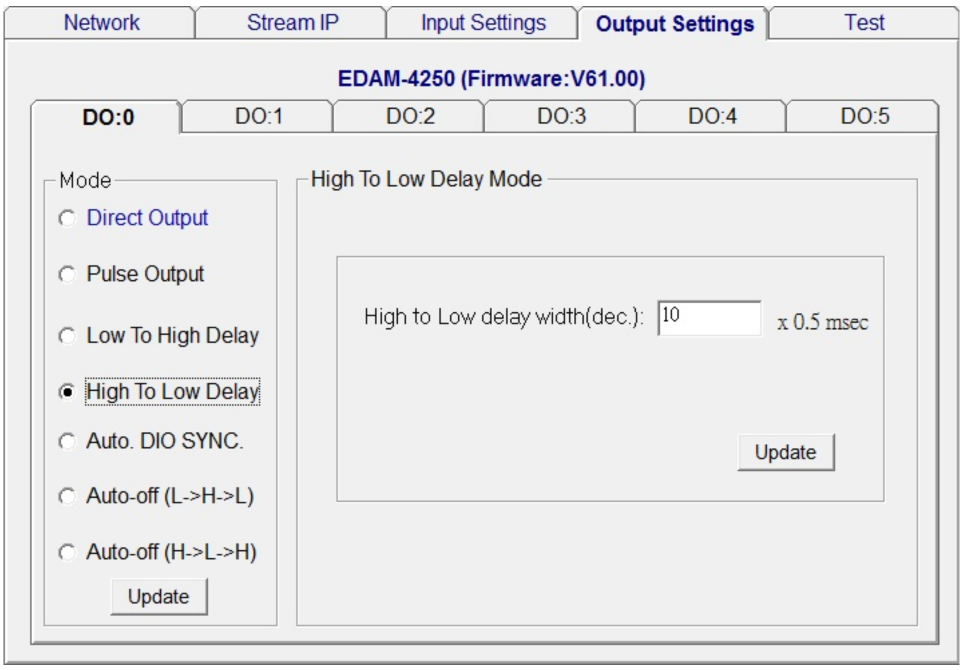
EDAM-4250 (Firmware:V61.00)

Mode: [✓ Low To High Delay]

Low To High Delay Mode

Low to High delay width(dec.): 10 x 0.5 msec

Figure 5-16 **Low to High Delay mode**



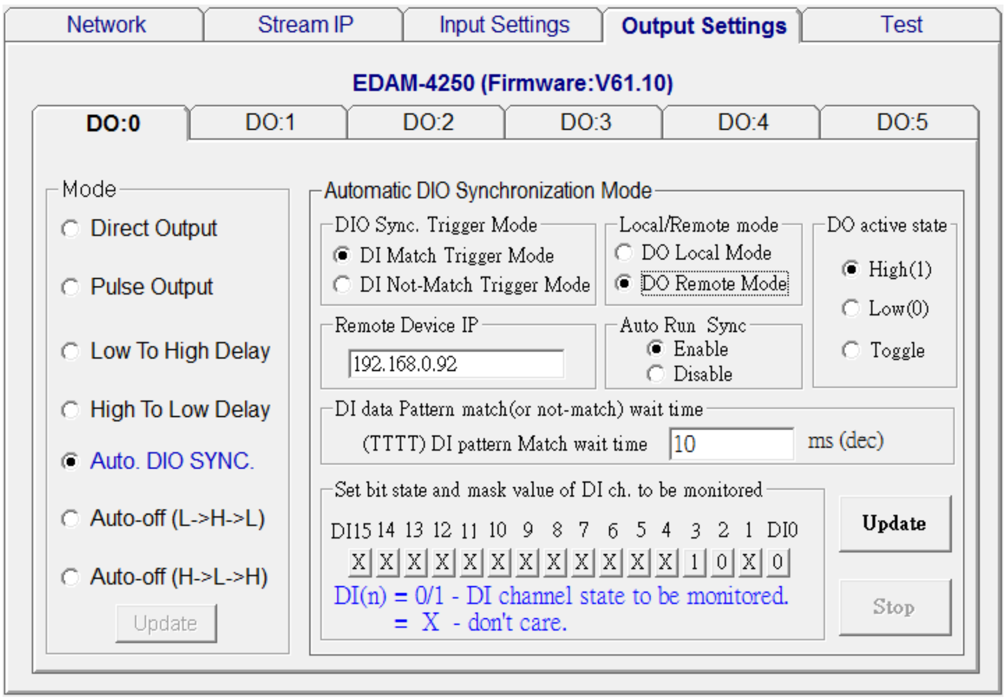
EDAM-4250 (Firmware:V61.00)

Mode: [✓ High To Low Delay]

High To Low Delay Mode

High to Low delay width(dec.): 10 x 0.5 msec

Figure 5-17 **High to Low Delay mode**


Network | Stream IP | Input Settings | **Output Settings** | Test

EDAM-4250 (Firmware:V61.10)

DO:0 | DO:1 | DO:2 | DO:3 | DO:4 | DO:5

Mode
- Direct Output
- Pulse Output
- Low To High Delay
- High To Low Delay
- [✓ Auto. DIO SYNC.]
- Auto-off (L->H->L)
- Auto-off (H->L->H)

Update

Automatic DIO Synchronization Mode

DIO Sync. Trigger Mode: [✓ DI Match Trigger Mode] DI Not-Match Trigger Mode

Local/Remote mode: DO Local Mode [✓ DO Remote Mode]

DO active state: [✓ High(1)] Low(0) Toggle

Remote Device IP: 192.168.0.92

Auto Run Sync: [✓ Enable] Disable

DI data Pattern match(or not-match) wait time

(TTTT) DI pattern Match wait time 10 ms (dec)

Set bit state and mask value of DI ch. to be monitored

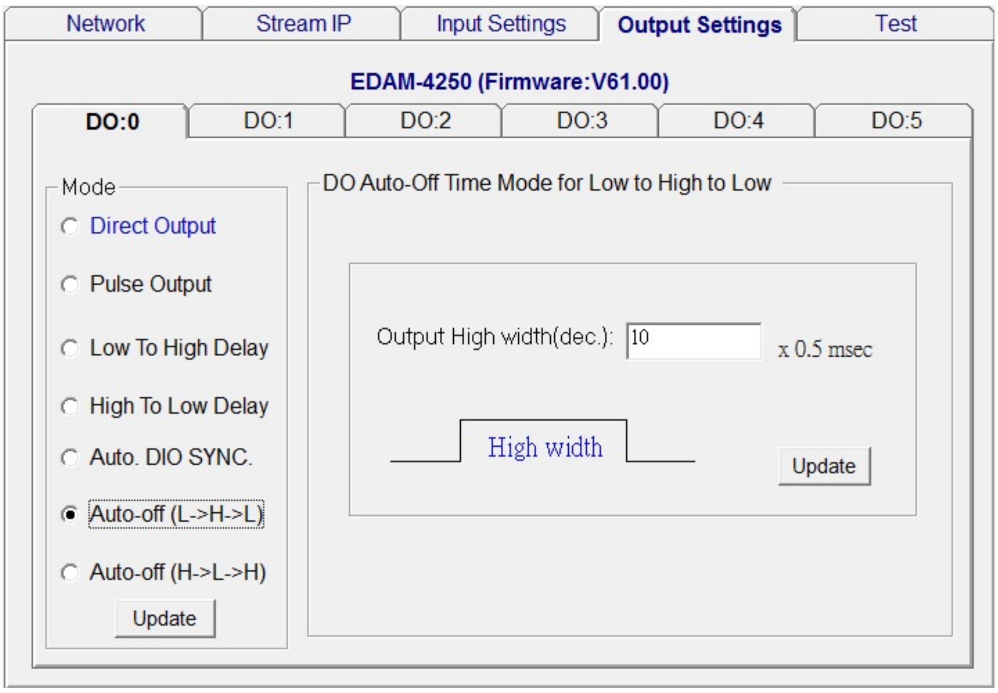
DI15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	DI0
X	X	X	X	X	X	X	X	X	X	X	X	1	0	X	0

DI(n) = 0/1 - DI channel state to be monitored.
= X - don't care.

Update

Stop

Figure 5-18 **Auto DIO SYNC. mode**


Network | Stream IP | Input Settings | **Output Settings** | Test

EDAM-4250 (Firmware:V61.00)

DO:0 | DO:1 | DO:2 | DO:3 | DO:4 | DO:5

Mode
- Direct Output
- Pulse Output
- Low To High Delay
- High To Low Delay
- Auto. DIO SYNC.
- [✓ Auto-off (L->H->L)]
- Auto-off (H->L->H)

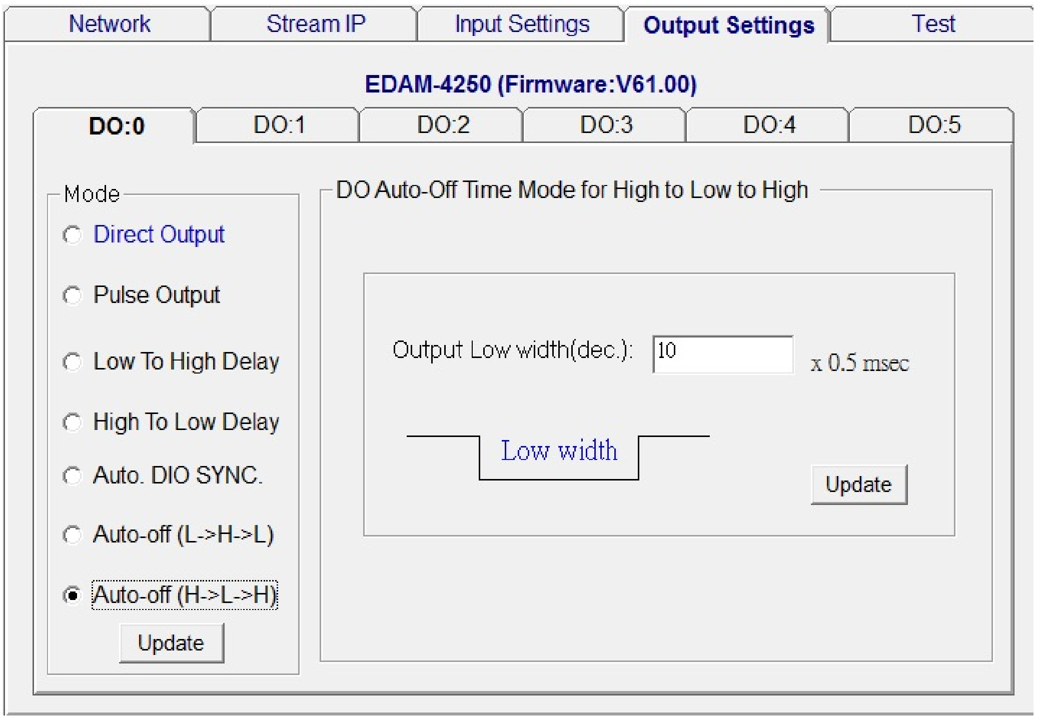
Update

DO Auto-Off Time Mode for Low to High to Low

Output High width(dec.): 10 x 0.5 msec

Update

Figure 5-19 **Auto-Off Time(L->H->L) Mode**



EDAM-4250 (Firmware:V61.00)

Mode: Auto-off (H->L->H) ●

DO Auto-Off Time Mode for High to Low to High

Output Low width(dec.): 10 x 0.5 msec

Figure 5-20 **Auto-Off Time(H->L->H) Mode**

Chapter 6 What is TCPDAQ ActiveX Control ?

TCPDAQ.OCX is a collection of ActiveX controls for performing I/O operations within any compatible ActiveX control container, such as Visual Basic, Delphi, etc. You can easily perform the I/O operations through properties, events and methods. Specific information about the properties, methods, and events of the TCPDAQ ActiveX controls can be found later in this manual.

With TCPDAQ ActiveX Control, you can perform versatile I/O operations to control your Inlog EDAM-4200 module series.

The TCPDAQ ActiveX Control setup program installs TCPDAQ.OCX through a process that may take several minutes. Installing the necessary software to use the TCPDAQ.OCX in your application involves two main steps: Installing the TCPDAQ ActiveX Control

Use the Inlog EDAM-4200 utility to configure the modules that is attached to your computer.

You can use these ActiveX controls in any development tool that supports them, including Microsoft Visual C++, Microsoft Visual Basic, Borland C++ Builder, Borland Delphi

6.1 Installing the TCPDAQ ActiveX Controls

Before using the TCPDAQ ActiveX Control, you must install the TCPDAQ.OCX first

♦ Insert the TCPDAQ installation CD-ROM disc into your computer.

♦ The installation program should start automatically. If autorun is not enabled on your computer, use your Windows Explorer or the Windows Run command to execute Setup.exe on the TCPDAQ installation CD-ROM disc (Assume "d" is the letter of your CD-ROM disc drive): **D: \Setup.exe**

6.2 Building TCPDAQ ActiveX Control with Various Tools

This chapter describes how you can use the TCPDAQ ActiveX Control with the following development tools:

♦ Microsoft Visual C++ version 6.0 (SP5)

♦ Microsoft Visual Basic version 6.0 (SP5)

♦ Borland Delphi version 4.0 (with the Delphi 6 Update Pack fixes for ActiveX installed)

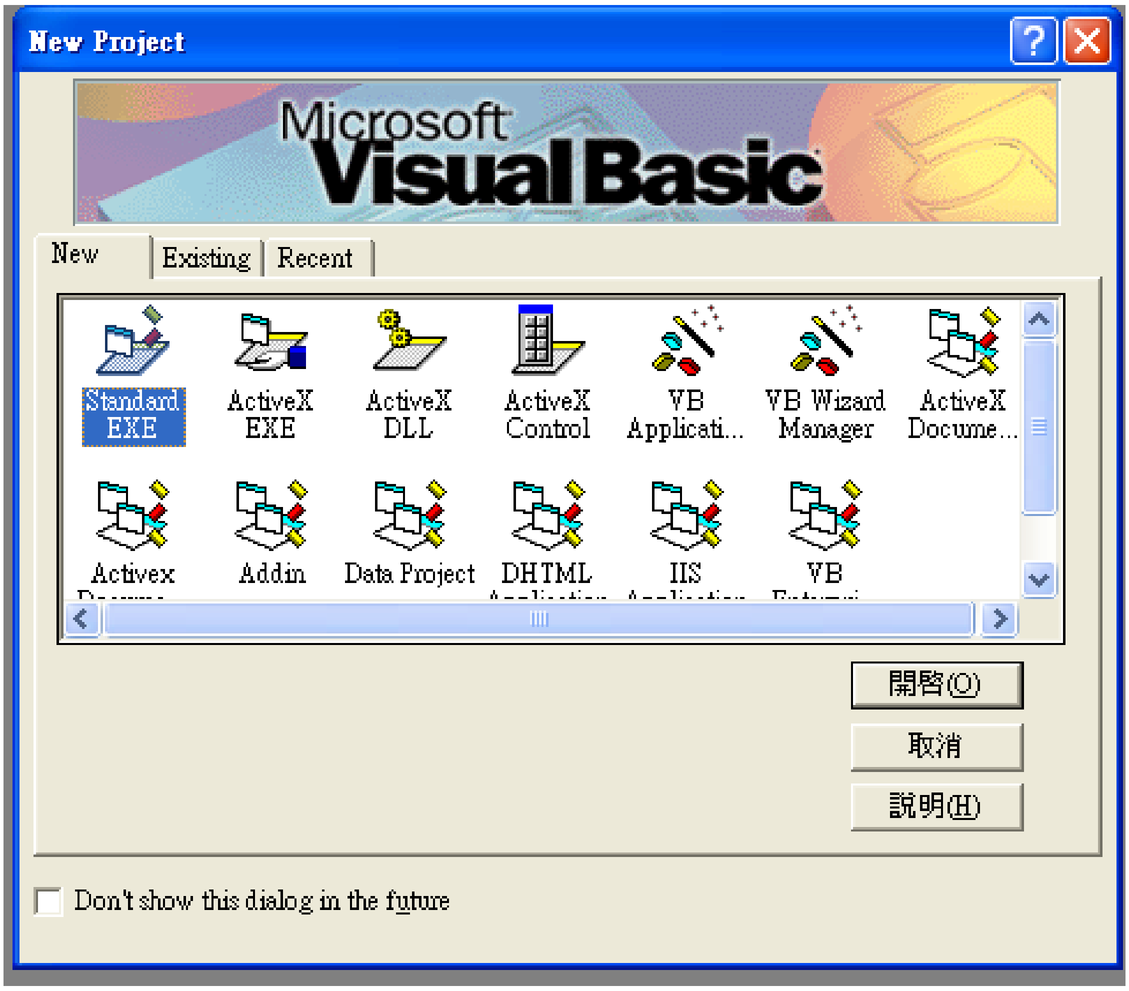
♦ Borland C++ Builder version 5.0

This chapter assumes that you are familiar with the basic concepts of using Visual Basic, Delphi, Borland C++ Builder, and Visual C++, including selecting the type of application, designing the form, placing the control on the form, configuring the properties of the control, creating the code (event handler routines) for this control.

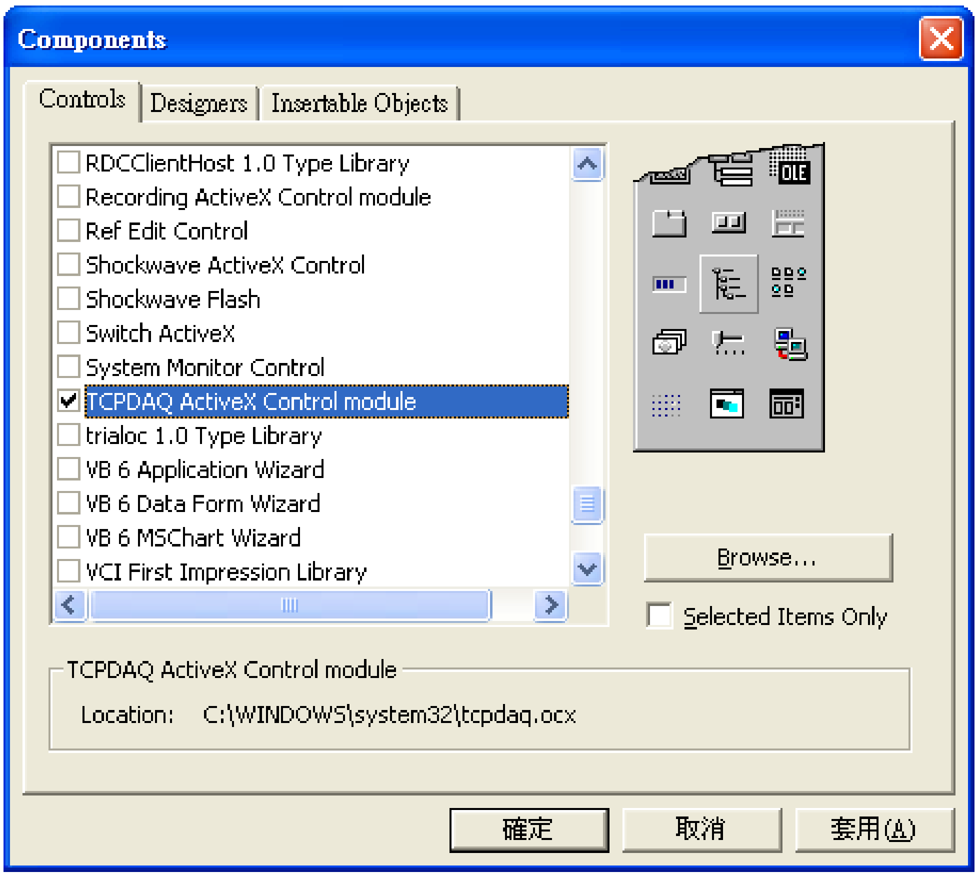
Note: For Borland Delphi 6, the Delphi 6 Update Pack fixes for ActiveX must be installed.

6.3 Building TCPDAQ Applications with Visual Basic

♦ Start Visual Basic.



♦ Select **Standard EXE** icon and press the **Open** button. A new project is created. Click on **Components...** from the **Project** menu. The Components dialog box is loaded as shown below:

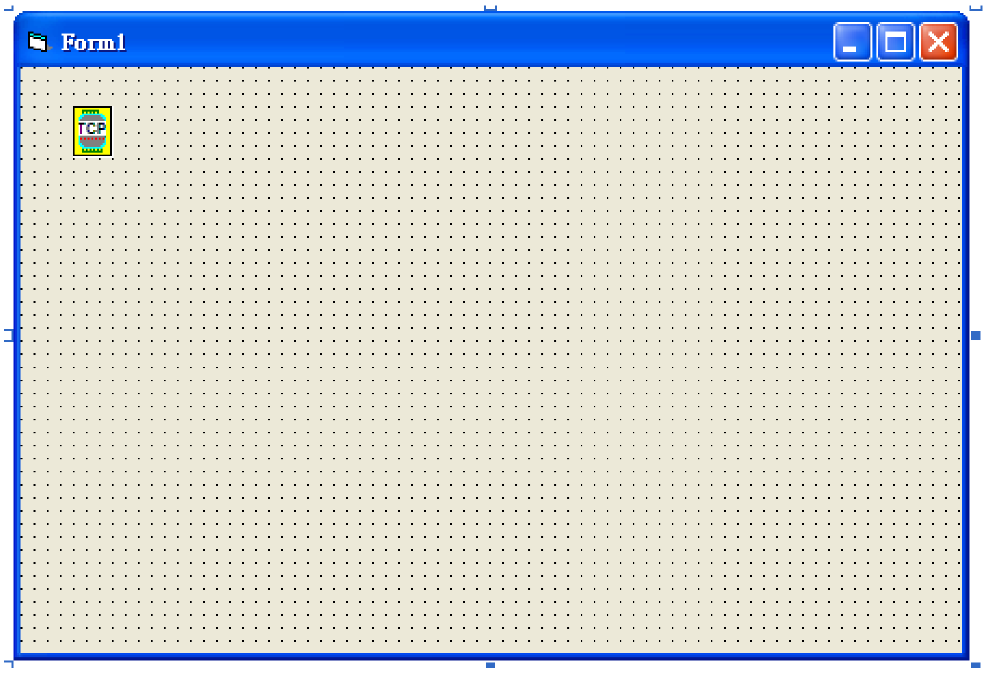


- [ ] RDCClientHost 1.0 Type Library
- [ ] Recording ActiveX Control module
- [ ] Ref Edit Control
- [ ] Shockwave ActiveX Control
- [ ] Shockwave Flash
- [ ] Switch ActiveX
- [ ] System Monitor Control
- [x] TCPDAQ ActiveX Control module
- [ ] trialoc 1.0 Type Library
- [ ] VB 6 Application Wizard
- [ ] VB 6 Data Form Wizard
- [ ] VB 6 MSChart Wizard
- [ ] VCI First Impression Library

TCPDAQ ActiveX Control module
Location: C:\WINDOWS\system32\tcpdaq.ocx

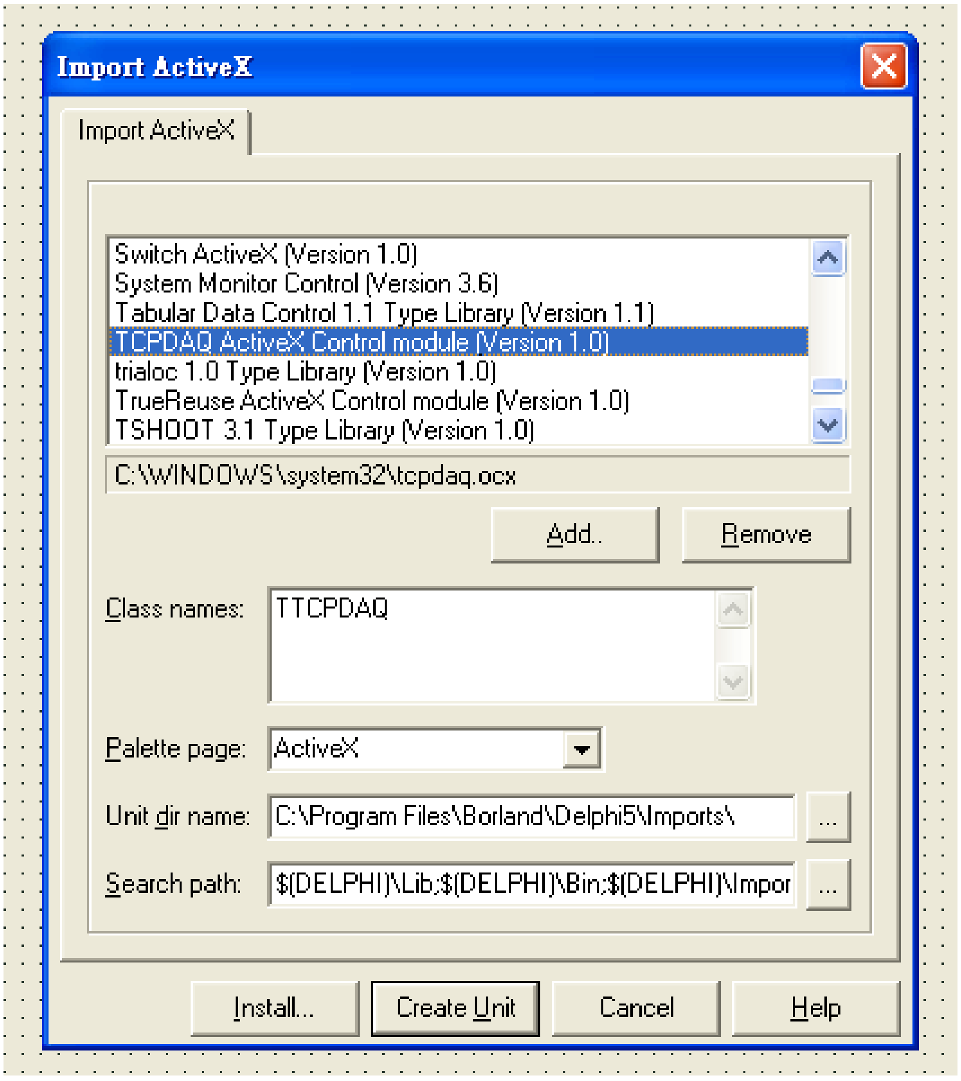
♦ Place a TCPDAQ control from the Toolbox on the form. Use the default name.

♦ Your form should look similar to the one shown below:



6.4 Building TCPDAQ Applications with Delphi

- ♦ Start Delphi, Delphi will launch as shown below:
- ♦ Select **Import ActiveX Control...** from the **Component** menu. The Import ActiveX dialog box loads:
- ♦ Select the TCPDAQ ActiveX Control Module and press the **Install...** button. A dialog box is displayed as follows:



Switch ActiveX (Version 1.0)
System Monitor Control (Version 3.6)
Tabular Data Control 1.1 Type Library (Version 1.1)
TCPDAQ ActiveX Control module (Version 1.0)
trialoc 1.0 Type Library (Version 1.0)
TrueReuse ActiveX Control module (Version 1.0)
TSHOOT 3.1 Type Library (Version 1.0)

C:\WINDOWS\system32\tcpdaq.ocx

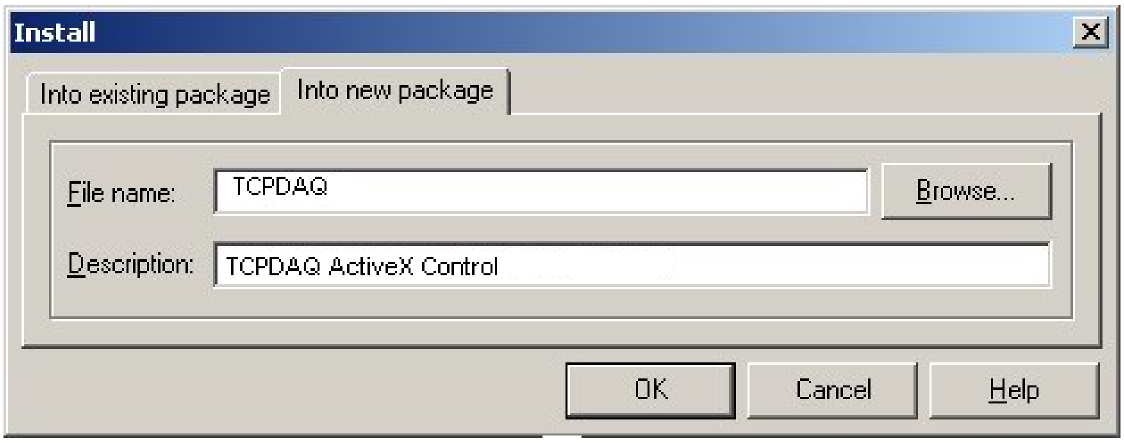
Class names: TTCPDAQ

Palette page: ActiveX

Unit dir name: C:\Program Files\Borland\Delphi5\Imports\

Search path: $(DELPHI)\Lib;$(DELPHI)\Bin;$(DELPHI)\Impor

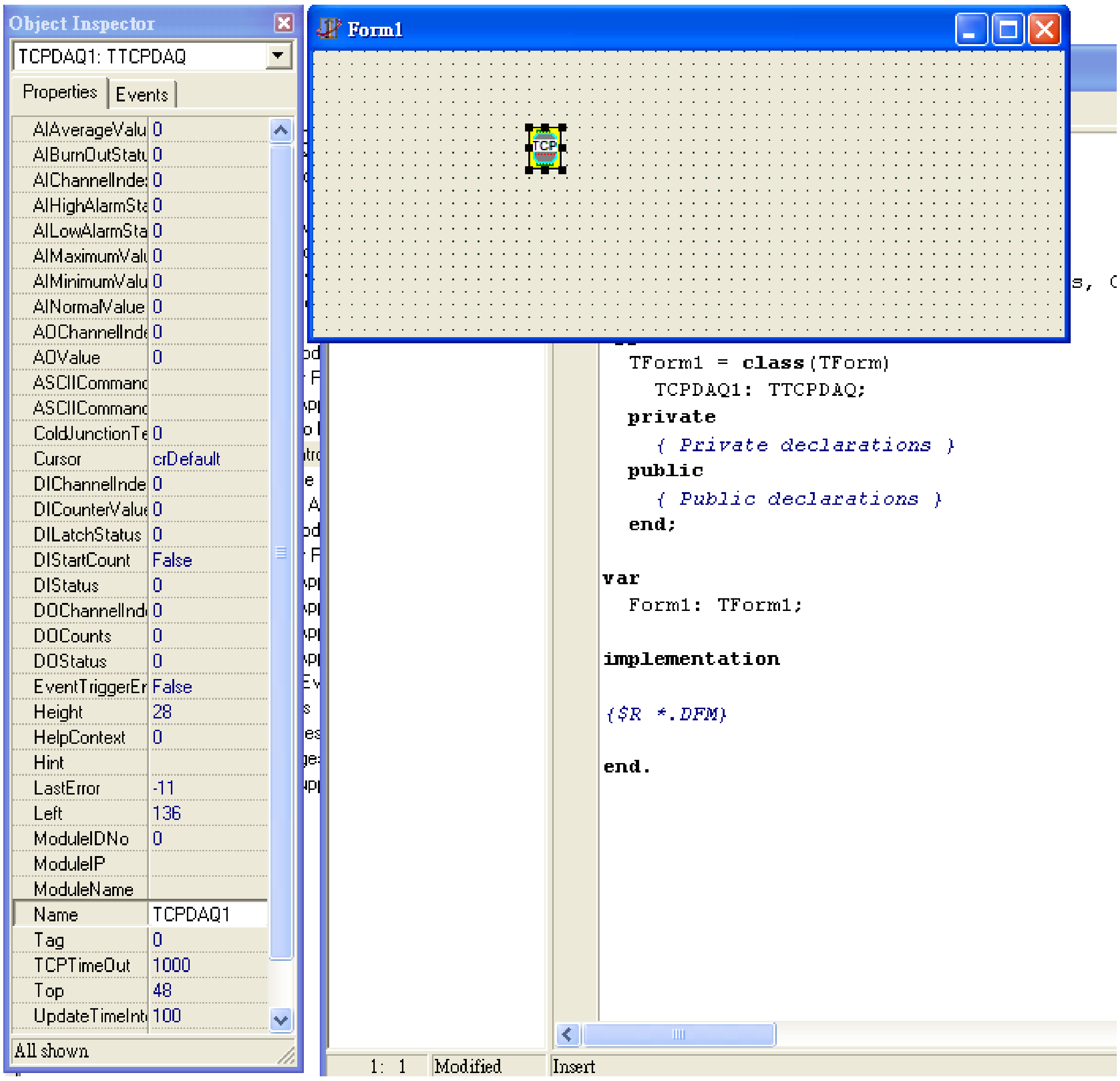
The TCPDAQ control is loaded into the **Component Palette**. You can check it by clicking on **Install Package...** from the **Component** menu. A dialog box is shown as below.



File name: TCPDAQ

Description: TCPDAQ ActiveX Control

- Switch to the form and select the ActiveX tab from the **Component Palette**.
- Place a TCPDAQ control from the **Component Palette** on the form. Use the default names TCPDAQ1.
- Your form should look similar to the one shown below:



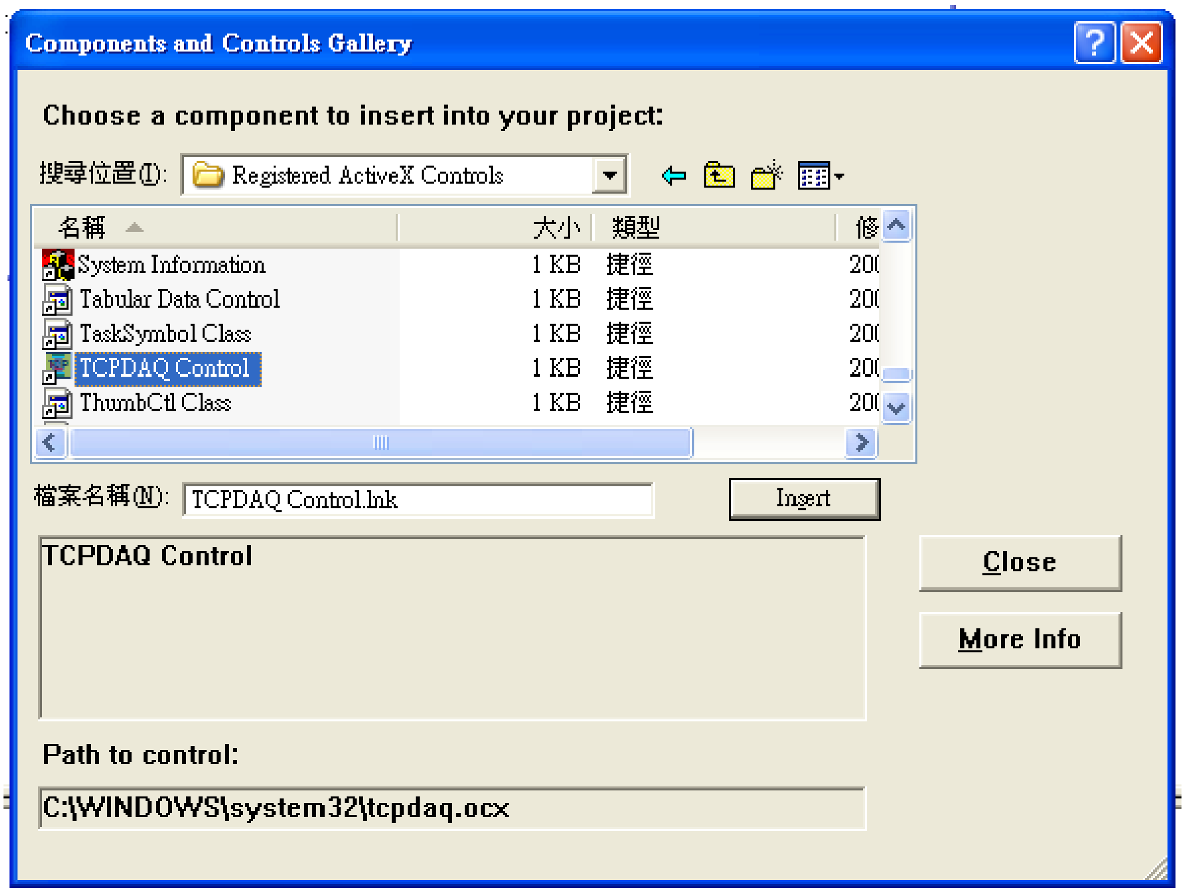
TCPDAQ1: TTCPDAQ

AIAverageValu	0
AIBurnOutStatu	0
AIChannelIndex	0
AIHighAlarmSta	0
AILowAlarmSta	0
AIMaximumValu	0
AIMinimumValu	0
AINormalValue	0
AOChannelInde	0
AOValue	0
ASCIICommanc	
ASCIICommanc	
ColdJunctionTe	0
Cursor	crDefault
DIChannelInde	0
DICounterValue	0
DILatchStatus	0
DIStartCount	False
DIStatus	0
DOChannelInd	0
DOCounts	0
DOStatus	0
EventTriggerEr	False
Height	28
HelpContext	0
Hint	
LastError	-11
Left	136
ModuleIDNo	0
ModuleIP	
ModuleName	
Name	TCPDAQ1
Tag	0
TCPTimeOut	1000
Top	48
UpdateTimeInt	100

```
  TForm1 = class(TForm)
    TCPDAQ1: TTCPDAQ;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

6.5 Building TCPDAQ Applications with Visual C++

- ♦ Start Visual C++ program.
- ♦ Select **Add to Project→ Components and Controls** from the **Project** menu, and double-click on **Registered ActiveX Controls**. The result should be as below:



Components and Controls Gallery
Choose a component to insert into your project:
搜尋位置(I): Registered ActiveX Controls
名稱
大小
類型
修
Insert
Close
More Info

名稱	大小	類型	修
System Information	1 KB	捷徑	20(
Tabular Data Control	1 KB	捷徑	20(
TaskSymbol Class	1 KB	捷徑	20(
TCPDAQ Control	1 KB	捷徑	20(
ThumbCtl Class	1 KB	捷徑	20(

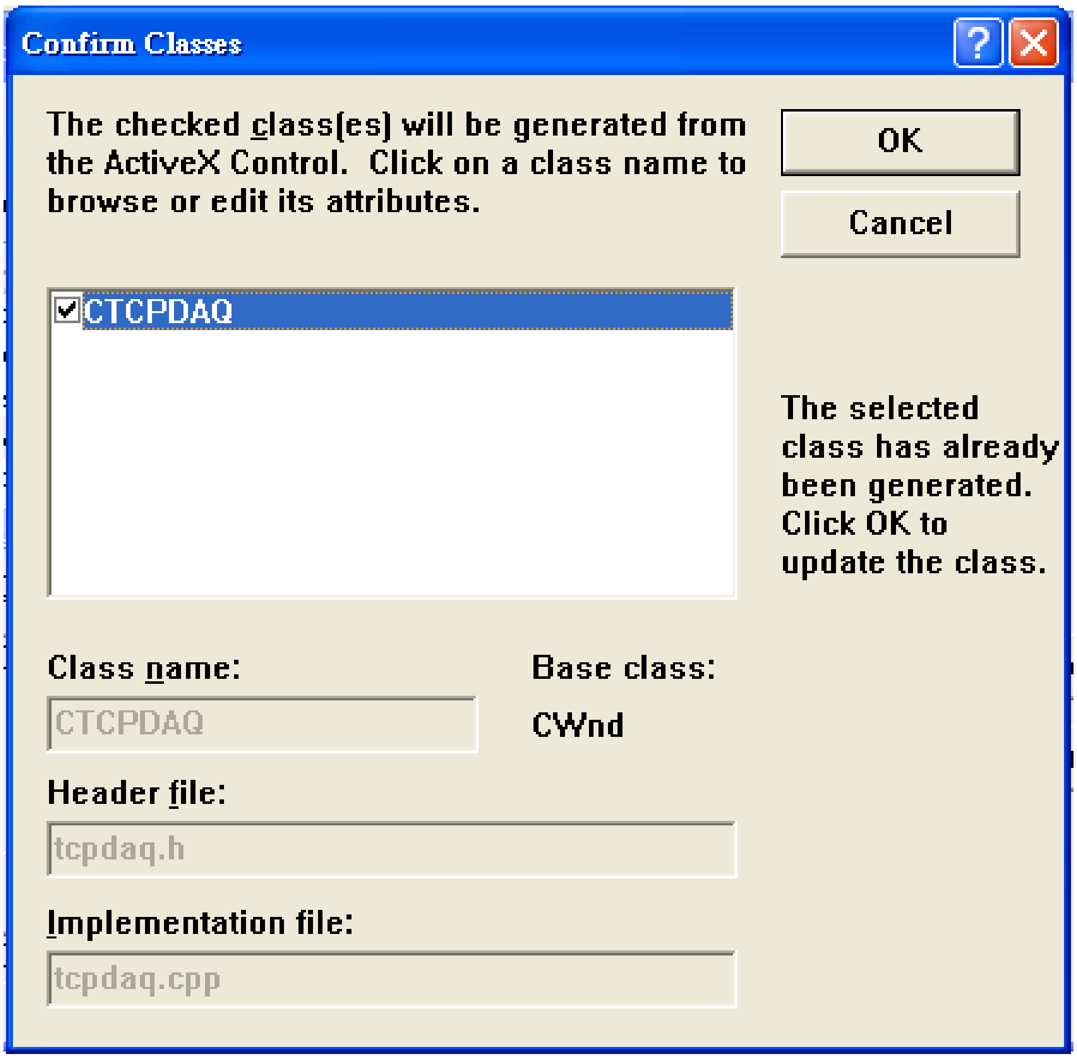
檔案名稱(N): TCPDAQ Control.lnk

TCPDAQ Control

Path to control:

C:\WINDOWS\system32\tcpdaq.ocx

♦ Scroll down to the TCPDAQ Control and press the **Insert** button. A Class Confirm dialog box is displayed, Press **OK** button.



Confirm Classes

The checked class(es) will be generated from the ActiveX Control. Click on a class name to browse or edit its attributes.

☑ CTCPDAQ

The selected class has already been generated. Click OK to update the class.

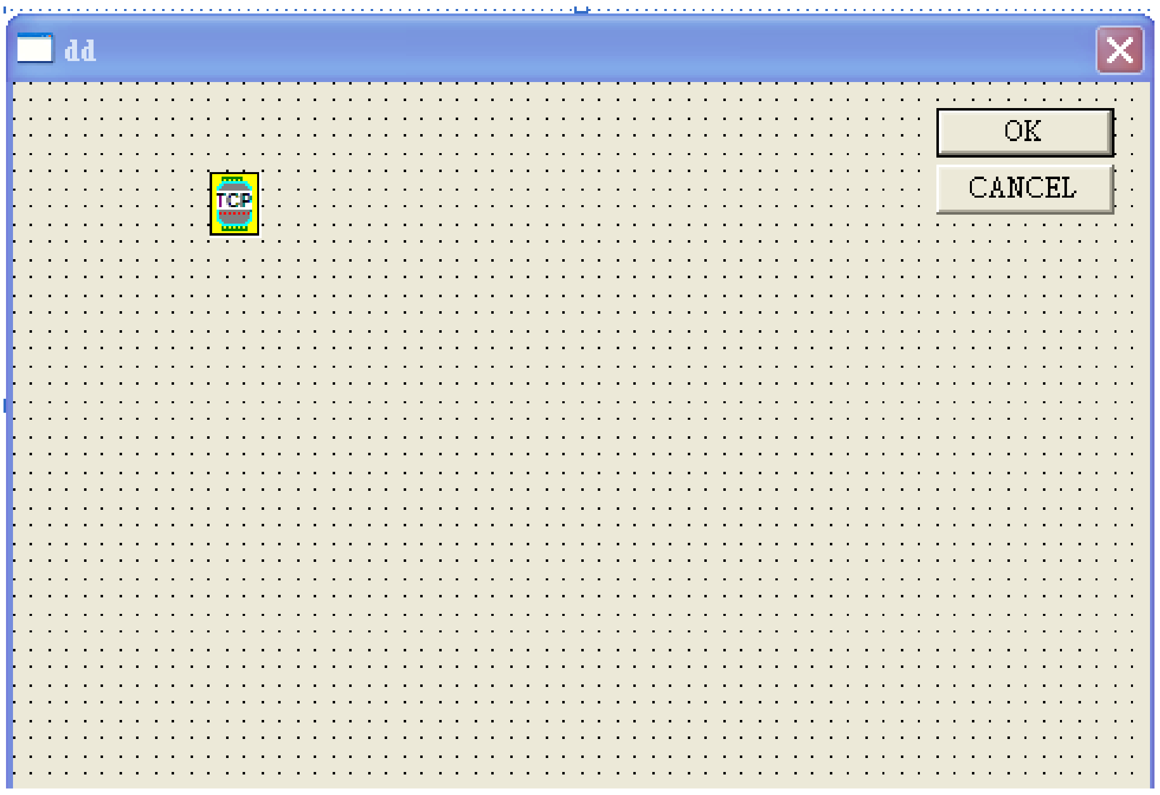
Class name: CTCPDAQ

Base class: CWnd

Header file: tcpdaq.h

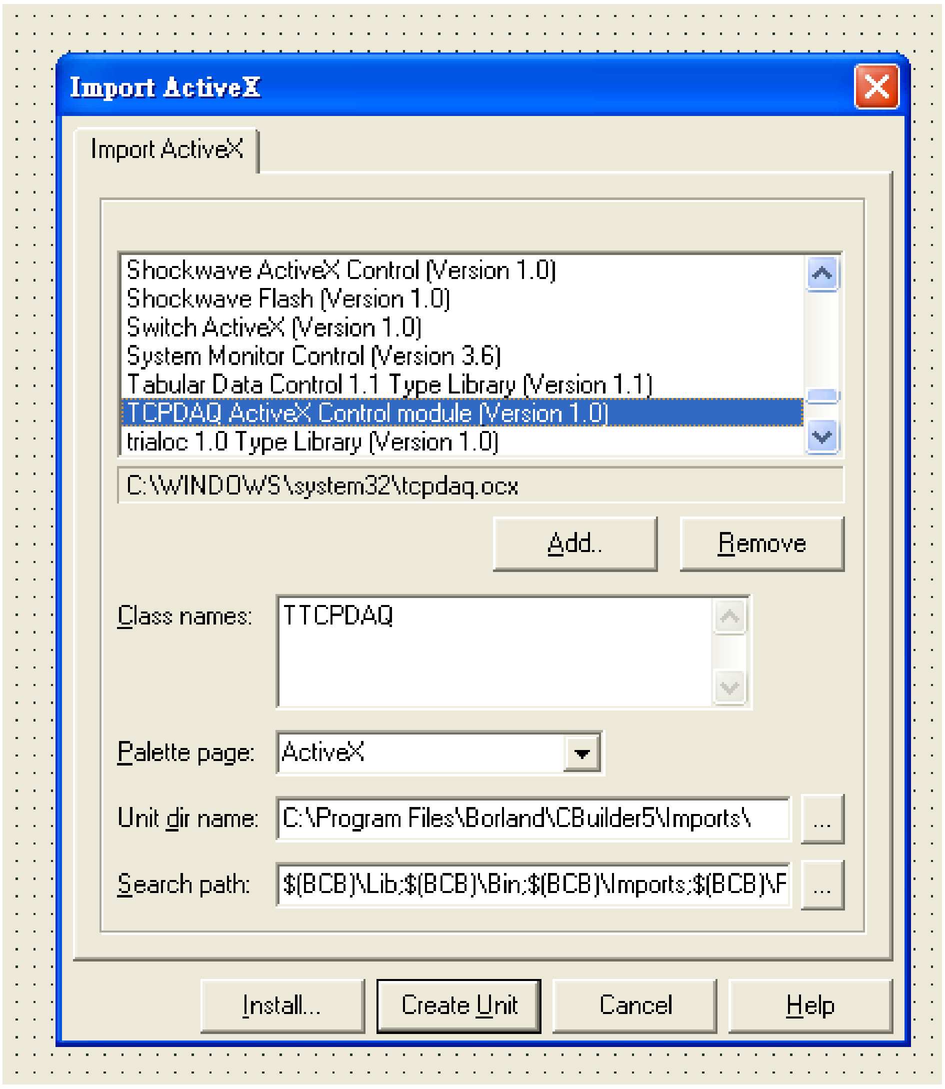
Implementation file: tcpdaq.cpp

- ♦ The TCPDAQ control will be showed in Visual C++ Toolbar.
- ♦ Place a TCPDAQ control from the Controls Toolbar on the dialog-based form.



6.6 Building TCPDAQ Applications with Borland C++ Builder

- ♦ Start Borland C++ Builder (BCB), BCB will launch as shown below:
- ♦ Select **Import ActiveX Control...** from the **Component** menu. The Import ActiveX dialog box loads:
- ♦ Select the TCPDAQ Control and press the **Install...** button. A dialog box is displayed as follows:



Shockwave ActiveX Control (Version 1.0)
Shockwave Flash (Version 1.0)
Switch ActiveX (Version 1.0)
System Monitor Control (Version 3.6)
Tabular Data Control 1.1 Type Library (Version 1.1)
TCPDAQ ActiveX Control module (Version 1.0)
trialoc 1.0 Type Library (Version 1.0)

C:\WINDOWS\system32\tcpdaq.ocx

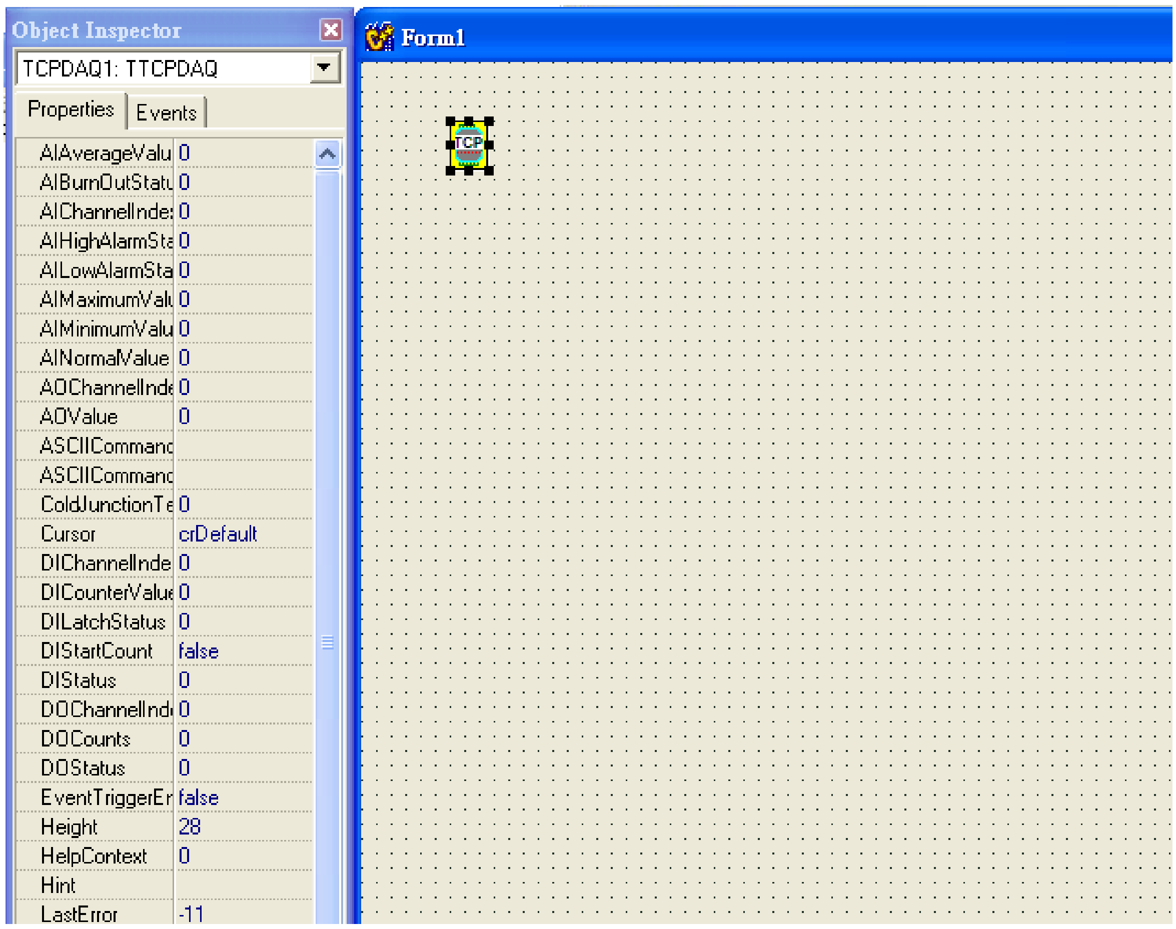
Class names: TTCPDAQ

Palette page: ActiveX

Unit dir name: C:\Program Files\Borland\CBuilder5\Imports\

Search path: $(BCB)\Lib;$(BCB)\Bin;$(BCB)\Imports;$(BCB)\F

- Enter "TCPDAQ" into the File name field under the **Into new package** tab, and press **OK** button. A Confirm dialog box is displayed. press **"Yes"** button.
- The TCPDAQ control is loaded into the **Component Palette**. You can check it by clicking on **Install Package...** from the **Component** menu. A dialog box is shown as below.



TCPDAQ1: TTCPDAQ

Properties | Events

AIAverageValu	0
AIBurnOutStatu	0
AIChannelInde	0
AIHighAlarmSta	0
AILowAlarmSta	0
AIMaximumValu	0
AIMinimumValu	0
AINormalValue	0
AOChannelInde	0
AOValue	0
ASCIICommand	
ASCIICommand	
ColdJunctionTe	0
Cursor	crDefault
DIChannelInde	0
DICounterValue	0
DILatchStatus	0
DIStartCount	false
DIStatus	0
DOChannelInd	0
DOCounts	0
DOStatus	0
EventTriggerEr	false
Height	28
HelpContext	0
Hint	
LastError	-11

6.7 Properties of TCPDAQ ActiveX Control

Name	Type	Description	Avaliable Model(s)
AOValue	double	Set the analog output voltage	All models
ASCIICommandReceive	string	Return the ASCII response message from module	All models
ASCIICommandSend	string	Send the ASII command message to module	All models
DIChannelIndex	short	Specifies the digital input channel to perform other DI properties read/write operation.	All DIO models
DIounterValue	long	Return the counting value for the specific DI channel which functions in "Count/Frequency mode"	All DIO models
DILatchStatus	short	Return the latch status for the specific DI channel which functions in "Lo-Hi/Hi-Lo latch mode" (1=Latched, 0=No latched)	All DIO models
DIStartCount	boolean	Start/stop counting for the specific DI channel which functions in "Count/Frequency mode" (True=Start, 0=Stop)	All DIO models
DIStatus	short	Return the status for the specific DI channel which functions in "DI mode" (1=Active, 0=Inactive)	All DIO models
DOChannelIndex	short	Specifies the digital output channel to perform other DO properties read/write operation.	All DIO models
DOCount	long	Set the output count value for the specific DO channel which functions in "Pulse output mode"	All DIO models
DOStatus	short	Return/set the status for the specific DO channel which functions in "D/O mode" (1=Active, 0=Inactive)	All DIO models
EventTriggerEnable	boolean	Enable/disable event trigger mode (True=Enable, False=Disable)	All models
LastError	short	Return the Error code of operation	All models
MoudleIDNo	short	Return the module ID number	All models
ModuleIP	string	Set the remote module IP address	All models
ModuelName	string	Return the module name	All models
TCPTimeOut	long	Return/set the TCP/IP Timeout (ms)	All models
UpdateTimeInterval	long	Return/set data update time interval(ms)	All models

6.8 Methods of TCPDAQ ActiveX Control

Name	Arguments	Returned type	Description
Open	None	None	Open TCPDAQ.OCX to start operation (Must be called before accessing properties at run time)
Close	None	None	Close TCPDAQ.OCX(Must be called before terminating the APP)
ModBusReadCoil	short Startaddress short Counts short coildata[]	None	Read coil data from remote module, and stored into coildata[] buffer

ModBusWriteCoil	shot StartAddress short Counts short coildata[]		Write coil data stored in coildata[] buffer to remote module
ModBusReadReg	short Startaddress short Counts short regdata[]	None	Read holding register data from remote module, and stored into regdata[] buffer
ModBusWriteReg	shot StartAddress short Counts short regdata[]		Write register data stored in regdata[] buffer to remote module

6.9 Events of TCPDAQ ActiveX Control

Name	Arguments	Returned type	Description
OnError	short ErrCode(out) string Errmsg(out)	None	be called when error occurred
EventDataArrival	string Datetime(out) short EventChannel(out) short EventType(out) short EventStatus(out) short EventValue(out)	None	be called when received an event data from the remote module **(*)**

(*): Please see *TCPDAQ_Data_Structure.pdf* file to understand the means of parameters

6.10 Building TCPDAQ ActiveX Applications with Various Development Tools

The demo programs of TCPDAQ AvtiveX control module are included in the provided DISC. The Installed folders include the demo programs for various development tools.

Chapter 7 TCPDAQ DLL API

7.1 Common Functions

NO.	Function Name	Description	Sec.
1	TCP_Open	To initiate the TCPDAQ.dll to use.	
2	TCP_Close	To terminates use of the TCPDAQ.dll.	
3	TCP_Connect	To create a Window TCP socket then establishing a connection to a specific EDAM-4200(A)	
4	TCP_Disconnect	Disconnecting the Window TCP socket from all EDAM-4200 modules	
5	TCP_ModuleDisconnect	Disconnecting the Window TCP socket from a specific EDAM-4200	
6	TCP_SendData	Send data to a specific EDAM-4200(A) module	
7	TCP_RecvData	Receive data to a specific EDAM-4200(A) module	
8	TCP_SendReceiveASCcmd	To accept an ASCII format string as a command, and transform it to meet the Modbus/TCP's specification. Then sending it to EDAM-4200(A) and receiving the response from EDAM-4200(A)	
9	UDP_Connect	To create a Window UDP socket then establishing a connection to a specific EDAM-4200(A)	
10	UDP_Disconnect	Disconnecting the Window UDP socket from all EDAM-4200(A) modules	
11	UDP_ModuleDisconnect	Disconnecting the Window UDP socket from a specific EDAM-4200(A)	
12	UDP_SendData	Send data to a specific EDAM-4200(A) module	
13	UDP_RecvData	Receive data to a specific EDAM-4200(A) module	
14	UDP_SendReceiveASCcmd	Direct send an ASCII format string as a command, and receive the response from EDAM-4200(A)	
15	TCP_GetModuleIPinfo	Return module IP information of a specific module	
16	TCP_GetModuleID	Return module ID number of a specific module	
17	TCP_GetIPFromID	Return IP address of a specific module ID number	
18	TCP_ScanOnLineModules	Scan all on-line EDAM-4200(A) modules	
19	TCP_GetDLLVersion	Return the DLL's version, that is the version of TCPDAQ.DLL	
20	TCP_GetModuleNo	Return the module name of a specific IP address	
21	TCP_GetLastError	Return the error code of the latest called function	
22	TCP_PingIP	Ping to Remote IP address	

7.2 Stream/Event Functions

TCP_StartStream	To instruct the PC to start to receive stream data that coming from EDAM-4200	
TCP_StopStream	To instruct the PC to stop receiving stream data from all modules	
TCP_ReadStreamData	To receive stream data that coming from the specific EDAM-4200(A)	
TCP_StartEvent	To instruct the PC to start to receive alarm event data that coming from EDAM-4200	
TCP_StopEvent	To instruct the PC to stop receiving alarm event data from all modules	
TCP_ReadEventData	To receive alarm event data that coming from the specific EDAM-4200(A)	

7.3 Digital I/O Functions

TCP_ReadDIOMode	To read the type for every D/I & D/O channels of an EDAM-4200(A) module	
TCP_ReadDIO	To read DI/DO's status for an EDAM-4200(A) module	
TCP_ReadDISignalWidth	To read the minimal high/low signal width of each D/I channel for an EDAM-4200(A) module	
TCP_WriteDISignalWidth	To set the minimal high/low signal width of each D/I channel for an EDAM-4200(A) module	
TCP_ReadDICounter	To read the counter value when a D/I channel function in 'Counter' mode	
TCP_ClearDICounter	To clear the counter value when a D/I channel function in 'Counter' mode	
TCP_StartDICounter	To start the counting when a D/I channel function in 'Counter' mode	
TCP_StopDICounter	To stop the counting when a D/I channel function in 'Counter' mode	
TCP_ClearDILatch	To clear the latch when a D/I channel function as 'Lo to Hi Latch' or 'Hi to Lo Latch'	
TCP_ReadDILatch	To read the counter value when a D/I channel function in 'Counter' mode	
TCP_WriteDO	To write some value to D/O channels for an EDAM-4200(A) module	
TCP_WriteDOPulseCount	To write the pulse output count for EDAM-4200(A) DIO modules during runtime	
TCP_WriteDODelayWidth	To set the pulse and delay signal widths to the specific EDAM-4200(A) DIO modules	
TCP_ReadDODelayWidth	To read the pulse and delay signal width from the specific EDAM-4200(A) DIO modules	

7.4 MODBUS/TCP Functions

TCP_MODBUS_ReadCoil	To read the coil values at a specific range described in parameters	
TCP_MODBUS_WriteCoil	To write the coil values at a specific range described in parameters.	
TCP_MODBUS_ReadReg	To read the holding register value at a specific range described in parameters	
TCP_MODBUS_WriteReg	To write values to the holding registers at a specific range described in parameters	

7.5 Function Description

The TCPDAQ.DLL function declarations are all included in following files that are attached with the provided DISC.

♦ TCPDAQ.h	:	Include file for both VC++ and Borland C++ Builder
♦ TCPDAQ.lib	:	Library file for VC++
♦ TCPDAQ_BC.lib	:	Library file for Borland C++ Builder
♦ TCPDAQ.bas	:	Module file for Visual Basic
♦ TCPDAQ.pas	:	Module file for Delphi

You need to add the above file into your AP project before using TCPDAQ.DLL functions

7.5.1 TCP_Open

Description: To initiate the TCPDAQ.dll to use.

Syntax:

- **Visual Basic:** (*see TCPDAQ.bas*)
  Declare Sub TCP_Open Lib "TCPDAQ.dll" Alias "_TCP_Open@0" ()
- **Borland C++ Builder:** (see TCPDAQ.h)
  int TCP_Open();
- **Delphi:** (*see TCPDAQ.pas*)
  function TCP_Open(); StdCal;
- **VC++:** (*see TCPDAQ.h*)
  int TCP_Open();

  Parameters: void

  Return Code: refer to the *Error code*.

7.5.2 TCP_Close

Description: To terminates use of the TCPDAQ.dll.

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*
  Declare Sub TCP_Close Lib "TCPDAQ.dll" Alias "_TCP_Close@0" ()
- **Borland C++ Builder:** (*see TCPDAQ.h*)
  int TCP_ Close();
- **Delphi:** *(see TCPDAQ.pas)*
  function TCP_ Close(); StdCall;
- **VC++:** *(see TCPDAQ.h)*
  int TCP_ Close();

  Parameters: void

  Return Code: refer to the *Error code*.

7.5.3 TCP_Connect

Description: to create a Window TCP socket then establishing a connection to a specific EDAM-4200

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*
  Declare Function TCP_Connect Lib "TCPDAQ.dll" Alias "_TCP_Connect@20" (ByVal szIP As String, ByVal port As Integer, ByVal ConnectionTimeout As Long, ByVal SendTimeout As Long, ByVal ReceiveTimeout As Long) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)
  int TCP_Connect(char szIP[],u_short port,int ConnectionTimeout, int SendTimeout,int ReceiveTimeout);
- **Delphi:** *(see TCPDAQ.pas)*

FunctionTCP_Connect (szIP: PChar; port: Integer; ConnectionTimeout: Longint; SendTimeout: Longint;ReceiveTimeout: Longint): Longint; StdCall;

♦ **VC++:** *(see TCPDAQ.h)*

int TCP_Connect(char szIP[],u_short port,int ConnectionTimeout, int SendTimeout, int ReceiveTimeout);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
port[in]	: the TCP/IP port used by Modbus/TCP, it is 502
ConnectionTimeout[in]	: Connection timeout value (msec)
SendTimeout[in]	: Send timeout value (msec)
ReceiveTimeout[in]	: Receive timeout value (msec)

Return Code: refer to the *Error code*.

7.5.4 TCP_Disconnect

Description: disconnecting the Window TCP socket from all EDAM-4200 modules

Syntax:

♦ **Visual Basic:** *(see TCPDAQ.bas)*

Declare Sub TCP_Disconnect Lib "TCPDAQ.dll" Alias "_TCP_Disconnect@0" ()

♦ **Borland C++ Builder:** (see TCPDAQ.h)

void TCP_Disconnect(void);

♦ **Delphi:** ***(see TCPDAQ.pas)***

procedure TCP_Disconnect ; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

void TCP_Disconnect(void);

Parameters: void

Return Code: none.

7.5.5 TCP_ModuleDisconnect

Description: disconnecting the Window TCP socket to a specific EDAM-4200

Syntax:

♦ **Visual Basic:** *(see TCPDAQ.bas)*

Declare Function TCP_ModuleDisconnect Lib "TCPDAQ.dll" Alias "_TCP_ModuleDisconnect@4" (ByVal szIP As String) As Long

♦ **Borland C++ Builder:** (see TCPDAQ.h)

Int TCP_ModuleDisconnect(char szIP[]);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_ModuleDisconnect (szIP: PChar): Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

Int TCP_ModuleDisconnect(char szIP[]);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

Return Code: refer to the *Error code*.

7.5.6 TCP_SendData

Description: to send data to a specific EDAM-4200 module

Syntax:

♦ **Visual Basic:** *(see TCPDAQ.bas)*

Declare Function TCP_SendData Lib "TCPDAQ.dll" Alias "_TCP_SendData@12" (ByVal szIP As String, ByRef pData As Byte, ByVal wDataLen As Integer) As Long

- **Borland C++ Builder:** (see TCPDAQ.h)

  Int TCP_SendData(char szIP[],char *pData,u_short wDataLen);
- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_SendData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_SendData(char szIP[],char *pData,u_short wDataLen);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
pData[in]	: 8 bit data array
wDataLen[in]	: length of data be sent

Return Code: refer to the *Error code*.

7.5.7 TCP_RecvData

Description: receive data to a specific EDAM-4200 module

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function TCP_RecvData Lib "TCPDAQ.dll" Alias "_TCP_RecvData@12" (ByVal szIP As String, ByRef pData As Byte, ByVal wDataLen As Integer) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  Int TCP_RecvData(char szIP[],char *pData,u_short wDataLen);
- **Delphi:** *(see TCPDAQ.pas)*

  Function TCP_RecvData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  Int TCP_RecvData(char szIP[],char *pData,u_short wDataLen);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
pData[out]	: 8 bit data array
wDataLen [in]	: length of data array

Return Code:

If return value >=0, it represents the length of received data

If return value<0, it represents *Error code*.

7.5.8 TCP_SendReceiveASCcmd

Description: to accept an ASCII format string as a command, and transform it to meet the Modbus/TCP's specification. Then sending it to EDAM-4200 and receiving the response from EDAM-4200

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function TCP_SendReceiveASCcmd Lib "TCPDAQ.dll" Alias "_TCP_SendReceiveASCcmd@12" (ByVal szIP As String, ByVal Sendbuf As String, ByVal Recvbuf As String) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  Int TCP_SendReceiveASCcmd(Char szIP[], char Sendbuf [], char Recvbuf []);
- **Delphi:** *(see TCPDAQ.pas)*

  Function TCP_SendReceiveasCcmd (szIP: PChar; Sendbuf: PChar; Recvbuf: PChar): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  Int TCP_SendReceiveASCcmd(Char szIP[], char Sendbuf[], char Recvbuf[]);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
Sendbuf [in]	: 8 bit data array to be sent
Recvbuf [out]	: 8 bit data array that stored the received data

Return Code: refer to the *Error code*.

7.5.9 UDP_Connect

Description: to create a Window UDP socket then establishing a connection to a specific EDAM-4200

Syntax:

- **Visual Basic:** (*see TCPDAQ.bas*)

  Declare Function UDP_Connect Lib "TCPDAQ.dll" Alias "_UDP_Connect@24" (ByVal szIP As String, ByVal s_port As Integer, ByVal d_port As Integer, ByVal ConnectionTimeout As Long, ByVal SendTimeout As Long, ByVal ReceiveTimeout As Long) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  Int UDP_Connect(char szIP[],u_short s_port,u_short d_port, int ConnectionTimeout, int SendTimeout, int ReceiveTimeout);
- **Delphi:** *(see TCPDAQ.pas)*

  Function UDP_Connect (szIP: PChar; s_port: word; d_port: word; ConnectionTimeout: Longint; SendTimeout: Longint; ReceiveTimeout: Longint): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  Int UDP_Connect(char szIP[],u_short s_port,u_short d_port,int ConnectionTimeout, int SendTimeout,int ReceiveTimeout);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
s_port	: source port number
d_port	: destination port number
ConnectionTimeout	: timeout value for connection (msec)
SendTimeout	: timeout value for sending (msec)
ReceiveTimeout	: timeout value for receiving (msec)

Return Code: refer to the *Error code*.

7.5.10 UDP_Disconnect

Description: disconnecting the Window UDP socket from all EDAM-4200 modules

Syntax:

- **Visual Basic**: (*see TCPDAQ.bas*)

  Declare Sub UDP_Disconnect Lib "TCPDAQ.dll" Alias "_UDP_Disconnect@0" ()
- **Borland C++ Builder:** (see TCPDAQ.h)

  void UDP_Disconnect(void);
- **Delphi:** *(see TCPDAQ.pas)*

  procedure UDP_Disconnect ; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  void UDP_Disconnect(void);

Parameters: void

Return Code: None

7.5.11 UDP_ModuleDisconnect

Description: disconnecting the Window UDP socket from a specific EDAM-4200

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function UDP_ModuleDisconnect Lib "TCPDAQ.dll" Alias "_UDP_ModuleDisconnect@4" (ByVal szIP As String) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  int UDP_ModuleDisconnect(Char szIP[]);

- **Delphi:** *(see TCPDAQ.pas)*
  Function UDP_ModuleDisconnect (szIP: PChar): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*
  int UDP_ModuleDisconnect(char szIP[]);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be disconnected

Return Code: refer to the *Error code*.

7.5.12 UDP_SendData

Description: send data to a specific EDAM-4200 module (Datagram)

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*
  Declare Function UDP_SendData Lib "TCPDAQ.dll" Alias "_UDP_SendData@12"
  (ByVal szIP As String, ByRef pData As Byte, ByVal wDataLen As Integer) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)
  int UDP_SendData(char szIP[],char *pData,u_short wDataLen);
- **Delphi:** *(see TCPDAQ.pas)*
  Function UDP_SendData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*
  int UDP_SendData(char szIP[],char *pData,u_short wDataLen);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
pData[in]	: points to data buffer
wDataLen[in]	: length of data be sent

Return Code: refer to the *Error code*.

7.5.13 UDP_RecvData

Description: receive data to a specific EDAM-4200 module (Datagram)

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*
  Declare Function UDP_RecvData Lib "TCPDAQ.dll" Alias "_UDP_RecvData@12"
  (ByVal szIP As String, ByRef pData As Byte, ByVal wDataLen As Integer) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)
  int UDP_RecvData(char szIP[],char *pData,u_short wDataLen);
- **Delphi:** *(see TCPDAQ.pas)*
  Function UDP_RecvData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*
  int UDP_RecvData(char szIP[],char *pData,u_short wDataLen);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
pData[out]	: 8 bit array that stored the received data
wDataLen [in]	: length of received data

Return Code: refer to the *Error code*.

7.5.14 UDP_SendReceiveASCcmd

Description: send an ASCII format string as a command to EDAM-4200 and receiving the response from EDAM-4200.

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function UDP_SendReceiveASCcmd Lib "TCPDAQ.dll" Alias "_UDP_SendReceiveASCcmd@12" (ByVal szIP As String, ByVal Txdata As _ String, ByVal Rxdata As String) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  int UDP_SendReceiveASCcmd(char szIP[],char Txdata [],char Rxdata []);
- **Delphi:** *(see TCPDAQ.pas)*

  Function UDP_SendReceiveAsCcmd (szIP: PChar; Txdata:PChar; Rxdata: PChar): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  int UDP_SendReceiveASCcmd(char szIP[],char Txdata [],char Rxdata []);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
Txdata [in]	: 8 bit array that stored the data to be sent
Rxdata [out]	: 8 bit array that stored the received data

Return Code: refer to the *Error code*.

7.5.15 TCP_GetModuleIPinfo

Description: return module IP information of a specific module

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function TCP_GetModuleIPinfo Lib "TCPDAQ.dll" Alias "_TCP_GetModuleIPinfo@8" (ByVal szIP As String, ByRef ModuleIP As ModuleInfo) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  Int TCP_GetModuleIPinfo(char szIP[],struct ModuleInfo *ModuleIP);
- **Delphi:** *(see TCPDAQ.pas)*

  Function TCP_GetModuleIPinfo (szIP: PChar; var ModuleIP: TModuleInfo): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  Int TCP_GetModuleIPinfo(char szIP[],struct ModuleInfo *ModuleIP);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
ModuleIP[out]	: a structure array that stroes the module IP information

Return Code: refer to the *Error code*.

7.5.16 TCP_GetModuleID

Description: return ID number of a specific module.

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function TCP_GetModuleID Lib "TCPDAQ.dll" Alias "_TCP_GetModuleID@8" (ByVal szIP As String, ByRef ModuleID As Byte) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  Int TCP_GetModuleID(char szIP[], char * ModuleID);
- **Delphi:** *(see TCPDAQ.pas)*

  Function TCP_GetModuleID(szIP: PChar; ModuleID: PByte): Longint; StdCall;;
- **VC++:** *(see TCPDAQ.h)*

  Int TCP_GetModuleID(char szIP[], char * ModuleID);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
ModuleID [in]	: the ID number

Return Code: refer to the *Error code*.

7.5.17 TCP_GetIPFromID

Description: get IP address for a specific module ID number. This function is helpful when the module is DHCP enabled

Syntax:

- **Visual Basic:** *(see TCPDAQ.bas)*

  Declare Function TCP_GetIPFromID Lib "TCPDAQ.dll" Alias "_TCP_GetIPFromID@8" (ByVal szID As Byte, ByRef szIP As String) As Long
- **Borland C++ Builder:** (see TCPDAQ.h)

  Int TCP_GetIPFromID(u_char szID ,char szIP[]);
- **Delphi:** *(see TCPDAQ.pas)*

  Function TCP_GetIPFromID(szID: Byte; szIP: PChar): Longint; StdCall;
- **VC++:** *(see TCPDAQ.h)*

  Int TCP_GetIPFromID(u_char szID ,char szIP[]);

Parameters:

szID[in]	: module ID number (0~255)
szIP[out]	: 8 bit array that stored the IP address string(such as "192.168.0.2")

Return Code: refer to the *Error code*.

7.5.18 TCP_ScanOnLineModules

Description: search on-line EDAM900 modules in the same subnet

Syntax:

- **Visual Basic: *(see TCPDAQ.bas)***

  Declare Function TCP_ScanOnLineModules Lib "TCPDAQ.dll" Alias "_TCP_ScanOnLineModules@8" (ModuleIP As ModuleInfo, ByVal Sortkey As Byte) As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_ScanOnLineModules(struct ModuleInfo ModuleIP[], u_char SortKey);
- **Delphi: *(see TCPDAQ.pas)***

  Function Scan_OnLineModules (var ModuleIP: TModuleInfo; Sortkey: Byte): Longint; StdCall;
- **VC++: *(see TCPDAQ.h)***

  Int TCP_ScanOnLineModules(struct ModuleInfo ModuleIP[], u_char SortKey);

Parameters:

ModuleIP[out]	: points to ModuleInfo structure array
SortKey[in]	: sortkey word (by IP address,by ID number, or by Module no) =SORT_MODULE_IP ,sort by IP address =SORT_MODULE_ID ,sort by ID number =SORT_MODULE_NO ,sort by module number

Return Code: refer to the *Error code*.

7.5.19 TCP_GetDLLVersion

Description: return the version number of TCPDAQ.dll

Syntax:

- **Visual Basic: *(see TCPDAQ.bas)***

  Declare Function TCP_GetDLLVersion Lib "TCPDAQ.dll" Alias "_TCP_GetDLLVersion@0" () As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_GetDLLVersion(void);
- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_GetDLLVersion: Longint; StdCall;
- **VC++: *(see TCPDAQ.h)***

  Int TCP_GetDLLVersion(void);

Parameters: void

Return Code: the version number.

7.5.20 TCP_GetModuleNo

Description: return the module name of a specific IP address

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_GetModuleNo Lib "TCPDAQ.dll" Alias "_TCP_GetModuleNo@8" _
  (ByVal szIP As String, ByRef Mname As Byte) As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_GetModuleNo(char szIP[], char Mname[]);
- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_GetModuleNo (szIP: PChar; Mname: PByte): Longint; StdCall;
- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_GetModuleNo(char szIP[], char Mname[]);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
Mname[out]	: 8 bit array that stored the module name string

Return Code: refer to the *Error code*.

7.5.21 TCP_GetLastError

Description: return the error code of the latest called function

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_GetLastError Lib "TCPDAQ.dll" Alias "_TCP_GetLastError@0" () As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_GetLastError(void);
- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_GetLastError: Longint ; StdCall;
- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_GetLastError(void);

Parameters: void

Return Code: refer to the *Error code*

7.5.22 TCP_PingIP

Description: ping to remote IP address

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_PingIP Lib "TCPDAQ.dll" Alias "_TCP_PingIP@8" (ByVal IPadr As String, ByVal PingTimes As
  Integer) As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_PingIP(char szIP[],int PingTimes);
- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_PingIP(szIP: PChar;PingTimes: Integer): Longint; StdCall;
- **VC++:** ***(see TCPDAQ.h)***

  int TCP_PingIP(char szIP[],int PingTimes);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

PingTimes [in] :Timeout value

Return Code: = -1, no response from remote IP

>0, response time from remote IP

7.5.23 TCP_StartStream

Description: to instruct the PC to start to receive stream data that coming from EDAM-4200

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_StartStream Lib "TCPDAQ.dll" Alias "_TCP_StartStream@8" (ByVal IP As String, ByVal EventFromApp As Long) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_StartStream(char szIP[],HANDLE EventFromApp);

- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_StartStream (szIP: PChar; EventFromApp: Longint): Longint; StdCall;

- **VC++:** ***(see TCPDAQ.h)***

  int TCP_StartStream(char szIP[],HANDLE EventFromApp);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

EventFromApp : event handle (be signaled, when stream data arrived)

Return Code: refer to the *Error code*.

7.5.24 TCP_StopStream

Description: to instruct the PC to stop receiving stream data from all modules.

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_StopStream Lib "TCPDAQ.dll" Alias "_TCP_StopStream@0" () As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_StopStream(void);

- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_StopStream: Longint; StdCall;

- **VC++:** ***(see TCPDAQ.h)***

  int TCP_StopStream(void);

Parameters: void

Return Code: refer to the *Error code.*

7.5.25 TCP_ReadStreamData

Description: to read stream data that coming from the specific EDAM-4200

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_ReadStreamData Lib "TCPDAQ.dll" Alias "_TCP_ReadStreamData@8" (ByVal szIP As String, ByRef lpData As StreamData) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_ReadStreamData (char szIP[], struct _StreamData *lpData);

- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_ReadStreamData (szIP: PChar; Var lpData: TStreamData): integer; StdCall;

- **VC++:** ***(see TCPDAQ.h)***

  int TCP_ReadStreamData (char szIP[], struct _StreamData *lpData);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

lpData[out] : points to stream data structure that stored the stream data

Return Code: refer to the *Error code.*

7.5.26 TCP_StartEvent

Description: to start listening the alarm event trigger

Syntax:

♦ **Visual Basic:** ***(see TCPDAQ.bas)***

Declare Function TCP_StartEvent Lib "TCPDAQ.dll" Alias "_TCP_StartEvent@8" (ByVal IPadr As String, ByVal EventFromApp As Long) As Long

♦ **Borland C++ Builder: (see TCPDAQ.h)**

int TCP_StartEvent(char szIP[],HANDLE EventFromApp);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_StartEvent(szIP: PChar; EventFromApp: Longint): Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

int TCP_StartEvent(char szIP[],HANDLE EventFromApp);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

EventFromApp : event handle (be signaled, when alarm event occured)

Return Code: refer to the *Error code.*

7.5.27 TCP_StopEvent

Description: to stop listening the alarm event trigger from all module

Syntax:

♦ **Visual Basic:** ***(see TCPDAQ.bas)***

Declare Function TCP_StopEvent Lib "TCPDAQ.dll" Alias "_TCP_StopEvent@0" () As Long

♦ **Borland C++ Builder: (see TCPDAQ.h)**

Int TCP_StopEvent(void);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_StopEvent: Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

Int TCP_StopEvent(void);

Parameters: void

Return Code: refer to the *Error code.*

7.5.28 TCP_ReadEventData

Description: to read triggered alarm event message

Syntax:

♦ **Visual Basic: (*see TCPDAQ.bas*)**

Declare Function TCP_ReadEventData Lib "TCPDAQ.dll" Alias "_TCP_ReadEventData@8" (ByVal szIP As String, ByRef lpData As AlarmData) As Long

♦ **Borland C++ Builder: (see TCPDAQ.h)**

int TCP_ReadEventData (char szIP[], struct _AlarmInfo *lpData);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_ReadEventData (SzIP: PChar; Var lpData: TEventInfo): integer; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

int TCP_ReadEventData (char szIP[], struct _AlarmInfo *lpData);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

lpData[out] : points to alarm event data structure that stored event message (ref. to TCPDAQ.H)

Return Code: refer to the *Error code*.

7.5.29 TCP_ReadDIOMode

Description: to read the mode of D/I & D/O channels of an EDAM-4200 module.

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_ReadDIOMode Lib "TCPDAQ.dll" Alias "_TCP_ReadDIOMode@12" _
  (ByVal szIP As String, ByRef DImode As Byte, ByRef DOmode As Byte) As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_ReadDIOMode(char szIP[],u_char DImode[],u_char DOmode[]);
- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_ReadDIOMode (szIP: PChar; DImode: PByte; DOmode: PByte): Longint; StdCall;
- **VC++:** ***(see TCPDAQ.h)***

  int TCP_ReadDIOMode(char szIP[],u_char DImode[],u_char DOmode[]);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

DImode[out] : an 8 bit array that stored the DI channel mode

DOmode[out] : an 8 bit array that stored the DO channel mode

Return Code: refer to the *Error code*.

7.5.30 TCP_ReadDIO

Description: to read DI/DO's status for an EDAM-4200 module

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_ReadDIO Lib "TCPDAQ.dll" Alias "_TCP_ReadDIO@12" _
  (ByVal szIP As String, ByRef ByDi As Byte, ByRef ByDo As Byte) As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_ReadDIO(char szIP[],u_char byDI[],u_char byDO[]);
- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_ReadDIO (szIP: PChar; ByDi: PByte; ByDo: PByte): Longint; StdCall;
- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_ReadDIO(char szIP[],u_char u_byDI[],u_char byDO[]);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

byDI[out] : an 8 bit array that stored the DI channel status (ex: byDI[0]= 0 → DI channel 0 = 0)

byDO[out] : an 8 bit array that stored the DO channel status (ex: byDO[3] = 1 → channel 3 = 1)

Return Code: refer to the *Error code*.

7.5.31 TCP_ReadDISignalWidth

Description: to read the minimal high/low signal width of all D/I channels

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_ReadDISignalWidth Lib "TCPDAQ.dll" Alias "_TCP_ReadDISignalWidth@12" (ByVal szIP As String, ByRef ulLoWidth As Long, ByRef ulHiWidth As Long) As Long
- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_ReadDISignalWidth(char szIP[],u_long ulLoWidth[],u_long ulHiWidth[]);

- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_ReadDISignalWidth (szIP: PChar; var ulLoWidth:array of Longword; var ulHiWidth:array of Longword): Longint; StdCall;

- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_ReadDISignalWidth(char szIP[],u_long ulLoWidth[],u_long ulHiWidth[]);

  Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
ulLoWidth[out]	: an 32 bit array that stored channel low width value
ulHiWidth[out]	: an 32 bit array that stored channel high width value

  Return Code: refer to the *Error code*.

7.5.32 TCP_WriteDISignalWidth

Description: to set the minimal high/low signal width of all D/I channels

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_WriteDISignalWidth Lib "TCPDAQ.dll" Alias "_TCP_WriteDISignalWidth@12" (ByVal szIP As String, ByRef ulLoWidth As Long, ByRef ulHiWidth As Long) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_WriteDISignalWidth(char szIP[],u_long ulLoWidth[],u_long ulHiWidth[]);

- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_WriteDISignalWidth(szIP: PChar; var ulLoWidth:array of Longword; var ulHiWidth:array of Longword): Longint; StdCall;

- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_WriteDISignalWidth(char szIP[],u_long ulLoWidth[],u_long ulHiWidth[]);

  Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
ulLoWidth[in]	: an unsigned 32 bits array that stored the minimal low signal width for each D/I channel. The unit is 0.5 mSec
ulHiWidth[in]	: an unsigned 32 bits array that stored the minimal high signal width for each D/I channel. The unit is 0.5 mSec

Return Code: refer to the *Error code*.

7.5.33 TCP_ReadDICounter

Description: to read the counter value of all D/I channels (the counter value is available only for channel that functions in 'Counter' mode

Syntax:

- **Visual Basic:** ***(see TCPDAQ.bas)***

  Declare Function TCP_ReadDICounter Lib "TCPDAQ.dll" Alias "_TCP_ReadDICounter@8" (ByVal szIP As String, ByRef ulCounterValue As Long) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  Int TCP_ReadDICounter(Char szIP[],u_long ulCounterValue[]);

- **Delphi:** ***(see TCPDAQ.pas)***

  Function TCP_ReadDICounter (szIP: PChar; var ulCounterValue:array of Longword): Longint; StdCall;

- **VC++:** ***(see TCPDAQ.h)***

  Int TCP_ReadDICounter(Char szIP[],u_long ulCounterValue[]);

  Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
ulCounterValue[out]	:an unsigned 32 bits array that stored the counter value for each D/I channel

Return Code: refer to the *Error code*.

7.5.34 TCP_ClearDICounter

Description: to clear the counter value when a D/I channel function in 'Counter' mode

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_ClearDICounter Lib "TCPDAQ.dll" Alias "_TCP_ClearDICounter@8"
  (ByVal szIP As String, ByVal wChno As Integer) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_ClearDICounter(char szIP[],u_short wChNo);

- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_ClearDICounter (szIP: PChar; wChno: Integer): Longint; StdCall;

- **VC++: *(see TCPDAQ.h)***

  int TCP_ClearDICounter(char szIP[],u_short wChNo);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wChNo[in]	: the D/I channel to be cleared.

Return Code: refer to the *Error code.*

7.5.35 TCP_StartDICounter

Description: to start the counting when a D/I channel function as 'Counter' mode

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_StartDICounter Lib "TCPDAQ.dll" Alias "_TCP_StartDICounter@8" (ByVal szIP As String,
  ByVal wChno As Integer) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_StartDICounter(Char szIP[],u_short wChNo);

- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_StartDICounter (szIP: PChar; wChno: Integer): Longint; StdCall;

- **VC++: *(see TCPDAQ.h)***

  int TCP_StartDICounter(Char szIP[],u_short wChNo);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wChNo[in]	: the channel number that is enabled to count

Return Code: refer to the *Error code.*

7.5.36 TCP_StopDICounter

Description: to stop the counting when a D/I channel function as 'Counter' mode

Syntax:

- **Visual Basic: *(see TCPDAQ.bas)***

  Declare Function TCP_StopDICounter Lib "TCPDAQ.dll" Alias "_TCP_StopDICounter@8"
  (ByVal szIP As String, ByVal wChno As Integer) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_StopDICounter(char szIP[],u_short wChNo);

- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_StopDICounter (szIP: PChar; wChno: Integer): Longint; StdCall;

- **VC++: *(see TCPDAQ.h)***

  int TCP_StopDICounter(char szIP[],u_short wChNo);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

wChNo[in] : the channel number that is disabled to count

Return Code: refer to the *Error code.*

7.5.37 TCP_ClearDILatch

Description: to clear the latch when a D/I channel function as 'Lo to Hi Latch' or 'Hi to Lo Latch'

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_ClearDILatch Lib "TCPDAQ.dll" Alias "_TCP_ClearDILatch@8" (ByVal szIP As String, ByVal wChno As Integer) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_ClearDILatch(char szIP[],u_short wChNo);

- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_ClearDILatch(szIP: PChar; wChno: Integer): Longint; StdCall;

- **VC++: *(see TCPDAQ.h)***

  int TCP_ClearDILatch(char szIP[],u_short wChNo);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

wChNo[in] : the channel number that latch status is cleared

Return Code: refer to the *Error code.*

7.5.38 TCP_ReadDILatch

Description: to read the DI latch status when a D/I channel function in 'Lo to Hi Latch' or 'Hi to Lo Latch'

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_ReadDILatch Lib "TCPDAQ.dll" Alias "_TCP_ReadDILatch@8" (ByVal szIP As String, ByRef wLatch As Byte) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_ReadDILatch(char szIP[],u_char wLatch[]);

- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_ReadDILatch (szIP: PChar; wLatch: PByte): Longint; StdCall;

- **VC++: *(see TCPDAQ.h)***

  int TCP_ReadDILatch(char szIP[],u_char wLatch[]);

Parameters:

szIP[in] : the IP address for an EDAM-4200 that to be connected

wLatch[out] : an unsigned 8 bits array that stored the latch stsatus for each D/I channel

Return Code: refer to the *Error code.*

7.5.39 TCP_WriteDO

Description: to write some value to D/O channels for an EDAM-4200 module

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_WriteDO Lib "TCPDAQ.dll" Alias "_TCP_WriteDO@16" _
  ByVal szIP As String, ByVal wStartDO As Integer, ByVal wCount As Integer,
  ByRef ByDo As Byte) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  int TCP_WriteDO(Char szIP[], u_short wStartDO, u_short wCount,u_char byDO[]);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_WriteDO(szIP: PChar; wStartDO: Integer; wCount: Integer;ByDo: PByte): Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

int   TCP_WriteDO(Char szIP[], u_short wStartDO, u_short wCount,u_char byDO[]);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wStartDO[in]	: the starting channel that to be written.
wCount[in]	: how many channels to be written.
byDO[in]	: an 8 bit array that stored the values that written to the connected EDAM-4200

Return Code: refer to the *Error code*.

7.5.40 TCP_WriteDOPulseCount

Description: to write the pulse output count for EDAM-4200 DIO modules during runtime

Syntax:

♦ **Visual Basic: (*see TCPDAQ.bas*)**

Declare Function TCP_WriteDOPulseCount Lib "TCPDAQ.dll" Alias _ "_TCP_WriteDOPulseCount@12" (ByVal szIP As String, ByVal wDoChannel As Integer, ByVal ulPulseCount As Long) As Long

♦ **Borland C++ Builder: (see TCPDAQ.h)**

int   TCP_WriteDOPulseCount(char szIP[],u_short wDoChannel,u_long ulPulseCount);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_WriteDOPulseCount(szIP: PChar; wDoChannel: Integer; ulPulseCount: Longint): Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

int   TCP_WriteDOPulseCount(char szIP[],u_short wDoChannel,u_long ulPulseCount);

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wDoChannel[in]	: the channel index for writing
ulPulseCount[in]	: the pulse output count.

Return Code: refer to the *Error code*.

7.5.41 TCP_WriteDODelayWidth

Description: to set the pulse and delay signal widths to specific EDAM-4200 DIO modules

Syntax:

♦ **Visual Basic: (*see TCPDAQ.bas*)**

Declare Function TCP_WriteDODelayWidth Lib "TCPDAQ.dll" Alias "_TCP_WriteDODelayWidth@24" (ByVal szIP As String, ByVal wChno As Integer, ByVal ulLoPulseWidth As Long, ByVal ulHiPulseWidth As Long, ByVal ulLoDelayWidth As Long, ByVal ulHiDelayWidth As Long) As Long

♦ **Borland C++ Builder: (see TCPDAQ.h)**

Int   TCP_WriteDODelayWidth(Char szIP[], u_short wChno,u_long ulLoPulseWidth,u_long ulHiPulseWidth, u_long ulLoDelayWidth,u_long ulHiDelayWidth);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function   TCP_WriteDODelayWidth (szIP: PChar; wChno: Integer; ulLoPulseWidth: Longint; ulHiPulseWidth: Longint;ulLoDelayWidth: Longint; ulHiDelayWidth: Longint): Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

```
int      TCP_WriteDODelayWidth(char szIP[], u_short wChno,
         u_long ulLoPulseWidth, u_long ulHiPulseWidth,
         u_long ulLoDelayWidth, u_long ulHiDelayWidth);
```

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wChno[in]	: the channel index for writing
ulLoPulseWidth[in]	: the output pulse signal width at low level.

ulHiPulseWidth[in]	: the output pulse signal width at high level.
ulLoDelayWidth[in]	: the output signal delay width when set DO from high to low level.
ulHiDelayWidth[in]	: the output signal delay width when set DO from low to high level.

Return Code: refer to the *Error code*.

7.5.42 TCP_ReadDODelayWidth

Description: to read the pulse and delay signal widths from specific EDAM-6000 DIO modules

Syntax:

- **Visual Basic:** (*see TCPDAQ.bas*)

  Declare Function TCP_ReadDODelayWidth Lib "TCPDAQ.dll" Alias "_TCP_ReadDODelayWidth@24" (ByVal szIP As String, ByVal wChno As Integer, ByRef ulLoPulseWidth As Long, ByRef ulHiPulseWidth As Long, ByRef ulLoDelayWidth As Long, ByRef ulHiDelayWidth As Long) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  ```
  int      TCP_ReadDODelayWidth(char szIP[],u_short wChno,
            u_long *ulLoPulseWidth,u_long *ulHiPulseWidth,
            u_long *ulLoDelayWidth,u_long *ulHiDelayWidth);
  ```

- **Delphi: (see TCPDAQ.pas)**

  Function TCP_ReadDODelayWidth (szIP: PChar; wChno: Integer; ulLoPulseWidth: Longint; ulHiPulseWidth: Longint;ulLoDelayWidth: Longint; ulHiDelayWidth: Longint): Longint; StdCall;

- **VC++: (see TCPDAQ.h)**

  ```
  int      TCP_ReadDODelayWidth(char szIP[],u_short wChno,   u_long *ulLoPulseWidth,lu_long *ulHiPulseWidth,
                                  u_long *ulLoDelayWidth,u_long *ulHiDelayWidth);
  ```

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wChno[in]	: the channel index for reading
ulLoPulseWidth[out]	: the pulse output signal width at low level
ulHiPulseWidth[out]	: the pulse output signal width at high level
ulLoDelayWidth[out]	: the delay output signal width at low level
ulHiDelayWidth) [out]	: the delay output signal width at high level

Return Code: refer to the *Error code*.

7.5.43 TCP_MODBUS_ReadCoil

Description: to read the coil values at a specific range described in parameters

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

  Declare Function TCP_MODBUS_ReadCoil Lib "TCPDAQ.dll" Alias "_TCP_MODBUS_ReadCoil@16" (ByVal szIP As String, ByVal wStartAddress As Integer, ByVal wCount As Integer, ByRef DATA As Byte) As Long

- **Borland C++ Builder: (see TCPDAQ.h)**

  ```
  Int   TCP_MODBUS_ReadCoil(char szIP[],u_short wStartaddress,u_short wCount,u_char byData[]);
  ```

- **Delphi: *(see TCPDAQ.pas)***

  Function TCP_MODBUS_ReadCoil (szIP: PChar; wStartAddress: Integer; wCount: Integer; Data: PByte): Longint; StdCall;

- **VC++: *(see TCPDAQ.h)***

  ```
  Int   TCP_MODBUS_ReadCoil(char szIP[],u_short wStartAddress,u_short wCount,u_char byData[]);
  ```

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wStartAddress[in]	: start address of coil registers (1 ~ 255)
wCount[in]	: the count that coil data be read
byData[in]	: the 8 bit array that stored the coil data (0=set, 1=reset)

Return Code: refer to the *Error code*.

7.5.44 TCP_MODBUS_WriteCoil

Description: to write the coil values at a specific range described in parameters.

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

```
Declare Function TCP_MODBUS_WriteCoil Lib "TCPDAQ.dll" Alias "_TCP_MODBUS_WriteCoil@16"
                    (ByVal szIP As String, ByVal wStartAddress As Integer, ByVal wCount As Integer,
                    ByRef DATA As Byte) As Long
```

- **Borland C++ Builder: (see TCPDAQ.h)**

```
int     TCP_MODBUS_WriteCoil(char szIP[],u_short wStartAddress,u_short wCount,u_char byData[]);
```

- **Delphi: *(see TCPDAQ.pas)***

```
Function  TCP_MODBUS_WriteCoil(szIP: PChar; wStartAddress: Integer; wCount: Integer; Data: PByte):
          Longint; StdCall;
```

- **VC++: *(see TCPDAQ.h)***

```
int     TCP_MODBUS_WriteCoil(char szIP[],u_short wStartAddress,u_short wCount,u_char byData[]);
```

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wStartAddress[in]	: start address of coil registers (1 ~ 255)
wCount[in]	: the count that coil data be written
byData[in]	: the 8 bit array that stored the coil data (0=set, 1=reset)

Return Code: refer to the *Error code*.

7.5.45 TCP_MODBUS_ReadReg

Description: to read the holding register value at a specific range described in parameters

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

```
Declare Function TCP_MODBUS_ReadReg Lib "TCPDAQ.dll" Alias "_TCP_MODBUS_ReadReg@16"
                    (ByVal szIP As String, ByVal wStartAddress As Integer, ByVal wCount As Integer,
                     ByRef DATA As Integer) As Long
```

- **Borland C++ Builder: (see TCPDAQ.h)**

```
Int   TCP_MODBUS_ReadReg(char szIP[],u_short wStartAddress,u_short wCount,u_short wData[]);
```

- **Delphi: *(see TCPDAQ.pas)***

```
Function      TCP_MODBUS_ReadReg (szIP: PChar; wStartAddress: Integer; wCount: Integer; Data: PWord):
              Longint; StdCall;
```

- **VC++: *(see TCPDAQ.h)***

```
Int   TCP_MODBUS_ReadReg(char szIP[],u_short wStartAddress,u_short wCount,u_short wData[]);
```

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wStartAddress[in]	: start address of holding registers (1 ~ 255)
wCount[in]	: the count that holding data be read
byData[in]	: the 16 bit array that stored the holding data

Return Code: refer to the *Error code*.

7.5.46 TCP_MODBUS_WriteReg

Description: to write values to the holding registers at a specific range described in parameters

Syntax:

- **Visual Basic: (*see TCPDAQ.bas*)**

```
Declare Function TCP_MODBUS_WriteReg Lib "TCPDAQ.dll" Alias "_TCP_MODBUS_WriteReg@16"
                    (ByVal szIP As String, ByVal wStartAddress As Integer, ByVal wCount As Integer,
                    ByRef DATA As Integer) As Long
```

- **Borland C++ Builder: (see TCPDAQ.h)**

```
Int   TCP_MODBUS_WriteReg(char szIP[],u_short wStartAddress,u_short wCount,u_short wData[]);
```

- **Delphi:** ***(see TCPDAQ.pas)***

```
Function     TCP_MODBUS_WriteReg(szIP: PChar; wStartAddress: Integer; wCount: Integer; Data: PWord):
             Longint; StdCall;
```

- **VC++:** ***(see TCPDAQ.h)***

```
Int   TCP_MODBUS_WriteReg(char szIP[],u_short wStartAddress,u_short wCount,u_short wData[]);
```

Parameters:

szIP[in]	: the IP address for an EDAM-4200 that to be connected
wStartAddress[in]	: start address of holding registers (1 ~ 255)
wCount[in]	: the count that holding data be read
byData[in]	: the 16 bit array that stored the holding data

Return Code: refer to the *Error code.*

Chapter 8 ASCII Commands for EDAM-4200 Modules

8.1 About ASCII Commands

For users do not familiar to Modbus protocol, Inlog offers a function library as a protocol translator, integrating ASCII command into Modbus/TCP structure. Therefore, users familiar to ASCII command can access EDAM-4200 easily. Before explaining the structure of ASCII command packed with Modbus/TCP format. Let's see how to use an ASCII command and how many are available for your program.

EDAM-4200 series also integrate ASCII command into ***UDP protocol with port 1025****. User can simply send the Command of ASCII format through UDP protocol (such as UPD_send (Dest_IP, "$01M")).*

8.2 Syntax of ASCII

Command Syntax: [delimiter character][address][channel][command][data][checksum][carriage return] Every command begins with a delimiter character.

There are two valid characters: $ and # .The delimiter character is followed by a two-character address (hex-decimal) that specifies the target system. The two characters following the address specific the module and channel.

Depending on the command, an optional data segment may follow the command string. An optional two-character checksum may also be appended to the command string. Every command is terminated with a carriage return(cr).

The command set is divided into the following five categories:

- ♦ System Command Set
- ♦ Analog Input Command Set
- ♦ Analog Input Alarm Command Set
- ♦ Universal I/O Command Set
- ♦ Digital I/O Command Set

Every command set category starts with a command summary of the particular type of module, followed by datasheets that give detailed information about individual commands. Although commands in different subsections sometime share the same format, the effect they have on a certain module can be completely different than that of another. Therefore, the full command sets for each type of modules are listed along with a description of the effect the command has on the given module.

Note: All commands should be issued in UPPERCASE characters only!

8.3 ASCII Command Set

8.3.1 General commands

Command	Function	Description	Modules	Pg.
$AAM	Read Module Name	Returns the module name from a specific module	All	86
$AAF	Read Firmware Version	Returns the firmware version from a specific module.	All	87
$AAID	Read module ID	Read user define module address(ID) (for tcpdaq.dll driver)	All	88
$AAIDFF	Set Module ID number	Set user define module address(ID) (for tcpdaq.dll device driver).	All	89
$AAMD(data)	Write module description	Write module description(max 30 characters)	All	90
$AAMD	Read module description	Read module description	All	91
$AAS1	Reloads the module factory default	Reloads the module factory default	All	92
~AAI	Soft INIT* command (Enable)	Soft INIT* command (Enable)	All	93
~AATNN	Sets the soft INIT* timeout value	Sets the soft INIT* timeout value	All	94
$AARS	Reboot the module	Reboot the module to the power-on state	All	95
$AA5	Reads the Reset Status of a module	Reads the Reset Status of a module	All	96
~AADNNNNN	Set timout to search DHCP	Set timout to search DHCP	All	97
~AAD	Read timout to search DHCP	Read timout to search DHCP	All	98
^AAMAC	Read MAC address	Read MAC address	All	99

8.3.2 Digital I/O commands

Command	Function	Description	Modules	Pg.
$AACONNDD	Set a single DO mode for channel N	Set a single DO mode for channel N	All	100
$AACONN	Read a single DO mode for channel N	Read a single DO mode for channel N	All	101
$AACINNDD	Set a single DI mode for channel N	Set a single DI mode for channel N	All	102
$AACINN	Read a single DI mode for channel N	Read a single DI mode for channel N	All	103
~AADSMN	Set DI/O active state	Set digital input/output active state	All	104
~AADS	Read DI/O active state	Read digital input/output active state	All	105
$AA6	Read DI/O Channel Status	Read the status of all DI(0~15) and DO(0~15) channels. (Same as "@AA")	All	106
@AA	Read DI/O Status	Read the status of all DI(0~15) and DO(0~15) channels. (Same as "@AA6")	All	107
#AA00DD	Write DO channels(0~7)	Write a value to digital output channels(0~7)	All	108
#AA1NDD	Set a single DO channel	Set a single digital output channel	All	109
@AA6DDDD	Write DO channels	Write a value to digital output channels(0~15)	All	110
@AA6ONSS	Set a single DO channel	Set a single digital output channel(0~15)	All	111
@AA6ON	Read single digital	Read a single digital output for channel (0~15)	All	112

	output for channel N			
@AA6	Read the status of all DIO channels	Read the status of all DIO(0~15) channels.	All	113
@AA6IN	Read a single digital input for channel N	Read a single digital input for channel (0~15)	All	114
$AA7	Read DI latch status	Read DI latch status	All	115
$AACLSNN	Clear DI latch status	Clear DI latch status for channel N	All	116

8.3.3 DIO Synchronization Mode(Mirror Local DI to Local /Remote DO) Commands

Command	Function	Description	Modules	Pg.
$AAYM5CRPS TTTT (data)	Set *DI match DO latch mode of DIO Sync.*	Set *DI match DO latch mode* of DIO Sync. (For Mirror Local DI to *Local DO channel* & Remote DO channel)	All	117
$AAYM6CRPS TTTT (data)	Set DI mismatch DO latch mode of DIO Sync.	Set *DI mismatch DO latch mode* forDIO Sync. (For Mirror Local DI to *Local DO channel* & Remote DO channel)	All	118
$AAYMNDDDD DDDD	Set remote device IP for DO channel N (for DIO sync. mode)	Set remote device IP for DO channel N (for DIO sync. mode)	All	119
$AAYMRCS	Start(Run)/Stop DIO synchronize	Start(Run)/Stop DIO sync. operation	All	121
$AAYM4C	Read DIO Sync. Mode parameters	Read DIO Sync. Mode parameters	All	120
$AAYMS	Read DIO Sync. DO active status	Read DIO Sync. DO active status	All	122
$AAYMPN	Read DO Sync. remote device IP for DO ch. N	Read remote device IP for DO channel N (for DIO sync. mode)	All	123

8.3.4 DO Pulse Output mode & Digital output Auto-Off Time mode commands

Command	Function	Description	Modules	Pg.
#AA2NPPPPPPPP	Write DO pulse counts	Write DO pulse counts to the specific DO channel (For *High/Low delay mode*)	All	124
$AA9PNNLLLL HHHH	Set DO pulse low/high width for channel N	Set DO pulse low/high width for channel N. (For High/Low delay mode)	All	125
$AA9NN	Read DO pulse low/high width for channel N	Read DO pulse low/high width and DO Low/high delay output width for ch. N, (For *High/Low delay mode* and *Digital output Auto-Off Time Mode*)	All	126
$AA9DNNHHHH LLLL	Set DO low/high delay width for channel N	Set DO Low/high delay output width for channel N (00~15), unit: 0.5ms. (For *High/Low delay mode* and *Digital output Auto-Off Time Mode*)	All	127

8.3.5 Digital Input Counter commands

Command	Function	Description	Modules	Pg.
$AA0MCC	Read DI counter filter (debounce time)	Read DI counter pre-debounce and post-debounce of channel N (unit = 0.5ms)	All Except 4251	129
$AA0MCC (data1)(data2)	Set DI counter filter (debounce time)	Set DI counter pre-debounce and post-debounce of channel N (unit = 0.5ms)	Except 4251	130
$AAECN	Start/Stop DI counter	Start/Stop counter of the specific DI channel.	All	131
$AACN	Clear DI counter	Clear DI counter of the specific DI channel.	All	132
#AAN	Read DI counter	Read counter value of the specific DI channel.	All	133
#AARN	Read DI counter with overflow	Read a single DI channel counter with overflow.	All	134

8.3.6 WatchDog commands

Command	Function	Description	Modules	Pg.
~**	Host ok	host ok. Refresh WDT counter of all modules via broadcast, (No reply from modbus response	All	135
~AA**	Host ok	host ok. Refresh WDT counter of the specific module (Response : !01)	All	136
~AA0	Read host watchdog timeout status	Read host watchdog timeout status. bit(7) - watchdog Enable/Disable, bit(2) - watchdog timeout(1) status)	All	137
~AA1	Reset host watchdog timeout status	Reset host watchdog timeout status	All	138
~AA2	Read host communication Timeout value.	Read host communication Timeout value. (0.1sec)	All	139
~AA3EVVV	Set Host watchdog timeout interval.	Enable Host watchdog and set timeout interval (unit=0.1sec)	All	140
~AA4V	Read the power-on DO value or the safe DO value of a module	Read the power-on DO value or the safe DO value of a module	All	142
~AA5V	Sets the current value as the power-on DO value or the safe DO value	Sets the current value as the power-on DO value or the safe DO value	All	143
~AA3PPP	Set module Power-on delay time.	Set module Power-on delay time (unit=0.1sec) to start host communication timeout watchdog	All	144
~AA3P	read module Power-on delay time	Read module Power-on delay time (unit=0.1sec) to start host communication timeout watchdog	All	145

8.4 ASCII Command Description

8.4.1 $AAM Read Module Name

Description:	Read the module name	
Command:	$AAM[CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	M	Command for Read Module Name
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA(data)[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	(data)	A string showing the name of the module (max. 8 chars.)
	CHK	Check sum
	(cr)	Carriage return

Example:

The command requests the system at address 01h to send its module name. The system at address 01h responds with module name 9050A indicating that there is an EDAM-9050A at address 01h.

Command: $01M(cr)

Response: !019050A(cr)

8.4.2 $AAF Read Firmware Version

Description:	Returns the firmware version from a specific module.	
Command:	$AAF[CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	F	is the Firmware Version command
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA(data)[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	(data)	A string showing the firmware version of the module (max. 6 chars.)
	CHK	Check sum
	(cr)	Carriage return

Example:

The command requests the system at address 01h to send its firmware version. The system responds with firmware version 6.080

Command: $01F(cr)

Response: !01 6.080 (cr)

8.4.3 **$AAID** Read module ID number

Description:	Read user define module address(ID). (for tcpdaq.dll device driver)	
Command:	$AAID[CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	ID	is the ID command.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AANN[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	NN	represents the ID number of the module.
	CHK	Check sum
	(cr)	Carriage return

Example:

The command requests the system at address 01h to send its ID number. The system responds with ID number 10(0AH).

Command: $01ID (cr)

Response: !010A (cr)

Related command: $AAIDFF

8.4.4 **$AAIDFF** Set module ID number

Description:	Set user define module address(ID) (for tcpdaq.dll device driver)	
Command:	$AAIDFF[CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	ID	is the ID command.
	FF	Module address (range 01-FF)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

The command Sets the ID of the module 01 to be "1A " and returns a valid response.

Command: $01ID1A (cr)

Response: !01 (cr)

Related command: $AAID

8.4.5 $AAMD(data) Set module description

Description:	Set module description	
Command:	$AAMD(data)[CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	MD	Set module description command.
	(data)	module description (max 30 characters)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

Set the desc. of the module 01 to be "12DI8DO " and returns a valid response.

Command: $01MD12DI8DO(cr)

Response: !01 (cr)

Related command: $AAMD

8.4.6 $AAMD Read module description

Description:	Read module description	
Command:	$AAMD [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	MD	Set module description command.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA(data)[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	(data)	module description (max 30 characters)
	CHK	Check sum
	(cr)	Carriage return

Example:

Set the desc. of the module 01 to be "12DI8DO " and returns a valid response.

Command: $01MD12DI8DO(cr)

Response: !01 (cr)

Related command: $AAMD(data)

8.4.7 **$AAS1** Reloads the module factory default

Description:	Reloads the module factory default	
Command:	$AAS1 [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	S1	command to reload the factory default.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Note: Before the command is issued, The Soft INIT* switch should be set to enable via "set the soft INIT* " command. (ref. ~AAI, ~AATnn)

Example :

(1) Sets the soft INIT* timeout value of module 01 to 32 seconds and returns a valid response.
Command: ~01T32(cr)
Response: !01(cr)

(2) Sets the soft INIT* enable and returns a valid response.
Command: ~01I(cr)
Response: !01(cr)

(3) Reloads the module factory default
Command: $01S1(cr)
Response: !01(cr)

Related command: ~AATnn, ~AAI

8.4.8 **~AAI** Set the Soft INIT*

Description:	The Soft INIT* command is used to enable modification of the IP, Gateway and communication protocol settings using software only.	
Command:	~AAI [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	I	command to set the Soft INIT* enable
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example :

1. Sets the soft INIT* timeout value of module 01 to 16 seconds and returns a valid response.
   Command: ~01T10(cr)
   Response: !01(cr)

2. Sets the soft INIT* enable and returns a valid response.
   Command: ~01I(cr)
   Response: !01(cr)

3. Reloads the module factory default
   Command: $01S1(cr)
   Response: !01(cr)

Related command: ~AATnn, $AAS1

8.4.9 **~AATNN** Sets the soft INIT* timeout value

Description:	The Soft INIT* command is used to enable modification of the IP, Gateway and communication protocol settings using software only.	
Command:	~AATNN [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	T	command to Sets the soft INIT* timeout value
	NN	Two hexadecimal digits representing the timeout value in seconds. The maximum timeout value is 60 seconds. When changing the IP or Gateway settings without altering the INIT* slide switch, the ~AAI and (or $AAS1) commands should be sent consecutively and the time interval between the two commands should be less than the soft INIT* timeout. If the soft INIT* timeout is 0, then the IP and Gateway settings cannot be changed using software only. The power-on reset value of the soft INIT timeout is 0.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example :

1. Sets the soft INIT* timeout value of module 01 to 16 seconds and returns a valid response.
   Command: ~01T10(cr)
   Response: !01(cr)

2. Sets the soft INIT* enable and returns a valid response.
   Command: ~01I(cr)
   Response: !01(cr)

3. Reloads the module factory default
   Command: $01S1(cr)
   Response: !01(cr)

Related command: ~AAI, $AAS1

8.4.10 $AARS Reboot the module to the power-on state

Description:	Reboot the module to the power-on state	
Command:	$AARS [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	RS	command to reboot the module to the power-on state
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

command: $01RS(cr)

response: !01(cr) ; Reboot the module to the power-on state.

Related command: $AA5

8.4.11 $AA5 Reads the Reset Status of a module

Description:	Reads the Reset Status of a module	
Command:	$AA5 [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	5	Command for read reset status
	CHK	Check sum
	(cr)	Carriage return
Response:	!AAS [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	S	the Reset Status of a module. = 0 - the module is not been reseted. = 1 - the module is been reseted.
	CHK	Check sum
	(cr)	Carriage return

Example 1:

command: $015(cr)

response: !011(cr) ; the module is been reseted

Example 2:

command: $015(cr)

response: !010(cr) ; the module is not been reseted

Related command: $AARS

8.4.12 **~AADNNNNN** Set timout to search DHCP

Description:	Set timout to search DHCP	
Command:	~AADNNNNN [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	D	is set DHCP search timeout command.
	NNNNN	DHCP timeout value (10~1800 sec, dec)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

The command set timeout value to search DHCP servo. If there is no DHCP exist and timeout reached, the module will reboot and use static (Fixed) IP assigned by E9KUtiliy.exe

command:	~01D01200(cr)
response:	!01(cr)

Related command: ~AAD

8.4.13 ~AAD Read timout to search DHCP

Description:	Set timout to search DHCP	
Command:	~AAD [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	D	is read DHCP search timeout command.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AANNNNN,D [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	NNNNN	DHCP timeout value (10~1800 sec, dec)
	,	Delimiter for DHCP IP OK
	D	DHCP IP OK, = 1 - DHCP IP OK, otherwise = 0 - none
	CHK	Check sum
	(cr)	Carriage return

Example:

The command read timeout is 1200 seconds and None DHCP IP.

command: **~01D(cr)**

response: **!0101200,0(cr)**

Related command: ~AADNNNNN

8.4.14 ^AAMAC Read MAC address

Description:	Read MAC address	
Command:	^AAMAC [CHK](cr)	
Syntax:	^	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	MAC	ccommand for read MAC address
	CHK	Check sum
	(cr)	Carriage return
Response:	!AAMMMMMM MMMMMM [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	MMMMMMMMMMMM	MAC address(hex)
	CHK	Check sum
	(cr)	Carriage return

Example:

command: ^01MAC(cr)

response: !0100E04C360629(cr) ; MAC Address: 00E04C360629

8.4.15 **$AACONNDD** Set a single DO channel mode

Description:	Set a single DO channel mode.	
Command:	$AACONNDD [CHK](cr)	
Syntax:	^	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	CO	command for set a single DO mode for channel N
	NN	Channel number (00~1F)
	DD	DO mode setting (2 characters), = 00 - Direct DO output(default) = 01 - Pluse output mode = 02 - Low to high delay = 03 - High to low delay = 04 - Automatic DIO Synchronization Mode = 05 - reserved = 06 - DO Auto-Off Time Mode for DO L*ow to High to Low*. = 07 - DO Auto-Off Time Mode for DO *High to Low to High*.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

Set DO channel(0) to *Low to high delay mode*.

Command: $01CO0002(cr)

Response: !01(cr)

Related command: $AACONN, $AACIINNDD, $AACINN

8.4.16 $AACONN Read a single DO channel mode

Description:	Read a single DO channel mode	
Command:	$AACONN [CHK](cr)	
Syntax:	^	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	CO	command for read a single DO mode for channel N
	NN	Channel number (00~1F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AADD [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DD	DO mode setting (2 characters), = 00 - Direct DO output(default) = 01 - Pluse output mode = 02 - Low to high delay = 03 - High to low delay = 04 - Automatic DIO Synchronization Mode = 06 - DO Auto-Off Time Mode for DO *Low to High to Low*. = 07 - DO Auto-Off Time Mode for DO *High to Low to High*.
	CHK	Check sum
	(cr)	Carriage return

Example:

read DO channel(0) *Low to high delay mode*

Command: $01CO00(cr)

Response: !0102(cr)

Related command: $AACONNDD, $AACIINNDD, $AACINN

8.4.17 $AACINNDD Set a single DI channel mode

Description:	Set a single DI channel mode	
Command:	$AACINNDD [CHK](cr)	
Syntax:	^	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	CI	command for set a single DI mode for channel N
	NN	Channel number (00~1F)
	DD	DI mode setting (2 characters), bit(2,1,0) - DI mode setting(default). = 0x000 - Direct DI input = 0x001 - Counter Mode = 0x010 - Low to high latch = 0x011 - High to low latch = 0x100 - Input frequency mode bit(5,4,3) - always 0, bit(6) - DI counter H/L width filter (Default H/L width =5ms) = 0 - disable (off), (default) = 1 - enable (on) bit(7) - always 0,
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

Set DI channel(0) to _Counter Mode_, _DI counter H/L width filter_ enable,

Command: $01CI00C1(cr) // bit(2,1,0) = 0x001, bit(6)=1

Response: !01(cr)

Related command: $AACINN, $AACONNDD, $AACONN

8.4.18 **$AACINN** Read a single DI channel mode

Description:	Read a single DI channel mode	
Command:	$AACINN [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	CI	command for read a single DI mode for channel N
	NN	Channel number (00~1F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AADD [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DD	DI mode setting (2 characters), bit(2,1,0) - DI mode setting. = 0x000 - Direct DI input = 0x001 - Counter Mode = 0x010 - Low to high latch = 0x011 - High to low latch = 0x100 - Input frequency mode bit(5,4,3) - always 0, bit(6) - DI counter H/L width filter (Default H/L width =5ms) = 0 - disable (off). = 1 - enable (on). bit(7) - always 0,
	CHK	Check sum
	(cr)	Carriage return

Related command: $AACINNDD, $AACONNDD, $AACONN

8.4.19 ~AADSMN Set DI/O active state

Description:	Set digital input/output active state.	
Command:	~AADSMN [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	DS	Command for setting DIO active status
	M	Digital input channel active values, = 0 - represent input value=0 is activate (ON), input value=1 is deactivate (OFF or OPEN). **(Default)** = 1 - represent input value=1 is activate (ON), input value=0 is deactivate (OFF or OPEN).
	N	Digital output channel active values, = 0 - represent output value=1 is activate (ON), output value=0 is deactivate (OFF). **(Default)** = 1 - represent output value=1 is deactivate (OFF), output value=0 is activate (ON).
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: For EDAM-9050A (10-DI/6-DO channels)

- Reads the value of the DI/DO channels
  Command: @016(cr)
  Response: >000003FF (cr) ; DO=0000 & DI=03FF

- Read DI active state is 0 and DO active state is 0.
  Command: ~01DS(cr)
  Response: !0100(cr)

- Set input active state= 1, when input value=1 and set output active state =1, when output value=high/open,
  Command: ~01DS11(cr)
  Response: !01(cr)

- Reads the value of the DI/DO channels
  Command: @016(cr)
  Response: >003F0000 (cr) ; DO=003F & DI=0000

Related command: ~AADS

8.4.20 **~AADS** Read DI/O active state

Description:	Read digital input/output active state.	
Command:	~AADS [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	DS	Command for read DIO active status
	CHK	Check sum
	(cr)	Carriage return
Response:	!AAMN [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	M	Digital input channel active values, = 0 - represent input value=0 is activate (ON), input value=1 is deactivate (OFF or OPEN). **(Default)** = 1 - represent input value=1 is activate (ON), input value=0 is deactivate (OFF or OPEN).
	N	Digital output channel active values, = 0 - represent output value=1 is activate (ON), output value=0 is deactivate (OFF). **(Default)** = 1 - represent output value=1 is deactivate (OFF), output value=0 is activate (ON).
	CHK	Check sum
	(cr)	Carriage return

Example: Ref. command ~AADSMN

Related command: ~AADSMN

8.4.21 **$AA6** Read DI /DO Channel Status

Description:	This command requests that the specific EDAM-4200 module return the status of its digital input and digital output channels. (same as "@AA")	
Command:	$AA6 [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	6	Command for read DIO channel status
	CHK	Check sum
	(cr)	Carriage return
Response:	>DDDDFFFF[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DDDD	Hexadecimal DO status (0000~FFFF).
	FFFF	Hexadecimal DI status (0000~FFFF).
	CHK	Check sum
	(cr)	Carriage return

Example 2: Read EDAM-4250 DIO status

command:	$016(cr)	; read EDAM-4250 DIO status
response :	!0100030004(cr)	; represents DO0, DO1 are ON and DI2 is ON

Related command: @AA, @AA6

8.4.22 **@AA** Read DIO status

Description:	This command requests that the specific EDAM-4200 module return the status of its digital input and digital output channels. (same as "@AA6")	
Command:	@AA [CHK](cr)	
Syntax:	@	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	CHK	Check sum
	(cr)	Carriage return
Response:	>DDDDFFFF[CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DDDD	Hexadecimal DO status (0000~FFFF).
	FFFF	Hexadecimal DI status (0000~FFFF).
	CHK	Check sum
	(cr)	Carriage return

Example 1: Read EDAM-4250 DIO status

```
command:    @01(cr)             ; read EDAM-4250 DIO status
response :  >00030004(cr)       ; represents DO0, DO1 are ON and DI2 is ON
```

Related command: @AA6, $AA6

8.4.23 **#AA00DD** Write DO channels

Description:	Write a value to digital output channels).	
Command:	#AA00DD [CHK](cr)	
Syntax:	#	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	00	Represents Writing to all channels (write a byte) command
	DD	Represents the data be written to digital output(00~FF)
	CHK	Check sum
	(cr)	Carriage return
Response:	>AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: An output byte with value 33h (00110011) is sent to the digital output module at address 01h.
The Output channel 0,1,4,5 = ON, Output channel 2,3,6,7 = OFF

command: #010033(cr)
response: >01(cr)

Related command: @AA, $AA6, @AA6, @AA6DDD

8.4.24 #AA1NDD Set a single Digital Output Channel

Description:	Write a single digital output channel	
Command:	#AA1NDD [CHK](cr)	
Syntax:	#	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	1	represents writing to a single DO channel command
	N	Channel number (0-F).
	DD	represents the status you want to set to the specific channel. = 00 – output Deactivate. = 01 – output Activate.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example 1: The command set digital channel 2 Activate (ON) status for the specific module at address 01h.

command: #011201(cr)

response: !01

Example 2: The command set digital channel 2 Deactivate (OFF) status for the specific module at address 01h.

command: #011200(cr)

response: !01

Related command: @AA, $AA6, @AA6, @AA6DDD , #AA00DD

8.4.25 **@AA6DDDD** Write DO channels (0~15)

Description:	Write value to digital output channels (0~15)	
Command:	@AA6DDDD [CHK](cr)	
Syntax:	@	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	6	represents write value to digital output channels command
	DDDD	Represents the data be written to digital output(0000~FFFF)
	CHK	Check sum
	(cr)	Carriage return
Response:	> [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: For E-9050A, write DO(0,2) to Activate (ON)

command: @0160005(cr)

response: >(cr)

Related command: @AA, $AA6, @AA6, @AA6DDD , #AA00DD, #AA1NDD

8.4.26 @AA6ONSS Set a single digital output channel

Description:	Write a single digital output channel	
Command:	@AA6ONSS [CHK](cr)	
Syntax:	@	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	6O	represents writing to a single DO channel command
	N	Channel number (0-F).
	SS	represents the status you want to set to the specific channel. = 00 – output Deactivate. = 01 – output Activate.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: Set DO(1) to active state

```
command:  @016O1(cr)        ; Read channel(1) DO status
response: >00(cr)           ; 00 represents DO channel(1) is deactivated

command:  @016O101(cr)      ; set DO(1) to activate state
response: !01(cr)

command:  @016O1(cr)        ; Read DO channel(1) value
response: >01(cr)           ; 01 represents DO channel(1) is activated
```

Related command: @AA, $AA6, @AA6, @AA6DDD , #AA00DD, #AA1NDD, @AA6DDDD

8.4.27 @AA6ON Read a single digital output channel

Description:	Read status of a single digital output channel	
Command:	@AA6ON [CHK](cr)	
Syntax:	@	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	6O	command to read status of a single digital output channel.
	N	Channel number (0-F).
	CHK	Check sum
	(cr)	Carriage return
Response:	>DD [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DD	Represents the status you want to set to the specific channel. = 00 – output Deactivate. = 01 – output Activate.
	CHK	Check sum
	(cr)	Carriage return

Example: Read DO(1) status

command: @016O1(cr)

response: >01(cr) ; 01 represents DO channel(1) is activated

Related command: @AA, $AA6, @AA6, @AA6DDD , #AA00DD, #AA1NDD, @AA6DDDD, @AA6ONSS

8.4.28 **@AA6** Read the status of all DIO channels

Description:	Read the status of all 16 DO and 16 DI channels	
Command:	@AA6 [CHK](cr)	
Syntax:	@	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	6	command for read DIO status
	CHK	Check sum
	(cr)	Carriage return
Response:	>TTTTNNNN [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	TTTT	Represents the 4-character hexadecimal DO status (0000~FFFF)
	NNNN	Represents the 4-character hexadecimal DI status (0000~FFFF)
	CHK	Check sum
	(cr)	Carriage return

Example: For E-9050A,

command:	@016(cr)	
response:	>00030004(cr)	; 0003 represents DO0, DO1 are ON and DO2~DO15 are OFF
		; 0004 represents DI2 is ON and DI0, DI1, and DI3~DI15 are OFF

Related command: @AA, $AA6, @AA6, @AA6DDD , #AA00DD, #AA1NDD, @AA6DDDD, @AA6ONSS

8.4.29 **@AA6IN** Read a single digital input channel

Description:	Read status of a single digital input channel	
Command:	@AA6IN [CHK](cr)	
Syntax:	@	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	6I	command to read status of a single digital input channel
	N	Channel number (0-F).
	CHK	Check sum
	(cr)	Carriage return
Response:	>DD [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	>	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DD	Input value = 00 – Inactive(OFF). = 01 – Active(ON).
	CHK	Check sum
	(cr)	Carriage return

Example: Read the status of DI channel(1)

command: @016I1(cr)

response: >01(cr) ; 01 represents DI channel(1) is active

Related command: @AA, $AA6, @AA6, @AA6DDD, #AA00DD, #AA1NDD, @AA6DDDD, @AA6ONSS, @AA6ON

8.4.30 $AA7 Read DI latch status

Description:	Read DI latch status	
Command:	$AA7 [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	7	Represents read DI latch status command
	CHK	Check sum
	(cr)	Carriage return
Response:	!AADDDD [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DDDD	Represent DI latch status, The discrete DI latch status in the response message are packed as one input per bit of the data field. Status is indicated as 1= latched; 0= no latched.
	CHK	Check sum
	(cr)	Carriage return

Example: The command read DI latch status= 0003, DI #0 latched, DI #1 latched, and DI #2 ~ DI #15 no latched

command: $017(cr)

response: !010003(cr)

Related command: $AACLSNN

8.4.31 **$AACLSNN** Clear DI latch status for channel N

Description:	Read DI latch status	
Command:	$AACLSNN [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	CLS	Represents clear DI latch status command
	NN	Represents DI channel to be cleared, = 0x00 ~ 0x0F - channel number. = 0xFF - Clear all DI channels
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: This command clears DI(1) latch status

command: $01CLS01(cr)

response: !01(cr)

Example: This command clears all DI latch status

command: $01CLSFF(cr)

response: !01(cr)

Related command: $AA7

8.4.32 **$AAYM5CRPSTTTT (data)** Set *DI match DO latch Mode* of DIO Sync.

Description:	Set DO to DI match DO latch Mode of DIO Synchronization. A single digital output channel is activated (1 or 0) and latched, when DI input value match DI mask pattern.	
Command:	$AAYM5CRPSTTTT (data) [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YM5	command to set DI match/ DO latch mode of Auto DIO Sync.
	C	Mirrored DO channel number (hex 0~F).
	R	write DO to local module or remote module when DIO sysnc active. = 0 - Local Mode, (Local DI->Local DO) = 1 - remote Mode,(Local DI->Remote DO)
	P	Enable/Disable Auto Run(Start) DIO Synchronization operation when power-on. = 0 - Disiable = 1 - Enable
	S	digital output active state when DI data pattern match = 0 - digital output 0 when DI data pattern match = 1 - digital output 1 when DI data pattern match = 2 - toggle digital output
	TTTT	DI channel pre-bounce time (hex 0000~FFFF ms)
	(data)	represent DI mask pattern is used to indicate the monitored input channels(0~15) and mask state (16char), bits(15..0) - indicate DI channel(15..0) to be monitored and mask state, = '1' - indicate DI channel n is monitored and mask state is '1'. = '0' - indicate DI channel n is monitored and mask state is '0'. = 'X' - don't care (not be monitored)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Note: *Before running DIO Synchronization function, you must set DO Mode to " DIO Sync. Mode " via command $AACONNDD.*

Example: Assume DI channel 0,2,5 are monitored and the ASCII form of DI mask pattern is XXXXXXXXXX1XX0X1.
When DI input ch(0,5)=1 and ch(2)=0, the corresponding DO(0) will be set to ON(1) and latched.
DI pre-debounce time is 300 msec. (Ref. to 12.7)

command: $01YM501110 12C XXXXXXXXXX1XX0X1(cr)

where R = 1 - remote Mode,(Local DI->Remote DO).
P = 1 - Enable Auto Run(Start) DIO Synchronization operation when power-on.
S = 1 - digital output active state(=1) when DI value match DI mask pattern
TTTT = 0x012C (300ms)

response: !01(cr) ; valid

Related command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM4CRPHHHHLLLL (data), $AAYM6CRPSTTTT(data)

8.4.33 $AAYM6CRPSTTTT (data) Set *DI mismatch DO latch Mode* of DIO Sync.

Description:	Set DO to DI mismatch DO latch Mode for Automatic DIO Synchronization. A single digital output channel is activated (1 or 0) and latched, when DI input value mismatch DI mask pattern.	
Command:	$AAYM6CPSTTTT (data) [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YM6	command to set DI mismatch/DO latch mode of Auto DIO Sync.
	C	Mirrored DO channel number (hex 0~F).
	R	write DO to local module or remote module when DIO sysnc active. = 0 - Local Mode, (Local DI->Local DO) = 1 - remote Mode,(Local DI->Remote DO)
	P	Enable/Disable Auto Run(Start) DIO Synchronization operation when power-on. = 0 - Disiable = 1 - Enable
	S	digital output active state when DI data pattern match = 0 - digital output 0 when DI data pattern match = 1 - digital output 1 when DI data pattern match = 2 - toggle digital output
	TTTT	DI channel pre-debounce time (hex 0000~FFFF ms)
	(data)	represent DI mask pattern is used to indicate the monitored input channels(0~15) and mask state, (16 char), bits(15..0) - indicate DI channel(15..0) to be monitored and mask state, = '1' - indicate DI channel n is monitored and mask state is '1'. = '0' - indicate DI channel n is monitored and mask state is '0'. = 'X' - don't care (not be monitored)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Note: *Before starting Automatic DIO Synchronization function, you must set DO Mode to " DIO Sync. Mode " (Ref. $AACONNDD).*

Example: Assume DI channel 0,2,5 are monitored and the ASCII form of DI mask pattern is XXXXXXXXXX1XX0X1. When DI input ch(0)≠1 or ch(5)≠1 or ch(2)≠0(mismatch DI mask pattern), the corresponding DO(0) will be set to ON(1) and latched. DI pre-debounce time is 300 msec.

command: $01YM6011012CXXXXXXXXXX1XX0X1(cr)

where R = 1 - remote Mode,(Local DI->Remote DO).
P = 1 - enable Auto Run(Start) DIO Synchronization operation when power-on.
S = 1 - digital output active state(=1) when DI value mismatch DI mask pattern
TTTT = 0x012C (300ms)

response: !01(cr) ; valid

Related command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM1CPSHHHHLLLL (data), $AAYM2CPSTTTT(data)

8.4.34 $AAYMNDDDDDDDD Set DIO Sync. remote device IP for DO channel N

Description:	Set remote device IP for DO channel N (for DIO sync. mode)	
Command:	$AAYMNDDDDDDDD [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YM	command to set DI mismatch/DO latch mode of Auto DIO Sync.
	N	Mirrored DO channel number (hex 0~F).
	DDDDDDDD	IP number(hex). (Ex: IP 192.168.0.122 = C0A8007A) (IP= 0xFFFFFFFF(255.255.255.255) for all remote modules)
	CHK	Check sum
	(cr)	Carriage return
	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
Response:	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: Set DIO Sync. Remote device IP(0xC0A8007A/192.168.0.122) for DO channel 0.

command: $01YM0C0A8007A (cr)

response: !01(cr) ; valid

Related command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM5CRPHHHHLLLL (data), $AAYM5CRPSTTTT(data), $AAYM6CRPSTTTT(data)

8.4.35 $AAYM4C Read DIO Synchronization Mode parameters

Description:	This command is used to read parameters of DIO Synchronization Mode	
Command:	$AAYM4C [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YM4	command to read parameters of DIO Synchronization Mode.
	C	mirrored DO channel number (0~F).
	CHK	Check sum
	(cr)	Carriage return
Response:	!AARDPSTTTT(data)(cr)	For "*DI match/mismatch DO latch Mode*" Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	R	write DO to local module or remote module when DIO sysnc active. = 0 - Local Mode, (Local DI->Local DO) = 1 - remote Mode,(Local DI->Remote DO)
	D	Specific DO channel mode. = 0 - specific DO channel isn't mirrored to DI hannel(s). = 2 - DIO Synchronization- DI match DO latch Mode. = 3 - DIO Synchronization- DI mismatch DO latch Mode.
	P	Enable/Disable Auto Run(Start) DIO Synchronization operation when power-on. = 0 - Disiable = 1 - Enable
	S	digital output active state when DI data pattern match = 0 - digital output 0 when DI data pattern match = 1 - digital output 1 when DI data pattern match = 2 - toggle digital output
	TTTT	DI channels pre-debounce time(0000~FFFF/ms).
	(data)	This parameter is used to indicate the monitored input channels(15~0) (16 char). bits(15..0) - indicate DI channel(15..0) state to be monitored, bit n = '1' or '0' - indicate DI channel n(0~15) is monitored and mask state is 1 or 0 = 'X' - indicate the DI channel n(0~15) isn't monitored, Example: DI channel(0,1,2,4,5,7) are monitored, (data) = "XXXXXXXX0X10X011"
	CHK	Check sum
	(cr)	Carriage return

Example: Read parametersof DO(0) DIO sync Mode.

command: $01YM4(cr) ; Read DIO sync Mode parameters of DO(0).

response: !011100000A000AXXXXXXXXXX1XXXXXX (cr) ; valid

Related command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MS, $AAYM4CRPHHHHLLLL (data), $AAYM5CRPSTTTT(data), $AAYM6CRPSTTTT(data)

8.4.36 $AAYMRCS Start(Run)/Stop DIO Synchronization operation

Description:	This command is used to start (run)/stop the DIO Synchronization operation.	
Command:	$AAYMRCS [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YMR	represent this command is used to start/stop DIO Synchronization operation.
	C	mirrored DO channel number (0~F).
	S	Start/Stop DIO Synchronization operation. = 0 - Stop DIO Synchronization operation (default) = 1 - Start(Run) DIO Synchronization operation.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA [CHK](cr)	Valid command
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Note: *Before starting Automatic DIO Synchronization function, you must set DO Mode to " DIO Sync. Mode " (Ref. $AACONNDD).*

Example: Assume DI channel 0,2,5 are monitored and the ASCII form of DI mask pattern is XXXXXXXXXX1XX0X1. When DI input ch(0,5)=1 and ch(2)=0, the corresponding DO(0) will be set to ON(1) (*DO toggle mode*), otherwise DO(0) will be set to OFF(0). DI pre-debounce time is 300 msec and post-debounce time=150msec.

Step 1: set DO mode(DD=**04**) to " DIO Sync. Mode " for DO channel **0** (Ref. $AACONNDD)

command: $01CO0004(cr)

response: !01(cr) ; valid

Step 2: set DI channels(ch(0,5)=1,ch(2)=**0**) DI mask pattern for DO Toggle Mode.

command: $01YM1011012C0096XXXXXXXXXX1XX0X1(cr)

response: !01(cr) ; valid

Step 3: set the digital output channel to OFF. (ref. @AA6ONSS)

command: @016O0000(cr)

response: !01(cr) ; valid

Step 4: Start(Run) DO(0) DIO Sync. operarion

command: $01YMR01(cr)

response: !01(cr) ; valid

Related command: $AACONNDD, @AA6ONSS, $AAY6MC, $AAY6MS, $AAYM1CPSHHHHLLLL (data), $AAYM2CPSTTTT(data)

8.4.37 **$AAYMS** Read current DO activated status during DIO Sync. operation

Description:	The DO activity status bit is sets when th channel of output has occurred (for *DIO Sync. mode*). (Ref. to 12.7).	
Command:	$AAYMS [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YMS	command to read DO current state during DIO Sync. operation
	CHK	Check sum
	(cr)	Carriage return
Response:	!AADDDD (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DDDD	DO channels (15..0) status(1=active, 0=inactive) (After read the all of DO activity status are cleared)
	CHK	Check sum
	(cr)	Carriage return

Example: Read DO current status of DIO Synchronization(Ref. to 12.7).

command: $01YMS(cr)

response: !010005(cr) ; valid and DO(0,2) have been activated

Related command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAYM1CPSHHHHLLLL (data), $AAYM2CPSTTTT(data)

8.4.38 $AAYMPN Read remote device IP for DO channel N(for DIO sync. mode)

Description:	Read remote device IP for DO channel N(for DIO sync. mode) . (for firmware version 6.100 or later)	
Command:	$AAYMPN [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	YMP	command to Read remote device IP
	N	DO channel number (0~F)
	(cr)	Carriage return
Response:	!AADDDDDDDD (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DDDDDDDD	IP addressr(hex). (ex: C0A8007A = 192.168.0.122) (IP= 0xFFFFFFFF(255.255.255.255) for all remote modules)
	CHK	Check sum
	(cr)	Carriage return

Example: Read remote device IP for DO channel 0.

command: $01YMP0(cr)

response: !01C0A8005C (cr) ; IP addr.: C0A8005C = 192.168.0.92

Related command: $AAYMNDDDDDDDD

8.4.39 **#AA2NPPPPPPPP** Write DO pulse counts

Description:	Write DO pulse counts to the specific DO channel (For *DO Pulse Output mode*)	
Command:	#AA2NPPPPPPPP [CHK](cr)	
Syntax:	#	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	2	Represent generates DO pulse output command.
	N	Channel number(0~F)

	PPPPPPPP	Represents pulse counts (8 digits, decimal 0~5242879)
		if pppppppp = 00000000, continue DO pulse
		if pppppppp = 00000001, stop DO pulse

	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example:

- ($AACONNDD) Set DO channel(3) to Pluse output mode.
  command: $01CO0301(cr)
  response: !01(cr)

- ($AA9PNNLLLLHHHH) Set DO pulse Low/high output width of channel 3.
  command: $019P0301230456(cr)
  response: !01(cr)

- (#AA2NPPPPPPPP) The command force the DO channel 3 to output 9 pulses.
  command: #012300000009 (cr)
  response: !01(cr)

Related command: $AA9PNNLLLLHHHH, $AA9NN , $AA9DNNHHHHLLLL

8.4.40 **$AA9PNNLLLLHHHH** Set DO pluse Low/High width of channel N

Description:	Set DO Pluse Low /High output width of channel N (unit: 0.5ms) (For *DO Pulse Output mode*)	
Command:	$AA9PNNLLLLHHHH [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	9P	command to set DO pluse high/low width of channel N.
	NN	DO channel number. = 00~0F (hex) - DO channel number. = FF (hex) - Copy the setting to all DO channels
	LLLL	4 char, DO pulse low signal width (hex 0001~ 3332, uint: 0.5ms)
	HHHH	4 char, DO pulse high signal width (hex 0001~ 3332, uint: 0.5ms)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: Ref. command "#AA2NPPPPPPPP"

Related command: #AA2NPPPPPPPP, $AA9NN , $AA9DNNHHHHLLLL

8.4.41 $AA9NN Read DO pulse and DO High/Low delay width for channel N

Description:	Read DO pulse low/high width and DO High/Low delay output width of channel N (unit: 0.5ms). (For *DO Pulse Output mode* , *High/Low delay mode* and *DO Auto-Off Time Mode*)	
Command:	$AA9NN [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	9	command for Read a single digital input..
	NN	Channel number(00~0F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AALLLLHHHH UUUUDDDD (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	LLLL	4 char, DO pulse low signal width (hex, uint: 0.5ms) (For *DO Pulse Output mode*)
	HHHH	4 char, DO pulse high signal width (hex, uint: 0.5ms) (For *DO Pulse Output mode*)
	UUUU	4 char, DO low to high delay width (hex, uint 0.5ms) (For *High/Low delay mode* and *DO Auto-Off Time Mode*)
	DDDD	4 char, DO high to low delay width (hex, uint: 0.5ms) (For *High/Low delay mode* and *DO Auto-Off Time Mode*)
	CHK	Check sum
	(cr)	Carriage return

Example: Read DO pulse and Low/high delay output width of DO channel 3

```
command:   $01903(cr)
response:  !0101230456000A000A (cr)        ; LLLL=hex 0123,    HHHH=hex 0456,
                                           ; UUUU=hex 000A,    DDDD=hex 000A
```

Related command: #AA2NPPPPPPPP, $AA9DNNHHHHLLLL, $AA9DNNHHHHLLLL

8.4.42 **$AA9DNNHHHHLLLL** Set DO low/high delay time

Description:	Set DO Low/high delay output width for channel N (unit: 0.5ms). (For *High/Low delay mode* and *DO Auto-Off Time Mode*)	
Command:	$AA9DNNHHHHLLLL [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	9D	Command to set DO low/high delay time of DO channel N
	NN	DO channel number. = 00~0F (hex) - DO channel number. = FF (hex) - Copy the setting to all DO channels
	HHHH	4 char, DO low to high delay width (hex 0001~hex 3332, uint: 0.5ms) (For "High->Low->High Auto-Off Time mode" HHHH always "0001")
	LLLL	4 char, DO high to low delay width (hex 0001~hex 3332, uint: 0.5ms) (For "Low->High->Low Auto-Off Time mode" LLLL always "0001")
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example(1): For DO *Low to High delay mode*

➢ ($AACONNDD) Set DO channel(1) to *Low to High delay mode*.
command: $01CO0102(cr)
response: !01(cr)

➢ ($AA9DNNHHHHLLLL) Set DO Low to high delay time(2000ms) for channel 1.
command: $019D010FA00001(cr) ; HHHH=0FA0, LLLL=0001
response: !01(cr)

➢ (#AA1NDD) Set DO(1) to High(1).
command: #011101 (cr) ; after 2000ms the DO(1) to high(1)
response: !01(cr)

Example(2): For DO *High to Low delay mode*

➢ ($AACONNDD) Set DO channel(1) to *High to Low delay mode*.
command: $01CO0203(cr)
response: !01(cr)

➢ ($AA9DNNHHHHLLLL) Set DO Low to high delay time(2000ms) for channel 2.
command: $019D0200010FA0 (cr) ; HHHH=0001, LLLL=0FA0
response: !01(cr)

➢ (#AA1NDD) Set DO(2) to High(1).
command: #011200 (cr) ; after 2000ms the DO(2) to high(0)
response: !01(cr)

Example(3): For DO *Low to High to Low of DO Auto-Off Time Mode*

➢ ($AACONNDD) Set DO channel(0) to DO *Low to High to Low of DO Auto-Off Time Mode*.

command: $01CO0006(cr)
response: !01(cr)

➢ ($AA9DNNHHHHLLLL) Set DO High time(2000ms) for channel 0.

command: $019D000FA00001 (cr) ; HHHH=0FA0, LLLL=0001
response: !01(cr)

➢ (#AA1NDD) Set DO(0) to High(1).

command: #011001 (cr) ; after 2000ms the DO(0) to Low(0)
response: !01(cr)

Example(4): For DO *High to Low to High of DO Auto-Off Time Mode*

➢ ($AACONNDD) Set DO channel(0) to *DO High to Low to High of DO Auto-Off Time Mode.*

command: $01CO0007(cr)
response: !01(cr)

➢ ($AA9DNNHHHHLLLL) Set DO Low time(2000ms) for channel 0.

command: $019D0000010FA0 (cr) ; HHHH=001, LLLL=00FA0
response: !01(cr)

➢ (#AA1NDD) Set DO(0) to Low(0).

command: #011000 (cr) ; after 2000ms the DO(0) to high(1)
response: !01(cr)

Related command: $AA9NN

8.4.43 $AA0MCC Read DI counter filter (debounce time)

Description:	Read DI counter pre-debounce and post-debounce of channel N (unit = 0.5ms)	
Command:	$AA0MCC [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	0M	command for read DI counter filter of channel N
	CC	Represents DI channel number (00~0F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA(data1)(data2) (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	(data1)	DI counter pre-debounce (Min. Low width) time, 8-chars, (unit=0.5ms)
	(data2)	DI counter post-debounce(Min. High width) time, 8-chars, (unit=0.5ms)
	CHK	Check sum
	(cr)	Carriage return

Example: Read DI counter filter (debounce time) for channel 0,

command: $010M00(cr)

response: !010000000200000003(cr) ; (data1) = 00000002 - represents channel(0) Low signal width.
; (data2) = 00000003 - represents channel(0) High signal width.

Ref. command: $AA0MCC(data1)(data2), $AAECN

8.4.44 **$AA0MCC(data1)(data2)** Set DI counter debounce time

Description:	Set DI counter pre-debounce and post-debounce of channel N (unit = 0.5ms)	
Command:	$AA0MCC(data1)(data2) [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	0M	command for read DI counter filter of channel N
	CC	Represents DI channel number (00~0F)
	(data1)	DI counter pre-debounce (Min. Low width) time, 8-chars, (unit=0.5ms)
	(data2)	DI counter post-debounce(Min. High width) time, 8-chars, (unit=0.5ms)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: Set DI counter channel(0) pre-debounce time to 00000002(=1 ms) and post-debounce time to 00000003(=1.5 ms)

command: $010M000000000200000003(cr)
response: !01(cr)

Ref. command: $AA0MCC, $AAECN

8.4.45 **$AAECN** Start/Stop single DI counter

Description:	Start/Stop single digital input counter (Falling Edge Trigger)	
Command:	$AAECN [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	E	Represents enable/disable DI counter command
	C	Represents DI counter channel number (0~F)
	N	Represents Start/Stop option (=0 –Stop, =1 -Start)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example 1: Start DI(2) counter
command: $01E21(cr)
response: !01(cr)

Example 2: Stop DI(2) counter
command: $01E20(cr)
response: !01(cr)

Ref. command: $AA0MCC, $AAECN, $AA0MCC(data1)(data2)

8.4.46 $AACN Clear single DI counter value and overflow flag

Description:	Clear single digital input counter value with overflow flag	
Command:	$AACN [CHK](cr)	
Syntax:	$	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	C	Represents clear DI counter command
	N	Represents DI counter channel number (0~F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: Clear DI counter channel 2

command: $01C2(cr)

response: !01(cr)

Related command: $AA0MCC, $AAECN, $AA0MCC(data1)(data2), $AAECN

8.4.47 #AAN Read single DI counter value

Description:	Read single digital input counter value	
Command:	#AAN [CHK](cr)	
Syntax:	#	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	N	Represents DI channel number (0~F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA(data) (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	(data)	10-characters(decimal) represents counter value
	CHK	Check sum
	(cr)	Carriage return

Example: Read single digital input channel(2) counter value

command: #012(cr)

response: !010000000123(cr) ; 0000000123 represents counter value is 123

Related command: $AA0MCC, $AAECN, $AA0MCC(data1)(data2), $AAECN, $AACN

8.4.48 #AARN Read single DI counter value and overflow flag

Description:	Read single digital input counter value with overflow	
Command:	#AARN [CHK](cr)	
Syntax:	#	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	R	represent read single DI counter value with overflow command
	N	represents DI channel number (0~F)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AAR(data) (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	R	DI Counter Overflow. = 0 - No counter overflow has occurred. = 1 - A counter overflow has occurred.
	(data)	10-characters(decimal) represents counter value
	CHK	Check sum
	(cr)	Carriage return

Example 1: Read single digital input channel(5) counter value with overflow

command: #01R5(cr)

response: !0110000000123(cr) ; Represents counter value is 0000000123 and counter overflow(1) has occurred.

Example 2: Read single digital input channel(5) counter value with overflow

command: #01R5(cr)

response: !0100000000123(cr) ; Represents counter value is 0000000123 and No counter overflow(0) has occurred.

Related command: $AA0MCC, $AAECN, $AA0MCC(data1)(data2), $AAECN, $AACN, #AAN

8.4.49 ~** Send "Host OK" to all modules via broadcast

Description:	Host send this command via broadcast IP to tell all modules that host and network are alive (No reply from modules) When host watchdog timer is enable, host computer must send this command to all module before timeout otherwise "Host watchdog timer enabled" module will go to safety state.	
Command:	~** [CHK](cr)	
Syntax:	~	Command leading code
	**	command for Host OK
	CHK	Check sum
	(cr)	Carriage return
Response:	No response.	

Related command: ~AA**, ~AA0, ~AA1, ~AA2, ~AA4V, ~AA5V, ~AA3EVVV, ~AA3PPP , ~AA3P

8.4.50 **~AA**** Send Host OK to the specific module

Description:	Host send this command via broadcast IP to tell all modules that host and network are alive (No reply from modules) When host watchdog timer is enable, host computer must send this command to all module before timeout otherwise "Host watchdog timer enabled" module will go to safety state.	
Command:	~AA** [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	**	command for Host OK
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Ref. command: ~**, ~AA0, ~AA1, ~AA2, ~AA4V, ~AA5V, ~AA3EVVV, ~AA3PPP , ~AA3P

8.4.51 **~AA0** Read watchdog timeout status

Description:	Read watchdog timeout status	
Command:	~AA0 [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	0	command for reading watchdog timeout status
	CHK	Check sum
	(cr)	Carriage return
Response:	!AASS (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	SS	Two hexadecimal digits that represent the host watchdog status. bit(7) - Host watchdog enable/disable , = 0 - Disable. = 1 - Enable. bit(2) - Host watchdog timeout status, = 0 - indicates that no host watchdog timeout has occurred. = 1 - indicates that a host watchdog timeout has occurred. bit(6,5,4,3,1,0) - reserved(always 0)
	CHK	Check sum
	(cr)	Carriage return

Note: *The host watchdog status is stored in EEPROM and can only be reset by using the ~AA1 command.*

Example 1: Reads the host watchdog status of module 01 and returns 00, meaning that the host watchdog is disabled or no host watchdog timeout has occurred.

Command: ~010(cr)
Response: !0100(cr)

Example 2: Reads the host watchdog status of module 01 and returns 04, meaning that a host watchdog timeout has occurred.

Command: ~010(cr)
Response: !0104(cr)

Ref. command: ~**, ~AA1, ~AA2, ~AA4V, ~AA5V, ~AA3EVVV, ~AA3PPP , ~AA3P

8.4.52 **~AA1** Reset host watchdog timeout status

Description:	Reset host watchdog timeout status	
Command:	~AA1[CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	1	command for resetting watchdog timeout status
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example 1: Reads the host watchdog status of module 01 and shows that a host watchdog timeout has occurred.
command: ~010(cr)
response: !0104(cr)

Example 2: Resets the host watchdog timeout status of module 01 and returns a valid response.
command: ~011(cr)
Response: !01(cr)

Example 3: Reads the host watchdog status of module 01 and shows that no host watchdog timeout has occurred.
command: ~010(cr)
response: !0100(cr)

Ref. command: ~**, ~AA*, ~AA0, ~AA2, ~AA4V, ~AA5V, ~AA3EVVV, ~AA3PPP , ~AA3P

8.4.53 **~AA2** Read host communication Timeout value

Description:	Read host communication Timeout value	
Command:	~AA2 [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	2	command for reading watchdog timeout value
	CHK	Check sum
	(cr)	Carriage return
Response:	!AAEVVV (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	E	Host watchdog enabled status = 1 – Enable. = 0 – Disable.
	VVV	Timeout value in hex format from hex 001 to 28F .(unit 0.1 sec), FF denotes 25.5 seconds
	CHK	Check sum
	(cr)	Carriage return

Example: Reads the host watchdog timeout value of module 01 and returns FF, which denotes that the host watchdog is enabled and the host watchdog timeout value is 25.5 seconds.

```
command:   ~012(cr)
response:  !011FF(cr)
```

Ref. command: ~**, ~AA1, ~AA0, ~AA4V, ~AA5V, ~AA3EVVV, ~AA3PPP , ~AA3P

8.4.54 ~AA3EVVV Set Host watchdog timeout interval

Description:	Enable/disable Host watchdog and set timeout interval (unit = 0.1sec)	
Command:	~AA3EVVV [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID (01 to FF , Always 01)
	3	command for setting host wdt Enable/ disable and host wdt timeout value.
	E	Host watchdog enabled status = 1 – Enable. = 0 – Disable.
	VVV	Timeout value in hex format from hex 001 to 28F .(unit 0.1 sec), FF denotes 25.5 seconds
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Note:

*If host watchdog timer is enabled, the host should send Host OK (see "~**" or ~AA**) command periodically within Timeout value to refresh the timer, otherwise the module will be forced to safet state (see "~AA5V") and The Power-LED on the module will go to flash. After timeout the all of D/O commands are disabled.*

Example:

➢ (~AA3EVVV) Set module (ID=01) to have watchdog timeout value 20.0 seconds and enable host watchdog.

command: ~01310C8(cr) ; enable host watchdog

response: !01(cr)

➢ (~AA2) Read watchdog timeout value form module (ID=01). The module returns 10C8, which denotes that the host watchdog is enabled and the host watchdog timeout value is 20.0 seconds.

command: ~012(cr)

response: !0110C8(cr)

➢ (~** or ~AA**) Host send this command to all modules that host and network are alive

command: ~**(cr) ; Host OK (to all modules)

or

command: ~01**(cr) ; Host OK (to the Specific module)

➢ Stop sending any command string to modules for at least 20.0 seconds. The Power- LED on the module will go to flash. The flash LED indicates the host watchdog is timeout and timeout status flag is set.

➢ (~AA0) Read watchdog timeout status, the module returns 01, which denotes that a host watchdog timeout has occurred.

command: ~010(cr)

response: !0184(cr) ; bit(7)=1 - Host watchdog enabled
; bit(2)=1 - indicates that a host watchdog timeout has occurred.

- (~AA1) Reset watchdog timeout status. Watchdog timeout is cleared and LED stops flashing, and host watchdog is disabled

  command: ~011(cr)

  response: !01(cr)

- Reads the host watchdog status of module 01 and returns 00, meaning that the host watchdog is disabled and no host watchdog timeout has occurred.

  command: ~010(cr)

  response: !0180(cr) ; bit(7)=1 - Host watchdog enabled and Timeout status is cleared

- (~AA3EVVV) Set module (ID=01) to have watchdog timeout value 20.0 seconds and disable host watchdog.

  command: ~01300C8(cr) ; disable host watchdog

  response: !01(cr)

- Reads the host watchdog status of module 01 and returns 00, meaning that the host watchdog is disabled and no host watchdog timeout has occurred.

  command: ~010(cr)

  response: !0180(cr) ; bit(7)=1 - Host watchdog enabled and Timeout status is cleared

Ref. command: ~**, ~AA1, ~AA0, ~AA4V, ~AA5V, ~AA2, ~AA3PPP , ~AA3P

8.4.55 ~AA4V Read Power-on or Safe DO value of module

Description:	Read power-on or safe DO value of module	
Command:	~AA4V [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	4	command for read the power-on DO value or the safe DO value of a module
	V	Read power-on value or safe value = P - read power-on value, = S - read safe value.
	CHK	Check sum
	(cr)	Carriage return
Response:	!AADDDD (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	DDDD	powe-on or safe value.(0000~FFFF)
	CHK	Check sum
	(cr)	Carriage return

Example 1: Read Power on value and return power-on value 5A5A.

command: ~014P(cr)

response: !045A5A(cr)

Example 2: Read Power on value and return safe value AA00.

command: ~014S(cr)

response: !04AA00(cr)

Ref. command: ~**, ~AA1, ~AA0, ~AA4V, ~AA5V, ~AA2, ~AA3PPP , ~AA3P, ~AA3EVVV

8.4.56 **~AA5V** Sets the current DO value as power-on or safe value

Description:	Set the current DO value as power-on or safe value	
Command:	~AA5V [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	5	command for sets the current value as the power-on DO value or the safe DO value.
	V	Set power-on value or safe value = P - set power-on value, = S - set safe value.
	VVV	Timeout value in hex format from hex 001 to 28F .(unit 0.1 sec), FF denotes 25.5 seconds
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example 1: Set Safe value.

- Set module DO to output value 2A.
  Command: @012A(cr)
  Response: >(cr)

- Set current output value 2A as safe value.
  Command: ~015S(cr)
  Response: !01(cr)

- Read safe value and return safe value 2A.
  Command: ~014S(cr)
  Response: !01002A(cr)

Example 2: Set Power on value

- Set module to output value 15.
  Command: @0115(cr)
  Response: >(cr)

- Set current output value 15 as power-on value.
  Command: ~015P(cr)
  Response: !01(cr)

- Read Power on value and return power-on value 0015.
  Command: ~014P(cr)
  Response: !010015(cr)

Ref. command: ~**, ~AA1, ~AA0, ~AA4V, ~AA2, ~AA3PPP , ~AA3P, ~AA3EVVV

8.4.57 **~AA3PPP** Set module Power-on delay time

Description:	Set module Power-on delay time	
Command:	~AA3PPP [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	3	command for set module Power-on delay time
	PPP	Power-on delay time(unit: 0.1sec) to start communication timeout (000~FFF). **Note:** Total Power-on delay time = PPP + Normal Power-on delay time(about 7 sec)
	CHK	Check sum
	(cr)	Carriage return
Response:	!AA (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	CHK	Check sum
	(cr)	Carriage return

Example: Set power-on delay time to 096 (15 sec)

Command: ~013096(cr)

Response: !01(cr)

Ref. command: ~AA3PPP , ~AA3P, ~AA5V, ~AA4V

8.4.58 ~AA3P Read module Power-on delay time

Description:	Set module Power-on delay time	
Command:	~AA3P [CHK](cr)	
Syntax:	~	Command leading code
	AA	Module address ID(01 to FF , Always 01)
	3P	command for read module Power-on delay time
	CHK	Check sum
	(cr)	Carriage return
Response:	!AAPPP (cr)	Valid command.
	?AA[CHK](cr)	Invalid command
	!	Delimiter for valid command
	?	Delimiter for invalid command
	AA	Module address ID
	PPP	Power-on delay time(unit: 0.1sec) to start communication timeout. (range hex 000~FFF)
	CHK	Check sum
	(cr)	Carriage return

Example 1: Set power-on delay time to 096 (15 sec)
command: ~013096(cr)
response: !01(cr)

Example 2: Read power-on delay time and return 096 (15 sec)
command: ~013P(cr)
response: !01096(cr)

Ref. command: ~AA3PPP

Chapter 9 MODBUS/TCP Command structure

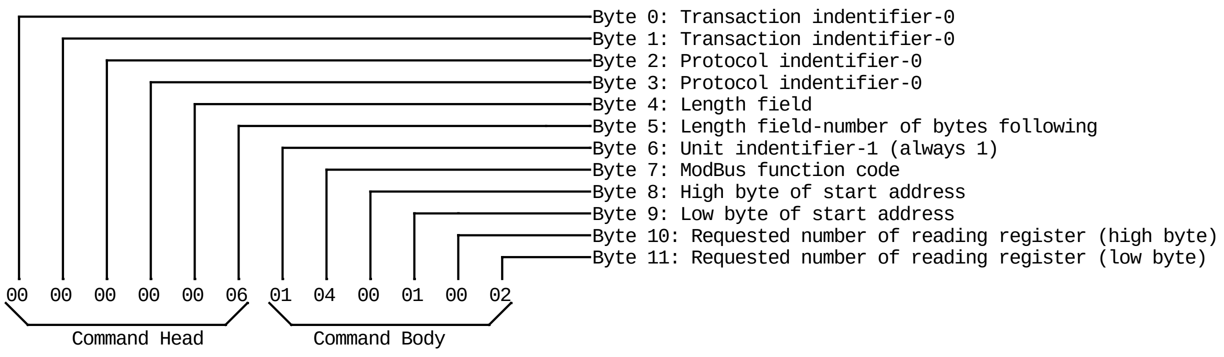
EDAM-4200 system accepts a command/response form with the host computer. When systems are not MODBUS/TCP Command structure. EDAM-4200 system accepts a command/response form with the host computer. When systems are not transmitting they are in listen mode. The host issues a command to a system with a specific address and waits a certain amount of time for the system to respond. If no response arrives, a time-out aborts the sequence and returns control to the host. This chapter explains the structure of the commands with Modbus/TCP protocol, and guides to use these command sets to implement user's programs.

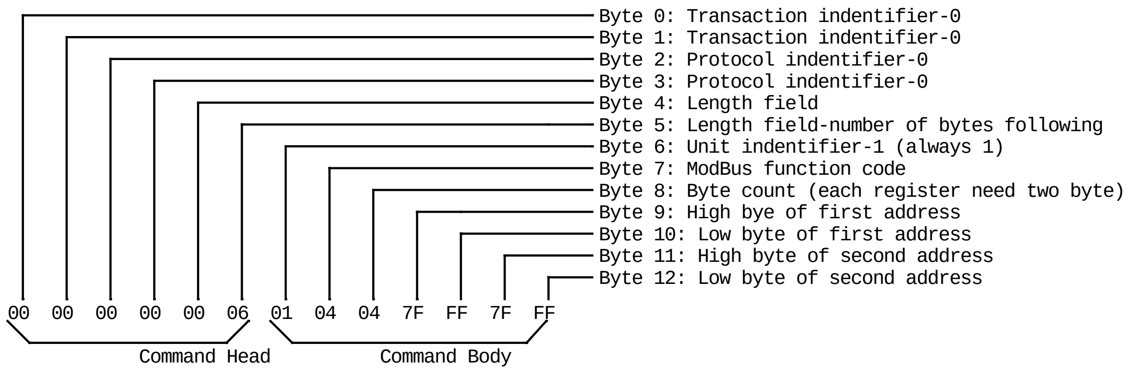
9.1 Command Structure

It is important to understand the encapsulation of a Modbus request or response carried on the Modbus/TCP network. A complete command is consisted of command head and command body. The command head is prefixed by six bytes and responded to pack Modbus format; the command body defines target device and requested action. Following example will help you to realize this structure quickly.

Example:

If you want to read digital input channels(0~11) of EADM-4050 (address: 00001~00012),
the Request command should be: 00 00 00 00 00 06 01 01 00 00 00 0C
and the Response should be: 00 00 00 00 00 05 01 01 02 00 00


Byte 0: Transaction indentifier-0
Byte 1: Transaction indentifier-0
Byte 2: Protocol indentifier-0
Byte 3: Protocol indentifier-0
Byte 4: Length field
Byte 5: Length field-number of bytes following
Byte 6: Unit indentifier-1 (always 1)
Byte 7: ModBus function code
Byte 8: High byte of start address
Byte 9: Low byte of start address
Byte 10: Requested number of reading register (high byte)
Byte 11: Requested number of reading register (low byte)
00 00 00 00 00 06 01 04 00 01 00 02
Command Head
Command Body


Byte 0: Transaction indentifier-0
Byte 1: Transaction indentifier-0
Byte 2: Protocol indentifier-0
Byte 3: Protocol indentifier-0
Byte 4: Length field
Byte 5: Length field-number of bytes following
Byte 6: Unit indentifier-1 (always 1)
Byte 7: ModBus function code
Byte 8: Byte count (each register need two byte)
Byte 9: High bye of first address
Byte 10: Low byte of first address
Byte 11: High byte of second address
Byte 12: Low byte of second address
00 00 00 00 00 06 01 04 04 7F FF 7F FF
Command Head
Command Body

Note: (Byte 6) Unit Indentifier 01 to FF(Always 01)

9.3 All Digital Input/Output Modules

All EDAM-4200 DIO modules use the same MODBUS address mapping

9.4 MODBUS/TCP address Mapping & Function Number

0xxxx	- Coils access,	(For 1, 5, 15 function code)
1xxxx	- Read discrete inputs,	(For 2 function code)
3xxxx	- Read input register,	(For 4 function code)
4xxxx	- Holding register access,	(For 3, 6, 16 function code)

xxxx - Element address of a data block, In the MODBUS data model each element within a data block is numbered from 1 to n.

Example:

00005	- Means Coils access and Starting address	= 0004 (0005-1)
10002	- Means Discrete inputs and Starting address	= 0001 (0002-1)
30257	- Means Input register and Starting address	= 0256 (0257-1)
40001	- Means Access holding register and Starting address	= 0000 (0001-1)

◆ MODBUS function code definition:

Function Code	Description
01 (0x01)	Read coils
02 (0x02)	Read Discrete Inputs
03 (0x03)	Read multiple Holding registers
04 (0x04)	Read multiple input registers
05 (0x05)	Write single coil
06 (0x06)	Write single register
15 (0x0F)	write Multiple coils
16 (0x10)	Write Multiple register
70 (0x46)	Read / write module settings

◆ Error Response:

If the function specified in the message is not supported, then the module Response as follows:

Offset	Function	Length	Description
00	Address	1 Byte	1 to 247
01	Function code	1 Byte	Function code | 0x80
02	Exception code	1 Byte	= 0x01 - invalid function code. = 0x02 - invalid data address. = 0x03 - invalid data value. = 0x04 - host WDT timeout = 0x05 – The Emergency input channel was activated.

9.5 Table of command sets

General command sets			
Address	Item	Func.	Attrib.
00272	Reload the module factory default = 0xFF00(or 1) - enable.	5,15	W
00273	Read module reset status = 1 - first read after powered on. = 0 - not the first read after powered on.	1	R
02210	Reset the module to initial power-on status = 0xFF00(1) - enable.	5,15	W
40481	Read Firmware version (version-1,version-2)	3	R
40483~40484	Read module name(name-1, name-2)	3	R

Digital Output command sets			
Address	Item	Func..	Attrib.
41453~41468	DO mode setting: (16 Channels) = 0000 - Direct DO output, (default) = 0001 - Pulse output mode, = 0002 - Low to high delay mode, = 0003 - High to low delay mode = 0004 - DIO Synchronization Mode = 0006 - DO "*Low to High to Low*" for "Auto-Off Time" Mode. = 0007 - DO "*High to Low to High* " for "Auto-Off Time" Mode.	3,6,16	R/W
00017~00032	Digital output (16 Channels, 1-bit /channel)	1,5,15	R/W
45609	Power-on Digital output value for DO0~DO15, (16 Channels)	3,6,16	R/W

DO Pulse Output command sets			
Address	Item	Func.	Attrib.
41065~41080	DO pulse output L level time (1 ~13107 , Unit: 0.5ms), (16 channels, 16-bit/ch)	3,6,16	R/W
41081~41096	DO pulse output H level time (1 ~13107, Unit: 0.5ms), (16 channels, 16-bit/ch)	3,6,16	R/W
41097~41138	DO pulse output count value (00000000~4FFFFFFF), (16 Channels, 32-bits/ch.) Example: ➢ (41097) for DO0(bit 15~0) & (41098) for DO0(bit 31~16), ➢ (41099) for DO1(bit 15~0) & (41100) for DO1(bit 31~16),	3,16	R/W
41139~41154	Start/stop DO Pulse output (16 channels, 16-bit/channel). = 0 - continue, = 1 - stop, = 2 - start	3,6,16	R/W

DIO Synchronization mode command sets			
Address	Item	Func.	Attrib.
41517~41532	Start(run) / Stop the DIO Synchronization operation. (16 DO channels, 16 bits/ch.) = 0 - Stop DIO Sync. = 1 - Start DIO Sync.	*3,6,16*	*R/W*
41547	Read DO(0~15) status in DIO Synchronization operation. The bit is set when output is activated. (bit/channel.) = 0 - output is inactivated. = 1 - output is activated. **Note:** After read the status will be cleared.	3	R

41548~41563	Monitored state of DI(0~15) in <u>*DO(x) Sync operation*</u> , *(16 DO channels, 16 bits/channel.)* = 0 - when DI(n) is inactivated = 1 - when DI(n) is activated Example: ➢ (41548) for Monitored state of DI(0~15) in DO(0) Sync operation , ➢ (41549) for Monitored state of DI(0~15) in DO(1) Sync operation ,	3,6,16	R/W
41564~41579	DI mask pattern in DIO Sync operation , (16 DO channels, 32 bits/channel) Example: (41564) for DI(0~15) mask pattern in DO(0) Sync operation. bit = '1' or '0' - indicate DI channel state to be monitor(16 bits/channel). (41565) for DI(0~15) mask pattern in DO(1) Sync operation(16 bits/channel). = 1 - Enable DI(n) to be monitored = 0 - Don't care.	3,6,16	R/W
41580~41595	DIO SYNC mode, DO(x) active output state and Enable/disable auto run when power-on for DO channel N. (16 DO channels, 16 bits/channel.) bit(1.0): DIO Sync. operation mode =02 - DI match DO latch mode =03 - DI mismatch DO latch mode. bit(2) : Enable/disable DIO Synchronization operation when power-on. = 0 - Disable. = 1 - Enable. bit(3) : digital output state when DI input value match(or not) DI mask pattern =0 - Inactive state =1 - Active state =2 – Toggle bit(4) : write DO to local module or remote module when DIO sysnc active. = 0 - Local Mode, (Local DI->Local DO) = 1 - remote Mode,(Local DI->Remote DO) bit(15~5): Don't care	3,6,16	R/W
41596~41611	DI <u>*Pre-debounce time*</u> of DIO SYNC . DI match /mismatch DO toggle mode. (16 channels, 16 bits/channel.) ✓ DI Pre-debounce time when DI value match DI mask pattern: =0x 0000 ~ 0xFFFF ms.	3,6,16	R/W
41612~41627	DI Post-debounce time of DIO SYNC . DI match /mismatch DO toggle mode. (16 channels, 16 bits/channel.) ✓ DI Post-debounce time when DI value match DI mask pattern: =0x 0000 ~ 0xFFFF ms.	3,6,16	R/W
41628~41643	DI Pre-debounce time of DIO SYNC . DI match /mismatch DO lacth mode. (16 channels, 16 bits/channel.) ✓ DI pre-debounce time when DI value match/mismatch DI mask pattern. =0x 0000 ~ 0xFFFF ms.	3,6,16	R/W

Digital output channel delay output mode command sets

Address	Item	Func.	Attrib.
41644~41659	Set DO channel Low to High delay output time (0x0001~0x3332, unit: **0.5ms)**. (16 DO channels, <u>16 Bits/channel</u>).	3,6,16	R/W
41660~41675	Set DO channel High to Low delay output time (0x0001~0x3332, unit: **0.5ms)**. (16 DO channels, <u>16 Bits/channel</u>).	3,6,16	R/W

Digital output *Auto-Off Time Mode* command sets

Address	Item	Func.	Attrib.
41676~41691	DO "*Low to High to Low mode*" High output delay time (0x1~0x3332 , unit: 0.5ms). (16 DO channels, 16 Bits/channel).	3,6,16	R/W
41692~41707	DO "*High to Low to High Mode*" Low output delay time (0x1~0x3332 , unit: 0.5ms). (16 DO channels, 16 Bits/channel).	3,6,16	R/W

Digital Input command sets

Address	Item	Func.	Attrib.
41485~41500	DI mode setting: (16 Channels) = 0000 - Direct DI input, (default) = 0001 - Counter Mode, = 0002 - Low to high latch = 0003 - High to low latch = 0004 - Input frequency mode(0.3 ~1000 Hz max)	3,6,16	R/W
00001~00016	Read digital Input status (16 Channels, 1-bit/channel)	1	R
00101~00116	Read DI latch status(after read the latch status will be cleared). Example: Read ch0~7 Latch Status request: 01 01 00 64 00 08 response: 01 01 01 07 ; ch0,1,2 are latched **Note:** After read the latch status will be cleared	1	R

Digital Input Counter command sets

Address	Item	Func.	Attrib.
00117~00132	Start/Stop DI Counter (0xFF00 = Start , 0x0000 = Stop)	1,5,15	R/W
00133~00148	Clear DI Counter (0xFF00= Clear), (16 Channels, 1-bit/channel)	5,15	W
00225~00240	Read counter overflow status (16 Channels, 1-bit/channel), = 1 - Overflow has occurred, = 0 - No overflow has occurred.	1	R
41001~41032	Read DI counter value (16 Channels, 32-bit/channel). Example: (41001) for DI0 (bit 15~0), (41002) for DI0 (bit 31~16). (41003) for DI1 (bit 15~0), (41004) for DI1 (bit 31~16).	3	R

WatchDog command sets

Address	Item	Func.	Attrib.
45678	Informs all modules that the host is OK. (No reply to modbus response)	3	R
45601	Host communication timeout value (dec. 1~655, unit: 0.1sec). (ref. (45604) Host watchdog timeout status)	3,6,16	R/W
45602	Host wdt timeout Safe DO(0~15) value. Note: After timeout the all of digital output commands are disabled.	3,6,16	R/W
45604	Host watchdog timeout status: ➢ 0xFF00 = host wdt timeout bit is set, ➢ Write (0xFF00) to clear host watchdog timeout.	3,6,16	R/W
45605	host wdt Enable(=0xFF00) and Disable(=0x0000), (Clear host wdt timeout status(45604) before disable host wdt) (ref. Ascii command: "~AA3EVVV")	3,6,16	R/W

9.6 Example of Modbus/TCP commands

- (00272) Reload the module factory default.

  request: 01 05 01 0F FF 00

  response: 01 05 01 0F FF 00 ; response: successful

- (02210) Reset the module to initial power-on status and return successful
  - (00273) Read reset status:

    request: 01 01 01 10 00 01

    response: 01 01 01 01 ; the module is been reseted,
  - (00273) read reset status:

    resquest: 01 01 01 10 00 01

    response: 01 01 01 00 ; the module is not been reseted,
  - (02210) Reset(reboot) the module to initial power-on state

    request : 01 05 08 A1 FF 00

    response: no respomse
  - (00273) Read reset status:

    request: 01 01 01 10 00 01

    response: 01 01 01 01 ; the module is been reseted,

- (40481) Read Firmware version.

  request: 01 03 01 E0 00 01

  response: 01 03 02 06 08 ; response: 06 08 (version: 06.08)

- (40483~40484) Read module name(name-1, name-2).

  request: 01 03 01 E2 00 02

  response: 01 03 04 00 42 50 00 ; response: module name(4250)

- (41453~41468) DO mode setting:
  - (41453) set DO(0) to *Automatic DIO Synchronization Mode*:

    request: 01 06 05 AC 00 04

    response: 01 06 05 AC 00 04 ; response: successful,
  - (41454) set DO(1) to Direct DO output mode:

    request: 01 06 05 AD 00 00

    response: 01 06 05 AD 00 00 ; response: successful,
  - (41458) set DO(5) to DO Auto-Off Time Mode for DO "Low to High to Low":

    request: 01 06 05 B1 00 06

    response: 01 06 05 B1 00 06 ; response: successful,

- (41453) read mode setting for channel 0~5:

  resquest: 01 03 05 AC 00 06

  response: 01 03 0C 00 04 00 00 00 04 00 03 00 02 00 06 ;DO0~5(04,00,04,03,03,06)

◆ (00017~00032) Digital output (16 Channels, 1-bit /channel):

- (41453) set DO(0~7) to *Direct DO output mode*
- (00017) write digital output DO(0,2,5) to ON and DO(1,3,4,6,7) to OFF:

  request: 01 0F 00 10 00 08 01 25

  response: 01 0F 00 10 00 08 ; response: successful,
- (00017) Read digital output channels from DO0~DO7:

  request: 01 01 00 10 00 08

  response: 01 01 01 25 ; DO value 0x25.
- (00017) write DO(1) to ON:

  request: 01 05 00 11 FF 00

  response: 01 05 00 11 FF 00 ; response: successful,
- (00017) Readback Digital output for DO1:

  request: 01 01 00 11 00 01

  response: 01 01 01 01 ; DO1 ON,
- (00017) write DO(0) to OFF and return successful.:

  request: 01 05 00 10 00 00

  response: 01 05 00 10 00 00 ; response: successful,
- (00017) Readback Digital output for DO0:

  request: 01 01 00 10 00 01

  response: 01 01 01 00 ; DO0 OFF,

◆ (45609) set power-on digital output value (16 Channels, 1-bit /channel):

- (45609) set power-on DO(0,2,1) value to ON

  request: 01 06 15 E8 00 25

  response: 01 06 15 E8 00 25 ; response: successful,
- (45609) read power-on DO(0~15) value

  request: 01 03 15 E8 00 01

  response: 01 03 02 00 25 ; power-on value DO(0,2,1) ON,

◆ Pulse Output command sets:

- (41453) set DO(0) to *Pulse output mode*

  request: 01 06 05 AC 00 01

  response: 01 06 05 AC 00 01 ; response: successful,
- (41065) write DO(0) pulse output L level value to 500ms

  request: 01 06 04 28 03 E8 ; unit=0.5ms

  response: 01 06 04 28 03 E8

- (41081) write DO(0) pulse output H level value to 500ms

  request: 01 06 04 38 03 E8 ; unit=0.5ms

  response: 01 06 04 38 03 E8 ; response: successful,

- (41097~41098) set DO(0) pulse output count value to 0x13121110 (32-bits/channel)

  request: 01 10 04 48 00 02 04 11 10 13 12

  response: 01 10 04 48 00 20 ; response: successful,

- (41097~41098) read DO(0) pulse output count value

  request: 01 03 04 48 00 02

  response: 01 03 04 11 10 13 12 ; response: successful,

- (41139) Start DO(0) Pulse output

  request: 01 06 04 72 00 02

  response: 01 06 04 72 00 02 ; response: successful,

  Wait seconds..............

- (41097~41098) read DO(0) pulse output count value

  request: 01 03 04 48 00 02

  response: 01 03 04 11 10 13 12 ; response: successful,

- (41139) Stop DO(0) Pulse output

  request: 01 06 04 72 00 01

  response: 01 06 04 72 00 01 ; response: successful,

◆ For *DIO Synchronization mode*:

- Ref. Appendix 13.7 " DIO Synchronization (Mirror Local DI to DO)"

◆ For Digital output channel delay output mode:

- (41453) set DO(0) to *High to Low delay mode* and DO(1) to *Low to High delay mode*

  request: 01 10 05 AC 00 02 04 00 03 00 02

  response: 01 10 05 AC 00 02 ; response: successful,

- (41644) Set DO(0) *high to low* delay output time (=4000ms).

  request: 01 06 06 6B 1F 40 (unit: 0.5ms)

  response: 01 06 06 6B 1F 40

- (41644) read DO(0) channel *high to low* delay output time (unit: 0.5ms).

  request: 01 03 06 6B 00 01

  response: 01 03 04 1F 40

- (41645) Set DO(1) *Low to High* delay output time (=3000ms).

  request: 01 06 06 6C 17 70 (unit: 0.5ms)

  response: 01 06 06 6C 17 70

- (41645) read DO(1) channel *Low to High* delay output time (unit: 0.5ms)

  request: 01 03 06 6C 00 01

  response: 01 03 02 17 70

- (00017) write digital output DO(0) to active(ON) and DO(1) to inactive(OFF).

  request: 01 0F 00 10 00 02 01 01

  response: 01 0F 00 10 00 02

  wait 4 sec…….., the DO(0) will be activated(ON) and the DO(1) to inactivated(OFF)
- (00017) write digital output DO(0) to inactive(OFF) and DO(1) to active(ON).

  request: 01 0F 00 10 00 02 01 02

  response: 01 0F 00 10 00 02

◆ DO channel "<u>*Low to High to Low*</u>" output for "<u>*Auto-Off Time Mode*</u>":

- (41455~41456) Set DO(2,3)to DO "Low to High to Low" for "Auto-Off Time" Mode.

  request: 01 10 05 AE 00 02 04 00 06 00 06

  response: 01 10 05 AE 00 02 ; response: successful,
- (41678) Set DO(2) output high delay timefor L->H->L (unit=0.5ms)

  request: 01 06 06 8D 0F A1 ; DO(2)=0x0FA1(2000.5)ms

  response: 01 06 06 8D 0F A1
- (41679) Set DO(3) output high delay timefor L->H->L (unit=0.5ms).

  request: 01 06 06 8E 07 D1

  response: 01 06 06 8E 07 D1
- (41678~41679) read DO(2,3) output high delay time for L->H->L (unit=0.5ms)

  request: 01 03 06 8D 00 02

  response: 01 03 04 0F A1 07 D1
- (00017) Write digital output DO(2,3) to active(ON).

  request: 01 0F 00 12 00 02 01 03

  response: 01 0F 00 12 00 02

wait 2 sec*…….., the DO(2,3) will be inactivated(OFF)*

◆ DO channel " <u>*High to Low to Hig*</u>" output for "<u>*Auto-Off Time Mode*</u>":

- (41453~41454) set DO(0,1)to DO "High to Low to High" for "Auto-Off Time" Mode.

  request: 01 10 05 AC 00 02 04 00 07 00 07

  response: 01 10 05 AC 00 02 ; response: successful,
- (41691~41692) set DO(0,1) output low delay timefor H->L->H (unit=0.5ms) ,

  request: 01 10 06 9B 00 02 04 0F A0 07 D0 ; DO(0)=0x0FA0(2000)ms, DO(1)=0x07D0(1000)ms

  response: 01 10 06 9B 00 02
- (41691~41692) read DO(0,1) output low delay timefor H->L->H (unit=0.5ms) ,

  request: 01 03 06 9B 00 02

  response: 01 03 04 0F A0 07 D0
- (00017) write digital output DO(0,1) to inactive(OFF)

  request: 01 0F 00 10 00 02 01 00

  response: 01 0F 00 10 00 02

wait 2 sec*…….., the DO(0,1) will be activated(ON)*

- For DI Counter Mode:
  - (41485) Set DI(0) to Counter Mode.

    request: 01 06 05 CC 00 01

    response: 01 06 05 CC 00 01 ; response: successful,
  - (00133) Clear DI(0) Counter Register.

    request: 01 05 00 84 FF 00

    response: 01 05 00 84 FF 00
  - (00117) Star DI(0) Counter.

    request: 01 05 00 74 FF 00

    response: 01 05 00 74 FF 00

wait for DI(0) input pulse........

  - (41001~41002) Read DI(0) counter value (32-bit/channel)..

    request: 01 03 03 E8 00 02

    response: 01 03 04 00 0A 00 00 ; response: DI(0) counter value = 0x0000000A,
  - (00225~00240) Read DI(0) counter overflow status(1-bit/channel).

    request: 01 01 00 E0 00 08

    response: 01 01 01 00 ; response: No overflow has occurred.

- Host watchdog timer:
  - (45601) Set host communication timeout value to 30sec.

    request: 01 06 15 E0 01 2C ; set timeout value = 30sec (unit=0.1s)

    response: 01 06 15 E0 01 2C ; response: successful,
  - (45602) Set host wdt timeout Safe DO(0~15) value.

    request: 01 06 15 E1 00 13 ; set Safe DO(0,1,4) to active(1)

    response: 01 06 15 E1 00 13
  - (45604) Clear host watchdog timeout status.

    request: 01 06 15 E3 FF 00 ; Write (0xFF00) to clear host watchdog timeout.

    response: 01 06 15 E3 FF 00
  - (45605) Enable host wdt.

    request: 01 06 15 E4 FF 00 ; set host wdt Enable(x0FF00)

    response: 01 06 15 E4 FF 00

wait 8 sec.......

  - (45678) Informs all modules that the host is OK.

    request: 01 06 16 2D 00 64 ; Informs all modules that the host is OK

    response: ; no response

wait 10 sec for host wdt timeout....................

- (45604) Read Host watchdog timeout status (0xFF00 = host wdt timeout bit is set).

  request: 01 03 15 E3 00 01

  response: 01 03 02 FF 00 ; response: 0xFF00(the host wdt timeout bit is set),

- (45604) Write (0xFF00) to clear host watchdog timeout..

  request: 01 06 15 E3 FF 00

  response: 01 06 15 E3 FF 00 ; response: No overflow has occurred.

- (45605) Disable host wdt (clear host wdt timeout status before disable host wdt) .

  request: 01 06 15 E4 00 00 ; set host wdt disable(x00000)

  response: 01 06 15 E4 00 00

Chapter 10 TCPDAQ Data Structure

10.1 Typedef struct _AlarmInfo

```
typedef   struct _AlarmInfo              //Alarm Event data structure
{
     u_cha         szIP[4];              //The IP address which cause the alarm change
     u_short       szDateTime[6];        //E.x[ 2001]/[09/][23][10]:[12]:[34]

   //      (Year/Month/Day Hour:Minute:Second)
     u_short       byChannel;            //The Channel of which cause the alarm change
     u_short       byAlarmType;          //0x00:AIO Low Alarm
                                         //0x01:AIO High Alarm
                                         //0x20:DIO Alarm
                                         //0xF0:Connection Alarm
     u_short       byAlarmStatus;        //0:Alarm ON to OFF, 1:Alarm OFF to ON
     u_short       wValue;               //Alarm value.For DIO, this value could be "0" or "1" means that "ON" or
                                               //"OFF"
                                         //For high or low alarm, this is the AIO value.
                                         //For connection lost, this value is '0'.
} _AlarmInfo;
```

10.2 Typedef struct _StreamData

```
Typedef   struct _StreamData             //Stream Event data structure
{
     u_char        szIP[4];              //The IP address which send the stream datae
     u_short       szDateTime[6];        //E.x [2001]/[09]/[23] [10]:[12]:[34]
                                         //      (Year/Month/Day Hour:Minute:Second)
     u_short       DIN;                  //Digital input data (DI#0~DI#15)
     u_short       DOUT;                 //Digital output data (DO#0~DO#15)
     u_short       wData[32];            //Digital input Counter (Each channel occupies 4 Byte)
} _StreamData;
```

10.3 Typedef struct ModuleInfo

```
typedef   struct ModuleInfo              // Used For Scan_Online_Modules(..)
{     u_char       szIP[4];              //IP address
     u_char        szGate[4];            //Gateway
     u_char        szMask[4];            //Submask
     u_char        szDHCP;               //DHCP status 01=enable, 00=disable
     u_char        szID;                 //Module ID number
     u_char        szMacAddr[6];         //MAC address of module
     u_short       szModuleNo;           //Module name
     u_char        szBuffer[12];         //Buffer reserved for TCPDAQ.DLL
} ModuleInfo;
```

10.4 Typedef struct ModuleData

```
typedef   struct ModuleData              //Used for function TCP_ReadAllDataFromModule (..)
{  u_char     Din[16];                   //Digital input data (DI#0~DI#15),avaliable for EDAM9050/51/52
   u_char          Dout[16];             //Digital output data (DO#0~DO#15),avaliable for
                                                    //EDAM9050/51/52/17/19
```

```
    u_char      DiLatch[16];          //Digital input latch status (DI#0~DI#15),avaliable for EDAM9050/51/52
    long        DiCounter[16];        //Digital input counter value (DI#0~DI#15),avaliable for EDAM9050/51/52
    double      AiNormalValue[16];    //Analog Input value(AI#0~AI#15),avaliable for EDAM9015/17/19
    double      AiMaxValue[16];       //Analog maximum value(AI#0~AI#15),avaliable for EDAM9015/17/19
    double      AiMinValue[16];       //Analog minimum value(AI#0~AI#15),avaliable for EDAM9015/17/19
    u_char      AiHighAlarm[16];      //Analog high alarm status(AI#0~AI#15),avaliable for EDAM9015/17/19
    u_char      AiLowAlarm[16];       //Analog low alarm status(AI#0~AI#15),avaliable for EDAM9015/17/19
    u_char      AiChannelType[16];    //Analog channel Type, avaliable for EDAM9015/17/19
    u_char      AiBurnOut[16] ;       //Analog channel burn out status,avaliable for EDAM9019/15 only
    double      CJCTemperature ;      //Cold junction temperature,avaliable for EDAM9019 only
} ModuleData;
```

Chapter 11 EDAM-9000/4200 Web Server

11.1 What is TCPDAQ Web Server?

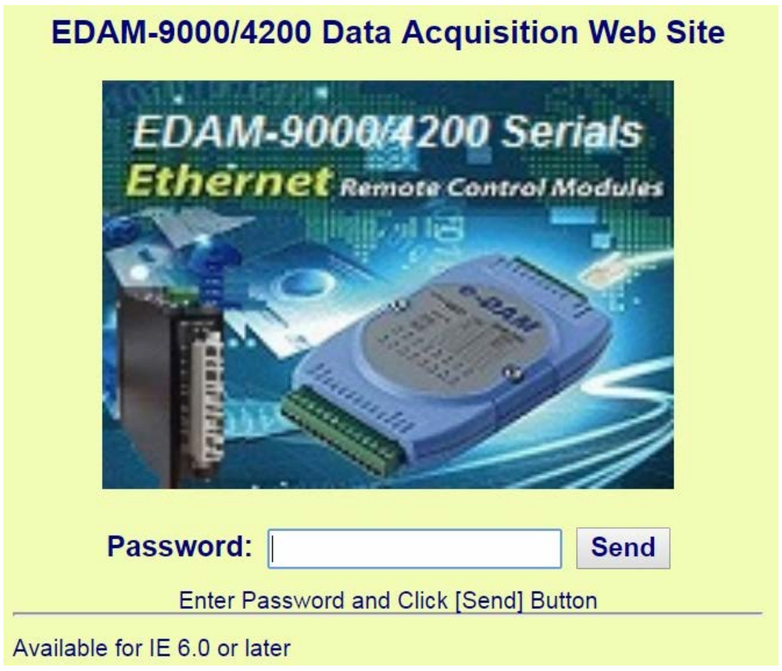
EDAM-9000/4200 I/O modules all features built-in web server. Remote computer or devices can monitor and control I/O status on EDAM-9000/4200 modules remotely through web browser. There is default built-in web page on EDAM-9000/4200 modules.

To use your computer to browse the web page on EDAM-9000/4200 module, you can simply type the IP address to connect to your EDAM-9000/4200 module in web browser. There will be one dialog window asking you to enter the password. After you have typed the correct password, you can start to monitor or control I/O on EDAM-9000/4200 modules.

Notice: Please use Windows Internet Explorer 5.5 (IE 5.5 or later version)

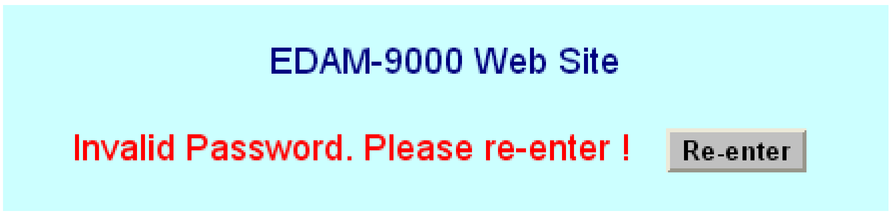
11.2 Home Page

- ♦ Type the **IP address** in the web browser (example: http:\\192.168.0.51)
- ♦ The home page will pop-up in the browser window to ask you to enter the password


EDAM-9000/4200 Data Acquisition Web Site

Enter Password and Click [Send] Button
Available for IE 6.0 or later

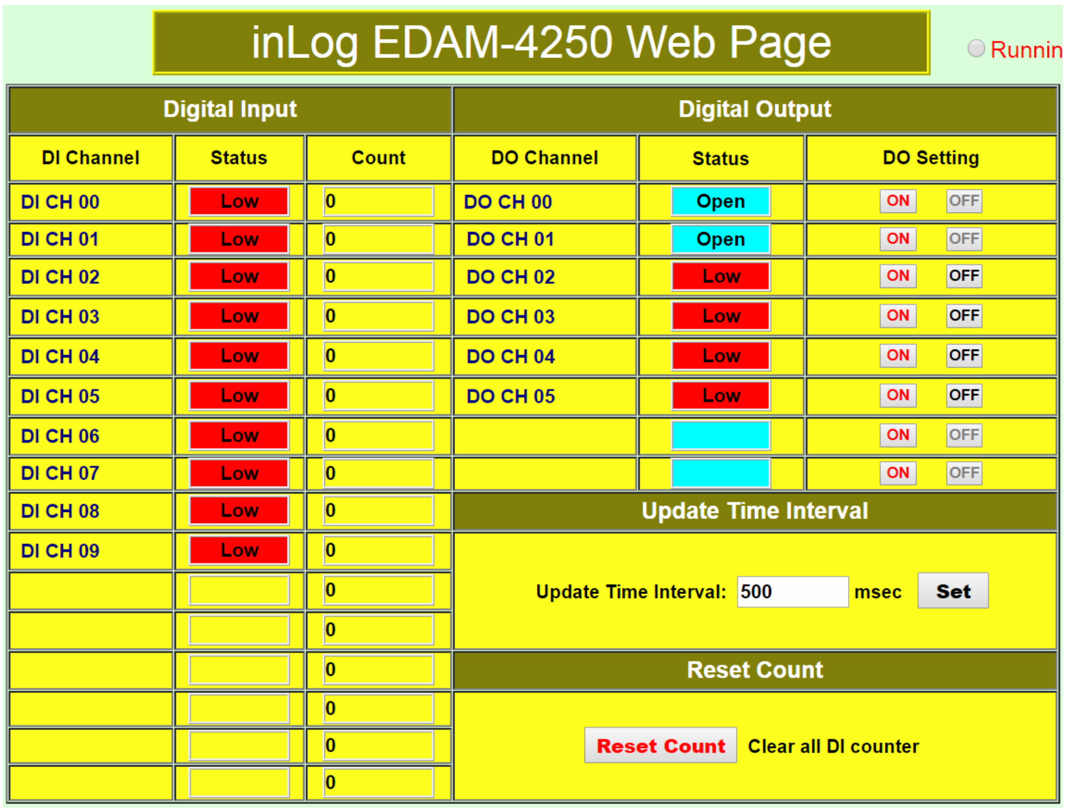
- ♦ Enter the correct password and click send button to verify the password. If the password is not correct, a warming message box will show up to remain you to reenter the password


EDAM-9000 Web Site
Invalid Password. Please re-enter !

- ♦ If the password is correct, the module monitoring page will pop up in the web browser.

11.3 Module monitoring page

11.3.1 EDAM-4250 monitoring page



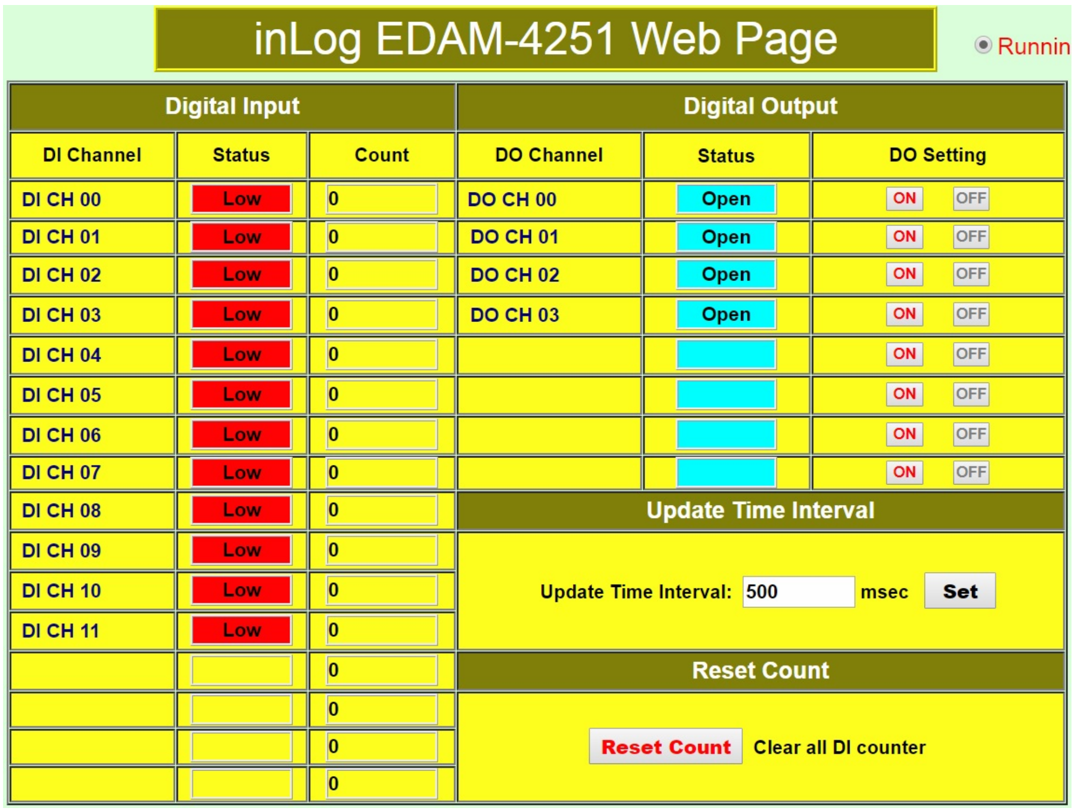
inLog EDAM-4250 Web Page

Runnin

Digital Input			Digital Output		
DI Channel	Status	Count	DO Channel	Status	DO Setting
DI CH 00	Low	0	DO CH 00	Open	ON OFF
DI CH 01	Low	0	DO CH 01	Open	ON OFF
DI CH 02	Low	0	DO CH 02	Low	ON OFF
DI CH 03	Low	0	DO CH 03	Low	ON OFF
DI CH 04	Low	0	DO CH 04	Low	ON OFF
DI CH 05	Low	0	DO CH 05	Low	ON OFF
DI CH 06	Low	0			ON OFF
DI CH 07	Low	0			ON OFF
DI CH 08	Low	0	Update Time Interval		
DI CH 09	Low	0	Update Time Interval: 500 msec Set		
		0			
		0			
		0	Reset Count		
		0	Reset Count Clear all DI counter		
		0			
		0			

Channel	: Channel number of digital input or output
Status	: Current input or output status
Count/Latch	: Counter value or latch status of digital input which functions at "Counter" or "Latch" mode
Mode	: Channel operating mode
DO Setting	: Set digital output on or off
Time interval	: I/O status update time interval

11.3.2 EDAM-4251 monitoring page



inLog EDAM-4251 Web Page

◉ Runnin

Digital Input			Digital Output		
DI Channel	Status	Count	DO Channel	Status	DO Setting
DI CH 00	Low	0	DO CH 00	Open	ON OFF
DI CH 01	Low	0	DO CH 01	Open	ON OFF
DI CH 02	Low	0	DO CH 02	Open	ON OFF
DI CH 03	Low	0	DO CH 03	Open	ON OFF
DI CH 04	Low	0			ON OFF
DI CH 05	Low	0			ON OFF
DI CH 06	Low	0			ON OFF
DI CH 07	Low	0			ON OFF
DI CH 08	Low	0			
DI CH 09	Low	0			
DI CH 10	Low	0			
DI CH 11	Low	0			
		0			
		0			
		0			
		0			

Update Time Interval

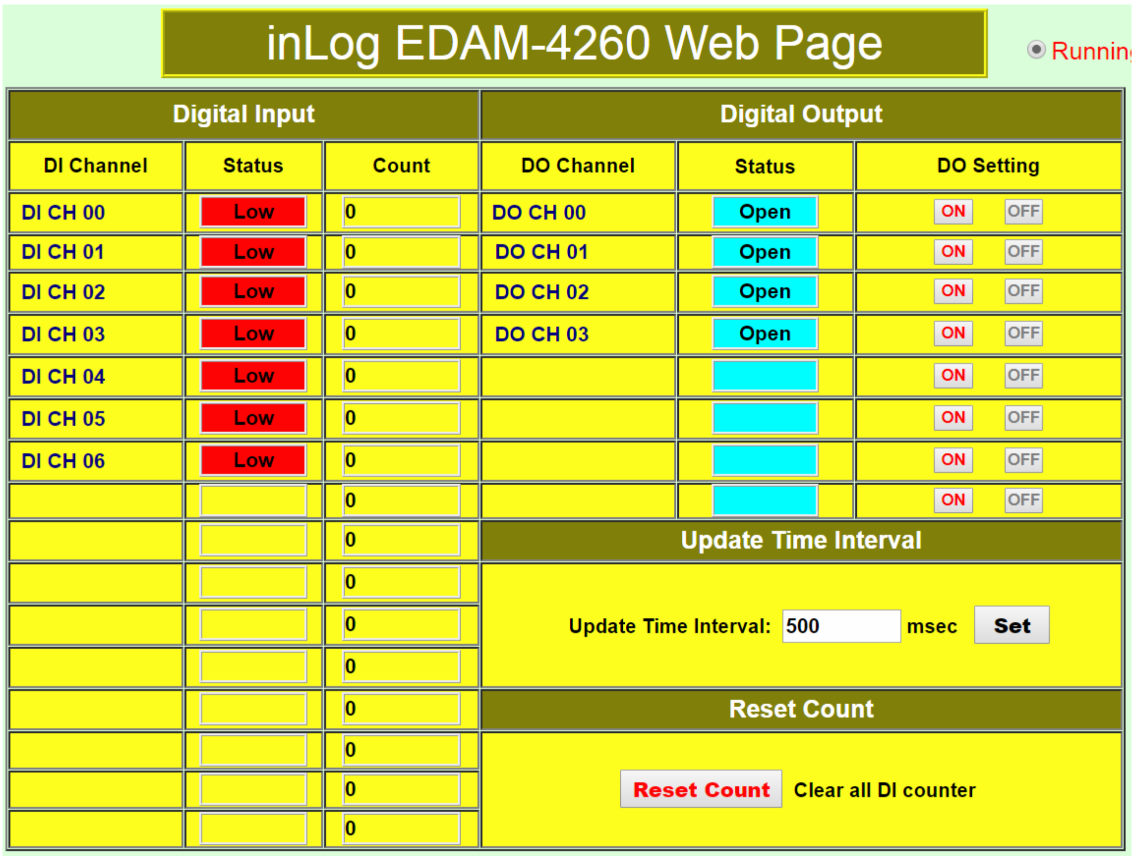
Update Time Interval: 500 msec Set

Reset Count

Reset Count Clear all DI counter

Channel	: Channel number of digital input or output
Status	: Current input or output status
Count/Latch	: Counter value or latch status of digital input which functions at "Counter" mode or "Latch" mode
Mode	: Channel operating mode
DO Setting	: Set digital output on or off
Time interval	: I/O status update time interval

11.3.3 EDAM-4260 monitoring page



inLog EDAM-4260 Web Page

● Runnin

Digital Input			Digital Output		
DI Channel	Status	Count	DO Channel	Status	DO Setting
DI CH 00	Low	0	DO CH 00	Open	ON OFF
DI CH 01	Low	0	DO CH 01	Open	ON OFF
DI CH 02	Low	0	DO CH 02	Open	ON OFF
DI CH 03	Low	0	DO CH 03	Open	ON OFF
DI CH 04	Low	0			ON OFF
DI CH 05	Low	0			ON OFF
DI CH 06	Low	0			ON OFF
		0			ON OFF
		0			
		0			
		0			
		0			
		0			
		0			
		0			

Update Time Interval

Update Time Interval: 500 msec Set

Reset Count

Reset Count Clear all DI counter

Channel	: Channel number of digital input or output
Status	: Current input or output status
Count/Latch	: Counter value or latch status of digital input which functions at "Counter" mode or "Latch" mode
Mode	: Channel operating mode
DO Setting	: Set digital output on or off
Time interval	: I/O status update time interval

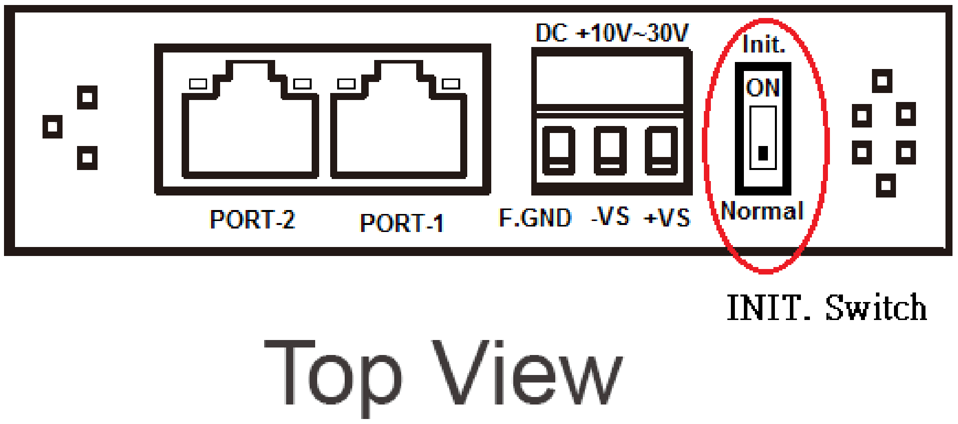
Chapter 12 Appendix

12.1 INIT* switch operation

The EDAM-4200 "INIT*mode" has two purposes, one for reading module current configuration, and another for configuring the module **IP Address, Subnet Mask, and Gateway**.

♦ Reading module current configuration
Each EDAM module has a built-in EEPROM which is used to store the configuration information such as address ID, type, DIO mode etc.. If the user unfurtunally forget the configuration of the module. User may use a special mode called "INIT* mode" to resolve the problem When the module is set to "INIT* mode", the default settings are IP Address, Subnet Mask, and Default Gateway (10.0.0.1, 255.255.255.0 and 10.0.0.1)

♦ Originally, the INIT mode is accessed by connecting the INIT* terminal to the GND terminal. New EDAM-4200 modules have the INIT switch located on the rear side of the module to allow easier access to the INIT mode. For these modules, INIT mode is accessed by sliding the INIT switch to the Init position as shown below.



♦ The following steps show you how to enable INIT* mode and read the current configuration:

1. Power off the module.
2. Sliding the INIT switch to the "Init" position.
3. Power on the module.
4. Start up the Windows Utility, it will search all EDAM-4200 I/O modules on the host PC' to read the current configuration stored in the EEPROM and set new **IP Address, Subnet Mask, and Default Gateway**,
5. Power off the module again
6. Sliding the INIT switch to the "Normal" position.

♦ Factory default settings:

1.	IP Address	: 10.0.0.1
2.	Subnet Mask	: 255.255.255.0
3.	Gateway	: 10.0.0.1
4.	DHCP	: Disabled
5.	Web Server	: Disabled
6.	Module ID	: 01
7.	Password	: 00000000

12.2 Module Status

Power-On Reset will let all output go to Power-On Value. The module may accept the host's command to change the output value. Host Watchdog Timeout will let all digital output go to Safe Value if the host watchdog timeout flag is set, and the output command will be ignored. The module's LED will go to flash and user must reset the module status via command to restore normal operation.

12.3 Dual Watchdog Operation

Dual Watchdog = Module Watchdog + Host Watchdog

The Module Watchdog is a hardware reset circuit to monitor the module's operating status. While working in harsh or noisy environment, the module may be down by the external signal. The circuit may let the module to work continues and never halt. The Host Watchdog is a software function to monitor the host's operating status. Its purpose is to prevent the network/communication from problem or host halt. While the timeout occurred, the module will turn the all output into safe state to prevent from unexpected problem of controlled target. The E-4200 module with Dual Watchdog may let the control system more reliable and stable.

12.4 Reset Status

The reset status of a module is set when the module is powered-on or when the module is reset by the module watchdog. It is cleared after the responding of the first $AA5 command. This can be used to check whether the module had been reset. When the $AA5 command responds that the reset status is cleared, that means the module has not been reset since the last $AA5 command was sent. When the $AA5 command responds that the reset status is set and it is not the first time $AA5 command is sent, it means the module has been reset and the digital output value had been changed to the power-on value.

12.5 Input counter and Input latch

Input counter:

Each input channel has internal counter used to software count the state change (*falling edge*) of input signal (max. 300Hz). The counting value can be read and cleared by sending "*Read digital input counter command*" or " *Clear digital input counter command*".

Input latch:

Each input channel has internal latch which is used to latch the pulse signal from the input. This latched state can be read by sending "*Read latched digital input* " command and cleared by sending "*Clear latched digital input*" command. For example, if the digital input is connected to a key switch. The key switch is a pulse signal. The user may lose the strike information by sending command $AA6. The digital input latch can latch the pulse and ready be read by sending "*Read latched digital input* " command. If the latched state=1 means that there is a key strike occurred.

12.6 Power-on & Safe value

Power-on value:

Power-on value is used to set the module default output value when the module is turned-on or watch dog timeout reset. This function is especially importance in some application where the specific initial output states are required User can set power on value by sending *Set power-on/safe value* command

Safe value:

Safe value are used to set the module outputs into the specific values when Host watchdog timeout If The host watchdog timer is enabled by sending *Set host watchdog timeout value*, the host should send *Host OK* command periodically within Timeout value to refresh the timer, otherwise the module will be forced to safety state.

12.7 DIO Synchronization (Mirror Local DI to Local/Remote DO)

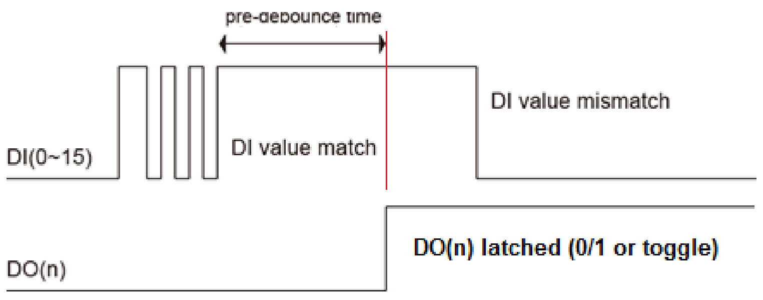
EDAM-4200 series modules also provide a *DIO Synchronization* function. A single digital output channel(or remote device) can be activated (1 or 0) dependent on the digital input channels value. When the specific DI channels value changed from "match" to "mismatch" (or "mismatch" to "match")DI mask pattern, the corresponding DO(or remote device) will be set to active state(1 or 0) dependent on the DO setting.

12.7.1 The *DIO Synchronization* is divided into two modes:

1. DI macth DO latch Mode.
2. DI mismatch DO latch Mode.

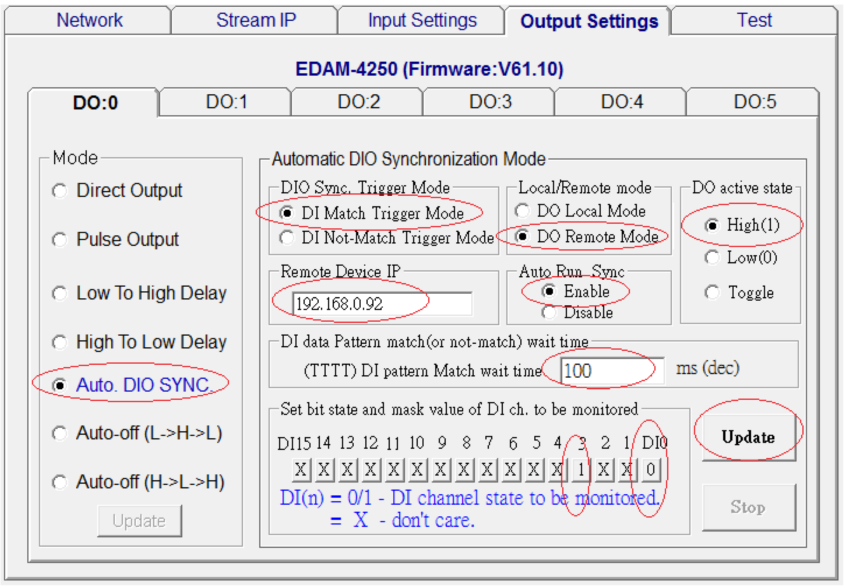
12.7.2 DIO Synchronization –DI match DO latch mode

When DI input value "match" DI mask pattern, the specific single digital output channel will be activated (1 or 0) and latched.



♦ **Example :** (**DI match DO latch mode & DO remote Mode**)

Assume DI channel 0,3 are monitored(the DI mask pattern is XXXXXXXXXXXX1XX0, DO remote mode (IP=192.168.0.92) and Auto Run Enable/DO Remote Mode, When DI input ch(0)=0 and ch(3)=1 (The DI pre-debounce time is 100 msec), the corresponding DO(0) will be set to activate(1).



EDAM-4250 (Firmware:V61.10)

Mode: Direct Output [ ] Pulse Output [ ] Low To High Delay [ ] High To Low Delay [ ] [✓ Auto. DIO SYNC.] Auto-off (L->H->L) [ ] Auto-off (H->L->H) [ ]

Automatic DIO Synchronization Mode

DIO Sync. Trigger Mode: [✓ DI Match Trigger Mode] DI Not-Match Trigger Mode

Local/Remote mode: DO Local Mode [✓ DO Remote Mode]

DO active state: [✓ High(1)] Low(0) Toggle

Remote Device IP: 192.168.0.92

Auto Run Sync: [✓ Enable] Disable

DI data Pattern match(or not-match) wait time
(TTTT) DI pattern Match wait time 100 ms (dec)

Set bit state and mask value of DI ch. to be monitored

DI15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	DI0
X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	0

DI(n) = 0/1 - DI channel state to be monitored.
= X - don't care.

♦ **Example** *(using ASCII command)*: DI match DO latch mode(for remote module)

When the specific DI channels (DI(0)=0, DI(3)=1) , the corresponding DO(0) of remote module will be set to ON(1). Assume DI pre-debounce time=0x0064(100)ms.

1. Set DO(0) to " DIO SYNC. Mode " (Ref. $AACONNDD)
   command: $01CO0004(cr)
   response: !01(cr) ; valid

2. Assume DI(0)=0 and DI(3)=1 are monitored and DI(1,2,4,5,6..)=X (don't care).
   R = 1 - remote Mode,(Local DI->Remote DO) .
   P = 1 - enable Auto Run(Start) DIO Synchronization operation when power-on.
   S = 1 - set digital output to active state (=1) when DI input data match DI mask pattern
   (Ref. $AAYM5CRPSTTTT (data))
   command: $01YM501110064XXXXXXXXXXXX1XX0(cr)
   response: !01(cr) ; valid

3. Set the remote module DO(0) to inactive state (OFF). *(Ref. @AA6ONSS)*

4. Start and wait for DI(0,3) DI pattern match.

♦ **Example** *(using ASCII command)***:** DI match DO latch mode(for local module)
When the specific DI channels (DI(0)=0, DI(3)=1) , the corresponding DO(0) of local module will be set to ON(1).
Assume DI pre-debounce time=0x0064(100)ms.

1. Set DO(0) to " DIO SYNC. Mode " (Ref. $AACONNDD)
   command: $01CO0004(cr)
   response: !01(cr) ; valid

2. Assume DI(0)=0 and DI(3)=1 are monitored and DI(1,2,4,5,6..)=X (don't care).
   R = 0 - Local Mode,(Local DI->Local DO) .
   P = 1 - enable Auto Run(Start) DIO Synchronization operation when power-on.
   S = 1 - set digital output to active state (=1) when DI input data match DI mask pattern
   (Ref. $AAYM5CRPSTTTT (data))
   command: $01YM500110064XXXXXXXXXXXX1XX0(cr)
   response: !01(cr) ; valid

3. Stop (S=0) DIO Sync. Operation on (Ref. $AAYMRCS)
   command: $01YMR00(cr)
   response: !01(cr) ; valid

4. Set the Local module DO(0) to inactive state (OFF). *(Ref. @AA6ONSS)*
   command: @016O000(cr)
   response: !01(cr) ; valid

5. Start/Run (S=1) DIO Sync. Operation on (Ref. $AAYMRCS)
   command: $01YMR01(cr)
   response: !01(cr) ; valid

6. Start and wait for DI(0,3) DI pattern match.

Ref. $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS,$AAYM3CPSTTTT(data)

♦ **Example (using Modbus command) :** DI match DO latch mode (for remote module)

When the specific DI channels (DI(0)=0, DI(3)=1) , the corresponding DO(0) of remote module will be set to ON(1). Assume DI pre-debounce time=0x0064(100)ms.

1. (41453) Set DO(0) mode to " *DIO SYNC. Mode* "
   Request: 01 06 A1 EC 00 04
   Response: 01 06 A1 EC 00 04 ; valid

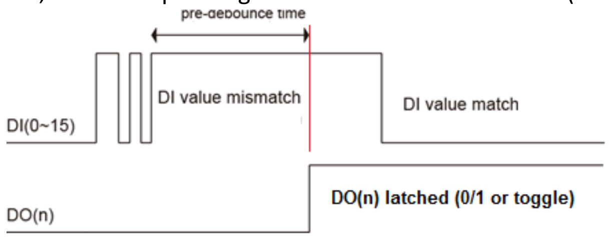
2. (41580) Set DO(0) to "*DI Match DO latch mode*" , enable auto run when power-on,
   DO active output =1, and remote mode.
   Request: 01 06 A2 6B 00 1E
   Response: 01 06 A2 6B 00 0E ; valid

3. (41564) Set DI channels DI(3,0) to be monitored and DI mask pattern
   high order word : = 0x0008 = (0000 0000 0000 1000) ; DI mask pattern
   low order word : = 0x0009 = (0000 0000 0000 1001) ; DI channels to be monitored
   ;(1=monitored DI chn,0=not monitored DI chn.)
   Request: 01 10 A2 5B 00 02 04 00 08 00 09
   Response: 01 10 A2 5B 00 02 ; valid

4. (41628) Set DI debounce time to=100(0x64)ms
   Request: 01 10 A2 9B 00 01 02 00 64
   Response: 01 10 A2 AE 00 02 ; valid

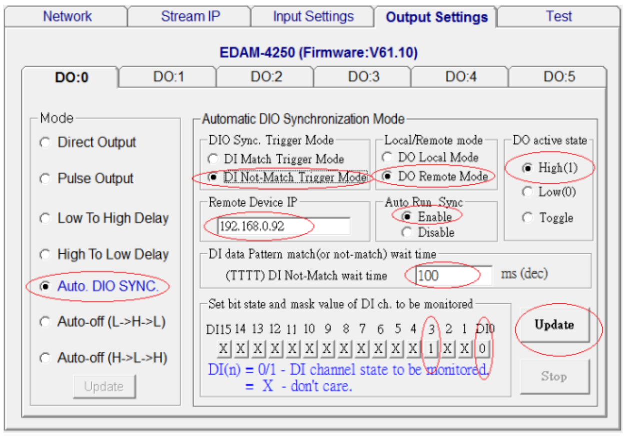
12.7.3 DIO Synchronization –DI mismatch DO latch mode

A single digital output channel is activated(1 or 0) dependent on the specific DI value, When the specific DI channels status mismatch DI mask pattern, the corresponding DO will be set to active state(1 or 0)



Example : (DI mismatch DO latch mode & DO remote Mode)

Assume DI channel 5,6 are monitored(the DI mask pattern is XXXXXXXXX10XXXXX), DO remote mode (IP=192.168.0.92) and Auto Run Enable/DO Remote Mode, When mismatch DI input ch(0)=1 and ch(2)=0 (The DI pre-debounce time is 150 msec), the corresponding DO(0) will be set to inactivate(0).



EDAM-4250 (Firmware:V61.10)

Mode: Direct Output [✓ Auto. DIO SYNC.] Pulse Output, Low To High Delay, High To Low Delay, Auto-off (L->H->L), Auto-off (H->L->H)

Automatic DIO Synchronization Mode

DIO Sync. Trigger Mode: DI Match Trigger Mode [✓ DI Not-Match Trigger Mode]

Local/Remote mode: DO Local Mode [✓ DO Remote Mode]

DO active state: [✓ High(1)] Low(0) Toggle

Remote Device IP: 192.168.0.92

Auto Run Sync: [✓ Enable] Disable

DI data Pattern match(or not-match) wait time
(TTTT) DI Not-Match wait time 100 ms (dec)

Set bit state and mask value of DI ch. to be monitored

DI15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	DI0
X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	0

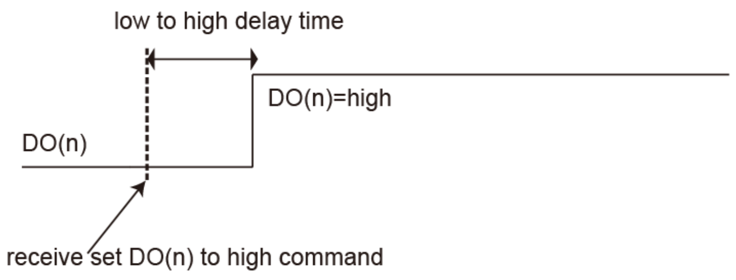
DI(n) = 0/1 - DI channel state to be monitored.
= X - don't care.

12.8 High/Low delay output mode

EDAM-4200 series modules supports **high-to-low and low-to-high delay** output function

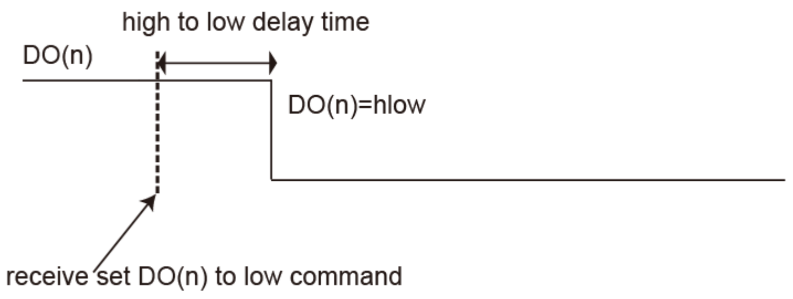
12.8.1 Low to High Delay output

When you choose *Low to High* delay mode, it is almost the same as choosing the DO direct output mode. The only difference is that there will be certain time delay when the output value changes from logic low to logic high. You can define the delay time by entering its value into the delay time text box in the setting area. After you complete the setting, click the" Apply" button. Then you can control the digital output value by the DO button and see its current value by the DO status LED display at the top of the module Display area.



12.8.2 High to Low Delay output

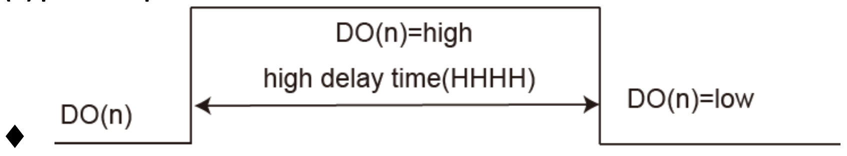
When you choose *High to Low* delay mode, it is almost the same as choosing the DO direct output mode. The only difference is that there will be certain time delay when the output value changes from logic high to logic low. You can define the delay time by entering its value into the Delay time text box in the Setting area. After you complete the setting, click the Apply button. Then you can control the digital output value by the DO button and see its current value by the DO status LED display at the top of the module Display area.



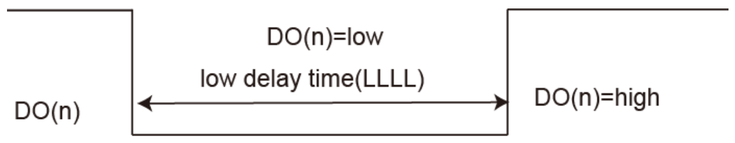
12.9 DO Auto-Off Time Mode

This function is used to force the specific DO channel to work as a monostable operation. After a certain period of time, the DO returns to the stable state until another triggering command is applied.

♦ **Low to High DO(n) pulse output**



♦ **High to Low DO(n) pulse output**



♦ **ASCII command:**

1.	$AACONNDD	;Set DO(n) to DO Auto-Off Time Mode
2.	$AA9DNNHHHHLLLL	;Set high /low delay width of DO(n)
3.	#AA1NDD	; Write DO channel to active

Example: Set the channel-0 of DO to active and the channel-0 of D/O will be auto-off(inactive) after 3 sec.

1. $01CO0006 - Se to DO(0) Auto-Off Time Mode for DO "Low->High->Low.
   (output low is active (ON), outputt high/open is inactive)
2. $019D0017700001 - Set DO High delay time(HHHH=3000ms) and Low delay time(LLLL= 0.5ms).
   (For " Low->High->Low Auto-Off Time mode" LLLL always "0001")
3. #011001 - Write DO channel(0) to active
   wait DO auto-off time (3sec)..........,
   ...
4. the DO channel(0) auto-off from active to inactive.

Example: Set the channel-1 of DO to inactive and the channel-1 of D/O will be auto-off(active) after 3 sec.

1. $01CO0107 - Se to DO(1) Auto-Off Time Mode for DO "High->Low->High.
   (output low is active (ON), outputt high/open is inactive)
2. $019D0100011770 - Set DO High delay time(HHHH=0.5ms) and Low delay time(LLLL= 3000ms).
   (For " High->Low->High Auto-Off Time mode" HHHH always "0001")
3. #011100 - Write DO channel(1) to inactive
   wait DO auto-off time (3sec)..........,
   ...
4. the DO channel(1) auto-off from inactive to active.

♦ **Modbus rtu command:**

1.	X+1453 ~ X+1484	;Set DO(0~31) to low to high delay mode(=2) or high to low delay mode(=3)
2.	X+1711 ~ X+1774	;Set high/low delay time of the specific DO(n)
3.	X+1775 ~ X+1806	;Start *auto-off time Mode* operation