EX9000-MTCP

Analog & DIO series

Data Acquisition Modules
User's Manual

Web site: www.topsccc.com.tw

Printed Date: June 1, 2016

Table Of Contents

Main Features

1.1 Ethernet I/O D/A Modules

EX9000-MTCP is based on the popular Ethernet networking standards used today in most business environments. Users can easily add EX9000-MTCP I/O modules to existing Ethernet networks or use EX9000-MTCP modules in

new Ethernet-enabled Manufacturing networks. EX9000-MTCP module features a 10/100 Mbps Ethernet chip and supports industrial popular Modus/TCP protocol over TCP/IP for data connection. EX9000-MTCP also supports UDP protocol over Ethernet networking. With UDP/IP, EX9000-MTCP 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 EX9000-MTCP Ethernet I/O D/A. And, these real-time data can be integrated with business system to create valuable, competitive business information immediately.

1.2 Ethernet I/O with Smart Function

Enhancing from traditional I/O modules, EX9000-MTCP I/O modules have pre-built Smart mathematic functions to empower the system capacity. The Digital Input modules provide Counter, Totalizer functions; the Digital Output modules provide pulse output, delay output functions; the Analog Input modules provide the Max./Min./Average data calculation; the Analog Output modules provide the PID loop control function.

1.3 Multi I/O in One Module to fit all Applications

EX9000-MTCP Multi 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 D/A system or standalone control unit in a middle or large scale, EX9000-MTCP Multi I/O design can easily fit application needs by one or two modules only. With additional embedded control modules, EX9000-MTCP can easily create a localized, less complex, and more distributed I/O architecture.

1.4 Industrial standard Modbus/TCP Protocol Supported for open connectivity

EX9000-MTCP modules support the popular industrial standard, Modbus/TCP protocol, to connect with Ethernet Controller or HMI/SCADA software built with Modbus/TCP driver..

1.5 Software Support

Based on the Modbus/TCP standard, the EX9000-MTCP firmware is a built-in Modbus/TCP server. Therefore, EXPERTDAQ provides the necessary DLL drivers, and Windows Utility for users for client data for the   EX9000-MTCP. Users can configure this D/A 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.

1.6 Technical Specification of EX9000-MTCP

Ethernet: 10 BASE-T IEEE 802.3 100 BASE-TX IEEE 802.3u

Wiring: UTP, category 5 or greater

Bus Connection: RJ45 modular jack

Common 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

Ethernet Communication: 1500 V DC

Isolation of I/O Module : 2000/2500/3000 VDC

Status Indicator: Power, CPU, Communication (Link, Collide, 10/100 Mbps, Tx, Rx)

Case: ABS with captive mounting hardware

Plug-in Screw Terminal Block: Accepts 0.5 mm 2 to 2.5 mm 2 , 1 - #12 or 2 - #14 to #22 AWG

Operating Temperature: - 25 to 75º C

Storage Temperature: - 25 to 85º C

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.

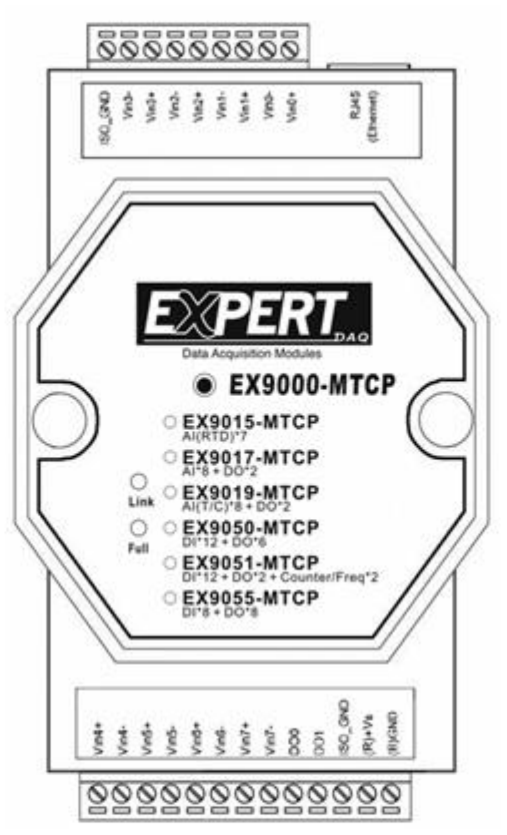
1.7 Dimensions

The following diagrams show the dimensions of the EX9000-MTCP I/O module in millimeters.

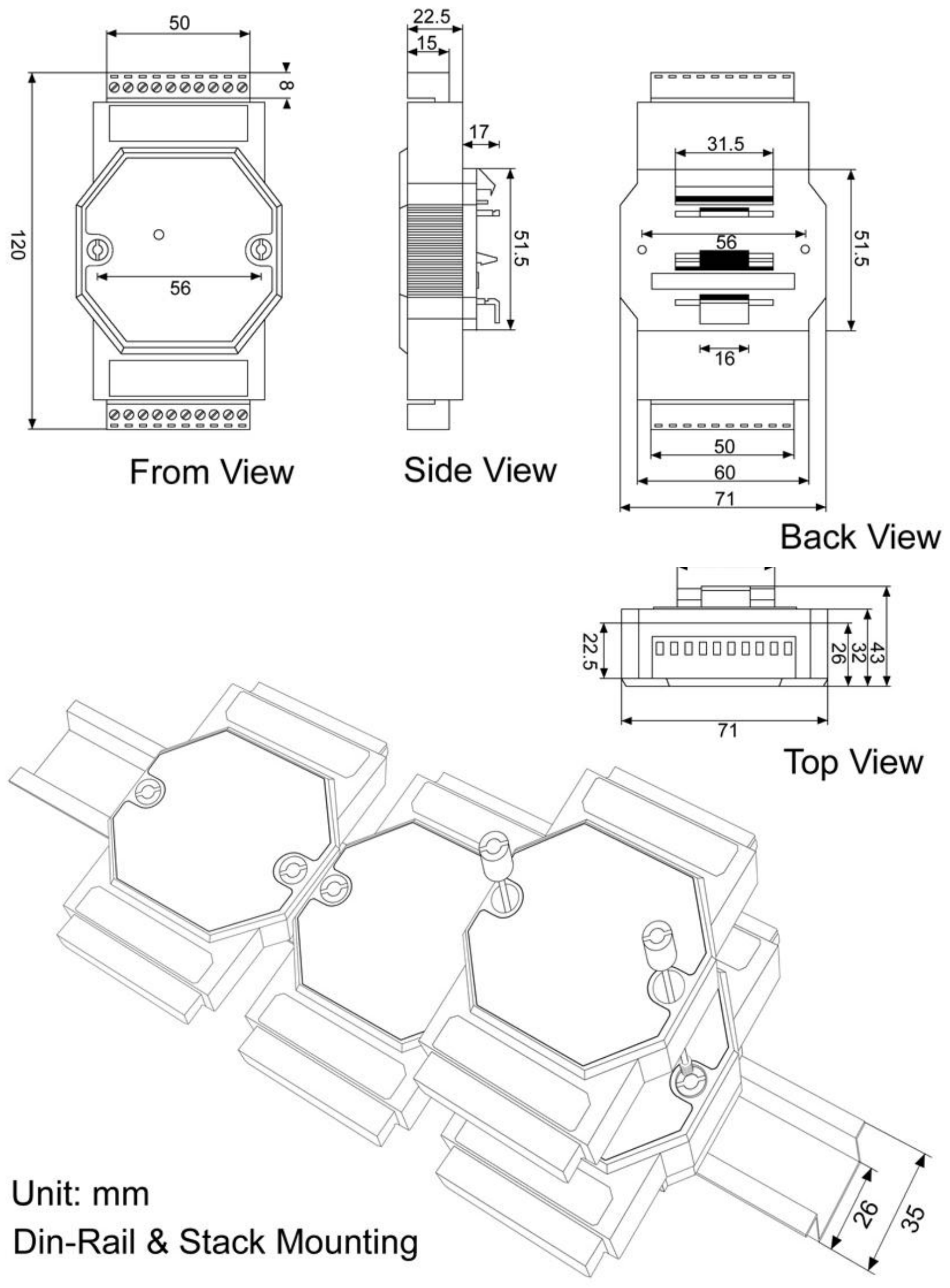
LED Status: Red indicator. one is for Power ; Link & full for normal on whenever EX9000-MTCP module is running.

EX9000-MTCP I/O Modules support Din-Rail & Wall Mount.

EX9000-MTCP IO Modules support stack Mounting also.



9000 Dimension



1.8 System Requirements

Host Computer

IBM PC compatible computer with 486 CPU (Pentium is recommended)

Microsoft 95/98/2000/NT 4.0 (SP3 or SP4)/XP 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

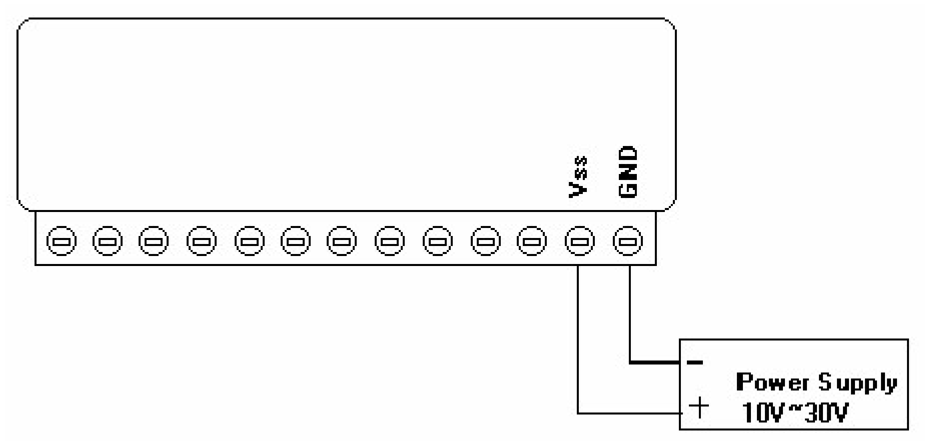
Power supply for EX9000-MTCP (+10 to +30 V unregulated)

1.9 Wiring and Connections

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

1.10 Wiring for Power supply

Although the EX9000-MTCP/TCP systems are designed for a standard industrial unregulated 24 VDC 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

1.11 Wiring for Ethernet I/O modules

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 0.5 mm to 2.5 mm.

Always use a continuous length of wire. Do not combine wires to make them longer.

Use the shortest possible wire length.

Use wire trays for routing where possible.

Avoid running wires near high-energy wiring.

Avoid running input wiring in close proximity to output wiring where possible.

Avoid creating sharp bends in the wires..

1.12 Summary of DIO modules

The EX9000-MTCP 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	Input ch.	Input type	Output ch.	Output type
9050-MTCP	12	Isolated single ended with **Wet** Contact (*common source or ground*)	6	Isolation with Open collector (NPN)
9051-MTCP	12	Isolated single ended with **Dry** Contact *(common source)*	2	Isolation with Open collector (NPN)
	2	Iso. with differential counter input		
9055-MTCP	8	Isolated single ended with **Dry/Wet** Contact*(common source or ground)*	8	Isolated with open drain (P-MOSFET), (3A/per channel)
9053-MTCP	12	Isolated single ended with **Dry/Wet** Contact*(common source or ground)*	4	Isolation with Open collector (NPN)

Relay Output and DC Input modules

Module	**In ch.**	**Input type**	**Out ch.**	**Output type**
9060-MTCP	5	Isolated single ended with ***common source or ground.***	3	0.6A@125VAC, 2A@30VDC. RL1 Form A, RL2,RL3 Form C
9063-MTCP	7	Isolated single ended with ***common source or ground.***	3	5A@250VAC, 5A@30VDC. (Form A)

2.0 Specification and Wiring

Analog input modules use an A/D converter to convert sensor voltage, current, thermocouple or RTD signals into digital data. The digital data is then translated into engineering units. When prompted by the host computer, the data is sent through a standard 10/100 based-T Ethernet interface. Users would able to read the current status via pre-built web page or any HMI software package supported Modbus/TCP protocol. The analog input modules protect your equipment from ground loops and power surges by providing opto-isolation of the A/D input andtrans-former based isolation up to 2,000/2,500/3,000 VDC .

2.1 EX9015-MTCP 7-channel RTD Input Module

The EX9015-MTCP is a 16-bit, 7-channel RTD input module that provides programmable input ranges on all channels. It accepts Various RTD inputs (PT100, PT1000, Balco 500 & Ni) and provides data to the host computer in engineering units (°C). In order to satisfy various temperature requirements in **one module, each analog channel is** allowed to configure an individual range for several applications.

9015MTCP Specification

Analog Input:

Effective resolution: 16-bit

Channels: 7

Input type: PT100, PT1000, Balco 500 & Ni

Input range:

PT100 -50°C ~ 150°C ,0°C ~ 100°C ,0°C ~ 200°C ,0°C ~ 400°C ,-200°C ~ 200°C ,

PT1000 -40°C ~ 160°C

Balco 500 -30°C ~ 120°C

Ni 604 -80°C ~ 100°C or 0°C ~ 100°C

Ni 1000 -60°C ~ 160°C

Isolation voltage:3000V

Sampling rate: 12 samples / sec.

Input impedance: 20 MΩ

Accuracy: ±0.05% or better

Zero drift: ±3 μV/°C

Span drift: ±25 ppm/°C

CMR @ 50/60 Hz: 150 dB

NMR @ 50/60 Hz: 100 dB

Built-in Watchdog Timer

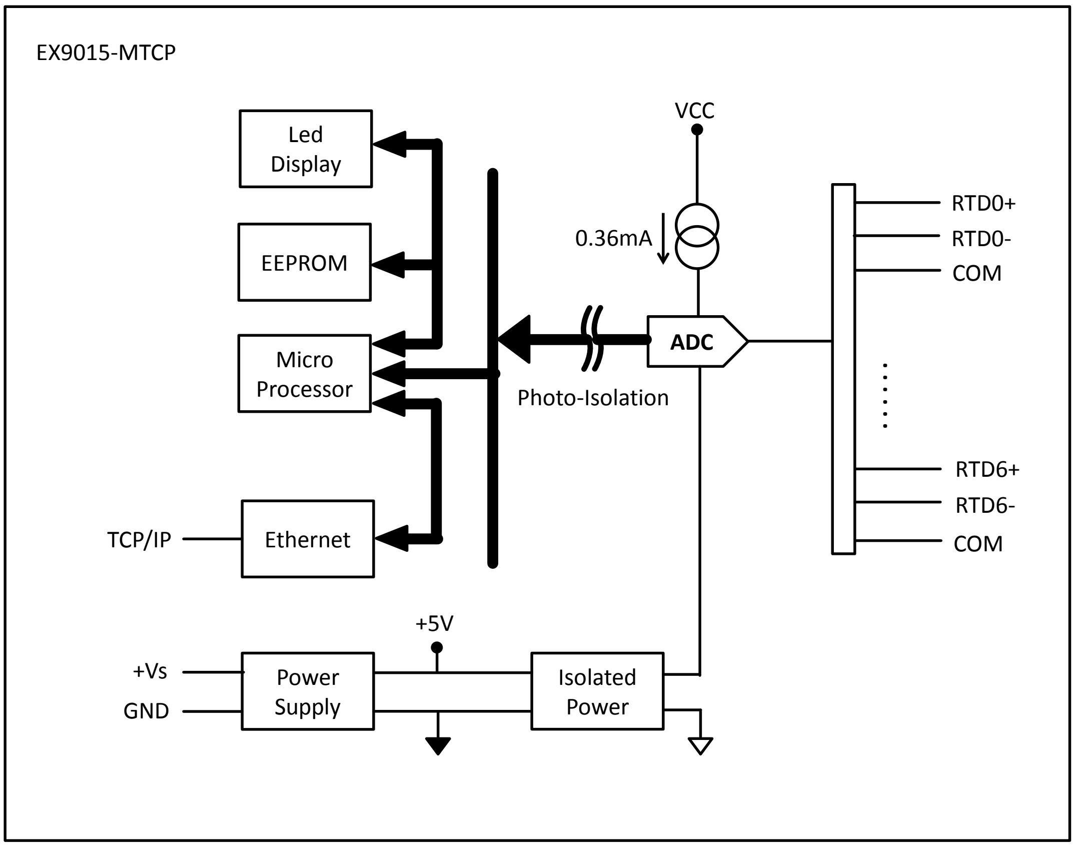
Power requirements: Unregulated +10 ~ +30 VDC

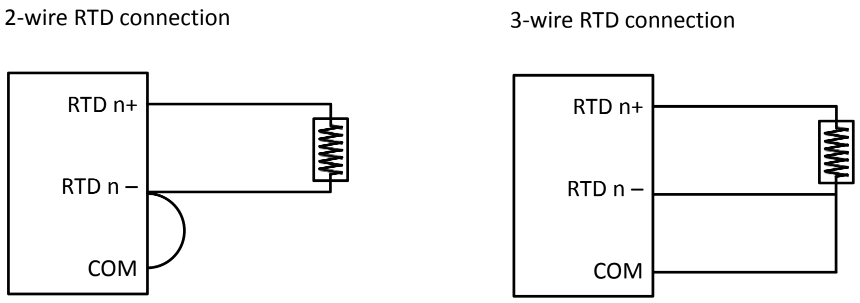
Power consumption: 2.2W

EX9015MTCP - MODBUS Data Format(Hex) Table

Input Type	-Full scale	+Full scale.	Formula
Pt100(-50~+150C)	0	FFFF	Modbus data > 0x4000 (Modbus data – 16384)*150/49152 Modbus data < 0x4000 (Modbus data – 16384)*50/16384
Pt100(-200C~+200C)	0	FFFF	(Modbus data – 32768)*200/32768
Pt100(0C~+100C)	0	FFFF	Modbus data*100/65536
Pt100(0C~+200C)	0	FFFF	Modbus data*200/65536
Pt100(0C~+400C)	0	FFFF	Modbus data*400/65536
Pt1000(-40C~+160C)	0	FFFF	Modbus data > 0x3333 (Modbus data – 13107)*160/52429 Modbus data < 0x3333 (Modbus data – 13107)*40/13107
Ni(604)0C~100C	0	FFFF	Modbus data*100/65536
Ni(604)-80C~100C	0	FFFF	Modbus data > 0x71C7 (Modbus data – 29127)*100/36409 Modbus data < 0x71C7 (Modbus data – 29127)*80/29127
Ni(1000)-60C~160C	0	FFFF	Modbus data > 0x45D1 (Modbus data – 17873)*160/47663 Modbus data < 0x45D1 (Modbus data – 17873)*60/17873
Balco(500)-30C~+120C	0	FFFF	Modbus data > 0x3333 (Modbus data – 13107)*120/52429 Modbus data < 0x3333 (Modbus data – 13107)*30/13107

Application Wiring





Assigning ModBus address

Based on the Modbus/TCP standard, the addresses of the I/O channels in EX9000-MTCP modules you place in the system are defined by a simple rule. Please Refer 7.0 to map the I/O address.

2.2 EX9017-MTCP 8-channel Analog Input with 2/DO Module

The EX9017-MTCP is a 16-bit, 8-channel analog differential input module that provides programmable input ranges on all channels. It accepts millivoltage inputs (±100mV, ±500mV), voltage inputs (±1V, ±5V and ±10V) and current input (±20 mA, 4~20 mA) and provides data to the host computer in engineering units (mV, V or mA). In order to satisfy all plant needs in one module, 9017MTCP has designed with 8 analog inputs and 2 digital outputs. Each analog channel is allowed to configure an individual range for variety of applications.

9017MTCP Specification

Analog Input:

Effective resolution: 16-bit
Channels: 8 differential
Input type: mV, V, Ma
Input range: ±150 mV, ±500 mV, 0-5 V, ±10 V, 0-20 mA, 4-20 mA
Isolation voltage: 3000 V
Fault and overvoltage protection: With stands over voltage up to ±35 V
Sampling rate: 10 samples / sec.
Input impedance: 20 MΩ
Bandwidth: 13.1 Hz @ 50 Hz, 15.72 Hz @ 60 Hz
Accuracy: ±0.1% or better
Zero drift: ±6 μV/°C
Span drift: ±25 ppm/°C
CMR @ 50/60 Hz: 92 dB min.

Digital Output:

Channel: 2
Open Collector(NPN) up to 30V/ 500 mA max. load
Optical Isolation: 3000V

Built-in Watchdog Timer
Power requirements: Unregulated +10 ~ +30 VDC
Power consumption: 2.2 W

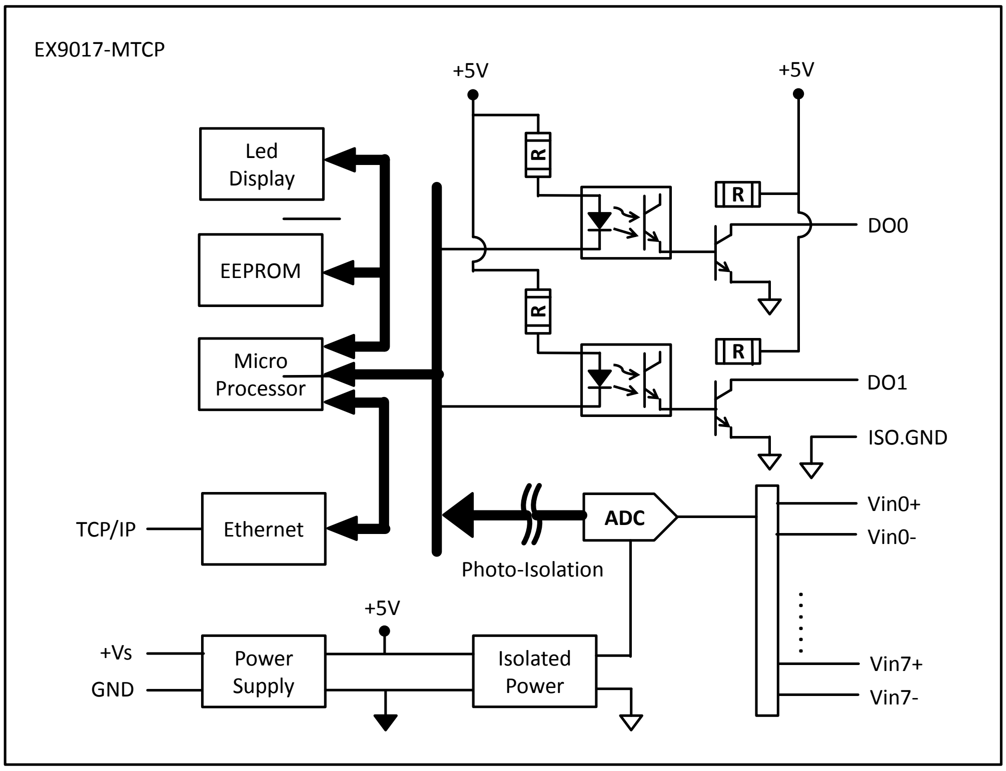
EX9017MTCP - MODBUS Data Format(Hex) Table

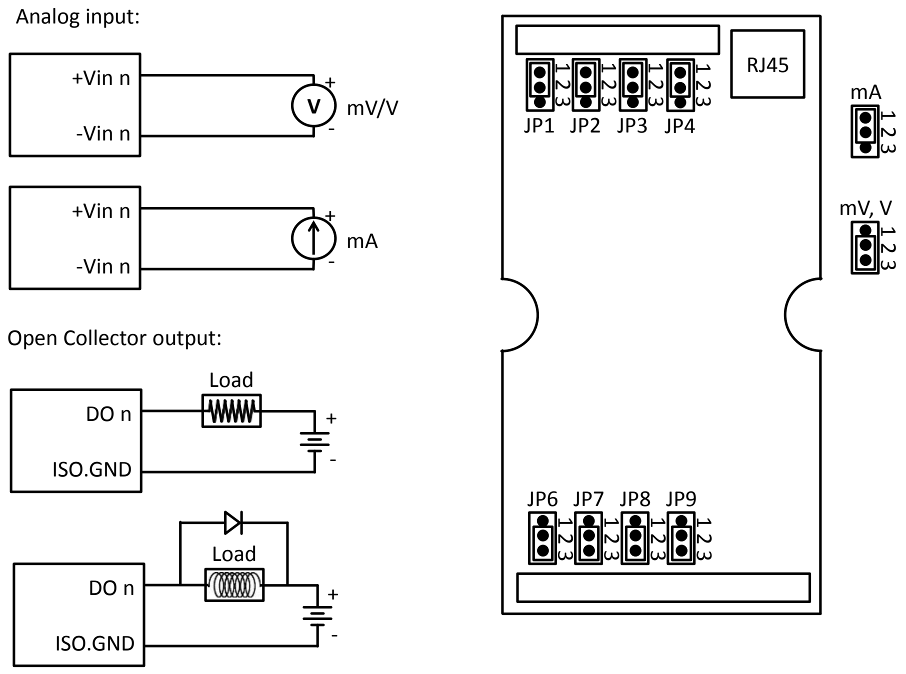
Input Type	-Full scale	+Full scale.	Formula	Unit
-10V ~ +10V	0	FFFF	Volt = (MODBUS Data - 32767)*10/32767	V
-5V ~ + 5V	0	FFFF	Volt = (MODBUS Data - 32767)*5/32767	V
-1V ~ +1V	0	FFFF	Volt = (MODBUS Data - 32767)*1/32767	V
-500mV ~ +500mV	0	FFFF	Volt = (MODBUS Data - 32767)*500/32767	mV
-150mV ~ +150mV	0	FFFF	Volt = (MODBUS Data - 32767)*150/32767	mV
0mA ~ 20mA	0	FFFF	Current = (MODBUS Data)/3276.75	mA
4mA~20mA	0	FFFF	Current=(MODBUS Data*16/65535)+4	mA

Example for 4~20mA
Modbus data=48A9(Hex)=18601(Dec)
Current=(18601*16/65535)+4=8.541mA

Application Wiring

9017MTCP has built with a 120 Ω resistor in each channel; users do not have to add any resistors in addition for current input measurement. Just adjust the jumper setting to choose the specific input type you need. Refer each analog input channel has built-in a jumper on the PCB for users to set as a voltage mode or current mode.





Assigning ModBus address

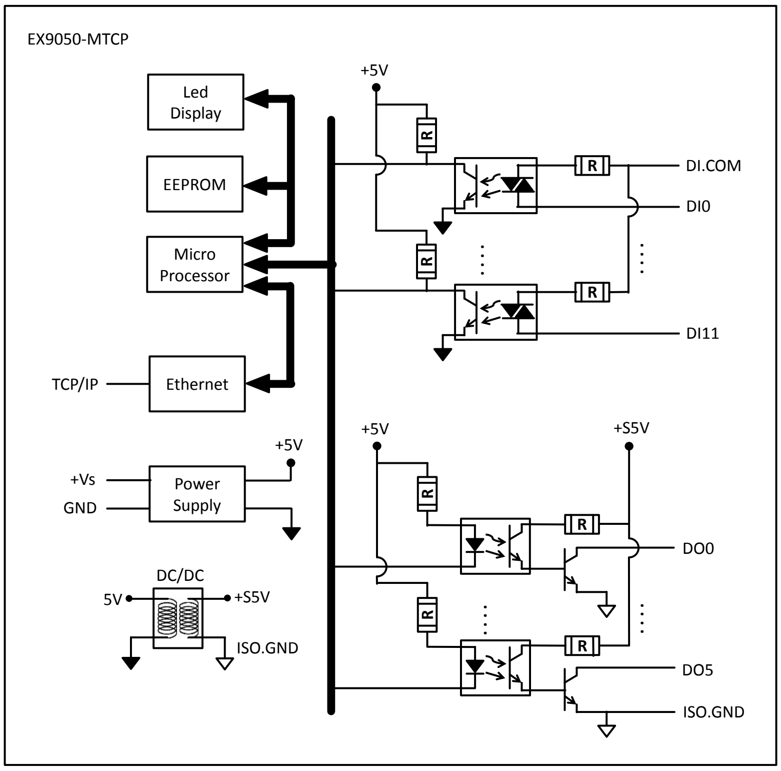
Basing on Modbus/TCP standard, the addresses of the I/O channels in EX9000-MTCP modules you place in the system are defined by a simple rule. Please Refer 7.0 to map the I/O address.

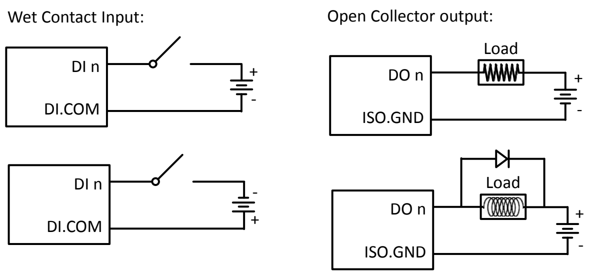
2.3 EX9050-MTCP 18-channel Digital I/O Module

The EX9050-MTCP is a high-density I/O module built-in a 10/100 based-T interface for seamless Ethernet connectivity. It provides 12 digital input and 6 digital output channels with 3000VDC 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 1 KHz counter. Opposite to the intelligent DI functions, the Digital Output channels also support pulse output function.

9050MTCP Specification
Digital input:
Channel: 12(single ended with common source/ground)
Input Type: (Logic level status can be inversed by Utility)
Wet Contact: Logic level 0: 0V~+2VDC max
Logic level 1: +10VDC~+50VDC max
Supports 1 kHz counter input (32-bit + 1-bit overflow)
Digital Output:
Channel: 6(sink)
Open Collector(NPN) up to 30V/500mA max. load
Power Consumption: 2 W (Typical)

Application Wiring:





Assigning ModBus address

Basing on Modbus/TCP standard, the addresses of the I/O channels in EX9000-MTCP modules you place in the system are defined by a simple rule. Please Refer 7.0 . All Digital Input channels in 9050MTCP are allowed to use as 32-bit counters (Each counter is consisted of two addresses, Low word and High word). Users could configure the specific DI channels to be counters via Windows Utility

2.4 EX9051-MTCP 16-channel Digital I/O Module

The EX9051-MTCP is a high-density I/O module built-in a 10/100 based-T interface for seamless Ethernet connectivity. It provides 12 digital input, 2 digital output, and 2 counter/ freq. (4.5 KHz) channels with 3000VDC 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 1 KHz counter. Opposite to the intelligent DI functions, the Digital Output channels also support pulse output function.

9051MTCP Specification

Digital Input:

Channel: 12(single ended with common source)

Input Type: (jumper select) (Logic level status can be inversed by Utility)

Dry Contact: Logic level 0: Close to GND

Logic level 1: Open

Optical Isolation: 3000VDC

Supports 1 kHz counter input (32-bit + 1-bit overflow)

Digital Output:

Channel: 2(sink)

Open Collector(NPN) up to 30V/ 500 mA max. load

Optical Isolation: 3000VDC

Counter/ Freq. :

Channel: 2

Maximum Count: 4,294,967,285(32 bit)

Input frequency: 0.3 ~ 4500 Hz max. (Frequency mode) ,4500 Hz max. (counter mode)

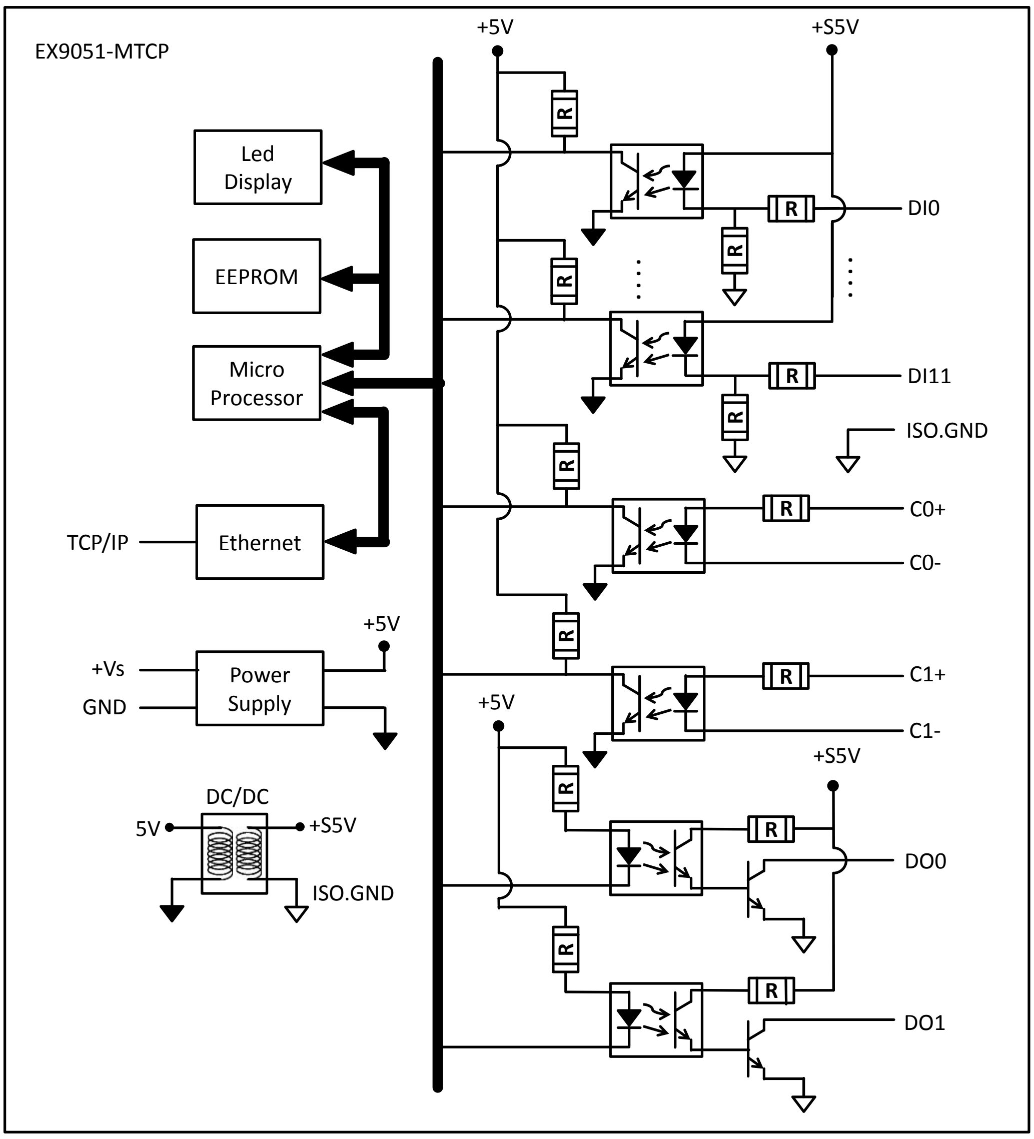
Isolation voltage: 3000VDC

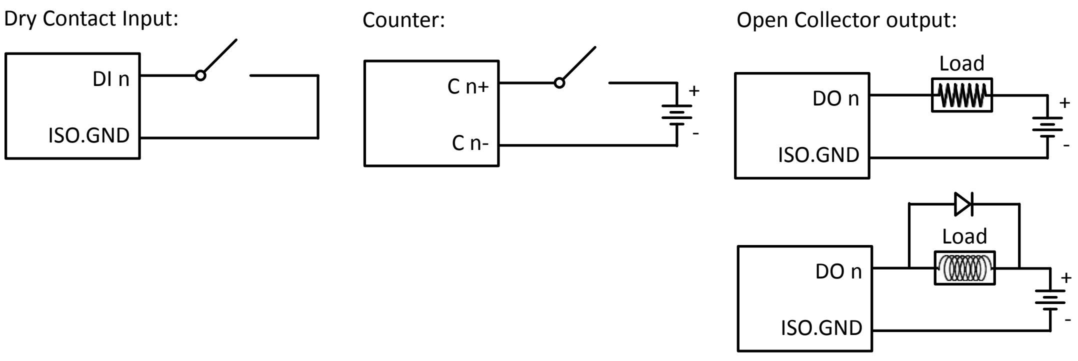
Mode: Counter, Frequency

Power Consumption: 2 W (Typical)

`

Application Wiring:





Assigning ModBus address

Basing on Modbus/TCP standard, the addresses of the I/O channels in EX9000-MTCP modules you place in the system are defined by a simple rule. Please Refer 7.0 to map the I/O address

All Digital Input channels in 9051MTCP are allowed to use as 32-bit counters (Each counter is consisted of two addresses, Low word and High word). Users could configure the specific DI channels to be counters via Windows Utility.

2.5 EX9053-MTCP 12 DI and 4 DO channels Digital I/O Module

The EX9053-MTCP is a high-density I/O module built-in a 10/100 based-T interface for seamless Ethernet connectivity. It provides 12 digital input, 4 digital output. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the Digital Output channels also support pulse output function.

9053MTCP Specification

Digital Input: Digital Input: Isolated single ended with common source/ground.

Channel: 12 (DI0~DI11)

Input Type: Logic level status can be inversed via ASCII/Modbus command

Dry Contact: Logic level 0: Close to GND

Logic level 1: open

Wet Contact: Logic level 0: 0V~+2VDC max

Logic level 1: +10VDC~+50VDC max

Supports 500Hz counter input (32-bit + 1-bit overflow)
Input Impedance : 10K ohm(Wet Contact)

Digital Output: Isolated Open collector (NPN) output channels.

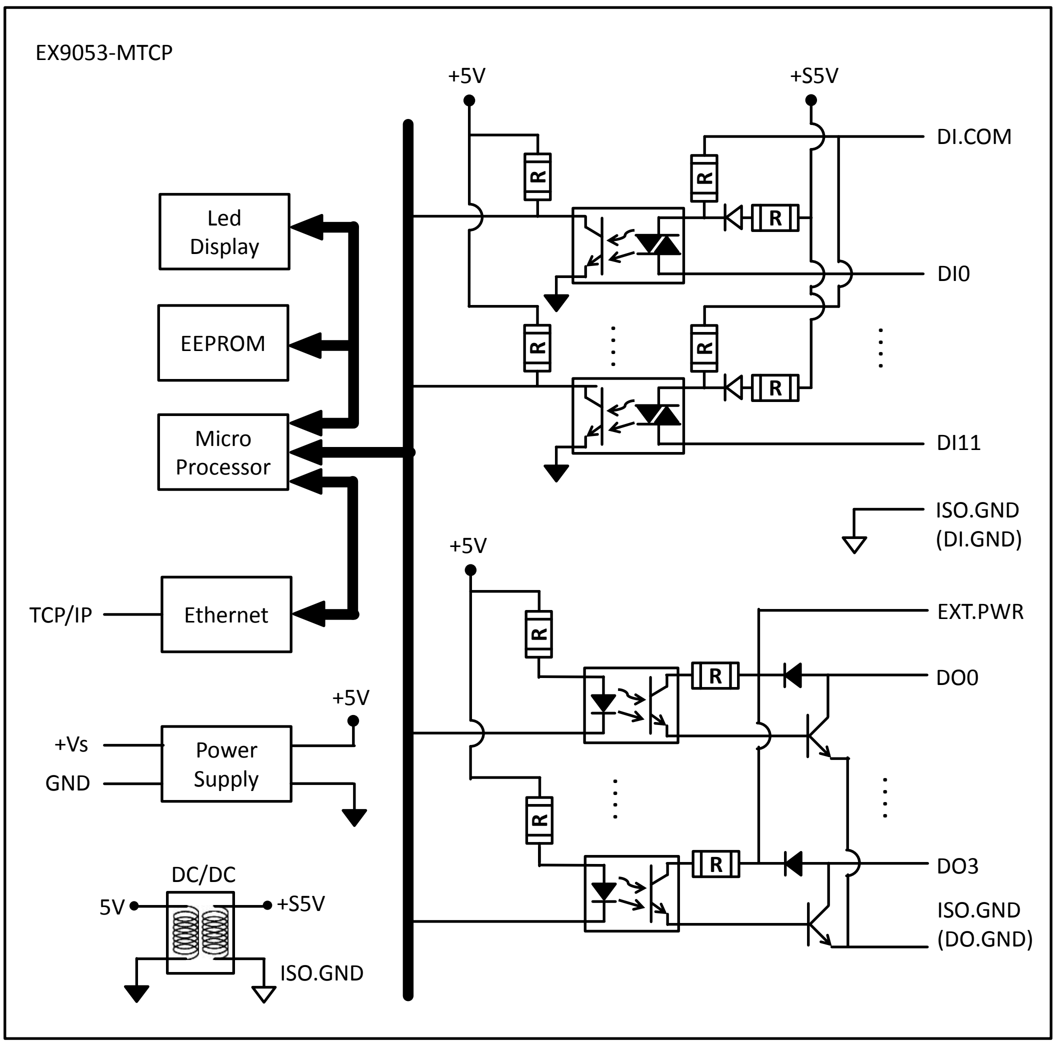
Channel: 4 (DO0~DO3)

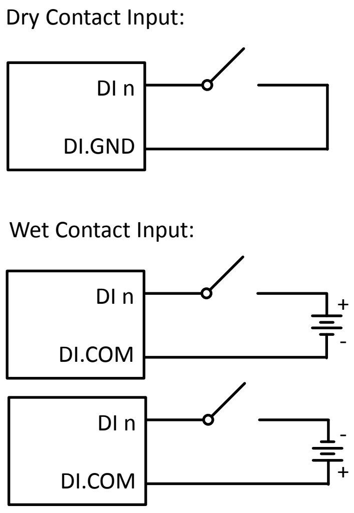
Output type : Logic level status can be inversed via ASCII/Modbus command
Open Collector to+5V~ 30V/ 500 mA max. load

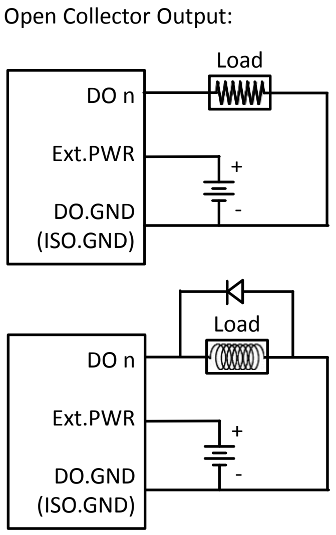
Optical Isolation: 3750Vrms
Pulse Output : Each channel supports 500Hz pulse output

Power Consumption: 1.8W (Typical)

Application Wiring:







Assigning ModBus addresses

Basing on Modbus/TCP standard, the addresses of the I/O channels in EX9000-MTCP modules you place in the system are defined by a simple rule. Please Refer 7.0 to map the I/O address

All Digital Input channels in 9053MTCP are allowed to use as 32-bit counters (Each counter is consisted of two addresses, Low word and High word). Users could configure the specific DI channels to be counters via Windows Utility.

2.6 EX9055-MTCP 16-channel Digital I/O Module

The EX9055-MTCP is a high-density digital I/O module designed with a 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.

9055MTCP Specification

I/O Type: 8 DI/ 8 DO

Digital Input: 8(single ended with common source/ ground)

Dry Contact :

Logic level 0 : Close to GND

Logic level 1 : Open

Wet Contact :

Logic level 0 : +2 VDC max

Logic level 1 : +10VDC to 50 VDC

Supports 1 kHz counter input (32-bit + 1-bit overflow)

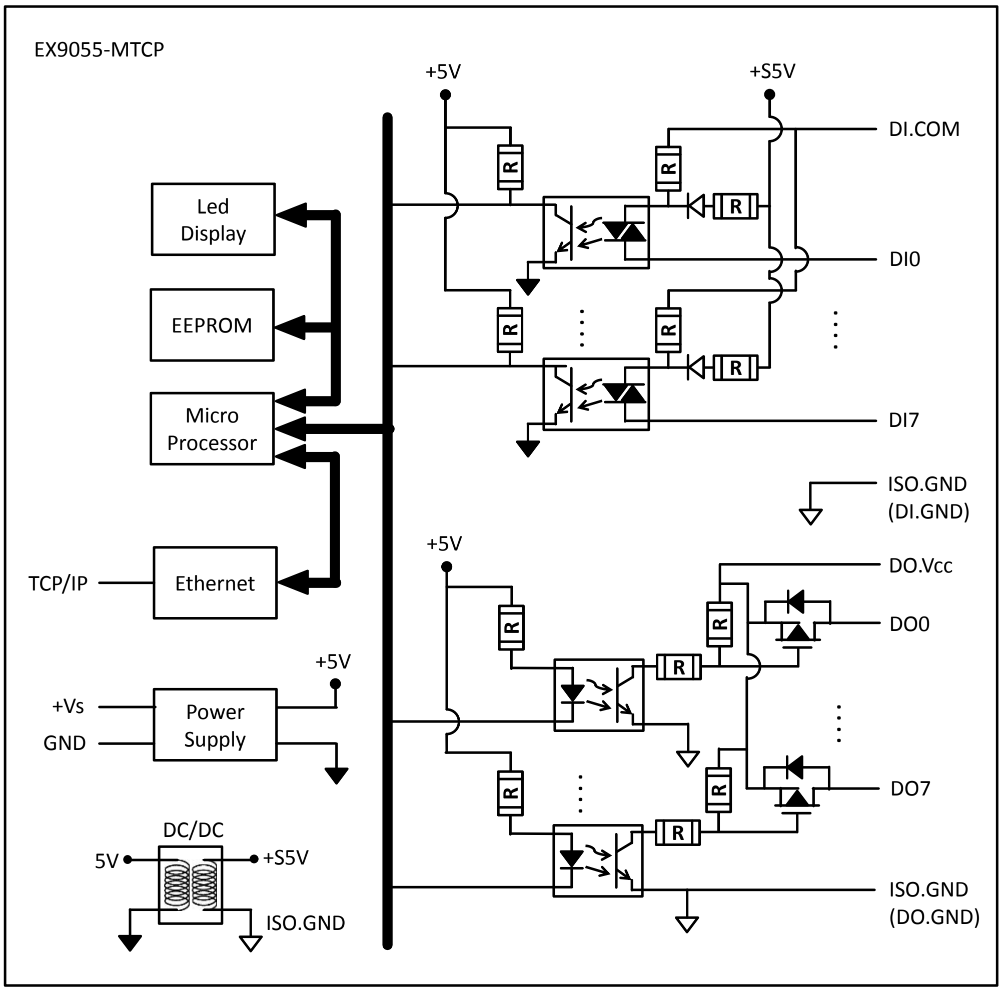
Digital Output: 8
Output Type : P-MOSFET(open drain) up to 30V/ 1A max. load

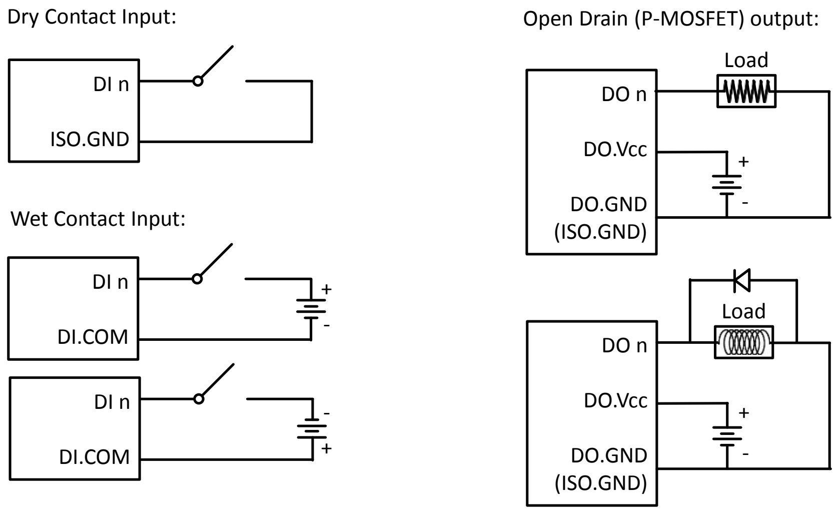
Optical Isolation: 3000 VDC

Power requirements: Unregulated +10 ~ +30 VDC

Power consumption: 2 W

Application Wiring:





Assigning ModBus addresses

Based on Modbus/TCP, the addresses of the I/O channels in EX9000-MTCP modules are defined by a simple rule. Please Refer 7.0 to map the I/O address. All digital input channels in 9055MTCP are allowed to use as 32-bit counters (Each counter is consisted of two addresses, Low word and High word). Users could configure the specific DI channels to be counters via Windows Utility. (Refer to 5.3)

2.7 EX9060-MTCP 5-channel Digital Input and 3 RELAY output Module

The EX9060-MTCP is a high-density digital I/O module designed with a 10/100 based-T interface for seamless Ethernet connectivity. It provides 5 isolated digital input channels and 3 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Meanwhile these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common..

9060MTCP Specification

Channel : 5 channels (DI0~DI4
Input Type: Logic level status can be inversed via ASCII/Modbus command
Digital Input: 5 (Isolated single ended with common source/ground)

Dry Contact

Logic level 0: Close to GND
Logic level 1: Open

Wet Contact

Logic level 0: +2 VDC max
Logic level 1: +10VDC to 50 VDC

Input Impedance : 3K ohm(Wet Contact)
Supports 500 kHz counter input (32-bit + 1-bit overflow)
Optical Isolation Voltage : 3750Vrms

Relay Output: RL1, RL2, RL3
Output channels : 3 relay output channels(RL1; Form A, RL2,RL3 Form C)

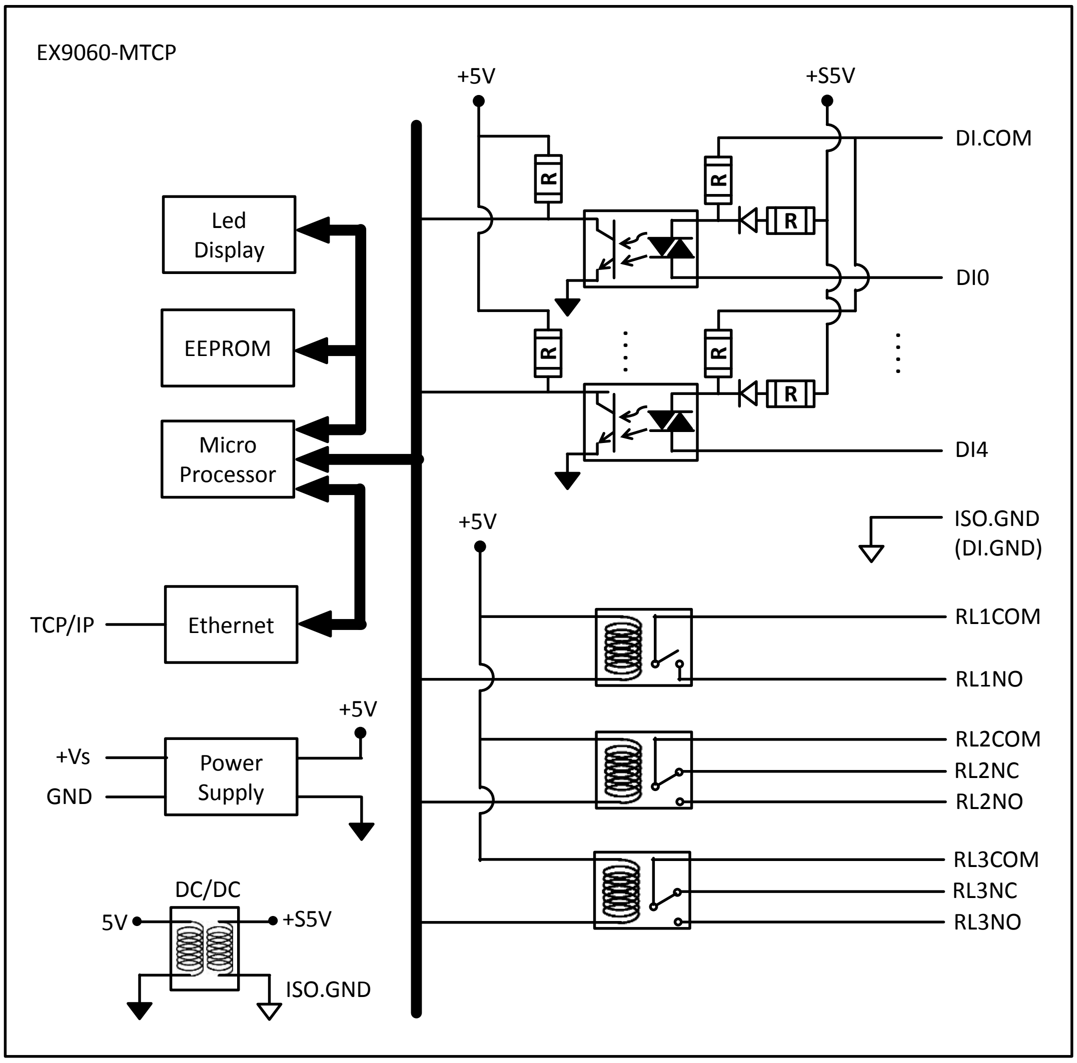
Surge strength : 500 V

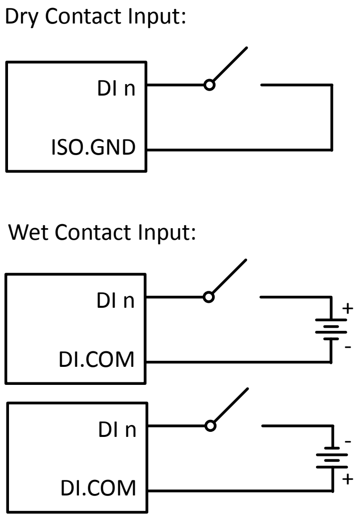
Relay contact rating: 0.6A/125Vac, 2A/30Vdc
Operate Time: 3mS max
Release Time: 2mS max

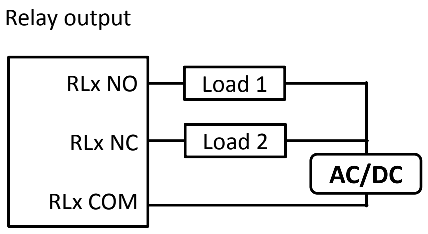
Min Life: $5*10^{5}$ ops
Pulse Output: Each channel supports 500Hz pulse output

Power Consumption: 2.0 W (Typical)

Application Wiring:







Assigning ModBus addresses

Based on Modbus/TCP, the addresses of the I/O channels in EX9000-MTCP modules are defined by a simple rule. Please Refer 7.0 to map the I/O address. All digital input channels in 9060MTCP are allowed to use as 32-bit counters (Each counter is consisted of two addresses, Low word and High word). Users could configure the specific DI channels to be counters via Windows Utility

2.8 EX9063-MTCP 7-channel Digital Input and 3 RELAY output Module

The EX9063-MTCP is a high-density digital I/O module designed with a 10/100 based-T interface for seamless Ethernet connectivity. It provides 7 isolated digital input channels and 3 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Meanwhile these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common..

9063MTCP Specification

Channel : 7 channels (DI0~DI6)
Input Type: Logic level status can be inversed via ASCII/Modbus command
Digital Input: 5 (Isolated single ended with common source/ground)

Dry Contact :

Logic level 0: Close to GND
Logic level 1: Open

Wet Contact

Logic level 0: +2 VDC max
Logic level 1: +10VDC to 50 VDC

Input Impedance : 10K ohm(Wet Contact)
Supports 500 kHz counter input (32-bit + 1-bit overflow)
Optical Isolation Voltage : 3750Vrms

Relay Output: RL1, RL2, RL3(DO0~DO2)
Output channels : 3 relay output channels(Form A)

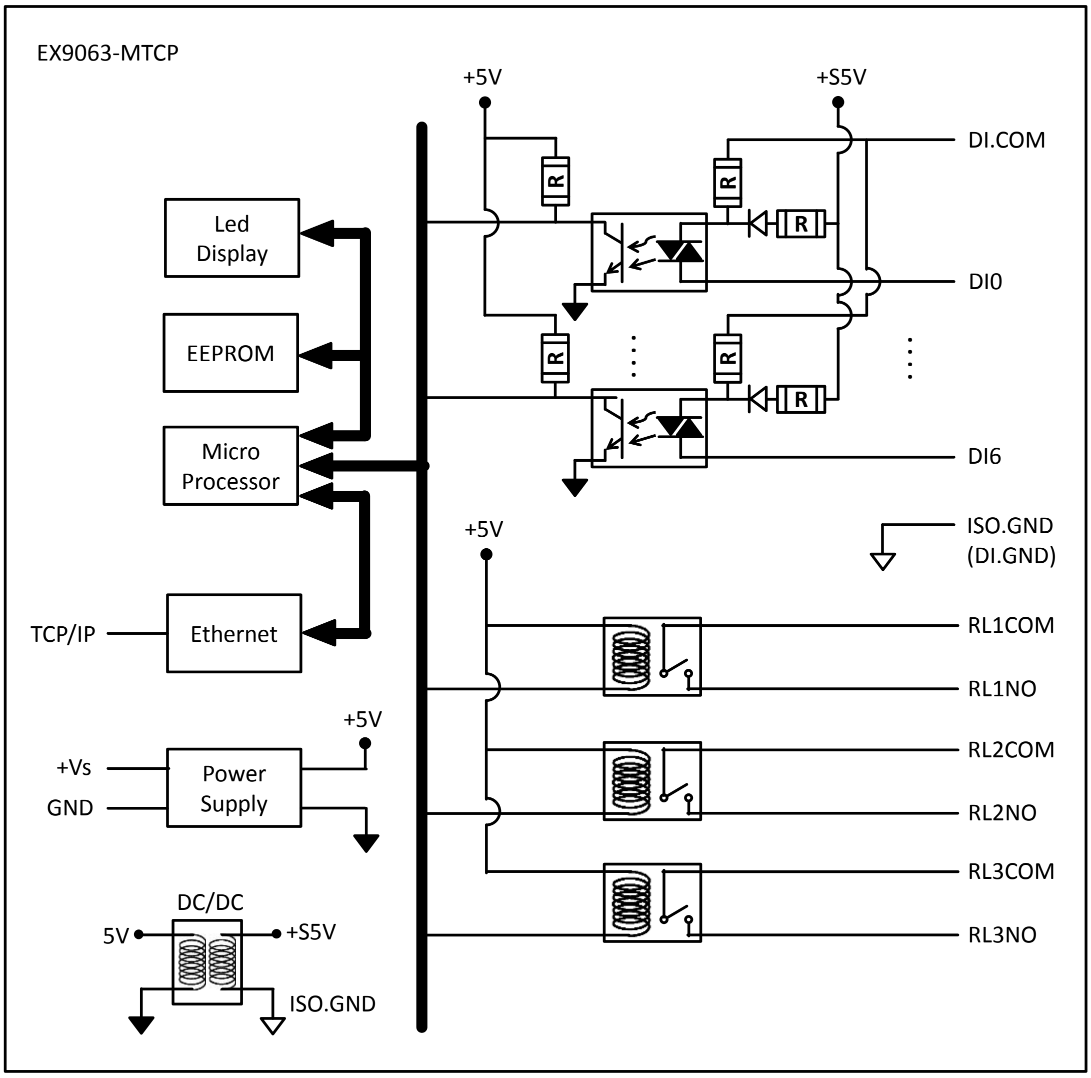
Surge strength : 4000 V

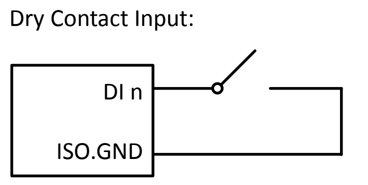
Relay contact rating: 5A/250Vac, 5A/30Vdc
Operate Time: 6mS max.
Release Time: 3mS max.

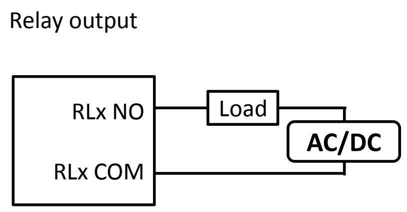
Min Life: $5*10^{5}$ ops
Pulse Output: Each channel supports 500Hz pulse output

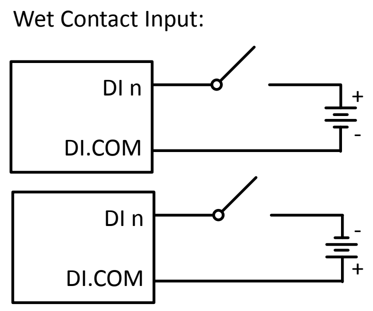
Power Consumption: 2.4 W (Typical)

Application Wiring:









Assigning ModBus addresses

Based on Modbus/TCP, the addresses of the I/O channels in EX9000-MTCP modules are defined by a simple rule. Please Refer 7.0 to map the I/O address. All digital input channels in 9063MTCP are allowed to use as

32-bit counters (Each counter is consisted of two addresses, Low word and High word). Users could configure the specific DI channels to be counters via Windows Utility.

2.9 EX9064-MTCP 5-channel Digital Input and 4 RELAY output Module

The EX9064-MTCP is a high-density digital I/O module designed with a 10/100 based-T interface for seamless Ethernet connectivity. It provides 5 isolated digital input channels and 4 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Meanwhile these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common..

9064MTCP Specification

Channel : 5 channels (DI0~DI4)
Input Type: Logic level status can be inversed via ASCII/Modbus command
Digital Input: 5 (Isolated single ended with common source.)

Dry Contact :

Logic level 0: open
Logic level 1: Close to GND

Wet Contact

Logic level 0: +2 VDC max
Logic level 1: +4VDC to 30 VDC

Input Impedance : 3K ohm(Wet Contact)
Supports 500 kHz counter input (32-bit + 1-bit overflow)
Optical Isolation Voltage : 3750Vrms

Relay Output: RL1, RL2, RL3, RL4
Output channels : 4 relay output channels(Form A)

Surge strength : 500 V

Relay contact rating: 0.6A/125Vac, 2A/30Vdc
Operate Time: 3mS max.
Release Time: 2mS max.

Min Life: $5*10^{5}$ ops
Pulse Output: Each channel supports 500Hz pulse output

Power Consumption: 1.8 W (Typical)

Chapter 3 EX9000-MTCP Utility Guide

In order to properly configure EX9000-MTCP series. You will need following items to complete your system hardware configuration.

3.1 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 or higher versions
- At least 32 MB RAM
- 20 MB of hard disk space available
- VGAcolor 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 EX9000-MTCP (+10 to +30 V unregulated)
- Make sure to prepare all of the items above, then connect the power and network wiring as Figure 3-1 Power wiring.

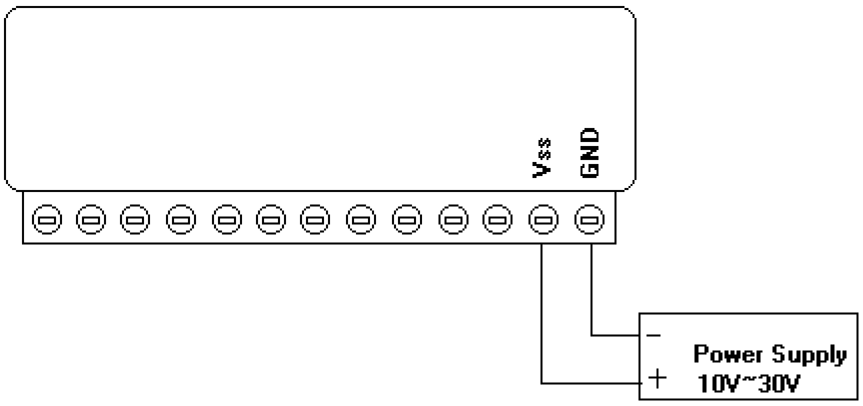


Figure 3-1 Power wiring

3.2Install Utility Software on Host PC

ExpertDAQ provides free download Manual and Utility software for EX9000-MTCP modules' operation and configuration. Link to the web site: www.topsccc.com.tw and click into the "Download Area" to get the latest version EX9000-MTCP 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.

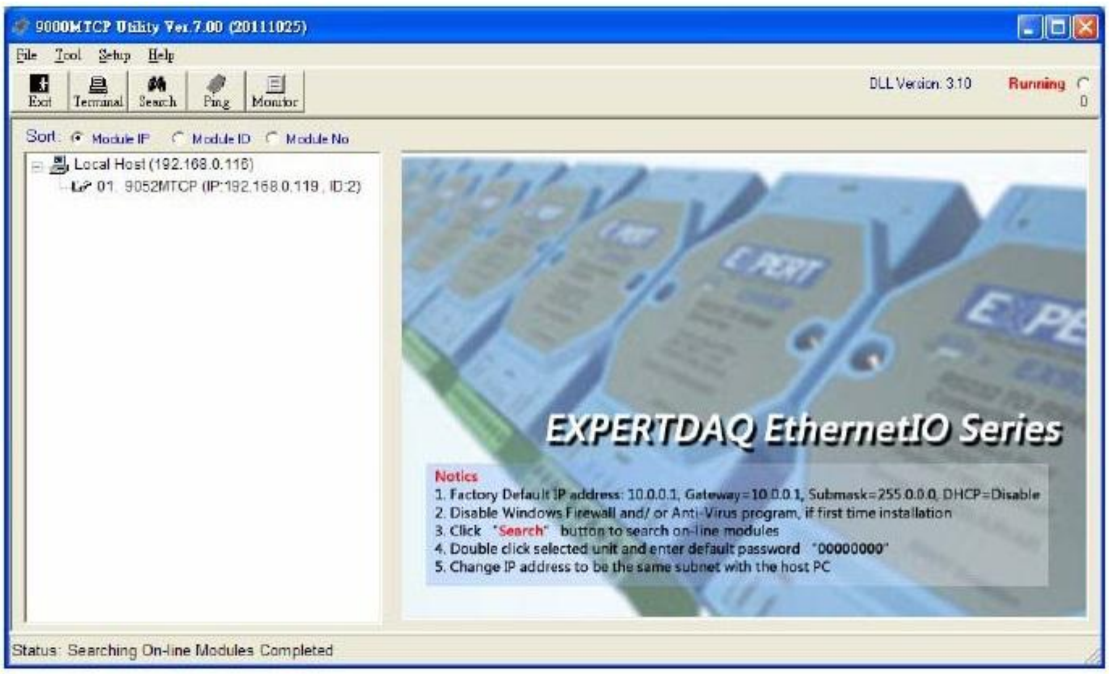
3.3 Ethernet I/O Utility Overview

The Utility software offers a graphical interface that helps you configure the EX9000-MTCP modules. It is also very convenient to test and monitor your remote ExpertDAQ 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

3.4 Main Menu

Double Click the icon of EX9000-MTCP thernet I/O Utility shortcut, the Operation screen will pop up as Figure3-2.


9000MTCP Utility Ver.7.00 (20111025)

DLL Version: 3.10 Running 0

Sort: [✓ Module IP] Module ID Module No

Local Host (192.168.0.116)
01. 9052MTCP (IP:192.168.0.119 , ID:2)

EXPERTDAQ EthernetIO Series

Notics
1. Factory Default IP address: 10.0.0.1, Gateway=10.0.0.1, Submask=255.0.0.0, DHCP=Disable
2. Disable Windows Firewall and/ or Anti-Virus program, if first time installation
3. Click "Search" button to search on-line modules
4. Double click selected unit and enter default password "00000000"
5. Change IP address to be the same subnet with the host PC

Status: Searching On-line Modules Completed

Figure3-2 main window

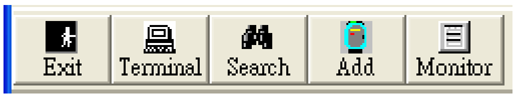
The top of the operation screen consists of a function menu and a tool bar for user's commonly operating functions.

3.5 Function Menu

- ✓ **File** contents "Exit" Function, using to exit this Utility program.
- ✓ **Tool** contents functions as below:
  - ➢ **Search** for Ethernet Device Search all EX9000-MTCP units in the specific Ethernet domination. (The same with host PC's Ethernet domination)
  - ➢ **Add Remote Ethernet Device:** Create a new EX9000-MTCP module located in other Ethernet domination, both available to local LAN and Internet application.
  - ✓ **Monitor Stream/Event Data:** comes from the remote I/O module
- ✓ **Terminal:** Call up the operation screen of Terminal emulation to do the request / response command execution.
- ✓ **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.

3.6 Tool Bar

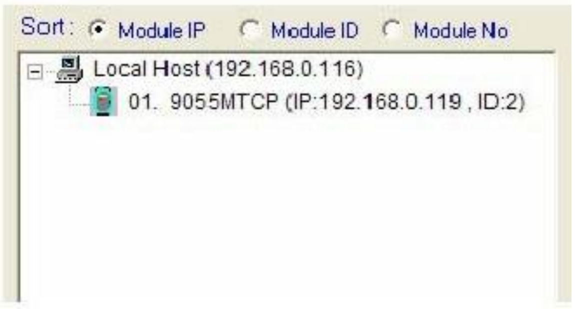
There are five push buttons in the tool bar.



- **Exit:** Exit utility program
- **Terminal:** Terminal emulation
- **Search:** Search ExpertDAQ module
- **Add:** Add remote ExpertDAQ I/O module
- **Monitor :** Monitor the Stream/Event Data

3.7List 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 (192.168.0.116)

01. 9055MTCP (IP:192.168.0.119 , ID:2)

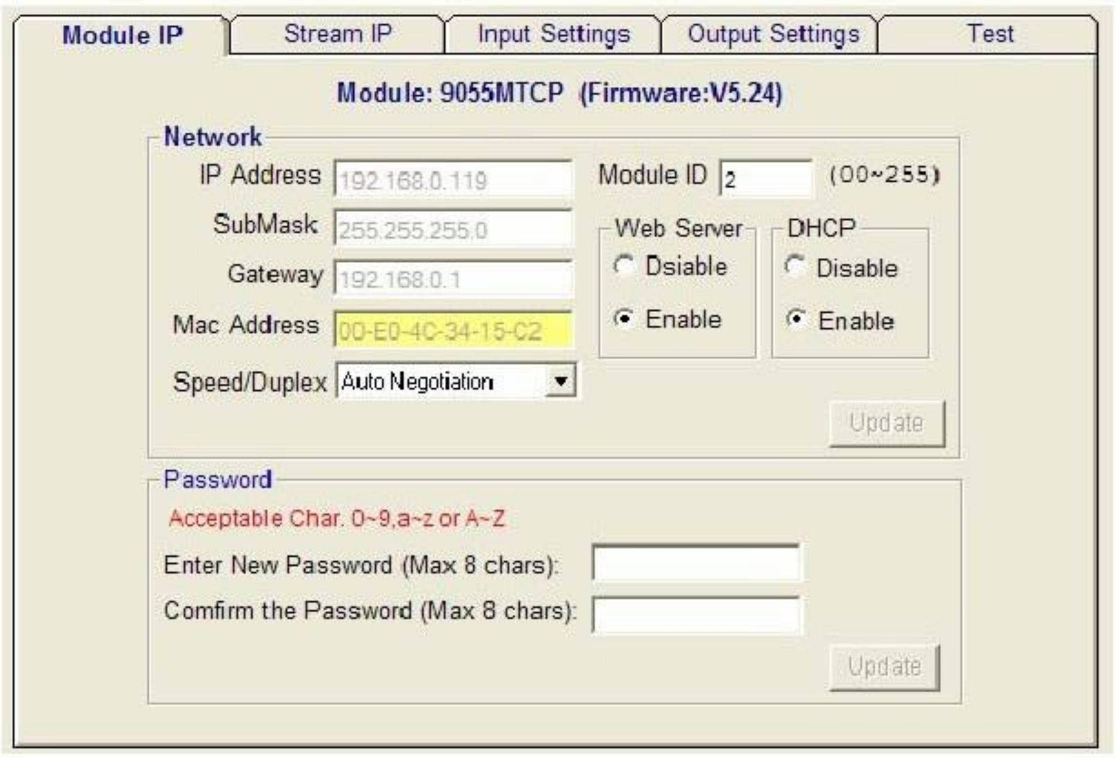
- **Moudle IP:** Sort by moudle IP
- **Module ID:** Sort by module ID
- **Module No:** Sort by module name

3.8 Network Setting

As the moment you start up this Windows Utility, it will search all EX9000-MTCP 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 EX9000-MTCP 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



Module: 9055MTCP (Firmware:V5.24)

Network

IP Address: 192.168.0.119

SubMask: 255.255.255.0

Gateway: 192.168.0.1

Mac Address: 00-E0-4C-34-15-C2

Speed/Duplex: Auto Negotiation

Module ID: 2 (00~255)

Web Server: Dsiable [✓ Enable]

DHCP: Disable [✓ Enable]

Password

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

Enter New Password (Max 8 chars):

Comfirm the Password (Max 8 chars):

Figure 3-3

3.8.1 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, 255.0.0.0 and 10.0.0.1)

The IP address identifies your EX9000-MTCP devices on the global network. Each EX9000-MTCP has same default IP address 10.0.0.1. Therefore, please do not initial many EX9000-MTCP at the same time to avoid the Ethernet collision. If you want to configure the EX9000-MTCP in the host PC's dominating network, only the IP address and Subnet Mask will need to set (The host PC and EX9000-MTCP Ethernet I/O must belong to same subnet Mask).

If you want to configure the EX9000-MTCP 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 EX9000-MTCP with the individual setting.

DHCP: (default Disabled)

Allow you to get IP address from the DHCP servo 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.

Web Server: (default Disabled)

Allow you monitor and control I/O status on EX9000-MTCP modules remotely through web browser.

Module ID: (default 00)

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) to get correct IP address for each ID number

Password: (default 00000000)

Allow you to change the password of the module

3.8.2 TCP/IP port:

EX9000-MTCP 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	HTTPD (web)

3.8.3 Stream/Event IP

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

Active Stream time interval: (default 1 sec)
set time interval for sending stream data

3.8.4 Input or Output Settings:

Configure Input or output channel type

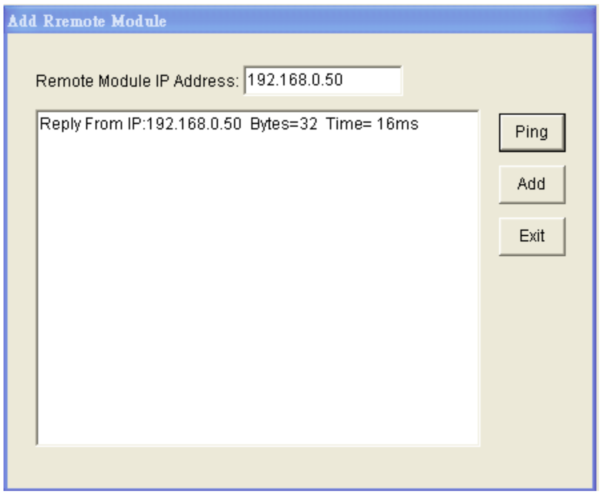
3.8.5 General Settings:

Misc. Setting and status (value) display

3.9 Add Remote Stations

To meet the remote monitoring and maintenance requirements, The EX9000-MTCP 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 EX9000-MTCP 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 Figure3-4. Then key-in the specific IP address and click the "Ping" button. If the communication success, click "Add" to add EX9000-MTCP
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

Figure3-4 Add remote module

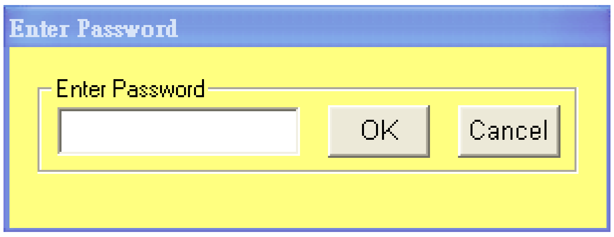
Note:

There is several conditions need to be sure before adding a remote EX9000-MTCP system in the windows 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.

3.10 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 EX9000-MTCP I/O modules. Once user setting the password into the EX9000-MTCP firmware, the important system configurations (Network, Firmware, Password) are only allowed to be changed by password verification.



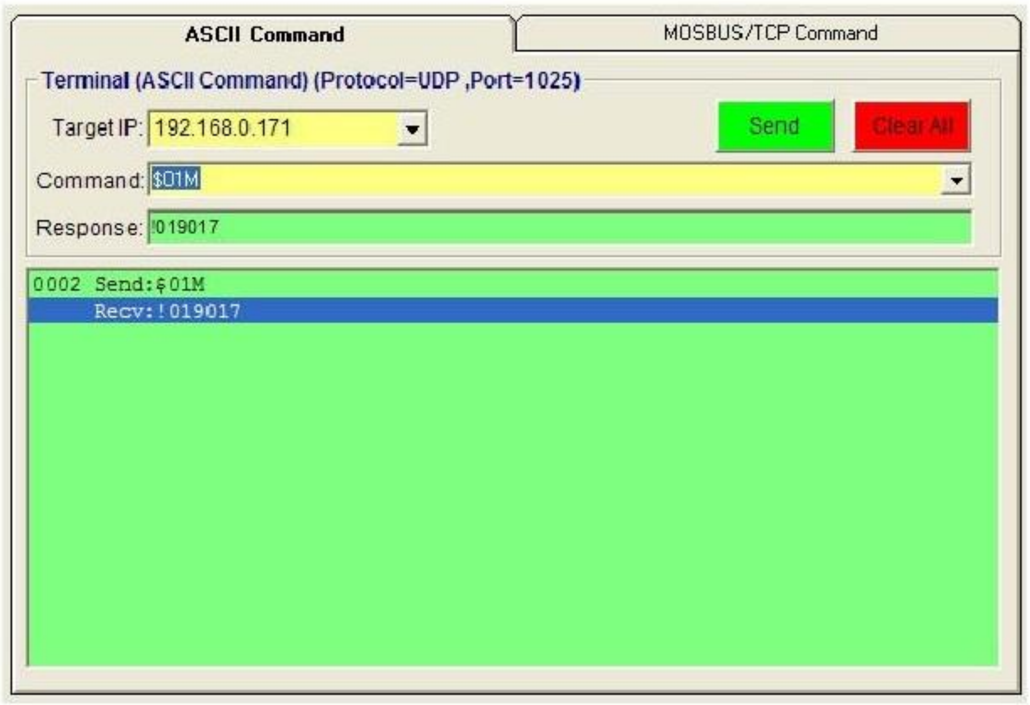
Note:

The default password of EX9000-MTCP is "**00000000**". Please make sure to keep the correct password by yourself. If you lose it, please contact to TOPS CCC's technical support center for help.

3.11 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 Figure 3-5.



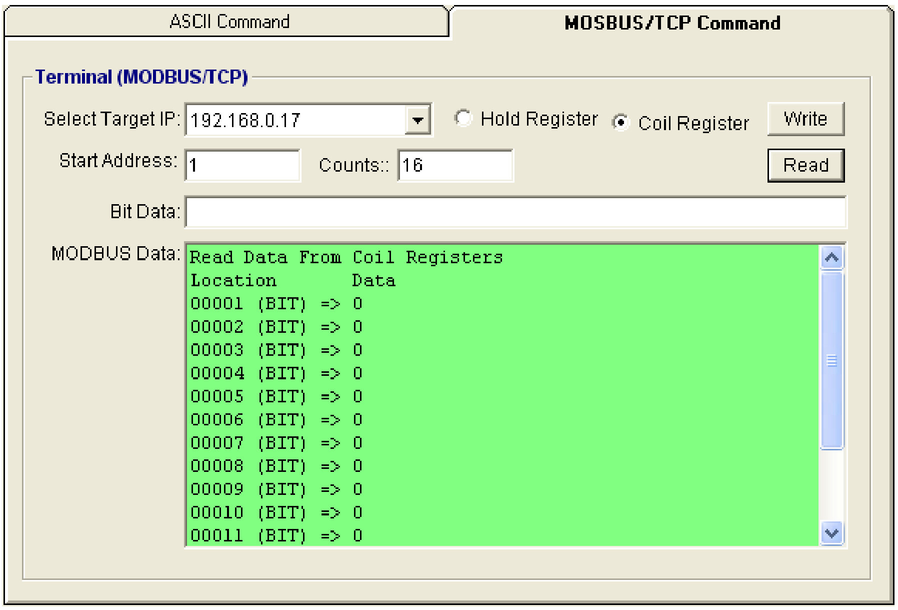
Target IP: 192.168.0.171

Command: $01M

Response: !019017

```
0002  Send:$01M
      Recv:!019017
```

ModBus Hexadecimal mode: shown as Figure 3-6.



Select Target IP: 192.168.0.17

[✓ 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 3-6 ModBus Terminal

3.12 Data /Event Stream

Data Stream Configuration

In addition to TCP/IP communication protocol, EX9000-MTCP 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 EX9000-MTCP 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, EX9000-MTCP 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 EX9000-MTCP 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 3-7 Stream display.

Select the IP address of the EX9000-MTCP you want to read data, then click "Start " button. The Utility software will begin to receive the stream data on this operation display.

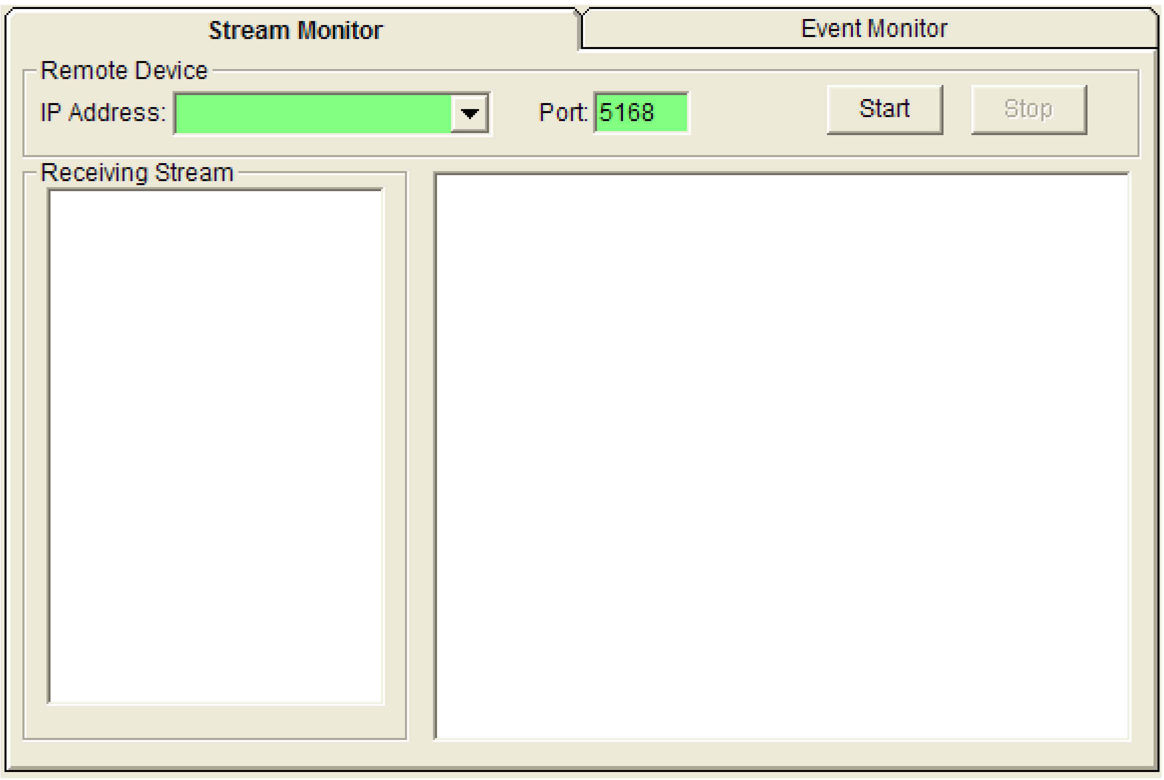


Figure 3-7 Stream display

Data Event Monitoring

After finishing the configuration of Data Event, you can select the tab "Event Monitor" in the function bar or click icon to call up operation display as Figure 3-8 Event display.

Select the IP address of the EX9000-MTCP you want to read data, then click "Start" button. The Utility software will begin to receive the stream data on this operation display.

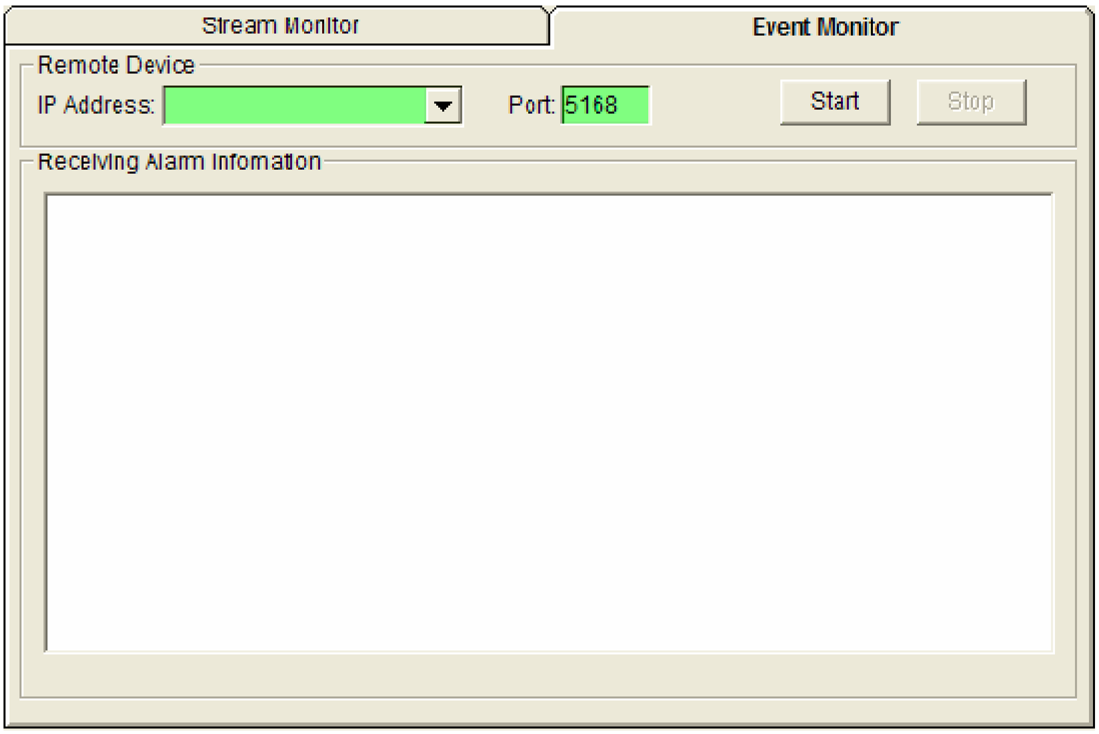
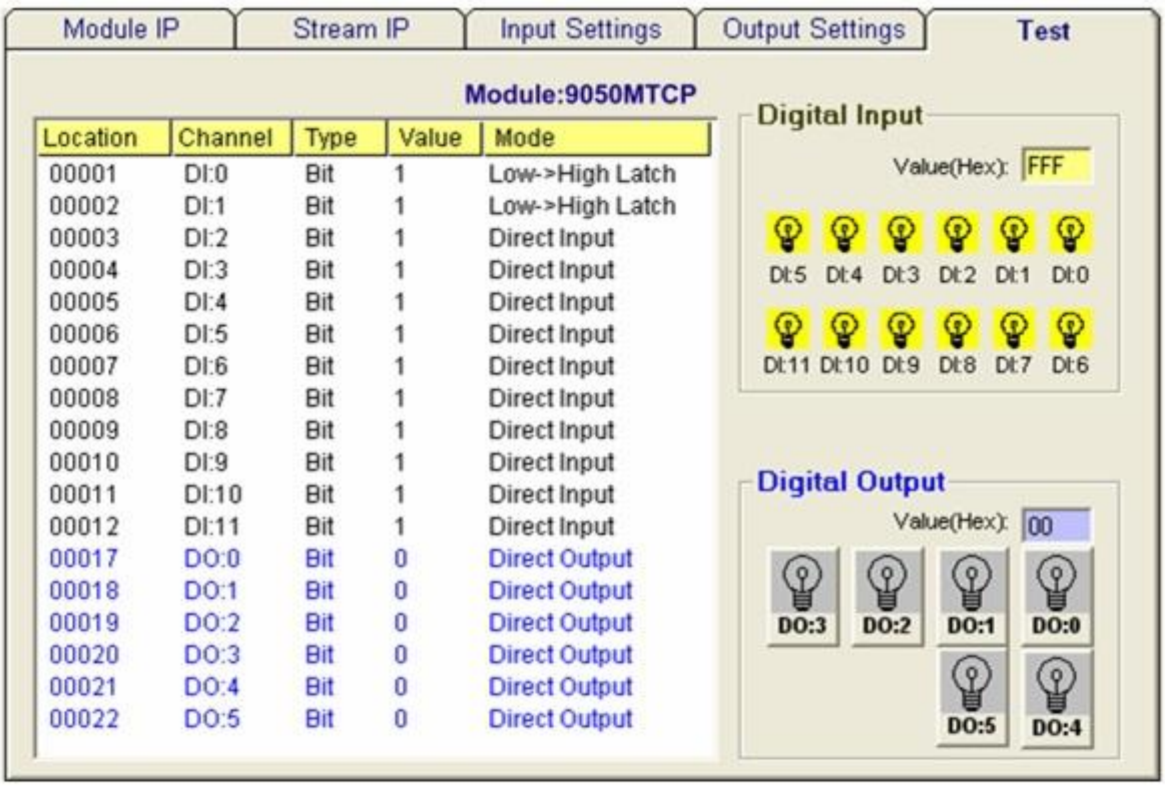


Figure 3-8 Event display

3.13 I/O Module Configurations

3.13.1 Digital Input/Output Module

Selecting EX9000-MTCP Digital Modules and select "Test" tab, user can read following information from the Utility.



Module:9050MTCP

Location	Channel	Type	Value	Mode
00001	DI:0	Bit	1	Low->High Latch
00002	DI:1	Bit	1	Low->High Latch
00003	DI:2	Bit	1	Direct Input
00004	DI:3	Bit	1	Direct Input
00005	DI:4	Bit	1	Direct Input
00006	DI:5	Bit	1	Direct Input
00007	DI:6	Bit	1	Direct Input
00008	DI:7	Bit	1	Direct Input
00009	DI:8	Bit	1	Direct Input
00010	DI:9	Bit	1	Direct Input
00011	DI:10	Bit	1	Direct Input
00012	DI:11	Bit	1	Direct Input
00017	DO:0	Bit	0	Direct Output
00018	DO:1	Bit	0	Direct Output
00019	DO:2	Bit	0	Direct Output
00020	DO:3	Bit	0	Direct Output
00021	DO:4	Bit	0	Direct Output
00022	DO:5	Bit	0	Direct Output

Digital Input Value(Hex): FFF

Digital Output Value(Hex): 00

Figure 3-9 ModBus location and I/O status

Location: Standard Modbus address. EX9000-MTCP 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 Figure3-10. 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.

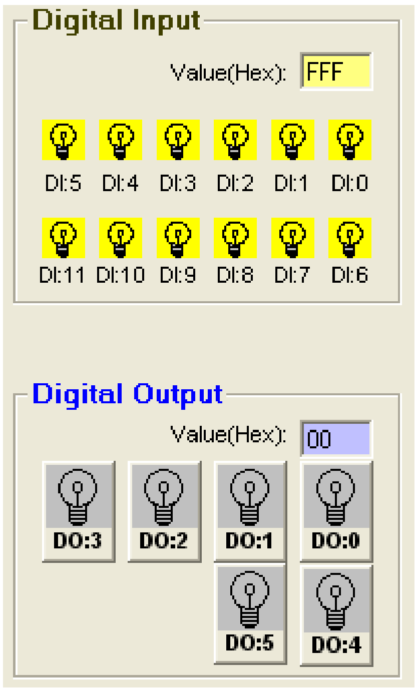


Figure3-10 DI/O status display

The digital input channels support counter(by software) and signal latch functions. Click the specific channel, there will be four working modes for choosing.

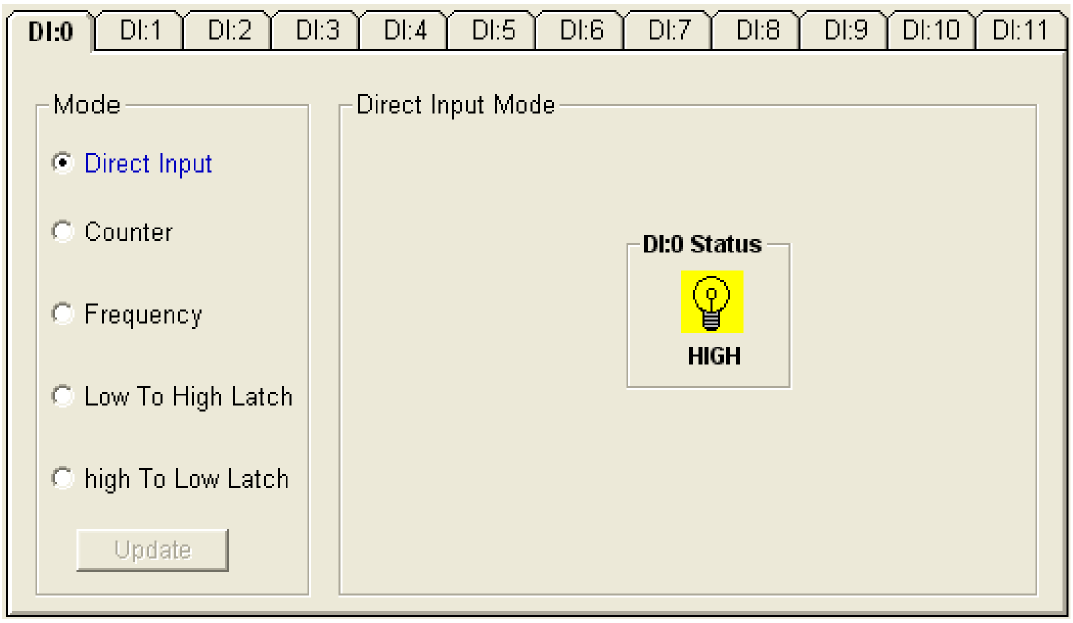
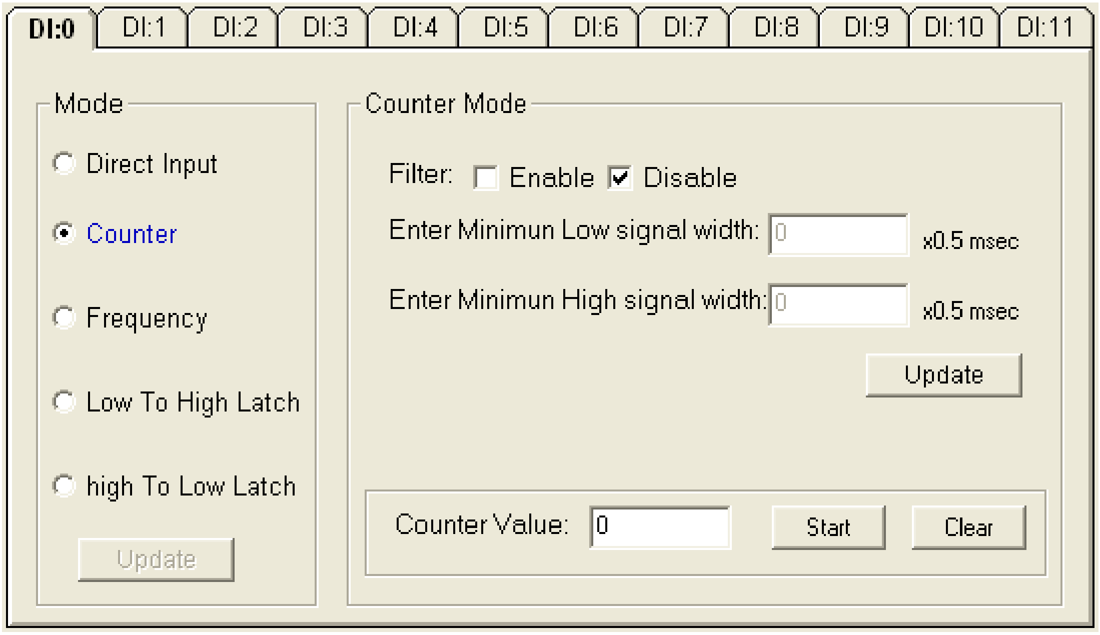


Figure 3-11 Direct input mode



Mode
- Direct Input
- [✓ Counter]
- Frequency
- Low To High Latch
- high To Low Latch

Counter Mode

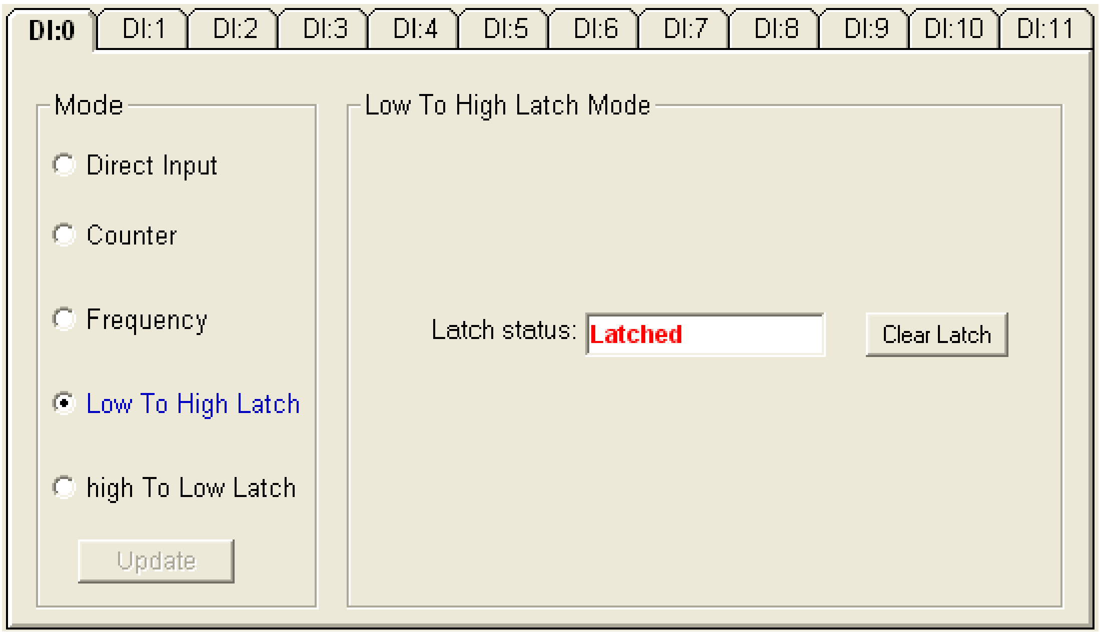
Filter: Enable [✓ Disable]

Enter Minimun Low signal width: 0 x0.5 msec

Enter Minimun High signal width: 0 x0.5 msec

Counter Value: 0

Figure 3-12 Counter setting



Mode
- Direct Input
- Counter
- Frequency
- [✓ Low To High Latch]
- high To Low Latch

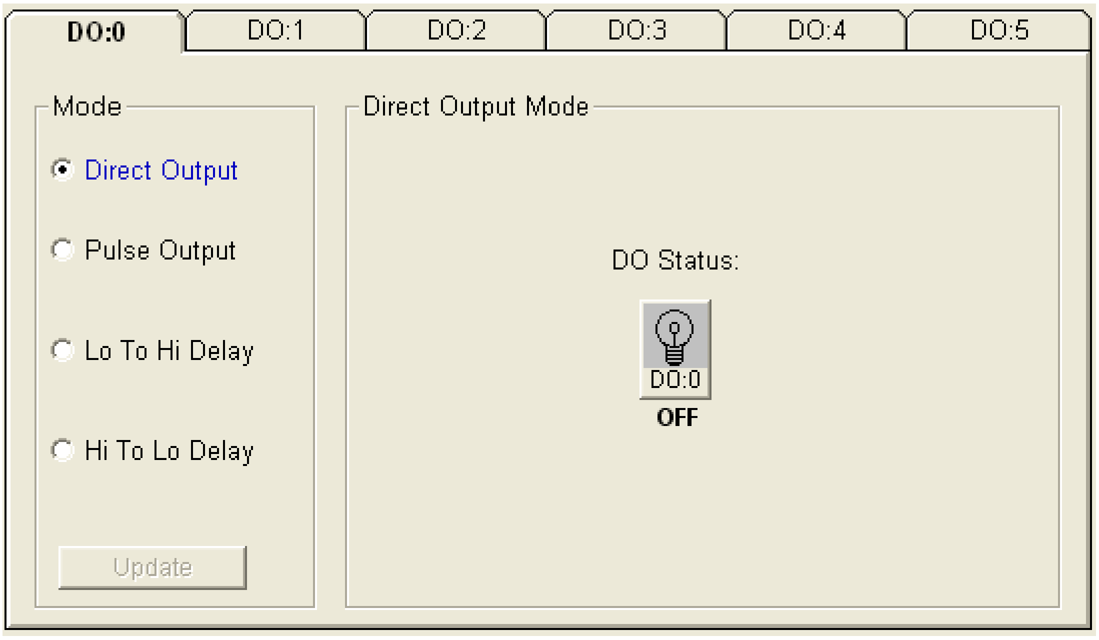
Low To High Latch Mode

Latch status: Latched

Figure 3-13 Input latch setting

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.

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

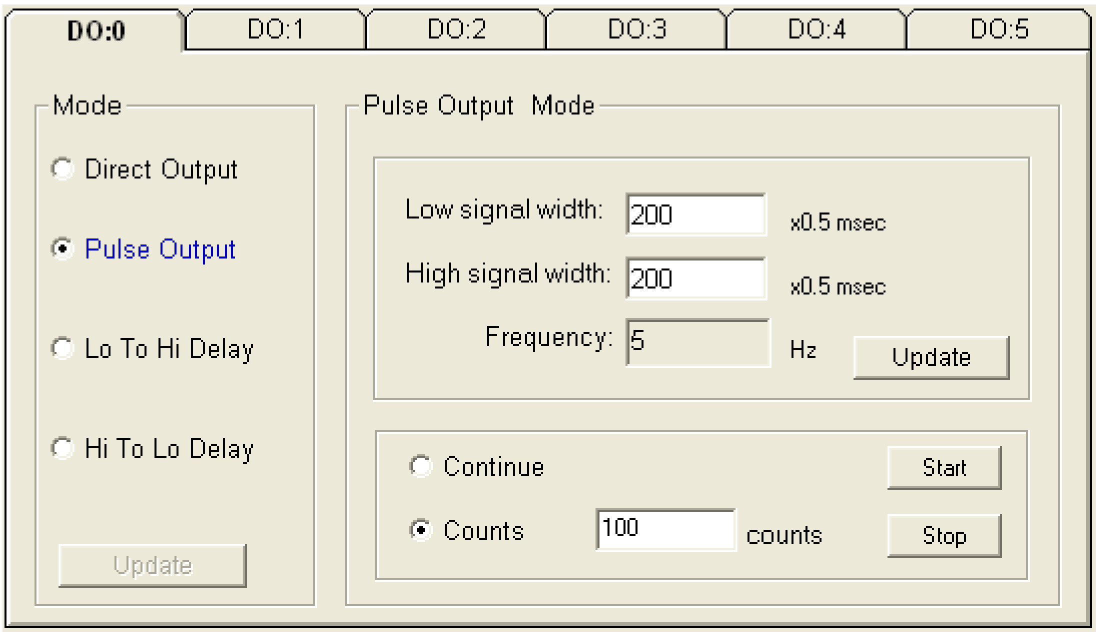


Mode
(● Direct Output) ○ Pulse Output ○ Lo To Hi Delay ○ Hi To Lo Delay

Direct Output Mode

DO Status: DO:0 OFF

Figure 3-14 riect output setting



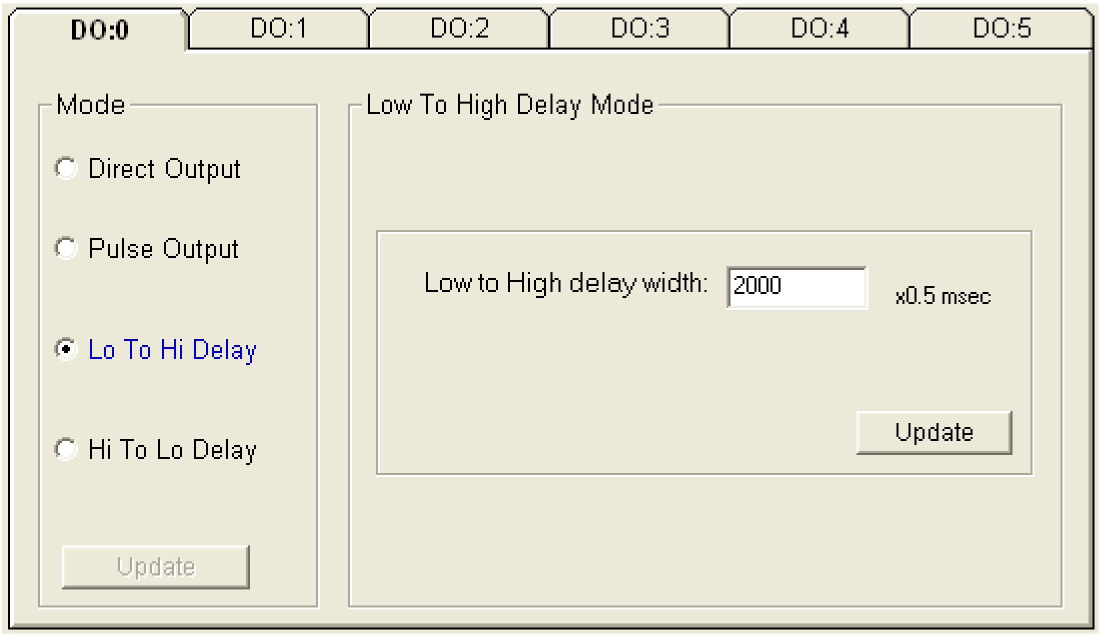
Mode
○ Direct Output (● Pulse Output) ○ Lo To Hi Delay ○ Hi To Lo Delay

Pulse Output Mode

Low signal width: 200 x0.5 msec
High signal width: 200 x0.5 msec
Frequency: 5 Hz

○ Continue
● Counts 100 counts

Figure 3-15 Pulse output setting



Mode
( ) Direct Output
( ) Pulse Output
[✓ Lo To Hi Delay]
( ) Hi To Lo Delay

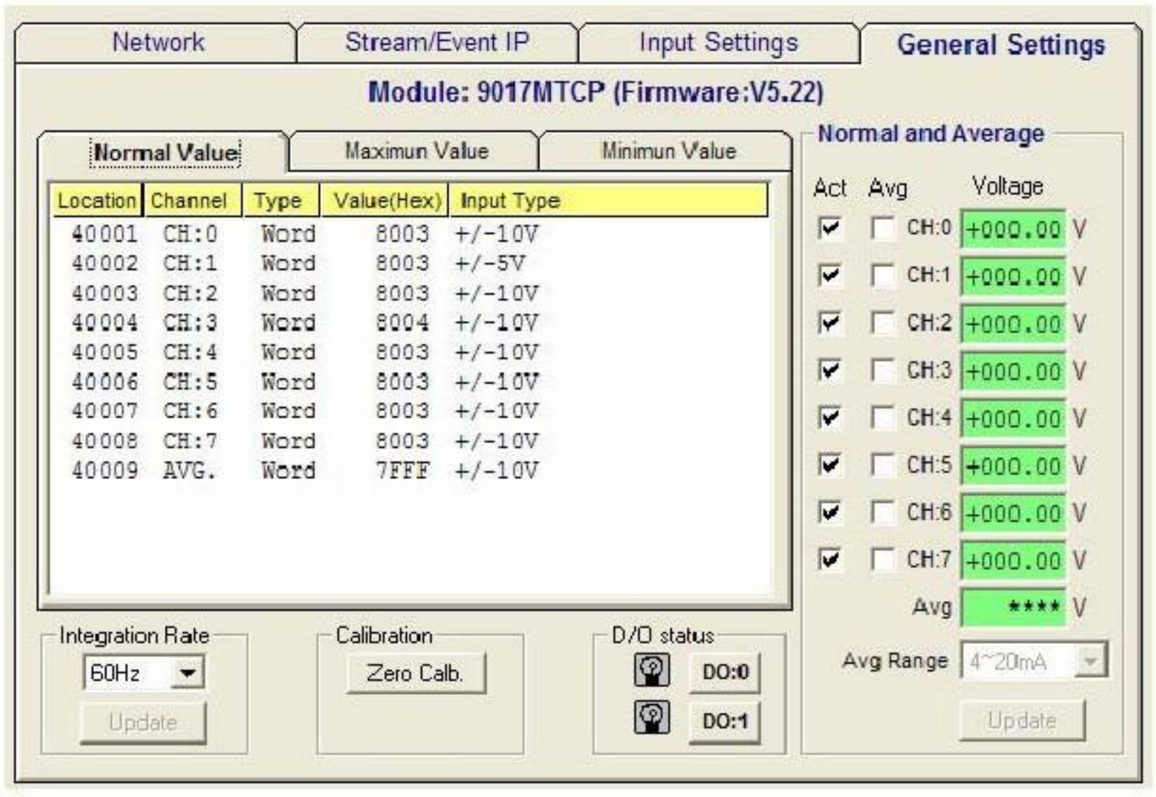
Low To High Delay Mode

Low to High delay width: 2000 x0.5 msec

Figure 3-16 Low to High Delay setting

3.13.2 Analog Input Module

Selecting EX9000-MTCP analog input Modules includes EX9017-MTCP and select **"General Settings"** tab, user can read following information from the Utility.



Module: 9017MTCP (Firmware:V5.22)

Location	Channel	Type	Value(Hex)	Input Type
40001	CH:0	Word	8003	+/-10V
40002	CH:1	Word	8003	+/-5V
40003	CH:2	Word	8003	+/-10V
40004	CH:3	Word	8004	+/-10V
40005	CH:4	Word	8003	+/-10V
40006	CH:5	Word	8003	+/-10V
40007	CH:6	Word	8003	+/-10V
40008	CH:7	Word	8003	+/-10V
40009	AVG.	Word	7FFF	+/-10V

Normal and Average

Act	Avg		Voltage	
✓		CH:0	+000.00	V
✓		CH:1	+000.00	V
✓		CH:2	+000.00	V
✓		CH:3	+000.00	V
✓		CH:4	+000.00	V
✓		CH:5	+000.00	V
✓		CH:6	+000.00	V
✓		CH:7	+000.00	V
		Avg	****	V

Avg Range: 4~20mA

Integration Rate: 60Hz

Calibration

D/O status

Figure 3-17 ModBus location and analog value

Location: Standard Modbus address. (Refer to Assigning address for I/O module in Chapter 4)

Channel: the channel number

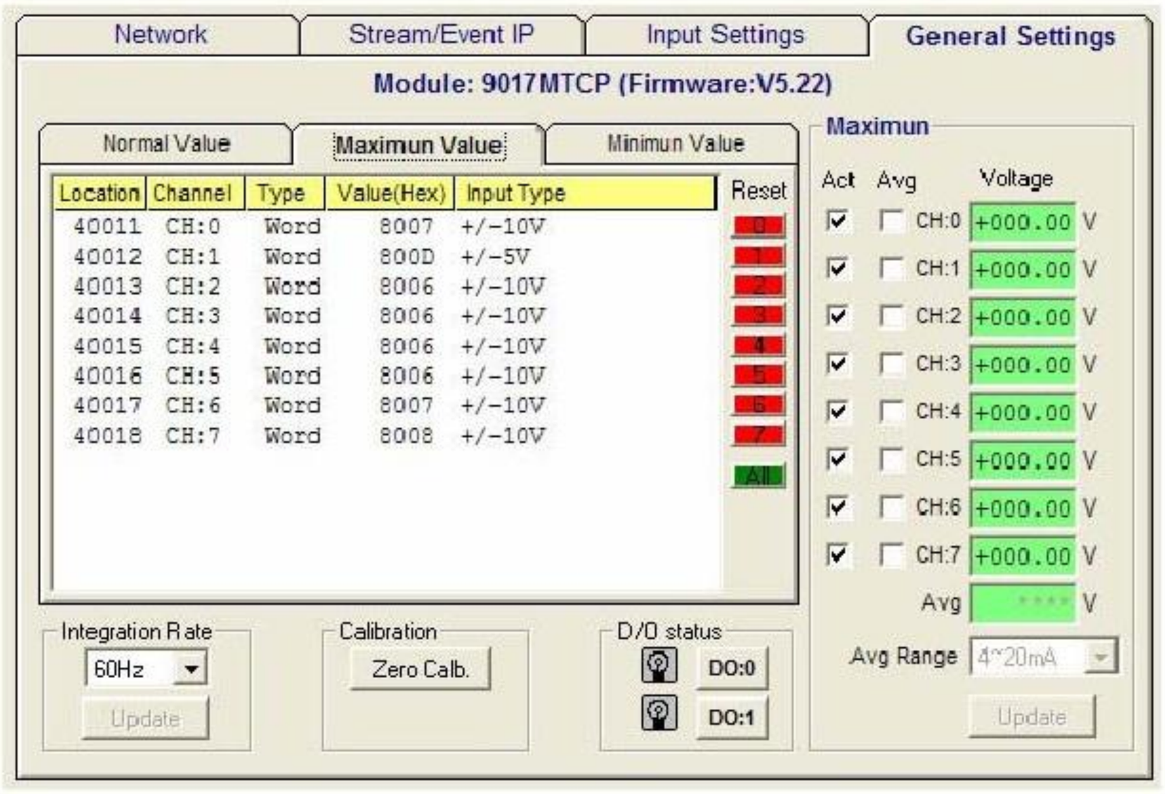
Type: Data type of the I/O channel. The data type of analog Input modules is always "word".

Value: The current status on each channel of I/O modules. Windows Utility provides both decimal and hexadecimal values used for different applications.

Input Type: Sensor types and measurement range of the specified module.

Before acquiring the current data of an analog input module, you have to select the input range and integration time. Then the input data will be scaled as the specified range with engineer unit.

To provide users more valuable information, the EX9000-MTCP analog modules have designed with calculation functions, includes Maximum, Minimum, and Average values of individual channels. Click the Maximum value tab, you will see the historical maximum values in each channel unless to press the against "Reset" buttons.



Module: 9017MTCP (Firmware:V5.22)

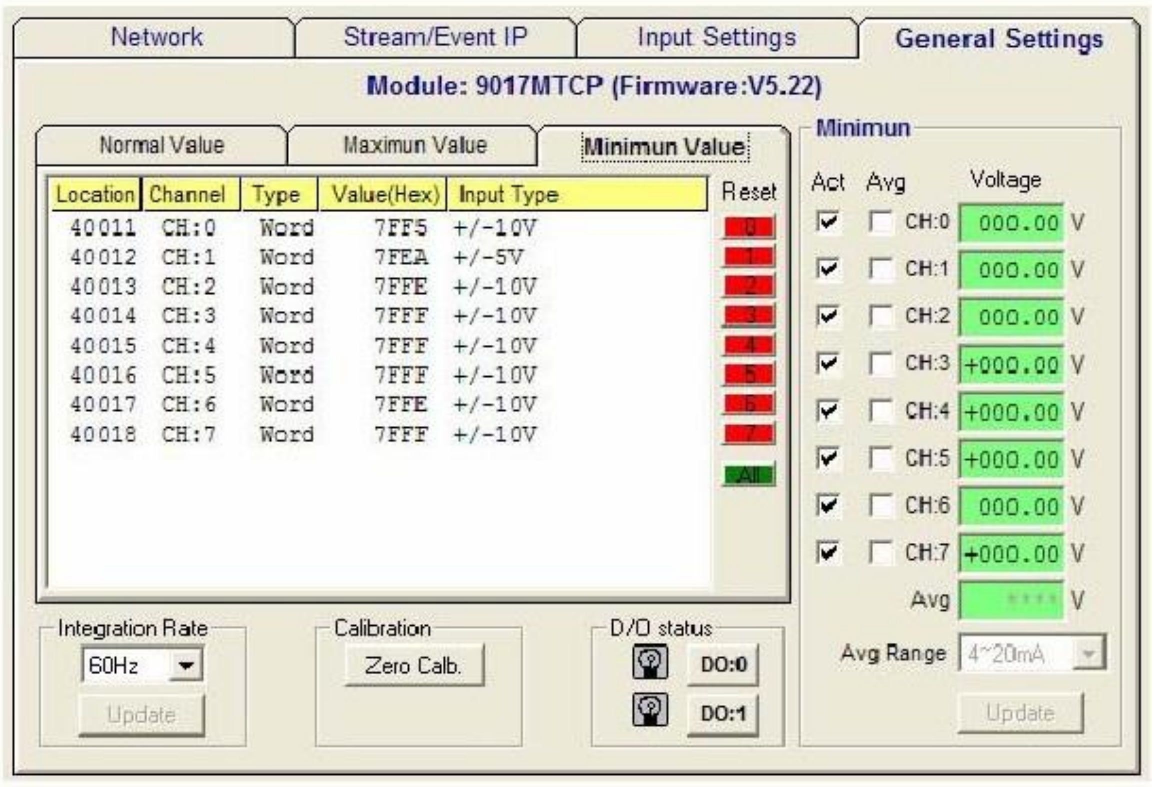
Location	Channel	Type	Value(Hex)	Input Type	Reset
40011	CH:0	Word	8007	+/-10V	0
40012	CH:1	Word	800D	+/-5V	1
40013	CH:2	Word	8006	+/-10V	2
40014	CH:3	Word	8006	+/-10V	3
40015	CH:4	Word	8006	+/-10V	4
40016	CH:5	Word	8006	+/-10V	5
40017	CH:6	Word	8007	+/-10V	6
40018	CH:7	Word	8008	+/-10V	7
					All

Maximun

Act	Avg		Voltage	
✓	☐	CH:0	+000.00	V
✓	☐	CH:1	+000.00	V
✓	☐	CH:2	+000.00	V
✓	☐	CH:3	+000.00	V
✓	☐	CH:4	+000.00	V
✓	☐	CH:5	+000.00	V
✓	☐	CH:6	+000.00	V
✓	☐	CH:7	+000.00	V
		Avg	****	V

Integration Rate: 60Hz
Avg Range: 4~20mA

Click the Minimum value tab, you will see the historical minimum values in each channel unless to press the against "Reset" buttons.



Module: 9017MTCP (Firmware:V5.22)

Location	Channel	Type	Value(Hex)	Input Type	Reset
40011	CH:0	Word	7FF5	+/-10V	0
40012	CH:1	Word	7FEA	+/-5V	1
40013	CH:2	Word	7FFE	+/-10V	2
40014	CH:3	Word	7FFF	+/-10V	3
40015	CH:4	Word	7FFF	+/-10V	4
40016	CH:5	Word	7FFF	+/-10V	5
40017	CH:6	Word	7FFE	+/-10V	6
40018	CH:7	Word	7FFF	+/-10V	7
					All

Minimun

Act	Avg		Voltage	
✓	☐	CH:0	000.00	V
✓	☐	CH:1	000.00	V
✓	☐	CH:2	000.00	V
✓	☐	CH:3	+000.00	V
✓	☐	CH:4	+000.00	V
✓	☐	CH:5	+000.00	V
✓	☐	CH:6	000.00	V
✓	☐	CH:7	+000.00	V
		Avg	****	V

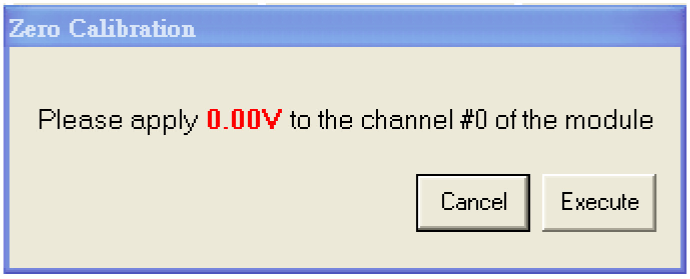
Integration Rate: 60Hz
Avg Range: 4~20mA

3.14 I/O Module Calibrations

Calibration is to adjust the accuracy of ExpertDAQ module. There are several modes for module's calibration: Zero calibration, Span calibration, CJC calibration, and Analog Output calibration. Only analog input and output modules can be calibrated, and the EX9017-MTCP is the first released analog module.

Zero Calibration

1. Apply power to the module and let it warm up for 30 minutes.
3. Make sure the module is correctly installed and properly configured for the input range you want to calibrate.
4. Short channel 0 to GND by wire as short as possible
5. Click the Execute button.


Zero Calibration

Please apply 0.00V to the channel #0 of the module

Span Calibration

1.Follow the same procedure of zero calibration

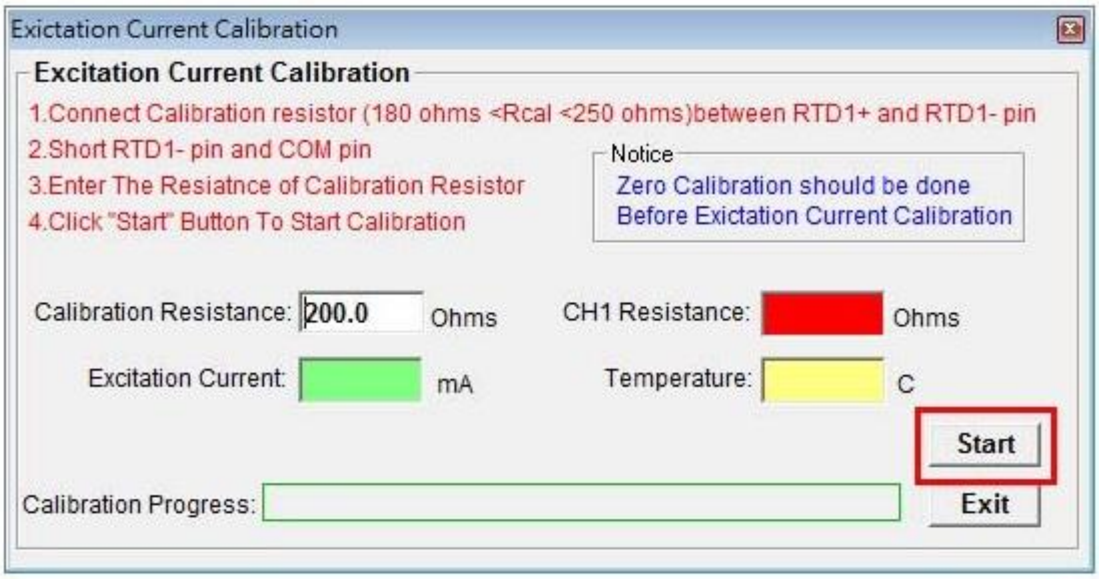
2.Use a precision voltage source to apply a calibration voltage to the V+ and V- terminals of the EX9017-MTCP module.

3.Click the Execute button.

Excitation Current (RTD) Calibration (for EX9015-MTCP):

1. Connect calibration resistor (180ohms<Rcal<250ohms) between CH1 RTD1+ & RTD1- pin, short CH1 RTD1- pin & COM pin.

2. Enter the resistance of calibration resistor and press Start button to start calibration


Exictation Current Calibration

Excitation Current Calibration

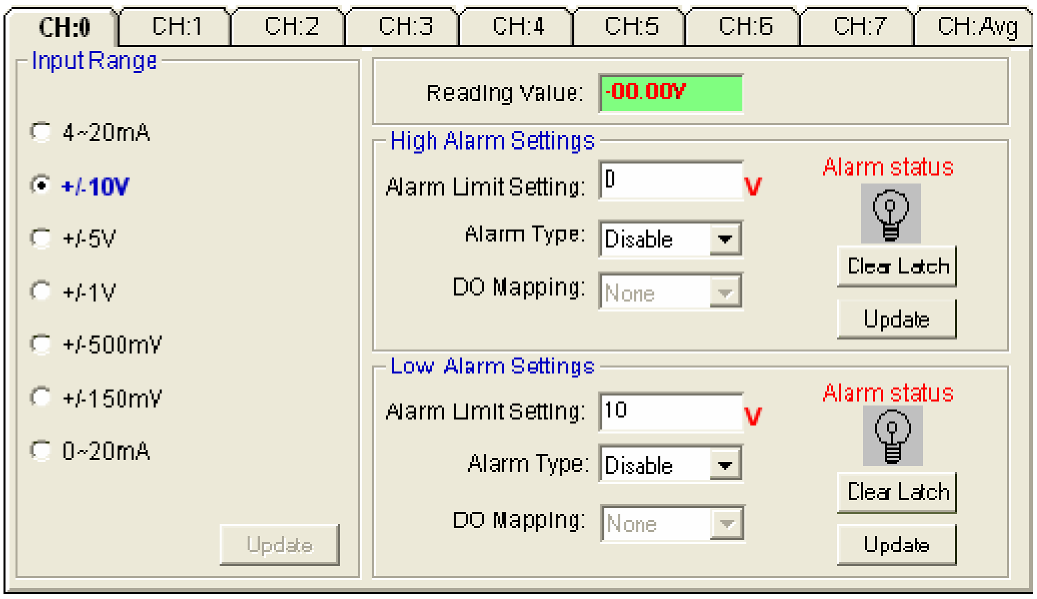
1.Connect Calibration resistor (180 ohms <Rcal <250 ohms)between RTD1+ and RTD1- pin
2.Short RTD1- pin and COM pin
3.Enter The Resiatnce of Calibration Resistor
4.Click "Start" Button To Start Calibration

Notice
Zero Calibration should be done
Before Exictation Current Calibration

Calibration Resistance: 200.0 Ohms
CH1 Resistance: Ohms
Excitation Current: mA
Temperature: C
Calibration Progress:

3.15 Input Type Settings

There is serval range of each channel of analog module. You should select properly type(range) before apply to the your applications


CH:0 CH:1 CH:2 CH:3 CH:4 CH:5 CH:6 CH:7 CH:Avg

Input Range
- 4~20mA
- [✓ +/-10V]
- +/-5V
- +/-1V
- +/-500mV
- +/-150mV
- 0~20mA

Update

Reading Value: -00.00V

High Alarm Settings
Alarm Limit Setting: 0 V
Alarm Type: Disable
DO Mapping: None
Alarm status
Clear Latch
Update

Low Alarm Settings
Alarm Limit Setting: 10 V
Alarm Type: Disable
DO Mapping: None
Alarm status
Clear Latch
Update

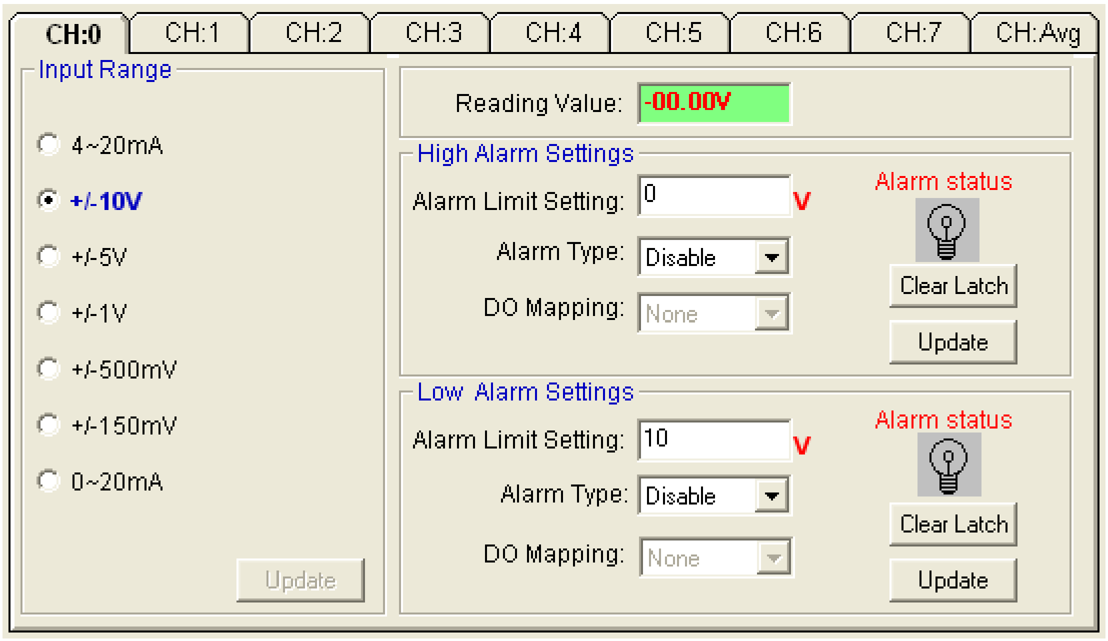
Figure 3-18 Input type setting

Note:

The new working mode setting will take effective after click the "Update" button.

3.16 Alarm Setting

Moreover, all of the analog channels are allowed to configure the High/Low limitation for alarm trigger function. Once the value of the specific channel is over or under the limitation, the alarm status could trigger a digital output channel in the EX9017-MTCP.


CH:0 CH:1 CH:2 CH:3 CH:4 CH:5 CH:6 CH:7 CH:Avg

Input Range
- 4~20mA
- [✓ +/-10V]
- +/-5V
- +/-1V
- +/-500mV
- +/-150mV
- 0~20mA

Update

Reading Value: -00.00V

High Alarm Settings
Alarm Limit Setting: 0 V
Alarm Type: Disable
DO Mapping: None
Alarm status
Clear Latch
Update

Low Alarm Settings
Alarm Limit Setting: 10 V
Alarm Type: Disable
DO Mapping: None
Alarm status
Clear Latch
Update

Figure 3-19 Alarm Setting

Chapter 4 What is TCPDAQ(Ethernet IO) 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 EX9000-MTCP 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 TCPDAQActiveX Control

Use the EX9000-MTCP 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

4.1 Installing the TCPDAQ ActiveX Controls

Before using the TCPDAQActiveX 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

4.2 Building TCPDAQ ActiveX Control with Various Tools

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

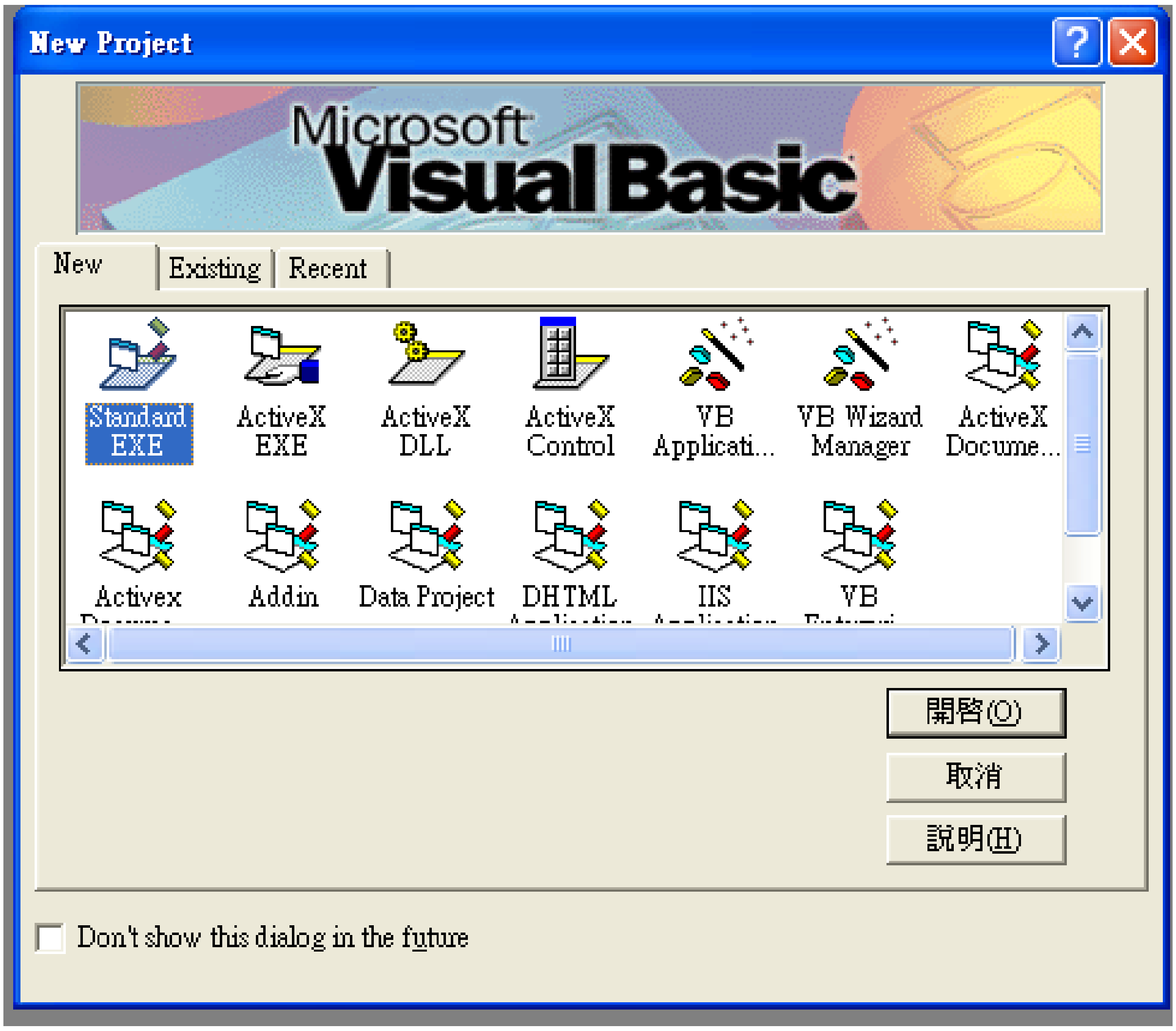
- Microsoft Visual C++ version 6.0 (SP5)
- Microsoft Visual Basic version 6.0 (SP5)
- Borland Delphi version 4.0 (with the Delphi 6 Update Pack fixes forActiveX installed)
- Borland C++ Builder version 5.0

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.

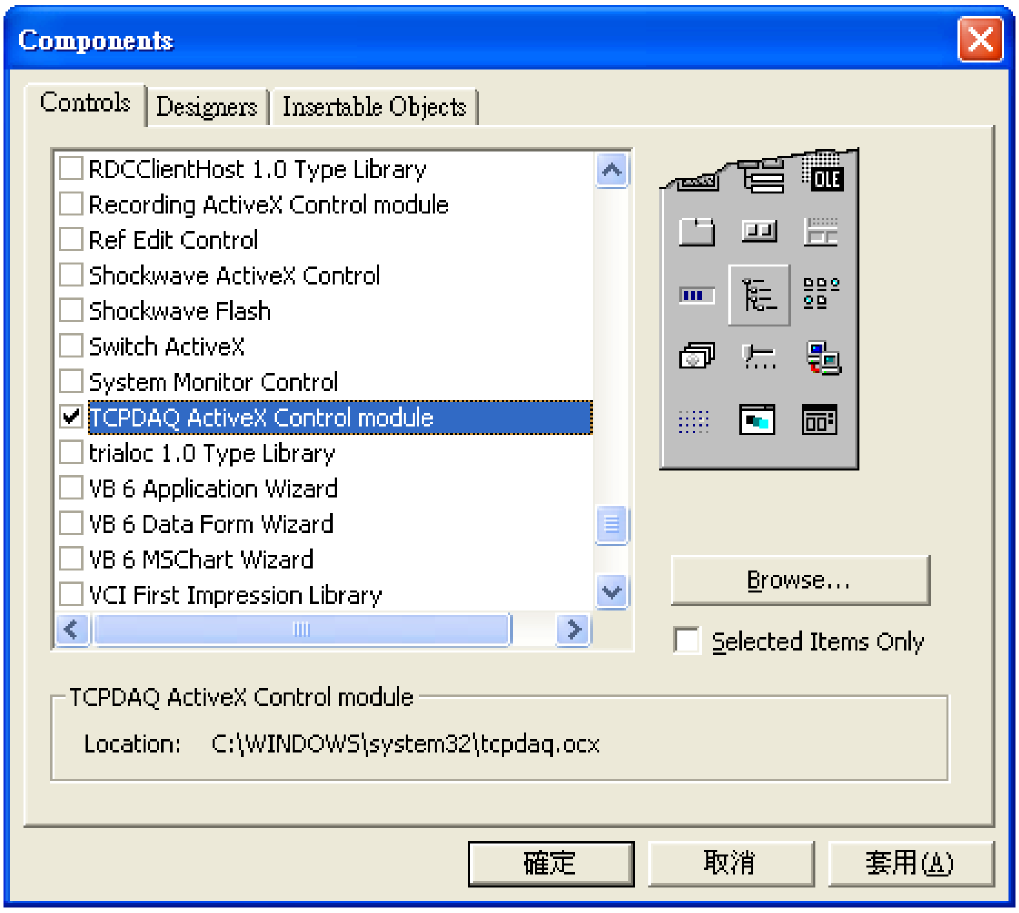
Note: For Borland Delphi 6, the Delphi 6 Update Pack fixes forActiveX must be installed.

4.2.1 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:


Components

Controls | Designers | Insertable Objects

- [ ] RDCClientHost 1.0 Type Library
- [ ] Recording ActiveX Control module
- [ ] Ref Edit Control
- [ ] Shockwave ActiveX Control
- [ ] Shockwave Flash
- [ ] Switch ActiveX
- [ ] System Monitor Control
- [✓] 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

Browse...

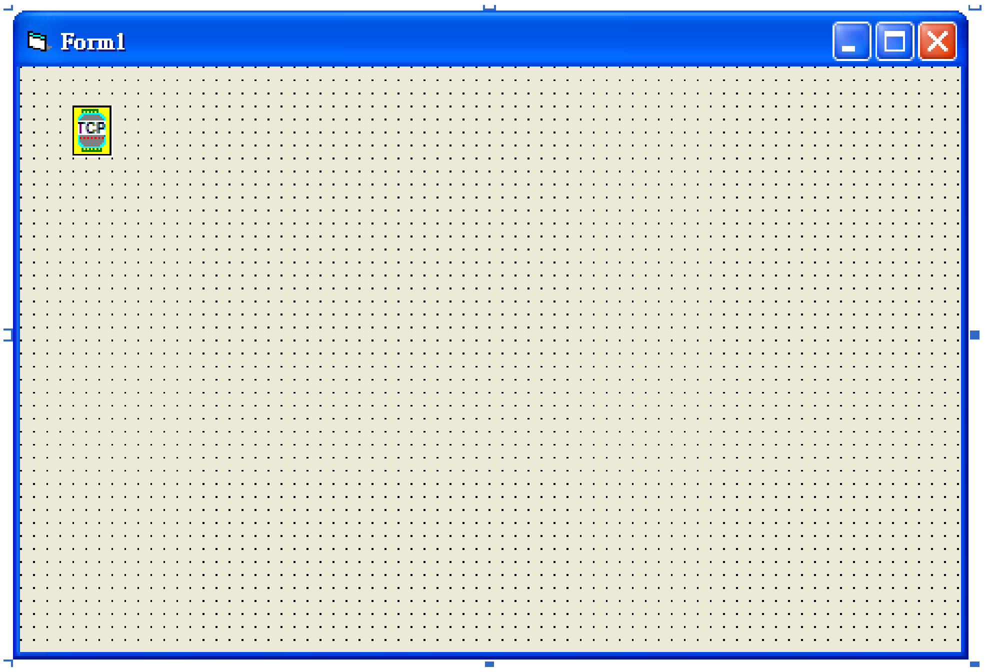
[ ] Selected Items Only

TCPDAQ ActiveX Control module

Location: C:\WINDOWS\system32\tcpdaq.ocx

確定 | 取消 | 套用(A)

- Place a TCPDAQ control from the Toolbox on the form. Use the default name.
- Your form should look similar to the one shown below:



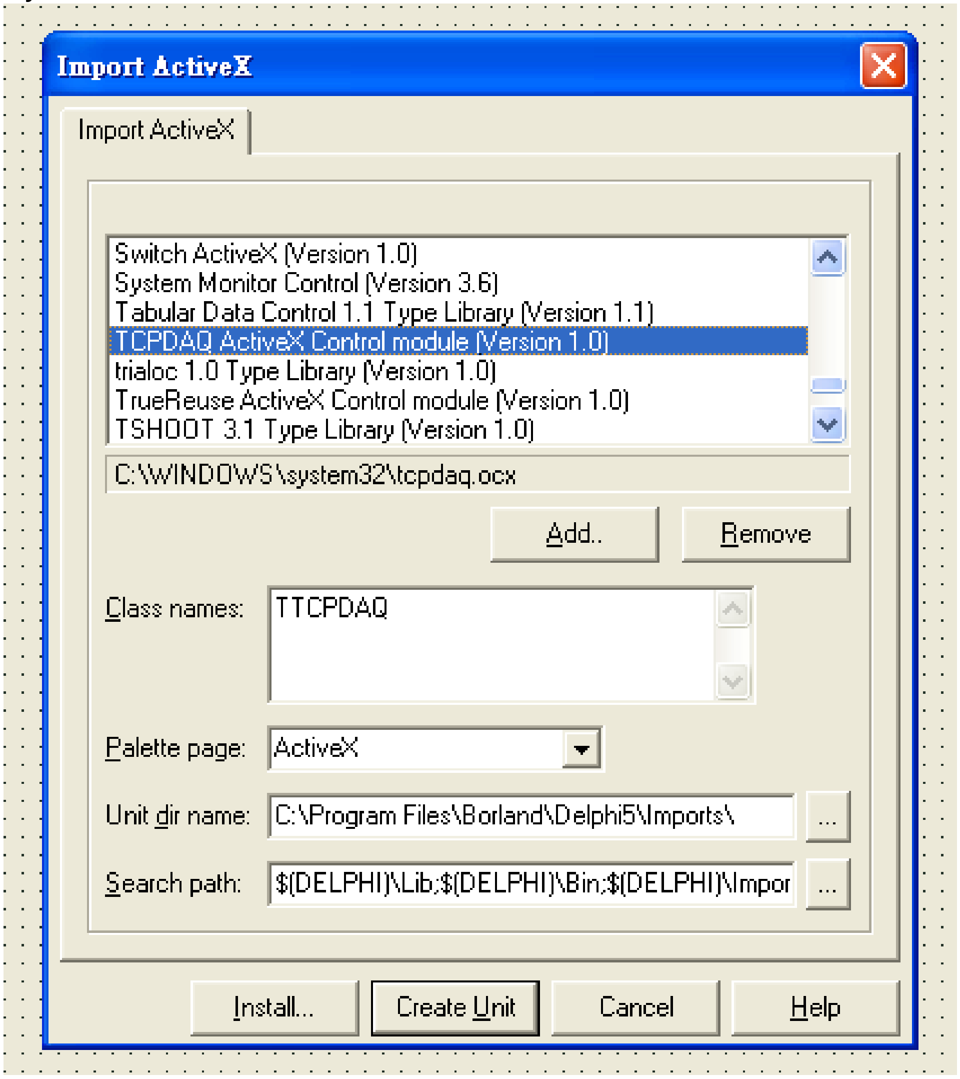
Printed Date: June 1, 2016

4.2.2 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

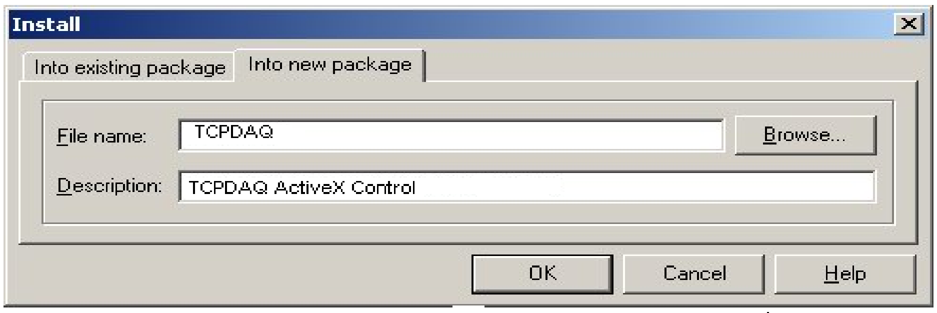
Class names: TTCPDAQ

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.



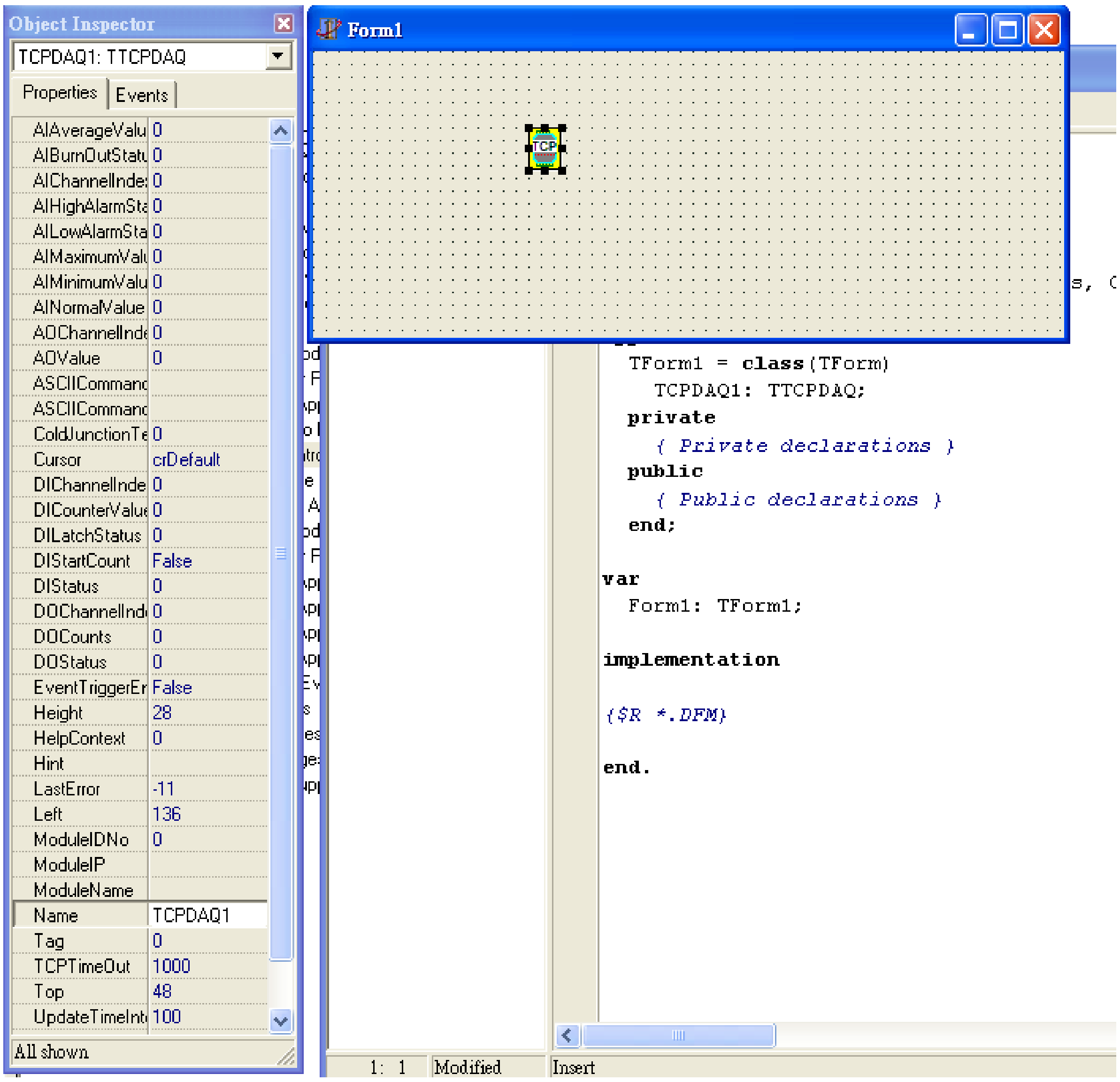
File name: TCPDAQ

Description: TCPDAQ ActiveX Control

Switch to the form and select theActiveX 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:



Object Inspector

TCPDAQ1: TTCPDAQ

Properties | Events

AIAverageValu	0
AIBurnOutStatu	0
AIChannelInde	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

All shown

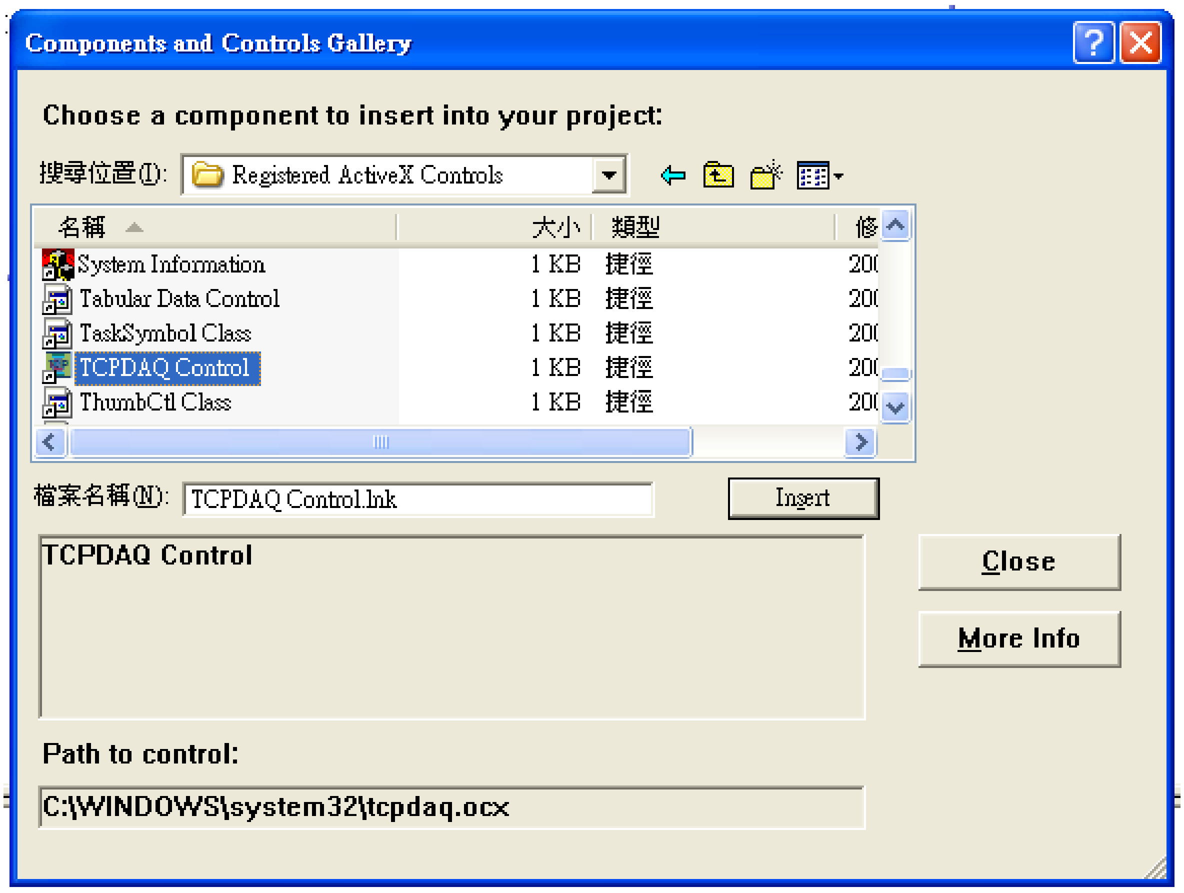
```
  TForm1 = class(TForm)
    TCPDAQ1: TTCPDAQ;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

1: 1 Modified Insert

4.2.3 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
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(

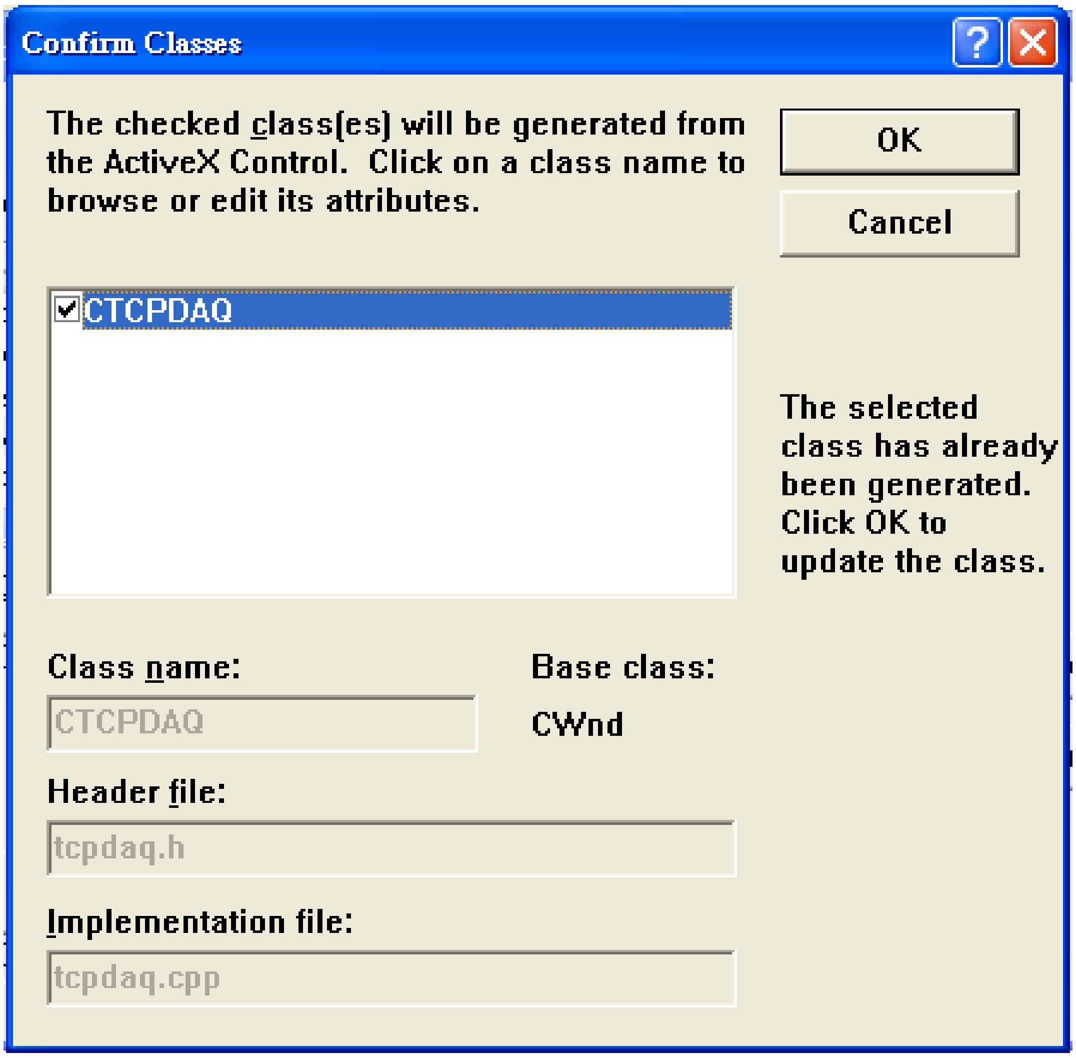
檔案名稱(N): TCPDAQ Control.lnk

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

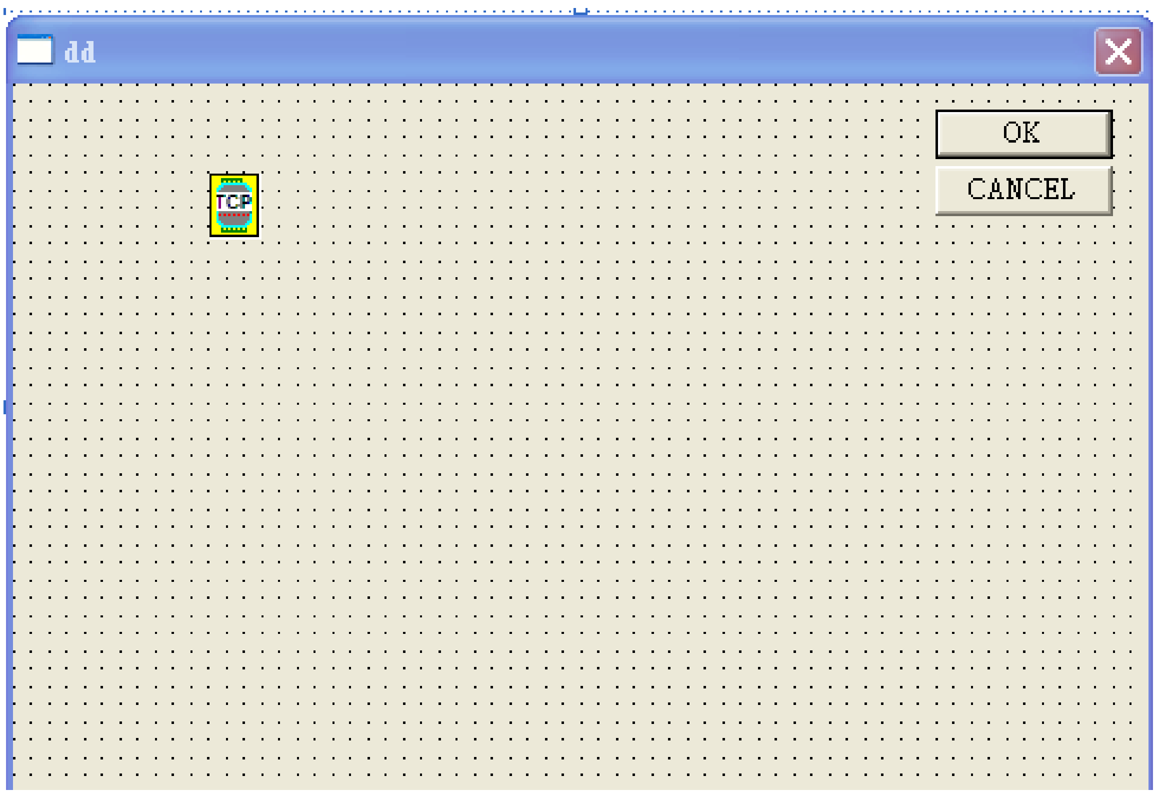
Base class: CWnd

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.

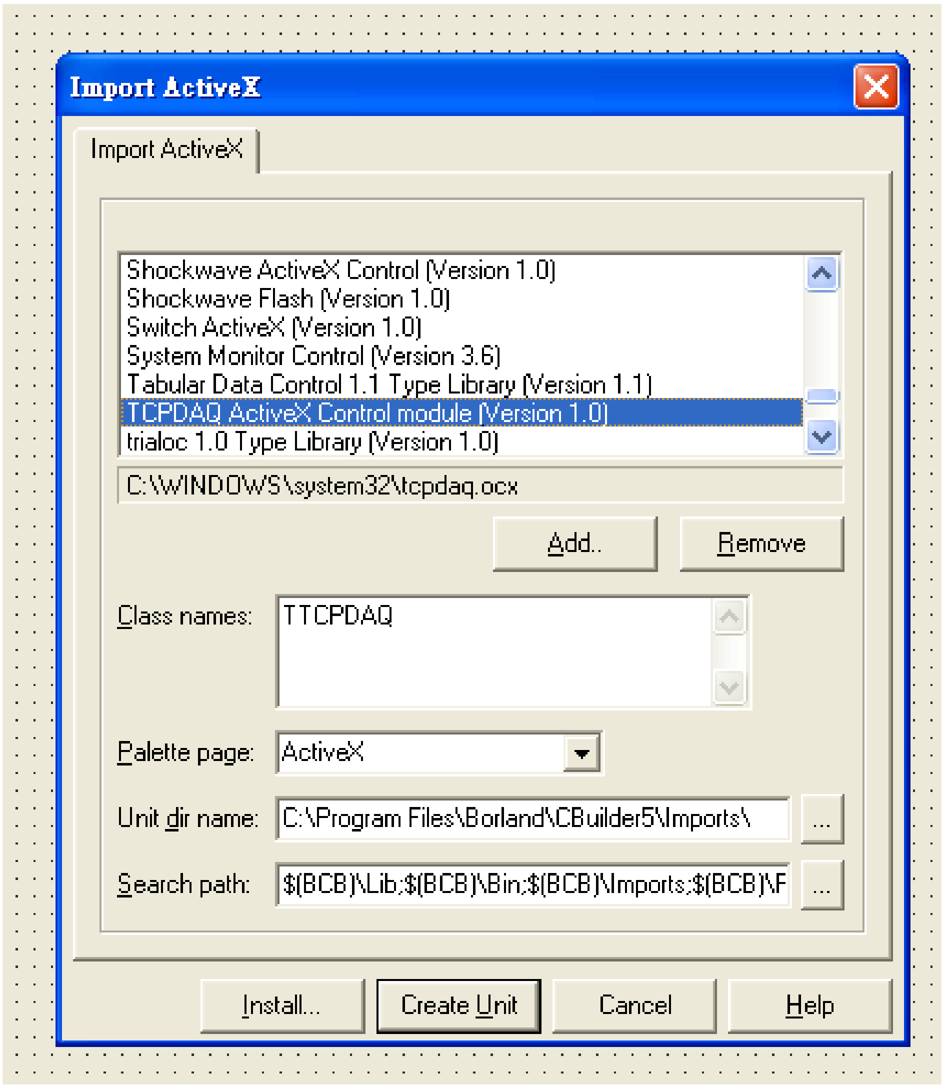


4.2.4 Building TCPDAQ Applications with Borland C++ Builder

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:


Import ActiveX

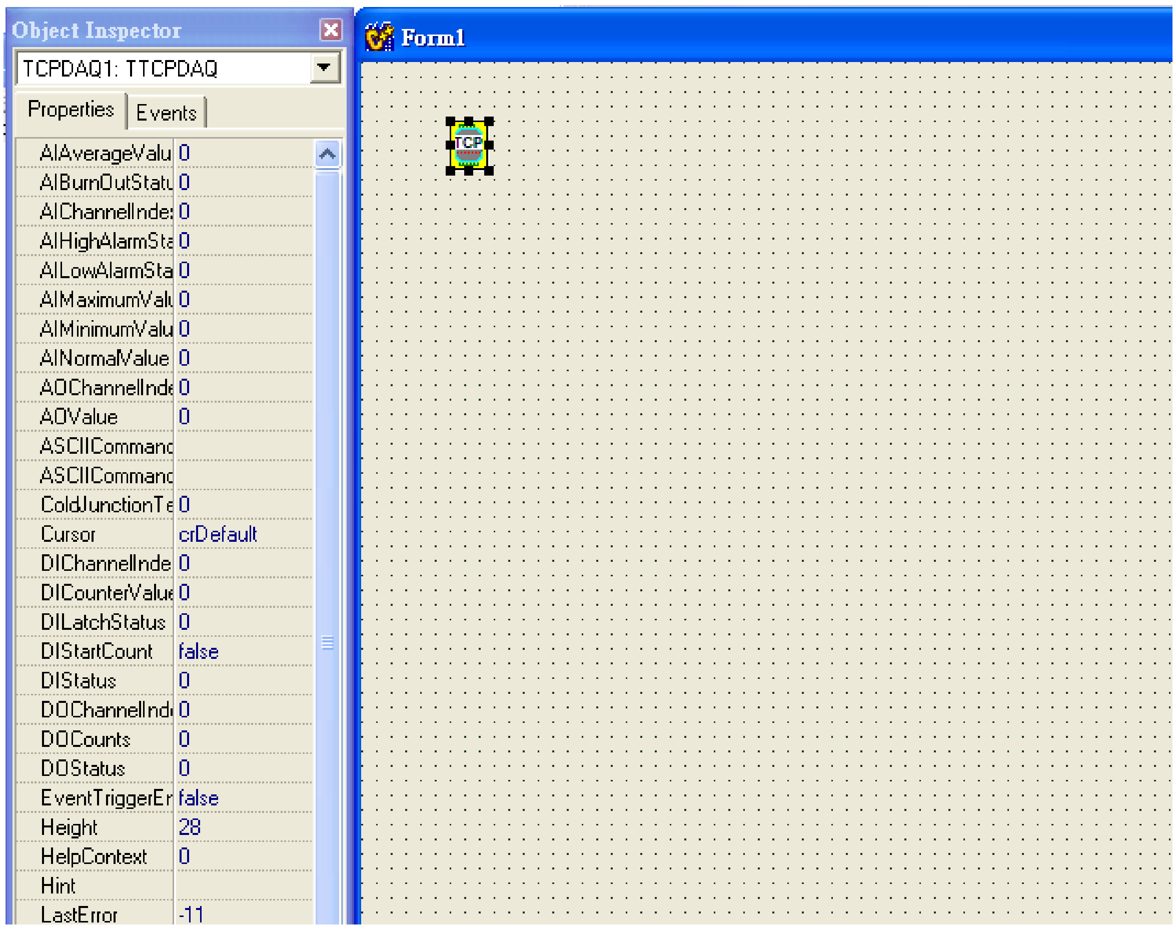
Shockwave ActiveX Control (Version 1.0)
Shockwave Flash (Version 1.0)
Switch ActiveX (Version 1.0)
System Monitor Control (Version 3.6)
Tabular Data Control 1.1 Type Library (Version 1.1)
TCPDAQ ActiveX Control module (Version 1.0)
trialoc 1.0 Type Library (Version 1.0)

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. Adialog box is shown as below.



TCPDAQ1: TTCPDAQ

AIAverageValu	0
AIBurnOutStatu	0
AIChannelInde:	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
DOChannelInde	0
DOCounts	0
DOStatus	0
EventTriggerEr	false
Height	28
HelpContext	0
Hint	
LastError	-11

4.3 Properties of TCPDAQ ActiveX Control

Name	Type	Description	Avaliable Model(s)
AIChannelIndex	short	Specifies the analog input channel to perform other AI properties read/write operation.	9015-MTCP,9017-MTCP,9019-MTCP
AINormalValue	double	Normal voltage of specifies the analog channel	9015-MTCP,9017-MTCP,9019-MTCP
AIAveragevalue	double	Average voltage value of the channels that are in average	9015-MTCP,9017-MTCP,9019-MTCP
AIMaximumValue	double	Maximal voltage of specifies the analog channel	9015-MTCP,9017-MTCP,9019-MTCP
AIMinimumValue	double	Minimal voltage of specifies the analog channel	9015-MTCP,9017-MTCP,9019-MTCP
AILowAlarmStatus	short	Return the low alarm status of specifies the analog channel (1=Alarm occurred, 0=No alarm)	9015-MTCP,9017-MTCP,9019-MTCP
AIHighAlarmStatus	short	Return the high alarm status of specifies the analog channel (1=Alarm occurred, 0=No alarm)	9015-MTCP,9017-MTCP,9019-MTCP
AIBurnOutStatus	short	Return the Burnout status of specifies the analog channel (1=open, 0=normal)	9015-MTCP and 9019-MTCP
AOChannelIndex	short	Specifies the analog output channel to perform other properties read/write operation.	Reserved for Ver 1.0
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
ColdJunctionTemperature	double	Return the cold junction temperature	9019-MTCP
DIChannelIndex	short	Specifies the digital input channel to perform other DI properties read/write operation.	9050-MTCP,9051-MTCP,9055-MTCP
DIounterValue	long	Return the counting value for the specified DI channel which functions in "Count/Frequency mode"	9050-MTCP,9051-MTCP,9055-MTCP
DILatchStatus	short	Return the latch status for the specified DI channel which functions in "Lo-Hi/Hi-Lo latch mode" (1=Latched, 0=No latched)	9050-MTCP,9051-MTCP,9055-MTCP
DIStartCount	boolean	Start/stop counting for the specified DI channel which	9050-MTCP,9051-MTCP,9055-MTCP

		functions in "Count/Frequency mode" (True=Start, 0=Stop)	
DIStatus	short	Return the status for the specified DI channel which functions in "DI mode" (1=Active, 0=Inactive)	9050-MTCP,9051-MTCP,9055-MTCP
DOChannelIndex	short	Specifies the digital output channel to perform other DO properties read/write operation.	9017-MTCP,9019-MTCP,9050-MTCP,9051-MTCP,9055-MTCP
DOCount	long	Set the output count value for the specified DO channel which functions in "Pulse output mode"	9050-MTCP,9051-MTCP,9055-MTCP
DOStatus	short	Return/set the status for the specified DO channel which functions in "D/O mode" (1=Active, 0=Inactive)	9017-MTCP,9019-MTCP,9050-MTCP,9051-MTCP,9055-MTCP
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

4.4 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

4.5 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

4.6 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.

Printed Date: June 1, 2016

Chapter 5 TCPDAQ(Ethernet IO) DLL API

5.1 Common Functions

NO.	Function Name	Description	Sec.
1	TCP_Open	To initiate the TCPDAQ.dll to use.	5.6.1
2	TCP_Close	To terminates use of the TCPDAQ.dll.	5.6.2
3	TCP_Connect	To create a Window TCP socket then establishing a connection to a specific EX9000- MTCP	5.6.3
4	TCP_Disconnect	Disconnecting the Window TCP socket from all EX9000- MTCP modules	5.6.4
5	TCP_ModuleDisconnect	Disconnecting the Window TCP socket from a specific EX9000- MTCP	5.6.5
6	TCP_SendData	Send data to a specific EX9000- MTCP	5.6.6
7	TCP_RecvData	Receive data to a specific EX9000- MTCP module	5.6.7
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 EX9000- MTCP and receiving the response from EX9000-MTCP	5.6.8
9	UDP_Connect	To create a Window UDP socket then establishing a connection to a specific EX9000- MTCP	5.6.9
10	UDP_Disconnect	Disconnecting the Window UDP socket from all EX9000- MTCP modules	5.6.10
11	UDP_ModuleDisconnect	Disconnecting the Window UDP socket from a specific EX9000- MTCP	5.6.11
12	UDP_SendData	Send data to a specific EX9000- MTCP	5.6.12
13	UDP_RecvData	Receive data to a specific EX9000- MTCP module	5.6.13
14	UDP_SendReceiveASCcmd	Direct send an ASCII format string as a command, and receive the response from EX9000- MTCP	5.6.14
15	TCP_GetModuleIPinfo	Return module IP information of a specific module	5.6.15
16	TCP_GetModuleID	Return module ID number of a specific module	5.6.16
17	TCP_GetIPFromID	Return IP address of a specific module ID number	5.6.17
18	TCP_ScanOnLineModules	Scan all on-line EX9000- MTCP modules	5.6.18
19	TCP_GetDLLVersion	Return the DLL's version, that is the version of TCPDAQ.DLL	5.6.19
20	TCP_GetModuleNo	Return the module name of a specific IP address	5.6.20
21	TCP_GetLastError	Return the error code of the latest called function	5.6.21
22	TCP_PingIP	Ping to Remote IP address	5.6.22

5.2 Stream/Event Functions

TCP_StartStream	To instruct the PC to start to receive stream data that coming from EX9000- MTCP	5.6.23
TCP_StopStream	To instruct the PC to stop receiving stream data from all modules	5.6.24
TCP_ReadStreamData	To receive stream data that coming from the specific EX9000- MTCP	5.6.25
TCP_StartEvent	To instruct the PC to start to receive alarm event data that coming from EX9000-MTCP	5.6.26
TCP_StopEvent	To instruct the PC to stop receiving alarm event data from all modules	5.6.27
TCP_ReadEventData	To receive alarm event data that coming from the specific EX9000- MTCP	5.6.28

5.3 Digital I/O Functions

TCP_ReadDIOMode	To read the type for every D/I & D/O channels of an EX9000- MTCP module	5.6.29
TCP_ReadDIO	To read DI/DO's status for an EX9000- MTCP module	5.6.30
TCP_ReadDISignalWidth	To read the minimal high/low signal width of each D/I channel for an EX9000- MTCP module	5.6.31
TCP_WriteDISignalWidth	To set the minimal high/low signal width of each D/I channel for an EX9000- MTCP module	5.6.32
TCP_ReadDICounter	To read the counter value when a D/I channel function in 'Counter' mode	5.6.33
TCP_ClearDICounter	To clear the counter value when a D/I channel function in 'Counter' mode	5.6.34
TCP_StartDICounter	To start the counting when a D/I channel function in 'Counter' mode	5.6.35
TCP_StopDICounter	To stop the counting when a D/I channel function in 'Counter' mode	5.6.36
TCP_ClearDILatch	To clear the latch when a D/I channel function as 'Lo to Hi Latch' or 'Hi to Lo Latch'	5.6.37
TCP_ReadDILatch	To read the counter value when a D/I channel function in 'Counter' mode	5.6.38
TCP_WriteDO	To write some value to D/O channels for an EX9000- MTCP module	5.6.39
TCP_WriteDOPulseCount	To write the pulse output count for EX9000- MTCP DIO modules during runtime	5.6.40
TCP_WriteDODelayWidth	To set the pulse and delay signal widths to the specific EX9000- MTCP DIO modules	5.6.40
TCP_ReadDODelayWidth	To read the pulse and delay signal width from the specific EX9000- MTCP DIO modules	5.6.42

5.4 Analog I/O Functions

TCP_ReadAIAlarmTypes	To set all channel type	5.6.43
TCP_WriteAIAlarmType	To set all channel alarm type	5.6.44
TCP_ReadAITypes	To read type of all channels of a specific analog module	5.6.45
TCP_ReadAIValue	To read normal value of all channel	5.6.46
TCP_ReadAIMaxVal	To read maximum value of all channel	5.6.47
TCP_ReadAIMinVal	To read minimum value of all channel	5.6.48
TCP_ReadAIMultiplexChannel	To read active status of all channel	5.6.49
TCP_WriteAIMultiplexChannel	To set active status of all channel	5.6.50
TCP_ReadAIAverageChannel	To read in average status of all channel	5.6.51
TCP_WriteAIAverageChannel	To set/reset channels to be in average	5.6.52
TCP_ReadAIAlarmDOConnection	To read alarm DO connection status	5.6.53
TCP_WriteAIAlarmDOConnection	To set alarm DO connection	5.6.54
TCP_ReadAIAlarmStatus	To read alarm status	5.6.55
TCP_ClearAILatchAlarm	To clear alarm latch status when a A/I channel function in 'Alarm Latch mode' mode	5.6.56
TCP_ClearAIMaxVal	To clear maximum value to zero	5.6.57
TCP_ClearAIMinVal	To clear minimum value to zero	5.6.58
TCP_ReadAIBurnOutStatus	To read AI burn out status(EX9015-MTCP/9019-MTCP only)	5.6.59
TCP_ReadAIAlarmLimit	To read channel high/low alarm limit value	5.6.60
TCP_WriteAIAlarmLimit	To set channel high/low alarm limit value	5.6.61
TCP_StartAIAlarm	To set channel alarm type of a specific analog module	5.6.62
TCP_StopAIAlarm	To disable channel alarm of a specific analog module	5.6.63
TCP_WriteCJCOffset	To set cold junction offset of a specific EX9019-MTCP module	5.6.64
TCP_ReadCJCOffset	To read cold junction offset from a specific EX9019-MTCP module	5.6.65
TCP_ReadCJCTemperature	To read cold junction temperature from a specific EX9019-MTCP module	5.6.66

5.5 MODBUS/TCP Functions

TCP_MODBUS_ReadCoil	To read the coil values at a specific range described in parameters	5.6.67
TCP_MODBUS_WriteCoil	To write the coil values at a specific range described in parameters.	5.6.68
TCP_MODBUS_ReadReg	To read the holding register value at a specific range described in parameters	5.6.69
TCP_MODBUS_WriteReg	To write values to the holding registers at a specific range described in parameters	5.6.70

5.6 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

5.6.1 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.*

5.6.2 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.*

5.6.3 TCP_Connect

Description: to create a Window TCP socket then establishing a connection to a specific EX9000-MTCP

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)*

Function TCP_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 EX9000-MTCP 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*.

5.6.4 TCP_Disconnect

Description: disconnecting the Window TCP socket from all EX9000-MTCP 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.

5.6.5 TCP_ModuleDisconnect

Description: disconnecting the Window TCP socket to a specific EX9000-MTCP

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 EX9000-MTCP that to be
connected **Return Code:**

refer to the *Error code.*

5.6.6 TCP_SendData

Description: to send data to a specific EX9000-MTCP 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 EX9000-MTCP that to be
connected **pData[in]: 8 bit data array**
wDataLen[in]: length of data be sent

Return Code:

refer to the *Error code.*

5.6.7 TCP_RecvData

Description: receive data to a specific EX9000-MTCP 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 EX9000-MTCP 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*.

5.6.8 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 EX9000-MTCP and receiving the response from EX9000-MTCP

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 EX9000-MTCP 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.*

5.6.9 UDP_Connect

Description: to create a Window UDP socket then establishing a connection to a specific EX9000-MTCP

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 EX9000-MTCP 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*.

5.6.10 UDP_Disconnect

Description: disconnecting the Window UDP socket from all EX9000-MTCP 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

5.6.11 UDP_ModuleDisconnect

Description: disconnecting the Window UDP socket from a specific EX9000-MTCP

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 EX9000-MTCP that to be

disconnected **Return Code:**

refer to the *Error code.*

5.6.12 UDP_SendData

Description: send data to a specific EX9000-MTCP 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 EX9000-MTCP that to be

connected pData[in]: points to data buffer

wDataLen[in]: length of data be sent

Return Code:

refer to the *Error code.*

5.6.13 UDP_RecvData

Description: receive data to a specific EX9000-MTCP 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 EX9000-MTCP 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.*

5.6.14 UDP_SendReceiveASCcmd

Description: send an ASCII format string as a command to EX9000-MTCP and receiving the response from EX9000-MTCP

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(SOCKET UDPsock,char Txdata [],char Rxdata []);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP 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.*

5.6.15 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 EX9000-MTCP that to be connected

ModuleIP[out]: a structure array that stroes the module IP information

Return Code:

refer to the *Error code.*

5.6.16 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 EX9000-MTCP that to be

connected ModuleID [in]: the ID number

Return Code:

refer to the *Error code.*

5.6.17 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*.

5.6.18 TCP_ScanOnLineModules

Description: search on-line EX9000-MTCP 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*.

5.6.19 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.

5.6.20 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 EX9000-MTCP that to be
connected Mname[out]: 8 bit array that stored the module name
string

Return Code:

refer to the *Error code*.

5.6.21 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:

The error status for the last operation that failed.(refer to the *Error code*)

5.6.22 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 EX9000-MTCP that to be
connected PingTimes [in]:Timeout value

Return Code:

=-1, no response from remote IP
>0, response time from remote IP

5.6.23 TCP_StartStream

Description: to instruct the PC to start to receive stream data that coming from EX9000-MTCP

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 EX9000-MTCP that to be connected

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

Return Code:

refer to the *Error code*.

5.6.24 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*.

5.6.25 TCP_ReadStreamData

Description: to read stream data that coming from the specific EX9000-MTCP

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 EX9000-MTCP that to be connected

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

Return Code:

refer to the *Error code*.

5.6.26 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 EX9000-MTCP that to be connected

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

Return Code:

refer to the *Error code*.

5.6.27 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*.

5.6.28 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 EX9000-MTCP 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*.

5.6.29 TCP_ReadDIOMode

Description: to read the mode of D/I & D/O channels of an EX9000-MTCP 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 EX9000-MTCP 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*.

5.6.30 TCP_ReadDIO

Description: to read DI/DO's status for an EX9000-MTCP 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 EX9000-MTCP that to be connected
byDI[out]: an 8 bit array that stored the DI channel status (ex: DI[1] = 0 – channel(0) = 0)
byDO[out]: an 8 bit array that stored the DO channel status (ex: DO[3] = 1 – channel(3) = 1)

Return Code:

refer to the *Error code*.

5.6.31 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 EX9000-MTCP 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*.

5.6.32 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 EX9000-MTCP 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*.

5.6.33 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 EX9000-MTCP 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.*

5.6.34 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 EX9000-MTCP that to be
connected wChNo[in]: the D/I channel to be cleared.

Return Code:

refer to the *Error code.*

5.6.35 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 EX9000-MTCP that to be
connected wChNo[in]: the channel number that is enabled to
count

Return Code:

refer to the *Error code*.

5.6.36 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 EX9000-MTCP that to be
connected wChNo[in]: the channel number that is disabled to
count

Return Code:

refer to the *Error code*.

5.6.37 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 EX9000-MTCP that to be
connected wChNo[in]: the channel number that latch status is
cleared

Return Code:

refer to the *Error code.*

5.6.38 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 EX9000-MTCP 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.*

5.6.39 TCP_WriteDO

Description: to write some value to D/O channels for an EX9000-MTCP 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 EX9000-MTCP 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 EX9000-MTCP
refer to the *Error code.*

5.6.40 TCP_WriteDOPulseCount

Description: to write the pulse output count for EX9000-MTCP 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 EX9000-MTCP 0 that to be

connected wDoChannel[in]: the channel index for writing

ulPulseCount[in]: the pulse output count.

Return Code:

refer to the *Error code.*

5.6.41 TCP_WriteDODelayWidth

Description: to set the pulse and delay signal widths to specific EX9000-MTCP 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 EX9000-MTCP 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.*

5.6.42 TCP_ReadDODelayWidth

Description: to read the pulse and delay signal widths from specific EX9000-MTCP 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 EX9000-MTCP 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*.

5.6.43 TCP_ReadAIAlarmTypes

Description: to read channel alarm type of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIAlarmTypes Lib "TCPDAQ.dll" Alias "_TCP_ReadAIAlarmTypes@16"
(ByVal szIP As String, ByVal AIchno As Integer, ByRef HiAlarmType As Byte, ByRef LoAlarmType As Byte) As Long

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

Int TCP_ReadAIAlarmTypes(char szIP[],u_short AIchno,u_char *AIHialarmtype,
u_char *AILoalarmtype);

Delphi: (see TCPDAQ.pas)

Function TCP_ReadAIAlarmTypes(szIP: PChar; AIchno: Integer; HiAlarmType: PByte;
LoAlarmType: PByte): Longint; StdCall;

VC++: (see TCPDAQ.h)

Int TCP_ReadAIAlarmTypes(char szIP[],u_short AIchno, u_char *AIHialarmtype,
u_char *AILoalarmtype);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected AIchno[in]: the channel index for reading

AIHialarmtype[in]: high alarm type(=0 momemtary_alarm,=1 latch_alarm,=2 disable_alarm)

AILoalarmtype[in]: low alarm type(=0 momemtary_alarm,=1 latch_alarm,=2 disable_alarm)

Return Code:

refer to the *Error code*.

5.6.44 TCP_WriteAIAlarmType

Description: to set channel alarm type of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_WriteAIAlarmType Lib "TCPDAQ.dll" Alias "_TCP_WriteAIAlarmType@16"
(ByVal szIP As String, ByVal Chno As Integer, ByVal HiLoAlarm As Byte, ByVal AlarmType As Byte) As Long

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

Int TCP_WriteAIAlarmType(char szIP[],u_short AIchno,u_char HiorLow,u_char Alarmtype);

Delphi: *(see TCPDAQ.pas)*

Function TCP_WriteAIAlarmType (szIP: PChar; Chno: Integer; HiLoAlarm: Byte; AlarmType: Byte): Longint; StdCall;

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

Int TCP_WriteAIAlarmType(char szIP[],u_short AIchno, u_char HiorLow,u_char Alarmtype);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected AIchno[in]: the channel index for reading
HiorLow[in]: set high or low alarm(=0 low alarm, =1 high alarm)
Alarmtype[in]: alarm type (0=momemtary_alarm, 1=latch_alarm)

Return Code:

refer to the *Error code**.***

5.6.45 TCP_ReadAITypes

Description: to read all channel type of a specific ananlog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Fu5.6.46nction TCP_ReadAITypes Lib "TCPDAQ.dll" Alias "_TCP_ReadAITypes@8"
(ByVal szIP As String, ByRef szRange As Byte) As Long

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

int TCP_ReadAITypes(char szIP[],u_char szTypes[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAITypes (szIP: PChar; szRange: PByte): Longint; StdCall;

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

int TCP_ReadAITypes(char szIP[],u_char szTypes[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected szTypes[out]: an array that stored the types of all A/I
channels

Return Code:

refer to the *Error code*.

5.6.46 TCP_ReadAIValue

Description: to read all channel input value of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIValue Lib "TCPDAQ.dll" Alias "_TCP_ReadAIValue@8"
(ByVal szIP As String, ByRef dlValue As Double) As Long

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

int TCP_ReadAIValue(char szIP[],double dlValue[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIValue (szIP: PChar; dlValue: PDouble): Longint; StdCall;

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

int TCP_ReadAIValue(char szIP[],double dlValue[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected
dlValue[out]: an array that stored the analog values that reading from A/I channels.

Return Code:

refer to the *Error code.*

5.6.47 TCP_ReadAIMaxVal

Description: to read all channel maxmal value of a specific ananlog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIMaxVal Lib "TCPDAQ.dll" Alias "_TCP_ReadAIMaxVal@8"
(ByVal szIP As String, ByRef dMaxValue As Double) As Long

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

int TCP_ReadAIMaxVal(char szIP[],double dMaxValue[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIMaxVal (szIP: PChar; dMaxValue: PDouble): Longint; StdCall;

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

int TCP_ReadAIMaxVal(char szIP[],double dMaxValue[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected
dMaxValue[out]: an array that stored the maximal analog values of all A/I channels

Return Code:

refer to the *Error code.*

5.6.48 TCP_ReadAIMinVal

Description: to read all channel minimal value of a specific ananlog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIMinVal Lib "TCPDAQ.dll" Alias "_TCP_ReadAIMinVal@8"
(ByVal szIP As String, ByRef dMinValue As Double) As Long

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

int TCP_ReadAIMinVal(char szIP[],double dMinValue[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIMinVal (szIP: PChar; dMinValue: PDouble): Longint; StdCall;

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

int TCP_ReadAIMinVal(char szIP[],double dMinValue[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected

dMinValue[out]: an array that stored the minimal analog values of all A/I channels

Return Code:

refer to the *Error code.*

5.6.49 TCP_ReadAIMultiplexChannel

Description: to read all channel activation status of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIMultiplexChannel Lib "TCPDAQ.dll" Alias
"_TCP_ReadAIMultiplexChannel@8" (ByVal szIP As String, ByRef szchno As Byte)
As Long

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

int TCP_ReadAIMultiplexChannel(char szIP[],u_char szChno[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIMultiplexChannel(szIP: PChar; szchstatus: PByte): Longint;
StdCall;

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

int TCP_ReadAIMultiplexChannel(char szIP[],u_char szChno[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected

szChno[in]: an 8 bit array that stored the AI channel which represent in numeric.
The meanning for a value in an entity as follow:
szChno[n]:0 disable channel #n for multiplexing
szChno[n]:1 Enable channel #n for multiplexing

Return Code:

refer to the *Error code.*

5.6.50 TCP_WriteAIMultiplexChannel

Description: to enable/disable channel activation of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_WriteAIMultiplexChannel Lib "TCPDAQ.dll" Alias
"_TCP_WriteAIMultiplexChannel@8" (ByVal szIP As String, ByRef szchno As Byte)
As Long

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

int TCP_WriteAIMultiplexChannel(char szIP[],u_char szChno[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_WriteAIMultiplexChannel(szIP: PChar; szchstatus: PByte): Longint;
StdCall;

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

Int TCP_WriteAIMultiplexChannel(char szIP[],u_char szChno[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected

szChno[in]: an 8 bit array that stored the AI channel which represent in numeric.
The meanning for a value in an entity as follow:
szChno[n]:0 disable channel #n for multiplexing
szChno[n]:1 Enable channel #n for multiplexing

Return Code:

refer to the *Error code.*

5.6.51 TCP_ReadAIAverageChannel

Description: to read all channels in-average status of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIAverageChannel Lib "TCPDAQ.dll" Alias
"_TCP_ReadAIAverageChannel@8" (ByVal szIP As String, ByRef avgch As Byte) As
Long

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

int TCP_ReadAIAverageChannel(char szIP[],u_char avgch[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIAverageChannel(szIP: PChar; avgch: PByte): Longint; StdCall;

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

int TCP_ReadAIAverageChannel(char szIP[],u_char avgch[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected

avgch[in]: an 8 bit array that stored the AI channel which represent in numeric.
The meanning for a value in an entity as follow:
avgch [n]:0 the channel #n is in average
avgch [n]:1 the channel #n is not in average

Return Code:

refer to the *Error code.*

5.6.52 TCP_WriteAIAverageChannel

Description: to set all channels to be in-average or not of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

```
Declare Function TCP_WriteAIAverageChannel Lib "TCPDAQ.dll" Alias
"_TCP_WriteAIAverageChannel@8" (ByVal szIP As String, ByRef avgch As Byte) As Long
```

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

```
int       TCP_WriteAIAverageChannel(char szIP[],u_char avgch[]);
```

Delphi: *(see TCPDAQ.pas)*

```
Function                TCP_WriteAIAverageChannel(szIP: PChar; avgch: PByte): Longint; StdCall;
```

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

```
int       TCP_WriteAIAverageChannel(cChar szIP[],u_char avgch[]);
```

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected

avgch[in]: an 8 bit array that stored the AI channel which represent in numeric.
The meanning for a value in an entity as follow:
avgch [n]:0 disable channel #n to be in average
avgch [n]:1 enable channel #n to be in average

Return Code:

refer to the *Error code*.

5.6.53 TCP_ReadAIAlarmDOConnection

Description: to read alarm channel DO connection of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

```
Declare Function TCP_ReadAIAlarmDOConnection Lib "TCPDAQ.dll" Alias
                 "_TCP_ReadAIAlarmDOConnection@16" (ByVal szIP As String, ByVal AIchno As
                 Integer, ByRef AIHiAlarmDOchn As Integer, ByRef AILoAlarmDOchn As Integer) As
                 Long
```

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

```
int     TCP_ReadAIAlarmDOConnection(char szIP[],u_short AIchno, u_short *AIHiAlarmDOchn,
                                       u_short *AILoAlarmDOchn);
```

Delphi: (see TCPDAQ.pas)

```
Function    TCP_ReadAIAlarmDOConnection(szIP: PChar; AIchno: Integer; AIHiAlarmDOchn:
             PWORD;   AILoAlarmDOchn: PWORD): Longint; StdCall;
```

VC++: (see TCPDAQ.h)

```
int     TCP_ReadAIAlarmDOConnection(char szIP[],u_short AIchno,u_short *AIHiAlarmDOchn,
                                       u_short *AILoAlarmDOchn);
```

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected AIchno[in]: the channel index for reading

AIHiAlarmDOchn[out]: D/O channel number be connected to high alarm

AILoAlarmDOchn[out]: D/O channel number be connected to low alarm

Return Code:

refer to the *Error code.*

5.6.54 TCP_WriteAIAlarmDOConnection

Description: to set alarm channel DO connection of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_WriteAIAlarmDOConnection Lib "TCPDAQ.dll" Alias "_TCP_WriteAIAlarmDOConnection@16" (ByVal szIP As String, ByVal AIchno As Integer, ByVal HiAlarmDOchn As Integer, ByVal LoAlarmDOchn As Integer) As Long

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

int TCP_WriteAIAlarmDOConnection(char szIP[],u_short AIchno,u_short HiAlarmDOchn, u_short LoAlarmDOchn);

Delphi: *(see TCPDAQ.pas)*

Function TCP_WriteAIAlarmDOConnection (szIP: PChar; AIchno: Integer; HiAlarmDOchn: PWORD; LoAlarmDOchn: PWORD): Longint; StdCall;

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

int TCP_WriteAIAlarmDOConnection(char szIP[],u_short AIchno, u_short HiAlarmDOchn, u_short LoAlarmDOchn);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected AIchno[in]: the channel index for reading

AIHiAlarmDOchn[in] D/O channel number be connected to high alarm

AILoAlarmDOchn[in]: D/O channel number be connected to low alarm

Return Code:

refer to the *Error code.*

5.6.55 TCP_ReadAIAlarmStatus

Description: to read a channel alarm status of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

```
Declare Function TCP_ReadAIAlarmStatus Lib "TCPDAQ.dll" Alias "_TCP_ReadAIAlarmStatus@16"
                (ByVal szIP As String, ByVal Chno As Integer, ByRef szHighAlarm As Byte, ByRef
                szLowAlarm As Byte) As Long
```

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

```
int     TCP_ReadAIAlarmStatus(char szIP[],u_short Chno,u_char *szHighAlarm,
                              u_char *szLowAlarm);
```

Delphi: *(see TCPDAQ.pas)*

```
Function    TCP_ReadAIAlarmStatus (szIP: PChar; Chno: Integer; szHighAlarm: PByte; szLowAlarm:
            PByte): Longint; StdCall;
```

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

```
int     TCP_ReadAIAlarmStatus(char szIP[],u_short Chno,u_char *szHighAlarm,
                              u_char *szLowAlarm);
```

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected **Chno[in]: the channel index for reading**

szHighAlarm: high alarm status (1=alarm occurred, 0=no alarm)

szLowAlarm: low alarm status (1=alarm occurred, 0=no alarm)

Return Code:

refer to the *Error code.*

5.6.56 TCP_ClearAILatchAlarm

Description: to clear channel latch status when A/I channel function in "Latch alarm" mode

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

```
Declare Function TCP_ClearAILatchAlarm Lib "TCPDAQ.dll" Alias "_TCP_ClearAILatchAlarm@12"
                (ByVal szIP As String, ByVal Chno As Integer, ByVal alarmlevel As Byte) As Long
```

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

```
Int             TCP_ClearAILatchAlarm(char szIP[],u_short Chno,u_char Alarmlevel);
```

Delphi: *(see TCPDAQ.pas)*

```
Function    TCP_ClearAILatchAlarm(szIP: PChar; Chno: Integer; alarmlevel: Byte): Longint; StdCall;
```

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

```
Int             TCP_ClearAILatchAlarm(char szIP[],u_short Chno,u_char Alarmlevel);
```

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected Chno[in]: the channel index for writing

Alarmlevel[in]: alarm latch be cleared (0=low alarm latch , 1=high lalarm latch)

Return Code:

refer to the *Error code.*

5.6.57 TCP_ClearAIMaxVal

Description: to clear channel maxmal value of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ClearAIMaxVal Lib "TCPDAQ.dll" Alias "_TCP_ClearAIMaxVal@8"
(ByVal szIP As String, ByVal Chno As Integer) As Long

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

Int TCP_ClearAIMaxVal(char szIP[],u_short Chno);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ClearAIMaxVal (szIP: PChar; Chno: Integer): Longint; StdCall;

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

Int TCP_ClearAIMaxVal(char szIP[],u_short Chno);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected Chno[in]: the channel index for clearing

Return Code:

refer to the *Error code.*

5.6.58 TCP_ClearAIMinVal

Description: to clear channel minimal value of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ClearAIMinVal Lib "TCPDAQ.dll" Alias "_TCP_ClearAIMinVal@8"
(ByVal szIP As String, ByVal Chno As Integer) As Long

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

Int TCP_ClearAIMinVal(char szIP[],u_short Chno);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ClearAIMinVal (szIP: PChar; Chno: Integer): Longint; StdCall;

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

Int TCP_ClearAIMinVal(char szIP[],u_short Chno);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected Chno[in]: the channel index for clearing

Return Code:

refer to the *Error code.*

5.6.59 TCP_ReadAIBurnOutStatus

Description: to read all channel burn-out status of a specific analog module (EX-9015-MTCP, 9019-MTCP only)

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIBurnOutStatus Lib "TCPDAQ.dll" Alias "_TCP_ReadAIBurnOutStatus@8" (ByVal szIP As String, ByRef dlBurnout As Byte) As Long

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

int TCP_ReadAIBurnOutStatus(char szIP[],u_char dlBurnout[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIBurnOutStatus (szIP: PChar; dlBurnout: PByte): Longint; StdCall;

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

int TCP_ReadAIBurnOutStatus(char szIP[],u_char dlBurnout[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be connected

dlBurnout[out]: an 8 bit array that stored the burn-out status of EX-9019-MTCP,9015-MTCP module (=0 normal, =1 burn-out)

Return Code:

refer to the *Error code.*

5.6.60 TCP_ReadAIAlarmLimit

Description: to read all channel high/low alarm limit value

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadAIAlarmLimit Lib "TCPDAQ.dll" Alias "_TCP_ReadAIAlarmLimit@16" (ByVal szIP As String, ByVal Chno As Integer, ByRef dHighLimit As Double, ByRef dLowLimit As Double) As Long

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

int TCP_ReadAIAlarmLimit(char szIP[],u_short Chno, double dHighLimit[], double dLowLimit[]);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadAIAlarmLimit(szIP: PChar; Chno: Integer; dHighLimit: PDouble; dLowLimit: PDouble): Longint; StdCall;

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

int TCP_ReadAIAlarmLimit(char szIP[],u_short Chno, double dHighLimit[], double dLowLimit[]);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected Chno[in]: the channel index for reading

dHighLimit[out]: 32 bit array that stored the high larm limit value

dLowLimit[out]: 32 bit array that stored the low larm limit value

Return Code:

refer to the *Error code.*

5.6.61 TCP_WriteAIAlarmLimit

Description: to set every channel high/low alarm limit value

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_WriteAIAlarmLimit Lib "TCPDAQ.dll" Alias "_TCP_WriteAIAlarmLimit@24"
(ByVal szIP As String, ByVal Chno As Integer, ByVal dHighLimit As Double, ByVal dLowLimit As Double) As Long

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

Int TCP_WriteAIAlarmLimit(char szIP[],u_short Chno, double dHighLimit, double dLowLimit);

Delphi: *(see TCPDAQ.pas)*

Function TCP_WriteAIAlarmLimit (szIP: PChar; Chno: Integer; dHighLimit: Double; dLowLimit: Double): Longint; StdCall;

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

Int TCP_WriteAIAlarmLimit(char szIP[],u_short Chno, double dHighLimit, double dLowLimit);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected Chno[in]: the channel index for writing

dHighLimit[in]: high larm limit value (such as 2.321 or -2.321)

dLowLimit[in]: high larm limit value

Return Code:

refer to the *Error code*.

5.6.62 TCP_StartAIAlarm

Description: to start channel alarm of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_StartAIAlarm Lib "TCPDAQ.dll" Alias "_TCP_StartAIAlarm@12"
(ByVal szIP As String, ByVal Chno As Integer, ByVal alarmlevel As Byte) As Long

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

int TCP_StartAIAlarm(char szIP[],u_short Chno,u_char alarmLevel);

Delphi: *(see TCPDAQ.pas)*

Function TCP_StartAIAlarm (szIP: PChar; Chno: Integer; alarmlevel: Byte): Longint; StdCall;

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

Int TCP_StartAIAlarm(char szIP[],u_short Chno,u_char alarmLevel);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be

connected Chno[in]: the channel index for starting alarm

alarmLevel[in]: =0 start low alarm, =1 start high larm

Return Code:

refer to the *Error code*.

5.6.63 TCP_StopAIAlarm

Description: to disable channel alarm of a specific analog module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_StopAIAlarm Lib "TCPDAQ.dll" Alias "_TCP_StopAIAlarm@12"
(ByVal szIP As String, ByVal Chno As Integer, ByVal alarmlevel As Byte) As Long

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

Int TCP_StopAIAlarm(char szIP[],u_short Chno,u_char alarmlevel);

Delphi: *(see TCPDAQ.pas)*

Function TCP_StopAIAlarm (szIP: PChar; Chno: Integer; alarmlevel: Byte): Longint; StdCall;

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

Int TCP_StopAIAlarm(char szIP[],u_short Chno,u_char alarmlevel);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected Chno[in]: the channel index for writing
alarmlevel[in]: 0= disable low alarm , 1=disable high larm

Return Code:

refer to the *Error code.*

Notice: call this function will disable channel alarm forever.You should call TCP_WriteAIAlarmType to set alarm type and then call TCP_StartAlarm functions to re-start alarm

5.6.64 TCP_WriteCJCOffset

Description: to set cold junction offset of a specific EX9000-MTCP module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_WriteCJCOffset Lib "TCPDAQ.dll" Alias "_TCP_WriteCJCOffset@12"
(ByVal szIP As String, ByVal CJoffset As Double) As Long

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

Int TCP_WriteCJCOffset(char szIP[],double CJoffset);

Delphi: *(see TCPDAQ.pas)*

Function TCP_WriteCJCOffset (szIP: PChar; CJoffset: Double): Longint; StdCall;

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

Int TCP_WriteCJCOffset(char szIP[],double CJoffset);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected CJoffset[in]: cold junction temperature offset

Return Code:

refer to the *Error code.*

5.6.65 TCP_ReadCJCOffset

Description: to read cold junction offset from a specific EX9000-MTCP module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadCJCOffset Lib "TCPDAQ.dll" Alias "_TCP_ReadCJCOffset@8"
(ByVal szIP As String, ByRef CJoffset As Double) As Long

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

Int TCP_ReadCJCOffset(char szIP[],double *CJoffset);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadCJCOffset (szIP: PChar; CJoffset: Double): Longint; StdCall;

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

Int TCP_ReadCJCOffset(char szIP[],double *CJoffset);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected CJoffset[out]: cold junction offset

Return Code:

refer to the *Error code.*

5.6.66 TCP_ReadCJCTemperature

Description: to read cold junction temperature from a specific EX-9019-MTCP module

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

Declare Function TCP_ReadCJCTemperature Lib "TCPDAQ.dll" Alias
"_TCP_ReadCJCTemperature@8" (ByVal szIP As String, ByRef CJTemp As Double)
As Long

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

Int TCP_ReadCJCTemperature(char szIP[],double *CJTemp);

Delphi: *(see TCPDAQ.pas)*

Function TCP_ReadCJCTemperature (szIP: PChar; CJTemp: PDouble): Longint; StdCall;

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

Int TCP_ReadCJCTemperature(char szIP[],double *CJTemp);

Parameters:

szIP[in]: the IP address for an EX9000-MTCP that to be
connected CJTemp[out]: cold junction temperature

Return Code:

refer to the *Error code.*

5.6.67 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 EX9000-MTCP 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.*

5.6.68 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 EX9000-MTCP 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.*

5.6.69 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 EX9000-MTCP 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.*

5.6.70 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 EX9000-MTCP 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 6 ASCII Commands for EX9000-MTCP Modules

6.1 About ASCII Commands

For users do not familiar to Modbus protocol, TOPS CCC offers a function library as a protocol translator, integrating ASCII command into Modbus/TCP structure. Therefore, users familiar to ASCII command can access EX9000-MTCP 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.

EX9000-MTCP 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")).*

6.2 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 specified 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!

6.3 ASCII Command Set

6.3.1 Common commands

Command	Command Name	Description	modules	Sec.
$aaM	Read Module Name	Return the module name	All modules	6.4.1
$aaF	Read Firmware Version	Return the firmware version	All modules	6.4.2
$aaID	Read ID number	Return the ID number	All modules	6.4.3
#aan	Read single analog Input	Read the input value from the specified analog input channel	9015/9017/9019MTCP	6.4.4
$aaID	Read ID number	Return the ID number	All modules	6.4.5
$aaIDFF	Set Module ID number	Set module ID number (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.6
$aaMD(data)	Set Module Descript	set module description (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.7
$aaMD	Set Module Descript	read module description (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.8
^aaMAC	read MAC address	read MAC address (for version 6.000 or later)	New ver. of 9000-MTCP	6.4.9
$aaRS	Reboot the module to the power-on state	Reboot the module to the power-on state (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.10
$aaS1	Reloads the module factory default	Reloads the module factory default (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.11
~aaI	Soft INIT command (Enable)	Soft INIT* command (Enable) (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.12
~aaTNN	Sets the soft INIT timeout value	Sets the soft INIT* timeout value (for firmware version 6.000 or	New ver. of 9000-MTCP	6.4.13
$aa5	Reads the Reset Status of a module	Reads the Reset Status of a module (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.14
^aaMAC	Read MAC address	Read MAC address (for firmware version 6.000 or later)	New ver. of 9000-MTCP	6.4.15

Printed Date: June 1, 2016

6.3.2 Digital I/O commands

Commands	Command name	Descritpion	Modules	Sec.
$aa6	Read DI/O Channel Status	read the status of all D(0~11)I and DO (0~7) channels	**9050 / 9051 / 9055-MTCP**	6.4.16
		read the status of all DI(0~15) and DO(0~15) channels. (ref. "@AA6" for read 16bits DIO) (for firmware version 6.000 or later)	All modules **(Except 9050 /9051 / 9055-MTCP)**	
@aa	Read DI/O Status	read the status of all DI(0~11) and DO(0~7) channels. (ref. "@AA6" for read 16bits DIO)	**9050 / 9051 / 9055-MTCP**	6.4.17
		read the status of all DI(0~15) and DO(0~15) channels. (ref. "@AA6" for read 16bits DIO) (for firmware version 6.000 or later)	All modules **(Except 9050 /9051 / 9055-MTCP)**	
$aa7	Read DI latch status	Read DI latch status	All modules	6.4.18
#aa00dd	Write DO channels(0~	Write a value to digital output channels(0~7)	All modules	6.4.19
#aa1ndd	Set single DO channel	Set the single digital output channel(0~15)	All modules	6.4.20
#aa2npppppppp p	Write DO pulse counts	Write DO pulse counts to the specified DO channel	All modules	6.4.28
$aaCLS	Clear all latch status	Clear latch status of all DI channels	All modules	6.4.29
~aaDSMN	Set DI/O active state	Set digital input/output active state (for firmware version 6.000	All modules	6.4.30
~aaDS	read DI/O active state	read digital input/output active state (for firmware version 6.000	All modules	6.4.31
@aa6	read the status of all DO and DI channels	read the status of all DO(4char) and DI(4char) channels. (for firmware version 6.000 or later)	All modules	6.4.32
@aa6DDDD	Write DO channels(0~15)	Write a value to digital output channels(0~15) (for firmware version 6.000 or later)	All modules	6.4.33
@aa6ONSS	Set single DO channel	Set the single digital output channel(0~15) (for firmware version 6.000 or later)	All modules	6.4.34
@aa6ON	Read a single digital output for channel N	Read a single digital output for channel N (0~15) (for firmware version 6.000 or later)	All modules	6.4.35

@aa6IN	Read a single digital input for channel N	Read a single digital input for channel N (0~15) (for firmware version 6.000 or later)	All modules	6.4.36
$AA9NN(data)	Set DO pulse low/high width for channel N	Set DO pulse low/high width and Low/high delay output width for channel N (00~15), unit: 0.5ms. (for firmware version 6.000 or later)	All modules	6.4.37
$AA9NN	Read DO pulse low/high width for channel N	Read DO pulse low/high width and Low/high delay output width for ch. N (00~15), unit: 0.5ms. (for firmware version 6.000 or later)	All modules	6.4.38

6.3.3 DI Counter Commands

Command	Command Name	Description	modules	Sec.
$aa0MCC	read DI counter filter	read DI counter filter signal width of channel N (uint = 0.5ms) (for firmware version 6.000 or later)	**New ver. of 9000-MTCP (Except 9051-MTCP)**	6.4.21
$aa0MCC(data)	Set DI counter filter	Set DI counter filter signal width of channel N (uint = 0.5ms) (for firmware version 6.000 or later)	**New ver. of 9000-MTCP (Except 9051-MTCP)**	6.4.22
$aaEcn	Enable/disable DI counter	Start/stop counter of the specified DI channel.	All modules	6.4.23
$aaCn	Clear DI counter	Clear DI counter of the specified DI channel.	All modules	6.4.24
#aa	Read DI counter	Read all DI counter values.	All modules	6.4.25
#aan	Read DI counter	Read DI counter value of the specified DI channel.	All modules	6.4.26
#aarn	Read DI counter with overflow	Read a single DI channel counter with overflow. (for firmware version 6.000 or later)	**New ver. of 9000-MTCP**	6.4.27

6.3.4 Watchdog Commands **(All command for firmware version 6.000 or later)**

Command	Command Name	Description	modules	Sec.
~**	host ok	host ok (either, WDT counter can be refresh via broadcast) (No reply to modbus	**New ver. of 9000-MTCP**	6.4.39
~AA**	host ok	host ok. (either, WDT counter can be refresh via *.DlLL->TCP_ScanOnLineModules() and	**New ver. of 9000-MTCP**	6.4.40
~AA0	read host watchdog timeout status	read host watchdog timeout status. bit(7) - watchdog Enable/Disable, bit(2) -	**New ver. of 9000-MTCP**	6.4.41
~AA1	reset host watchdog timeout status	reset host watchdog timeout status	**New ver. of 9000-MTCP**	6.4.42
~AA2	read host communication Timeout value.	read host communication Timeout value. (0.1sec)	**New ver. of 9000-MTCP**	6.4.43
~AA3EVVV	Set Host watchdog timeout interval.	Enable Host watchdog and set timeout interval (unit=0.1sec)	**New ver. of 9000-MTCP**	6.4.44
~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	**New ver. of 9000-MTCP**	6.4.45
~AA5V	sets the current value as the power-on DO value or the safe DO	sets the current value as the power-on DO value or the safe DO value	**New ver. of 9000-MTCP**	6.4.46
~AA3PPP	set module Power-on delay time.	set module Power-on delay time (unit=0.1sec) to start host communication timeout	**New ver. of 9000-MTCP**	6.4.47
~AA3P	read module Power-on delay time	read module Power-on delay time (unit=0.1sec) to start host communication timeout	**New ver. of 9000-MTCP**	6.4.48

6.3.5 Analog commands

Command	Command name	Description	modules	Sec.
#aan	Read single analog Input	Read the input value from the specified analog	9015/9017/9019-MTCP	6.4.49
#aa	Read all analog Input	Read the input values from all analog input channels	9015/9017/9019-MTCP	6.4.50
#aaAcctt	Set analog input type	Set type of the specified analog input channel	9015/9017/9019-MTCP	6.4.51
$aaBhh	Read input type	read input type of the specified analog channel	9015/9017/9019-MTCP	6.4.52
$aa0	Span Calibration	Calibrate the analog input module to correct the gain error	9015/9017/9019-MTCP	6.4.53
$aa1	Offset Calibration	Calibrate the analog input module to correct the offset error	9015/9017/9019-MTCP	6.4.54
$aa6	Read Channel Enable/Disable	Read the Enable/Disable status of all analog input channels	99015/9017/9019-MTCP	6.4.55
$aa5mm	Enable/disable ananlog channel(s)	Enable/disable analog input channels	9015/9017/9019-MTCP	6.4.56
#aaMH	Read all Max. Data	Read the maximum data from all analog input	9015/9017/9019-MTCP	6.4.57
#aaMHn	Read single Max. Data	Read the maximum date from a specified analog input channel	9015/9017/9019-MTCP	6.4.58
#aaML	Read all Min. Data	Read the minimum data from all analog input	9015/9017/9019-MTCP	6.4.59
#aaMLn	Read single Min. Data	Read the minimum data from a specified analog input channel	9015/9017/9019-MTCP	6.4.60
$aaCjAhs	Set Alarm Mode	Set the High/Low alarm in either Momentary or Latching mode	9015/9017/9019-MTCP	6.4.61
$aaCjAh	Read Alarm Mode	Returns the alarm mode for the specified channels	9015/9017/9019-MTCP	6.4.62
$aaCjAhEs	Enable/Disable Alarm	Enables/Disables the high/low alarm of the specified channels	9015/9017/9019-MTCP	6.4.63
$aaCjCh	Clear Latch Alarm	Resets a latched alarm	9015/9017/9019-MTCP	6.4.64
$aaCjAhCCn	Set Alarm Connectio n	Connects the High/Low alarm of a specified input channel to interlock with a specified	9017/9019-MTCP	6.4.65
$aaCjRhC	Read Alarm Connection	Returns the alarm configuration of a specified	9017/9019-MTCP	6.4.66
$aaCjAhU	Set Alarm Limit	Sets the High/Low alarm limit value to a specified channel	9015/9017/9019-MTCP	6.4.67
$aaCjRhU	Read Alarm Limit	Returns the High/Low alarm limit value of the	9015/9017/9019-MTCP	6.4.68
$aaCjS	Read Alarm Status	Reads whether an alarm occurred in the specified Channel	9015/9017/9019-MTCP	6.4.69
$aa3	Read cold junction	Return the Cold Junction temperature	9019-MTCP	6.4.70
$aa9hhhhh	Set CJ offset	Set Cold Junction temperature offset	9019-MTCP	6.4.71
$aa9	Read CJ offset	Return Cold Junction temperature offset	9019-MTCP	6.4.72

6.4 ASCII Command Description

6.4.1 $aaM Read Module Name

Description: Returns the module name from a specified module.

Syntax: **$aaM(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
M	is the Module Name command.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was in-valid.

aa (range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.

(data) A string showing the name of the module (max. 6 chars.)

(cr) is the terminating character, carriage return (0Dh).

Example: command: **$01M(cr)**

response**:** **!019050(cr)**

The command requests the system at address 01h to send its module name. The system at address 01h responds with module name 9050-MTCP indicating that there is an EX9000-MTCP at address 01h.

6.4.2 $aaF Read Firmware Version

Description: Returns the firmware version from a specified module.

Syntax: **$aaF(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
F	is the Firmware Version command.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.
(data)	firmware version of module(max. 6 chars.)
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **$01F(cr)**
response: **!01M1.01(cr)**

The command requests the system at address 01h to send its firmware version. The system responds with firmware version M1.01.

Example: command: **$01F(cr)**
response: **!0160.01(cr)**

The command requests the system at address 01h to send its firmware version. The system responds with firmware version 60.01.

6.4.3 ~aaDnnnnn Set timout to search DHCP

Description: Set timout to search DHCP.

Syntax: ~**aaDnnnnn(cr)** **(**available for firmware V5.26 or later only)

~	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
D	is set DHCP search timeout command.
nnnnn	is DHCP timeout value (10~1800 sec) (dec**)**
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(cr)** if the command is valid.
?aa(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.
(cr)	is the terminating character, carriage return (0Dh).

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

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.

6.4.4 ~aaD Read timout to search DHCP

Description: Read timout to search DHCP.

Syntax: ~**aaD (cr)** (available for firmware V5.26 or later only)

~	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
D	is read DHCP search timeout command.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aannnnn(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.
nnnnn	Timeout value to search DHCP servo
(cr)	is the terminating character, carriage return (0Dh).

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

response: **!0101200 (cr)**

The command read timeout is 1200 seconds

6.4.5 $aaID Read module ID number

Description: Returns the ID number from a specified module.

Syntax: **$aaID(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
ID	is the ID command.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aann(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
nn	represents the ID number of the module.
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **$01ID(cr)**

response: **!010A(cr)**

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

6.4.6 $aaIDFF Set module ID number

Description: Set module ID number. **(command for version 6.000 or later)**

Syntax: **$aaIDFF(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
ID	is the ID command.
FF	Module ID (range 01-FF)
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa (cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **$01ID1A(cr)**

response: **!01(cr)**

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

6.4.7 $aaMD(data) set module description

Description: set module description **(command for version 6.000 or later)**

Syntax: **$aaMD(data)(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
MD	set module description command.
(data)	module description (max 30 characters)
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa (cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **$01MD12DI8DO(cr)**

response: **!01(cr)**

; Sets the desc. of the module 01 to be “**12DI8DO** " and returns a valid response.

6.4.8 $aaMD read module description

Description: read module description **(command for version 6.000 or later)**

Syntax: **$aaMD(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
MD	read module description command.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(data)	module description (max 30 characters)
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **$01MD (cr)**

response: **!0112DI8DO(cr)**

; Read desc. of module 01 and return the moduledesc "**12DI8DO**"

6.4.9 ^aaMAC read MAC address

Description: read MAC address **(command for version 6.000 or later)**

Syntax: **^aaMAC(cr)**

^	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
MAC	command for read MAC address.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aaMMMMMMMMMMMM (cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
MMMMMMMMMMMM	MAC address(hex)
(cr)	is the terminating character, carriage return (0Dh).

Example: command: ^01MAC **(cr)**

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

6.4.10 $aaRS Reboot the module to the power-on state

Description: Reboot the module to the power-on state. **(command for version 6.000 or later)**

Syntax: **$aaRS(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
RS	command for Reboot the module to the power-on state
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **$01RS (cr)**
response: **!01(cr)** **;** Reboot the module to the power-on state.

6.4.11 $aaS1 Reloads the module factory default

Description: Reloads the module factory default
(command for version 6.000 or later)

Syntax: **$aaS1(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
S1	Command to reload the factory default
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(cr)** if the command is valid.
?aa(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(cr)	is the terminating character, carriage return (0Dh).

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

6.4.12 ~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 for version 6.000 or later)

Syntax: **~aaI(cr)**

~	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
I	Command to set the Soft INIT* enable
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(cr)** if the command is valid.
?aa(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(cr)	is the terminating character, carriage return (0Dh).

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

6.4.13 ~aaTNN Sets the soft INIT* timeout value.

Description: Sets the soft INIT* timeout value. **(command for version 6.000 or later)**

Syntax: **~aaTNN(cr)**

~	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (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.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
(cr)	is the terminating character, carriage return (0Dh).

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

6.4.14 $aa5 Reads the Reset Status of a module

Description: Reads the Reset Status of a module. **(command for version 6.000 or later)**

Syntax: **$aa5(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
5	Command for read reset status.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!aaS(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
S	the Reset Status of a module. = 0 - the module is not been reseted = 1 - the module is been reseted
(cr)	is the terminating character, carriage return (0Dh).

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

6.4.15 ^aaMAC Read module MAC address

Description: Read module MAC address. **(command for version 6.000 or later)**

Syntax: **^aaMAC(cr)**

^	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
MAC	Command for Read module MAC address.
(cr)	is the terminating character, carriage return (0Dh).

Response: **!AANNNNNNNNNNNN(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error, communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal address of an EX9000-MTCP module.(always 01)
NNNNNNNNNNNN	The module MAC address(hex).
(cr)	is the terminating character, carriage return (0Dh).

Example: command: **^01MAC(cr)**

response: **!0100E04C360629 (cr)** **;** MAC addr.
00-E0-4C-36-06-29

6.4.16 $aa6 Read DI /DO Channel Status

Description: This command requests that the specified EX9000-MTCP module return the status of its digital input and digital output channels.

Syntax: **$aa6(cr)**

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)

6 is the Digital Data In command.

(cr) is the terminating character, carriage return (0Dh)

Response: **!aa0(data1)(data2) (cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module.

0 command for Read the Digital I/O Status.

For 9050(A)/9051(A)/9055(A) :

(data1) represents the 2-character hexadecimal DO status (00~FF).

(data2) represents the 3-character hexadecimal DI status (000~FFF).

For All modules (Except EX**-9050-MTCP/EX-9051-MTCP/EX-9055-MTCP) :**

(data1) represents the 4-character hexadecimal DO status (0000~FFFF).

(data2) represents the 4-character hexadecimal DI status (0000~FFFF).

(cr) is the terminating character, carriage return (0Dh)

Example: command: **$016(cr)** **; read EX-9050-MTCP DIO status**

response: **!01003004(cr)**

03 represents DO0, DO1 are ON and DO2~DO7 are OFF

004 represents DI2 is ON and DI0, DI1, DI3~DI 11 are OFF

Example: command: **$016(cr)** **; read EX-9053-MTCP DIO status**

response: **!010000030004(cr)**

0003 represents DO0, DO1 are ON and DO2~DO15 are OFF

0004 represents DI 2 is ON and DI0, DI1, DI3~DI15 are OFF

Related command: @AA, @AA6

6.4.17 @aa Read DIO status

Description: Read digital input and output status. All modules

Syntax: **@aa(cr)**

@	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Response: **>(data1)(data2)**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.

For EX**9050-MTCP/EX-9051-MTCP/EX-9055-MTCP :**

(data1)	represents the 2-character hexadecimal DO status (00~FF).
(data2)	represents the 3-character hexadecimal DI status (000~FFF).

For All modules (Except EX**9050-MTCP/EX-9051-MTCP/EX-9055-MTCP) :**

(data1)	represents the 4-character hexadecimal DO status (0000~FFFF).
(data2)	represents the 4-character hexadecimal DI status (0000~FFFF).

(cr)	is the terminating character, carriage return (0Dh)

Example: command: **@01(cr)** **; read EX-9050-MTCP DIO status**
response: **>03004(cr)**

03	represents DO0, DO1 are ON and DO2~DO7 are OFF
004	represents DI2 is ON and DI0, DI1, DI3~DI 11 are OFF

Example: command: **@01(cr)** **; read EX-9050-MTCP DIO status**
response: **>00030004(cr)**

0003	represents DO0, DO1 are ON and DO2~DO15 are OFF
0004	represents DI 2 is ON and DI0, DI1, DI3~DI15 are OFF

Note: (data1) is always 00 or 0000 for none DO modules and (data2) is always 000 or 0000 for none DI modules.

Related command: @AA6, $AA6

6.4.18 $aa7 Read DI latch status

Description: Read DI latch status.

Syntax: **$aa7(cr)**

$	is a delimiter character.
aa	(range 00-2D) represents the 2-character hexadecimal Modbus address (Always 01)
7	represents read DI latch status command.
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aaDDDD(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.
DDDD	represent DI latch status
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$017(cr)**

response: **!010003(cr)**

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

6.4.19 #aa00dd Write All Digital Output

Description: This command sets all digital output channels to the specific EX9000-MTCP module.

Syntax: **#aa00dd(cr)**

#	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)
00	represents Writing to all channels (write a byte) command
dd	represents the data be written to digital output

Response: **!01(cr)** if the command was valid.

?aa(cr) if an invalid command has been issued.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address of a module that is responding. (always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **#010033(cr)**
response: **!01(cr)**

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

6.4.20 #aa1ndd Set Single Digital Output Channel

Description: Set the digital output status of EX9000-MTCP digital output module.

Syntax: **#aa1ndd(cr)**

#	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
n	(range 0-F) represents the specific channel you want to set the output status.
dd	represents the status you want to set to the specific channel. = 00 – output inactive. = 01 - output active.
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.
?aa(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	Delimiter indicating a valid command was received.
?	Delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **#011201(cr)**
response: **!01**
The command set digital channel 2 "ON" status for the specific module at address 01h.

Example: command: **#011200(cr)**
response: **!01**
The command set digital channel 2 "OFF" status for the specific module at address 01h.

6.4.21 $aa0Mcc Read DI counter filter signal width of channel N(uint= 0.5ms)

Description: Read DI counter min. low and high filter(Debounce) signal width of channel N (**uint = 0.5ms**)

Syntax: **$aa0Mcc(cr)**

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
0M	command for read DI counter filter of channel N
cc	represents DI channel number (00~0F)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(data)	DI counter min. LOW/HIGH witdh, 16-chars(L/H) each channel, (hex/unit=0.5ms)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$010M00 (cr)**

response: **!010000000200000003 (cr)**

00000002	represents channel(0) Low signal width.
00000003	represents channel(0) High signal width.

6.4.22 $aa0M(data) Set DI counter filter signal width of channel N(uint= 0.5ms)

Description: Set DI counter min. low and high filter(Debounce) signal width of channel N (uint = 0.5ms)

Syntax: **$aa0M(data)(cr)**

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
0M	command for read DI counter filter of channel N
cc	represents DI channel number (00~0F)
(data)	DI counter min. LOW/HIGH witdh, 16-chars(L/H) each channel, (hex/**unit=0.5ms**)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$010M0000000200000003 (cr)**

response: **!01 (cr)**

set channel(0) Low signal width to 00000002. Set channel(0) High signal width to 00000003.

6.4.23 $aaEcn Start/stop single DI counter

Description: start/stop single digital input counter (*Falling Edge Trigger)*

Syntax: **$aaEcn(cr)**

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
E	represents enable/disable DI counter command
c	represents DI counter channel number
n	represents enable/disable option (n=0 disable / n=1 enable)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$01E21(cr)**

response: **!01(cr)**

1 represents enable DI counter channel 2

Example: command: **$01E20(cr)**

response: **!01(cr)**

0 represents disable DI counter channel 2

6.4.24 $aaCn Clear single DI counter value with overflow flag

Description: clear single digital input counter value with overflow flag

Syntax: **$aaCn(cr)**

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
C	represents clear DI counter command
n	represents DI channel number (0~F)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$01C2(cr)**

response: **!01(cr)**

2 represents DI counter channel 2

6.4.25 #aa Read all DI counter value

Description: read all digital input counter value

Syntax: **#aa(cr)**

#	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(data)(data)(data)….(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(data)…	10-characters(**decimal**) represents counter values
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **#01(cr)**

response: **!0100000001230230000001000000052300000050011010000043 2 ……(cr)**

0000000123	represents channel #0 counter value is **123** represents
0230000001	channel #1 counter value is **230000001** represents
0000000523	channel #2 counter value is **523** represents channel #3
0000005001	counter value is **5001**
	… so on

Note:

(1)This command is valid for EX9050/9051/9055-MTCP digital I/O modules only.

(2)This command is supported for EX9050/9051/9055-MTCP with firmware V2.21 or later.

6.4.26 #aan Read single DI counter value

Description: read single digital input counter value

Syntax: **#aan(cr)**

#	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
n	represents DI channel number (0~F)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(data)(cr)** if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(data)	10-characters(**decimal**) represents counter value
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **#012(cr)**

response: **!010000000123(cr)**

represents DI counter channel 2

0000000123 represents counter value is **123**

6.4.27 #aarn Read single DI counter value with overflow

Description: read single digital input counter value with overflow **(The Command for version 6.000 or later)**

Syntax: **#aarn(cr)**

#	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
r	represent read single DI counter value with overflow command.
n	represents DI channel number (0~F)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aar(data)(cr)** if the command is valid.
?01(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
r	DI Counter Overflow. = 0 - No counter overflow has occurred. = 1 - A counter overflow has occurred.
(data)	10-characters(**decimal**) represents counter value (0~4294967295)
(cr)	is the terminating character, carriage return (0Dh)

Example(1): command: **#01RC(cr)**
response: **!0110000000123(cr)**
represents DI counter channel 12,
10000000123 represents counter value is 123 and counter overflow has occurred.

Example(2): command: **#01RC(cr)**
response: **!0100000000123(cr)**
represents DI counter channel 12,
00000000123 represents counter value is 123 and No counter overflow has occurred..

6.4.28 #aa2npppppppp Write DO pulse counts

Description: Generates pulse output of the specified DO channel.

Syntax: **#aa2npppppppp (cr)**

#	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
2	represent generates DO pulse output command.
n	represents DO channel n
pppppppp	represents pulse counts (8 digits) (0000~FFFFFFFF) if pppppppp=00000000, continue DO pulse if pppppppp=00000001, stop DO pulse
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **#0123001F(cr)**

response: **!01(cr)**

The command force the DO channel #3 to output 31(1FH) pulses

6.4.29 $aaCLS Clear DI latch status

Description: Clear DI latch status.

Syntax: **$aaCLS(cr)**

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
CLS	represents clear DI latch status command.
(cr)	is the terminating character, carriage return (0Dh)

Response: !aa(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$01CLS(cr)**

response: **!01(cr)**

The command clears all DI latch status

6.4.30 ~aaDSMN Set DI/O active state

Description: Set digital input/output active state.
(The command for version 6.000 or later)

Syntax: **~aaDSMN(cr)**

~ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)

DS Command for setting DIO active status

M Digital input channel active values,

= 0 - input value 0 for input active(ON),
input value 1 for input inactive (OFF or non-signal/OPEN).

= 1 - input value 1 for input active(ON),
input value 0 for input inactive (OFFF or non-signal/OPEN).

N Digital output channel active values,

= 0 – output value 1 for output active(ON), output
value 0 for output inactive(OFF).

= 1 - output value 1 for output inactive(OFF),
output value 0 for output active(ON).

Response: **!aa(cr)** if the command was valid.

?aa(cr) if an invalid command has been issued.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of a module that is responding. (always 01)

(cr) is the terminating character, carriage return (0Dh)

Example:
Read EX9050-MTCP active status , Response: DI active read value is 0
Command: ~01DS(cr)
Response: !0101(cr)

For the EX9050-MTCP (ID=01), Reads the status of the DIO
Command: @01(cr)
Response: >3DFFF (cr)

For the EX9050-MTCP (ID=01), Set input active value to 1 and output value 1(ON) for output active,
Command: ~01DS11(cr)
Response: !01 (cr)

For the EX9050-MTCP(ID=01), Reads the status of the DIO
Command: @01(cr)
Response: >3D000 (cr)

6.4.31 ~aaDS Read DI/O active state

Description: Read digital input/output active state. **(Command for version 6.000 or later)**

Syntax: **~aaDS(cr)**

~ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)

DS Command for setting DIO active status

Response: **!aaMN(cr)** if the command was valid.

?aa(cr) if an invalid command has been issued.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of a module that is responding. (always 01)

M Digital input channel active values,

= 0 - input value 0 for input active(ON),
input value 1 for input inactive (OFF or non-signal/OPEN).

= 1 - input value 1 for input active(ON),
input value 0 for input inactive (OFFF or non-signal/OPEN).

N Digital output channel active values,

= 0 – output value 1 for output active(ON), output
value 0 for output inactive(OFF).

= 1 - output value 1 for output inactive(OFF),
output value 0 for output active(ON).

(cr) is the terminating character, carriage return (0Dh)

Example:
Read EX9050-MTCP active status , Response: DO active read value is 0
Command: ~01DS(cr)
Response: !0100(cr)

For the EX9050-MTCP (ID=01), Reads the status of the DIO
Command: @01(cr)
Response: >00FFF (cr)

For the EX9050-MTCP(ID=01), Set input active value to 0 and output value 1(ON) for output active,
Command: ~01DS01(cr)
Response: !01 (cr)

For the EX9050-MTCP(ID=01), Reads the status of the DIO
Command: @01(cr)
Response: >3FFFF (cr)

6.4.32 @aa6 Read DIO status

Description: read the status of all 16 DO and 16 DI channels. **(Command for version 6.000 or later)**

Syntax: **@aa6(cr)**

@	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
6	Command for read DIO status
(cr)	is the terminating character, carriage return (0Dh)

Response: **>TTTTNNNN**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
TTTT	represents the 4-character hexadecimal DO status (0000~FFFF).
NNNN	represents the 4-character hexadecimal DI status (0000~FFFF)
(cr)	is the terminating character, carriage return (0Dh)

Example: (for EX-9050-MTCP)

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

6.4.33 @aa6DDDD Write DO channels(0~15)

Description: Write a value to digital output channels(0~15). **(Command for version 6.000 or later)**

Syntax: **@aa6DDDD(cr)**

@	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
6	Command for read DIO status
DDDD	Represents the data be written to digital output(channels 0~15)
(cr)	is the terminating character, carriage return (0Dh)

Response: **> (cr)** if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

Example: (for EX9053-MTCP)

command: **@016123A(cr)**

response: **> (cr)**

6.4.34 @aa6Onss Set a single digital output for channel N

Description: Set a single digital output for channel N. **(Command for version 6.000 or later)**

Syntax: **@aa6Onss(cr)**

@	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
6O	Command for set a single digital output.
n	channel number(0~F)
ss	output ON(active) / OFF(inactive) state. = 00 - set the digital output channel to OFF(inactive). = 01 - set the digital output channel to ON(active).
(cr)	is the terminating character, carriage return (0Dh)

Response: **!aa**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

Example: (for EX-9050-MTCP) set DO(1) to active

command:	**@016O1(cr)**	; Read channel(1) DO status
response:	**>00 (cr)**	**;** 00 represents channel(1) DO inactive
command:	**@016O101(cr)**	; set DO(1) to active
response:	**!01 (cr)**	
command:	**@016O1(cr)**	; Read channel(1) DO status
response:	**>01 (cr)**	**;** 01 represents channel(1) DO active

6.4.35 @aa6ON Read a single digital output for channel N

Description: Read a single digital output for channel N. **(Command for version 6.000 or later)**

Syntax: **@aa6On(cr)**

@	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
6O	Command for Read a single digital output.
n	channel number(0~F)
(cr)	is the terminating character, carriage return (0Dh)

Response: **>DD**(cr) if the command is valid.
?01(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
DD	output status = 00 - OFF (inactive) = 01 - ON (active)
(cr)	is the terminating character, carriage return (0Dh)

Example: (for EX9050-MTCP) Read channel(1) DO status
command: **@016O1(cr)**
response: **>01 (cr)** **;** 01 represents channel(1) DO active

6.4.36 @aa6IN Read a single digital input for channel N

Description: Read a single digital input for channel N. **(Command for version 6.000 or later)**

Syntax: **@aa6IN (cr)**

@	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
6I	Command for Read a single digital input.
N	channel number(0~F)
(cr)	is the terminating character, carriage return (0Dh)

Response: **>DD)**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
DD	input status

= 00	OFF (inactive)
= 01	ON (active)
-	

(cr)	is the terminating character, carriage return (0Dh)

Example: (for EX9050-MTCP) Read channel(1) DI status command:**@016I1(cr)**

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

6.4.37 $AA9NN(data) Set DO pulse and low/high width for channel N

Description: Set DO pulse low/high width and Low/high delay output width for channel N (**unit: 0.5ms**). **(Command for version 6.000 or later)**

Syntax: **$aa9NN (data)(cr)**

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
9	Command for Read a single digital input.
NN	channel number.
	= 00~0F (hex) - channel number.
	= FF (hex) - for all channels
(data)	DO pulse time interval and low/high width (LLLLHHHHUUUUDDDD)

LLLL	- 4 char, DO pulse low signal width	(hex 0001~hex 3332/**uint 0.5ms**)
HHHH	- 4 char, DO pulse high signal width	(hex 0001~hex 3332/uint 0.5ms)
UUUU	- 4 char, DO low to high delay width	(hex 0001~hex 3332/uint 0.5ms)
DDDD	- 4 char, DO high to low delay width	(hex 0001~hex 3332/uint 0.5ms)

(cr) is the terminating character, carriage return (0Dh)

Response: **!AA**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address of a module that is responding. (always 01)
(cr)	is the terminating character, carriage return (0Dh)

Example: (for EX9063-MTCP) Set DO pulse and Low/high delay output width for channel 2

command: **$0190201F401F4012C00C8(cr)**

response: **!01 (cr)** ; valid

6.4.38 $AA9NN Read DO pulse and low/high width for channel N

Description: Read DO pulse low/high width and Low/high delay output width for channel N (**unit: 0.5ms**). **(Command for version 6.000 or later)**

Syntax: $aa9NN (cr)

$	is a delimiter character.
aa	represents the 2-character hexadecimal Modbus address (Always 01)
9	Command for Read a single digital input.
NN	channel number.
	= 00~0F (hex) - channel number.
	= FF (hex) - for all channels
(cr)	is the terminating character, carriage return (0Dh)

Response: !AA(data)(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address of a module that is responding. (always 01)
(data)	DO pulse time interval and low/high width (LLLLHHHHUUUUDDDD)

LLLL	- 4 char, DO pulse low signal width	(hex 0001~hex 3332/**uint 0.5ms**)
HHHH	- 4 char, DO pulse high signal width	(hex 0001~hex 3332/uint 0.5ms)
UUUU	- 4 char, DO low to high delay width	(hex 0001~hex 3332/uint 0.5ms)
DDDD	- 4 char, DO high to low delay width	(hex 0001~hex 3332/uint 0.5ms)

(cr) is the terminating character, carriage return (0Dh)

Example: (for EX9063-MTCP) Read DO pulse and Low/high delay output width for channel 2

command: **$01902 (cr)**

response: **!0101F401F4012C00C8 (cr)** **;** DO data

Printed Date: June 1, 2016

6.4.39 ~** Host OK (via broadcast)

Description: Host send this command to all modules via broadcast for send the information "Host OK". (No reply to response) .

(Command for firmware version 6.000 or later)

Note: When host watchdog timer is enable, host computer must send this command to all module before timeout otherwise "Host watchdog timer enabled" module's output value will go to safety state output value.

Syntax: **~** (cr)**

~	is a delimiter character.
**	Command for Host OK
(cr)	is the terminating character, carriage return (0Dh)

Response: No response.

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

6.4.40 ~AA** Host OK (to the Specific module)

Description: Host send this command to the Specific module for send the information "Host OK".

(Command for firmware version 6.000 or later)

Note: When host watchdog timer is enable, host computer must send this command to the specific module before timeout otherwise "Host watchdog timer enabled" module's output value will go to safety state output value.

Syntax: **~AA** (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
******	Command for Host OK
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AA**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

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

6.4.41 ~AA0 Read watchdog timeout status

Description: Read watchdog timeout status. **(Command for firmware version 6.000 or later)**

Syntax: **~AA0 (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
0	Command for reading timeout status
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AASS**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

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(=0)

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

? delimiter indicating the command was invalid.

(cr) is the terminating character, carriage return (0Dh)

Example(1): 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: !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

6.4.42 ~AA1 Reset host watchdog timeout status

Description: Reset host watchdog timeout status. **(Command for firmware version 6.000 or later)**

Syntax: **~AA1 (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
1	Command for resetting watchdog timeout status
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AA** (cr) if the command is valid.
?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

Example: Reads the host watchdog status of module 01 and shows that a host watchdog timeout has occurred.
Command: ~010 (cr)
Response: !0104 (cr)

Resets the host watchdog timeout status of module 01 and returns a valid response.
Command: ~011 (cr)
Response: !01 (cr)

Reads the host watchdog status of module 01 and shows that no host watchdog timeout has occurred.
Command: ~010
Response: (cr) !0100
(cr)

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

6.4.43 ~AA2 Read host communication Timeout value.

Description: read host communication Timeout value. **(Command for firmware version 6.000 or later)**

Syntax: **~AA2 (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
2	Command for reading watchdog timeout value
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AAEVVV**(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.

E	Host watchdog enabled status
	E = 1 – Enable.
	E = 0 – Disable..

VVV	Timeout value in hex format from hex 001 to 28F . (01 denotes 0.1 seconds and FF denotes 25.5 seconds)
(cr)	is the terminating character, carriage return (0Dh)

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

6.4.44 ~AA3EVVV Set Host watchdog timeout interval

Description: Enable/Disable Host watchdog and set timeout interval(unit = 0.1sec)
(Command for firmware version 6.000 or later)

Syntax: **~AA3EVVV (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
3	Command for set host wdt Enable/ disable and host wdt timeout value.

E	Host watchdog enabled status
	E = 1 – Enable.
	E = 0 – Disable..

VVV	Timeout value in hex format from hex 001 to 28F. (01 denotes 0.1 seconds and FF denotes 25.5 seconds)
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AA** (cr) if the command is valid.
?01(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.
(cr)	is the terminating character, carriage return (0Dh)

Note: 1. 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 safety state(see "~AA5V") and The Power-LED on the module will go to flash.

2. After timeout the all of D/O commands are disabled.

Example:
Set module (ID=01) to have watchdog timeout value 20.0 seconds and enable host watchdog.
Command: ~01310C8(cr)
Response: !01(cr)

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)

Host send this command to all modules for send the information "Host OK"
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. ThePower- LED on the module will go to flash. The flash LED indicates the host watchdog is timeout and timeout status is set.

Read watchdog timeout status, The module returns 01, which denotes that a host watchdog timeout has occurred.
Command: ~010(cr)
Response: !0104(cr)

Reset watchdog timeout status. Watchdog timeout is cleared and LED stop 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: !0100(cr) Timeout status is cleared

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

6.4.45 ~AA4V Read the power-on DO value or the safe DO value of a module

Description: read the power-on DO value or the safe DO value of a module
(Command for firmware version 6.000 or later)

Syntax: **~AA4V (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01) Command
4	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.
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AADDDD** (cr) if the command is valid.
?01(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.
DDDD	powe-on or safe value.
(cr)	is the terminating character, carriage return (0Dh)

Example:

Read Power on value and return power-on value 5A5A.
Command: ~014P(cr)
Response: !045A5A(cr)

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

6.4.46 ~AA5V Sets the current value as the power-on DO value or the safe DO value

Description: sets the current value as the power-on DO value or the safe DO value
(Command for firmware version 6.000 or later)

Syntax: **~AA5V (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
5	command for sets the current value as the power-on DO value or the safe DO value
V	sets the current value as the power-on DO value or the safe DO value

= P – set to power-on value,
= S - set to safe value.

(cr) is the terminating character, carriage return (0Dh)

Response: **!AA** (cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.
(cr)	is the terminating character, carriage return (0Dh)

Example:

For the EX9050-MTCP(ID=01), Set module to have output value 2A. **Command:** @012A(cr)
Response: > (cr)

For the EX9050-MTCP(ID=011), Set current output value 2A as safe value. **Command:** ~015S(cr)
Response: !01(cr)

For the EX9050-MTCP(ID=01), Set module to have output value 15.
Command: @0115(cr)
Response: > (cr)

For the EX9050-MTC0(ID=01), Set current output value 15 as power-on value. **Command:** ~015P(cr)
Response: !01(cr)

For the EX9050-MTCP(ID=01), Read Power on value and return power-on value 0015. **Command:** ~014P (cr)
Response: !010015 (cr)

For the EX9050-MTCP(ID=01), Read Power on value and return safe value 2A. **Command:** ~014S (cr)
Response: !01002A (cr)

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

6.4.47 ~AA3PPP set module Power-on delay time.

Description: set module Power-on delay time.
(Command for firmware version 6.000 or later)

Syntax: **~AA3PPP (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
3	command for set module Power-on delay time.
PPP	Power-on delay time(unit=0.1sec) to start Communication timeout.(range 001~28F) (default 5 sec).
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AA** (cr)if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

?	delimiter indicating the command was invalid. delimiter
!	indicating a valid command was received.
(cr)	is the terminating character, carriage return (0Dh)

Example:

set power-on delay time to 096 (15 sec)

Command: ~013096 (cr)
Response: !01 (cr)

Ref. command: ~AA3P

6.4.48 ~AA3P Read module Power-on delay time.

Description: Read module Power-on delay time.
(Command for firmware version 6.000 or later)

Syntax: **~AA3P (cr)**

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address (Always 01)
3P	command for read module Power-on delay time.
(cr)	is the terminating character, carriage return (0Dh)

Response: **!AAPPP** (cr) if the command is valid.
?01(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

?	delimiter indicating the command was invalid. delimiter
!	indicating a valid command was received.
PPP	Power-on delay time(unit=0.1sec) to start Communication timeout. (range 001~28F)
(cr)	is the terminating character, carriage return (0Dh)

Example:
set power-on delay time to 096 (15 sec)
Command: ~013096 (cr)
Response: !01 (cr)

Read power-on delay time and return 096 (15 sec)
Command: ~013P (cr)
Response: !01096 (cr)

Ref. command: ~AA3PPP

6.4.49 #aan Read Analog Input from Channel N

Description: Returns the input data from a specified analog input channel in a specified module.

Syntax: #aan(cr)

is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

n (range 0-8) represents the specific channel you want to read the input data.

(cr) is the terminating character, carriage return (0Dh).

Response: >(data)(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

(cr) is the terminating character, carriage return (0Dh).

Example: command: **#012(cr)**

response: **>+01.000(cr)**

Channel 2 of the EX9000 analog module at address 01h responds with an input value +01.000.

6.4.50 #aa Read Analog Input from All Channels

Description: Returns the input data from all analog input channels in a specified module.

Syntax: #aa(cr)

#	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
(cr)	is the terminating character, carriage return (0Dh).

Response: >(data)(data)(data)(data)(data)(data)(data)(data)(data)(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
Data	represents analog data
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh).

Note: The latest data returned is the average value of the preset channels in this module.

Example: command: **#01(cr)**

response: **>+00.000+01.000+02.000+03.800+04.000+05.000+06.000+07.000+04.320(cr)**

where channel #0 data is **+00.000**, channel #1 data is **+01.000**, channel #2 data is **+04.320,,,,** and average data is **+04.320**

6.4.51 $aaAcctt Set analog input type (range)

Description: Set the analog input type (range) in EX9000-MTCP analog input module.

Syntax: $aaAcctt(cr)

$ is a delimiter character.

aa represents the 2-character hexadecimal Modbus address (Always 01)

A represents the analog input setting command.

cc represents the specific channel you want to set the input type.

tt (range 00-FF) represents the type you want to set to the specific channel(see 6.4.51)

(cr) is the terminating character, carriage return (0Dh)

Response: !01(cr) if the command is valid.

?01(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! Delimiter indicating a valid command was received.

? Delimiter indicating the command was invalid.

01 represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module. (Always 01)

(cr) is the terminating character, carriage return (0Dh)

Example: command: **$01A030D(cr)**

response: **!01(cr)**

The command set analog input channel 3 to type 0D (0~20mA) for the specific analog input module

6.4.52 $aaBhh Read analog input type

Description: Return the input type of the specified analog channel

Syntax: **$aaBhh(cr)**

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)

B represents read the analog input type command.

hh is the analog input channel number represents the 2-character in hexadecimal format.

(cr) is the terminating character, carriage return (0Dh)

Response: **!aann(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa represents the 2-character hexadecimal Modbus network address of module(always 01).

nn a 2-character hexadecimal value representing the type of the analog input channel.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **$01B01(cr)**

response: **!0108(cr)**

The first 2-character portion of the response (exclude the "!" character) indicates the address of the EX9000-MTCP module. The second 2-character portion of the response is the type of channel (For each analog module, the type number is different, ref to Figure 6-1 Analog input types)

Code(Hex)	Type
0x07	4-20mA
0x08	+/-10V
0x09	+/-5V
0x0a	+/-1V
0x0b	+/-500mV
0x0c	+/-150mV
0x0d	0-20mA
0x0e	J type -8824 uV ~ 69536 uV,
0x0f	K type -5891 uV ~ 54807 uV
0x10	T type -5603 uV ~ 20869 uV
0x11	E type -9835 uV ~ 76373 uV
0x12	R type -0000 uV ~ 21101 uV
0x13	S type -0000 uV ~ 18693 uV
0x14	B type -0000 uV ~ 13820 uV
0x20	IEC Pt100 -50C ~ 150C
0x21	IEC Pt100 0C ~ 100C
0x22	IEC Pt100 0C ~ 200C
0x23	IEC Pt100 0C ~ 400C
0x24	IEC Pt100 -200C ~ 200C
0x25	JIS Pt100 -50C ~ 150C
0x26	JIS Pt100 0C ~ 100C
0x27	JIS Pt100 0C ~ 200C
0x28	JIS Pt100 0C ~ 400C
0x29	JIS Pt100 -200C ~ 200C
0x2A	Pt1000 -40C ~ 160C
0x2B	BALCO500 -30C ~ 120C
0x2C	Ni -80C ~ 100C
0x2D	Ni 0C ~ 100C

Figure 6-20 Analog input types

6.4.53 $aa0 Span Calibration

Description: Calibrates a specified module to correct for gain errors

Syntax: **$aa0(cr)**

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

0 represents the span calibration command.

(cr)is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.

(cr) is the terminating character, carriage return (0Dh)

Note: In order to successfully calibrate an analog input module's input range, a proper calibration input signal should be connected to the analog input module before and during the calibration process.

6.4.54 $aa1 Zero Calibration

Description: Calibrates a specified module to correct for offset errors

Syntax: **$aa1(cr)**

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

1 represents the zero calibration command.

(cr) is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000M module.

(cr) is the terminating character, carriage return (0Dh)

Note: In order to successfully calibrate an analog input module's input range, a proper calibration input signal should be connected to the analog input module before and during the calibration process.

6.4.55 $aa6 Read Channel Enable/Disable Status

Description: Asks a specified module to return the Enable/Disable status of all analog input channels

Syntax: $aa6(cr)

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
6	is the read channels status command.
(cr)	is the terminating character, carriage return (0Dh)

Response: !aamm(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.
mm	are two hexadecimal values. Each value is interpreted as 4 bits. The first 4-bit value represents the status of channels 7-4, the second 4 bits represents the status of channels 3-0. A value of 0 means the channel is disabled, while a value of 1 means the channel is enabled.
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$016(cr)**

response: **!01FF(cr)**

The command asks the specific module at address 01h to send Enable/Disable status of all analog input channels. The analog input module responds that all its channels are enabled (FF equals 1111 and 1111).

6.4.56 $aa5mm Set Channel Enable/Disable Status

Description: Set Enable/Disable status for all analog input channels

Syntax: **$aa5mm(cr)**

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

5 identifies the enable/disable channels command.

mm (range 00-FF) are two hexadecimal characters. Each character is interpreted as 4 bits. The first 4-bit value represents the status of channels 7-4; the second 4-bit value represents the status of channels 3-0. A value of 0 means the channel is disabled, while a value of 1 means the channel is enabled.

(cr) is the terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000 module.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **$01581(cr)**

response: **!01(cr)**

The command enables/disables channels of the analog input module at address 01h. Hexadecimal 8 equals binary 1000, which enables channel 7 and disables channels 4, 5 and 6. Hexadecimal 1 equals binary 0001, which enables channel 0 and disables channels 1, 2 and 3.

6.4.57 #aaMH Read Maximum Value

Description: Read the maximum values from all analog input channels in a specified analog module

Syntax: #aaMH(cr)

is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

MH represents the read maximum value command.

(cr) is the terminating character, carriage return (0Dh)

Response: >(data)(data)(data)(data)(data)(data)(data)(data)(data)(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **#01MH(cr)**

response:**>+01.000+02.000+00.000+06.000+10.000+09.000 +05.400+05.000**

The command asks the specific module at address 01h to send historic maximum value from all analog input channels.

6.4.58 #aaMHn Read Maximum Value from channel N

Description: Read the maximum value from a specific channel in a specified module

Syntax: **#aaMHn(cr)**

is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

MH represents the read maximum value command.

n (range 0-8) represents the specific channel you want to read the input data.

(cr) is the terminating character, carriage return (0Dh)

Response: **>(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **#01MH2(cr)**

response: **>+10.000(cr)**

The command asks the specific module at address 01h to send historic maximum value from analog input channel 2.

6.4.59 #aaML Read Minimum Value

Description: Read the minimum values from all analog input channels in a specified module

Syntax: **#aaML(cr)**

is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)

ML represents the read minimum value command.

(cr) is the terminating character, carriage return (0Dh)

Response: >(data)(data)(data)(data)(data)(data)(data)(data)(data)(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **#01ML(cr)**

response:**>+00.000-08.000-05.000+00.000+10.000+10.000+10.000+10.000+10.000(cr)**

The command asks the specific module at address 01h to send historic minimum value from all AI channels.

6.4.60 #aaMLn Read Minimum Value from channel N

Description: Read the minimum value from a specific analog input channel in a specified module

Syntax: **#aaMLn(cr)**

#	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address (Always 01)
ML	represents the read minimum value command.
n	(range 0-8) represents the specific channel you want to read the input data.
(cr)	is the terminating character, carriage return (0Dh)

Response: **>(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **#01ML3(cr)**

response: **>-07.000(cr)**

The command asks the specific module at address 01h to send historic minimum value from analog input channel 3.

6.4.61 $aaCjAhs Set Alarm Mode

Description: Sets the High/Low alarm of the specified input channel in the addressed EX9000-MTCP module to either Latching or Momentary mode.

Syntax: $aaCjAhs(cr)

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)
Cj	identifies the desired channel **j** (**j** : 0 to 7).
Ah	is the Set Alarm Mode command. **h** indicates alarm types (H = High alarm, L = Low alarm)
s	indicates alarm modes (M = Momentary mode,L = Latching mode)
(cr)	represents terminating character, carriage return (0Dh)

Response: !aa(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
aa	represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module.
(cr)	represents terminating character, carriage return (0Dh)

Example: command: **$01C1AHL(cr)**

response: **!01(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is instructed to set its High alarm in latching mode. The module confirms that the command has been received.

6.4.62 $aaCjAh Read Alarm Mode

Description: Returns the alarm mode for the specified channel in the specified EX9000-MTCP module.

Syntax: $aaCjAh(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)

Cj identifies the desired channel **j** (**j** : 0 to 7).

Ah is the Read Alarm Mode command.**h** indicates the alarm types (H = High alarm,L = Low alarm)

(cr) represents terminating character, carriage return (0Dh)

Response: !aas(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module.

s indicates alarm modes (M = Momentary mode, L = Latching mode)

(cr) represents terminating character, carriage return (0Dh)

Example: command: **$01C1AL(cr)**

response: **!01M(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is instructed to return its Low alarm mode. The system responds that it is in Momentary mode.

6.4.63 $aaCjAhEs Enable/Disable Alarm

Description: Enables/Disables the High/Low alarm of the specified input channel in the addressed EX9000-MTCP module

Syntax: $aaCjAhEs(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)

Cj identifies the desired channel **j** (**j** : 0 to 7).

AhE is the Set Alarm Mode command. **h** indicates alarm type (H = High alarm, L = Low alarm), and **s** indicates alarm enable/disable (E = Enable, D = Disable)

(cr) represents terminating character, carriage return (0Dh)

Response: !aa(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal address of the corresponding EX9000-MTCP module.

(cr) represents terminating character, carriage return (0Dh)

Example: command: **$01C1ALEE(cr)**

response: **!01(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is instructed to enable its Low alarm function. The module confirms that its Low alarm function has been enabled.

Note: An analog input module requires a maximum of 2 seconds after it receives an Enable/Disable Alarm command to let the setting take effect. During this interval, the module can not be addressed to perform any other actions.

6.4.64 $aaCjCh Clear Latch Alarm

Description: Sets the High/Low alarm to OFF (no alarm) for the specified input channel in the addressed EX9000-MTCP module

Syntax: **$aaCjCh(cr)**

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)

Cj identifies the desired channel **j** (**j** : 0 to 7).

Ch is the Clear Latch Alarm command. **h** indicates alarm type (H = High alarm, L = Low alarm)

(cr) represents terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal Modbus network address of the corresponding EX9000-MTCP module

(cr) represents terminating character, carriage return (0Dh)

Example: command: **$01C1CL(cr)**

response: **!01(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is instructed to set its Low alarm state to OFF. The system confirms it has done so accordingly.

6.4.65 $aaCjAhCCn Set Alarm Connection

Description: Connects the High/Low alarm of the specified input channel to interlock the specified digital output in the addressed EX9000-MTCP module

Syntax: $aaCjAhCCn(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)

Cj identifies the desired analog input channel **j** (**j** : 0 to 7).

AhC is the Set Alarm Connection command.**h** indicates alarm type (H = High alarm, L = Low alarm)

Cn identifies the desired digital output channel **n** (**n** : 0 to 1). To disconnect the digital output, n should be set as ˜*****

(cr) represents terminating character, carriage return (0Dh)

Response: !aa(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal Modbus network address of the corresponding EX9000-MTCP module.

(cr) represents terminating character, carriage return (0Dh)

Example: command: **$01C1ALCC0(cr)**

response: **!01(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is instructed to connect its Low alarm to the digital output of channel 0 in the specific module. The system confirms it has done so accordingly.

6.4.66 $aaCjRhC Read Alarm Connection

Description: Returns the High/Low alarm limit output connection of a specified input channel in the addressed module

Syntax: $aaCjRhC(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module. (Always 01)

Cj identifies the desired analog input channel **j** (**j** : 0 to 7).

RhC is the Read Alarm Connection command.**h** indicates alarm type (H = High alarm, L = Low alarm)

(cr) represents terminating character, carriage return (0Dh)

Response: !aaCn(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal Modbus network address of the corresponding EX9000-MTCP module.

Cn identifies the desired digital output channel n (n : 0 to 1) whether interlock with the alarm of the specific analog input channel. If the values of n are "*", the analog input has no connection with a digital output point.

(cr) represents terminating character, carriage return (0Dh)

Example: command: **$01C1RLC(cr)**

response: **!01C0(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is instructed to read its Low alarm output connection. The system responds that the Low alarm output connects to the digital output at channel 0 in the specific module.

6.4.67 $aaCjAhU Set Alarm Limit

Description: Sets the High/Low alarm limit value for the specified input channel of a specified EX9000-MTCP module.

Syntax: **$aaCjAhU(data)(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)
Cj	identifies the desired analog input channel **j** (**j** : 0 to 7).
AhU	is the Set Alarm Limit command.**h** indicates alarm type (H = High alarm, L = Low alarm)
(data)	represents the desired alarm limit setting. The format is always in engineering units.
(cr)	represents terminating character, carriage return (0Dh)

Response: **!aa(cr)** if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
aa	represents the 2-character hexadecimal Modbus network address of the corresponding EX9000-MTCP module.
(cr)	represents terminating character, carriage return (0Dh)

Example: command: **$01C1AHU+080.00(cr)**

response: **!01(cr)**

The high alarm limit of the channel 1 in the specific module at address 01h is been set +80. The system confirms the command has been received.

Note: An analog input module requires a maximum of 2 seconds after it receives a Set Alarm Limit command to let the settings take effect. During this interval, the module cannot be addressed to perform any other actions.

6.4.68 $aaCjRhU Read Alarm Limit

Description: Returns the High/Low alarm limit value for the specified input channel in the addressed EX9000-MTCP module

Syntax: $aaCjRhU(cr)

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)
Cj	identifies the desired analog input channel **j** (**j** : 0 to 7).
RhU	is the Read Alarm Limit command. **h** indicates alarm type (H = High alarm, L = Low alarm)
(cr)	represents terminating character, carriage return (0Dh)

Response: !aa(data)(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

!	delimiter indicating a valid command was received.
aa	represents the 2-character hexadecimal Modbus network address of the corresponding EX9000-MTCP module.
(data)	represents the desired alarm limit setting. The format is always in engineering units.
(cr)	represents terminating character, carriage return (0Dh)

Example: command: **$01C1RHU(cr)**

response: **!01+2.0500(cr)**

Channel 1 of the EX9000-MTCP module at address 01h is configured to accept 5V input. The command instructs the system to return the High alarm limit value for that channel. The system responds that the High alarm limit value in the desired channel is 2.0500 V.

6.4.69 $aaCjS Read Alarm Status

Description: Reads whether an alarm occurred to the specified input channel in the specified EX9000-MTCP module

Syntax: $aaCjS(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address of an EX9000-MTCP module(Always 01)

Cj identifies the desired analog input channel **j** (**j** : 0 to 7).

S is the Read Alarm Status command.

(cr) represents terminating character, carriage return (0Dh)

Response: !aahl(cr) if the command was valid

?aa(cr) if an invalid operation was entered.

There is no response if the system detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal address Modbus of the corresponding EX9000-MTCP module.

h represents the status of High alarm. "1" means the High alarm occurred, '0" means it did not occur.

l represents the status of Low alarm. ˜1" means the Low alarm occurred, ˜0" means it did not occur.

(cr) represents terminating character, carriage return (0Dh)

Example: command: **$01C1S(cr)**

response: **!0101(cr)**

The command asks the module at address 01h to return its alarm status for channel 1. The system responds that a High alarm has not occurred, but the Low alarm has occurred.

6.4.70 $aa3 Read cold junction temperature

Description: Return the Cold Junction temperature of EX9019-MTCP

Syntax: **$aa3(cr)**

$	is a delimiter character.
aa	(range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)
3	is the command to read cold junction temperature.
(cr)	is the terminating character, carriage return (0Dh)

Response: **>(data)(cr)** if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(data)	a 8-character hexadecimal value representing the cold junction temperature.
(cr)	is the terminating character, carriage return (0Dh)

Example: command: **$013(cr)**

response: **>+00017.5(cr)**

The command asks the specific module at address 01h to return the cold junction temperature of specified module. The response is +17.5C

6.4.71 $aa9hhhhh Set CJ offset

Description: Set Cold Junction temperature offset of EX9019-MTCP

Syntax: $aa9hhhhh(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)

9 is the command to set cold junction temperature offset.

hhhhh is the offset value times by 80 (5-character hexadecimal format)

(cr) terminating character, carriage return (0Dh)

Response: !aa(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

aa represents the 2-character hexadecimal address Modbus of the corresponding EX9019-MTCP module.

? delimiter indicating the command was invalid.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **$019000A0(cr)**

response: **!01(cr)**

This example need to set cold junction offset to 2C , then the actual ASCII value should be 2 *80=160 (hex=000A0). Hence the complete ASCII command string is $019000A0(cr)

6.4.72 $aa9 Read CJ offset

Description: Return Cold Junction temperature offset of EX9019-MTCP

Syntax: $aa9(cr)

$ is a delimiter character.

aa (range 00-FF) represents the 2-character hexadecimal Modbus network address (Always 01)

9 is the command to read cold junction temperature offset.

(cr) is the terminating character, carriage return (0Dh)

Response: >(data)(cr) if the command is valid.

?aa(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

(data) a 8-character hexadecimal value representing the cold junction temperature offset.

(cr) is the terminating character, carriage return (0Dh)

Example: command: **$019(cr)**

response: **>+00005.5(cr)**

The command asks the specific module at address 01h to return the cold junction temperature offset of specified module. The response is +5.5C.

Chapter 7 MODBUS/TCP Command structure

7.1 MODBUS/TCP Command structure

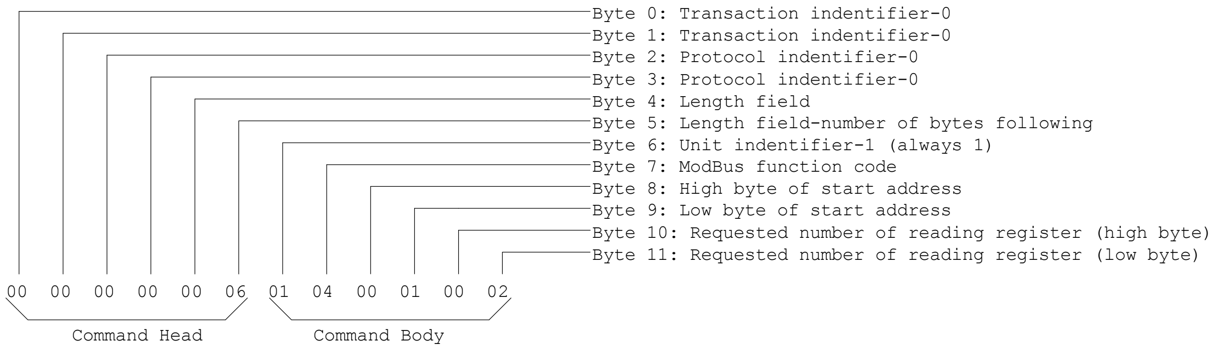
EX9000-MTCP system accepts a command/response form with the host computer. When systems are not MODBUS/TCP Command structure. EX9000-MTCP system accepts a command/response form with the host computer. When systems are not transmitting they are in listen mode. The host issues a command to a system with a specified 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.

7.2 Command Structure

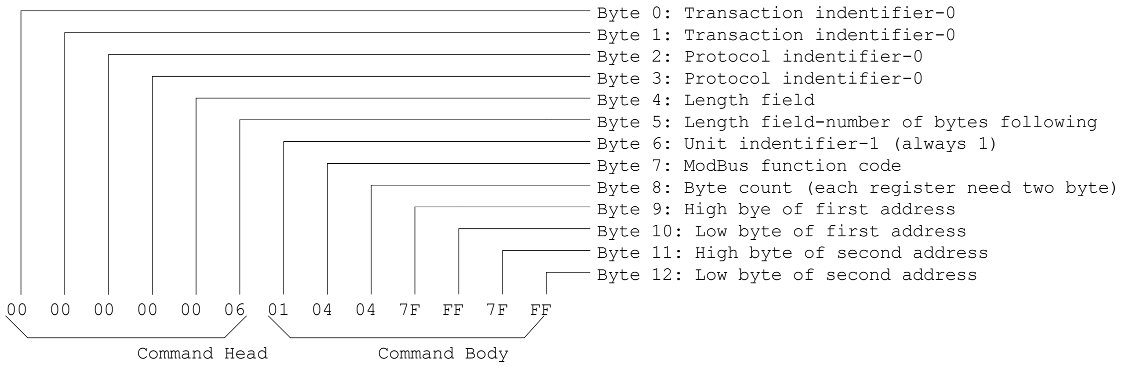
It is important to understand the encapsulation of a Modbus request or response carried on the Modbus/TCP network. A complete command is consisted of command head and command body. The command head is prefixed by six bytes and responded to pack Modbus format; the command body defines target device and requested action. Following example will help you to realize this structure quickly.

Example:

If you want to read the first two values of EX9017-MTCP (address: 40001~40002), the request command should be:


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

And the response should be:


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 Always 1

7.3 ModBus Function code introductions

Code (Hex)	**Name**	**Usage**
01	Read Coil Status	Read Discrete Output Bit
02	Read Input Status	Read Discrete Input Bit
03	Read Holding Registers	Read 16-bit register. Used to read integer or floating point process data.
04	Read Input Registers	
05	Force Single Coil	Write data to force coil ON/OFF
06	Preset Single Register	Write data in 16-bit integer format
0F	Force Multiple Coils	Write multiple data to force coil ON/OFF
10	Preset Multiple Registers	Write multiple data in 16-bit integer format

7.4 For DIO Modules: Register Address (Unit: 16bits)

Where X = 40000 for function 03, function 06, function 16
X = 30000 for function 04

7.4.1 For EX9050-MTCP: 12 Digital Input/6 Digital Output Module

Address	Channel	Item	type
X+0001~X+0024	Read DI Counter Count value, (For DI Counter Mode).	12 Channels, **32 Bits/per channel**.	R
X+0025~X+0036	For Pulse Output L level, time Unit:0.1ms	6 Channels, 32 Bits/channel.	R/W
X+0037~X+0048	For Pulse Output H level, time Unit:0.1ms	6 Channels, 32 Bits/channel.	R/W
X+0049~X+0060	Set DO pulse count value. (Set to 0= Continue mode & 1= stop)	6 Channels, 32 Bits/channel.	R/W

7.4.2 For EX9051-MTCP: 12 Digital Input/2 Counter/2 Output Module

Address	Channel	Item	Type
X+0001~X+0028	Read DI Counter Count value, (For DI Counter Mode).	14 Channels, **32 Bits per channel**	R
X+0029~X+0032	For Pulse Output L level, time Unit:0.1ms	2 Channels, 32 Bits/channel	R/W
X+0033~X+0036	For Pulse Output H level, time Unit:0.1ms	2 Channels, 32 Bits/channel	R/W
X+0037~X+0040	Set DO pulse count value. (Set to 0= Continue mode & 1= stop)	2 Channels, 32 Bits/channel	R/W

7.4.3 For EX9055-MTCP 8 channel digital Input /digital out Module

Address	Channel	Item	Type
X+0001~X+0016	Read DI Counter Count value, (For DI Counter Mode).	8 Channels, **32 Bits per channel**	R
X+0017~X+0032	For Pulse Output L level, time Unit:0.1ms	8 Channels, 32 Bits/channel	R/W
X+0033~X+0048	For Pulse Output H level, time Unit:0.1ms	8 Channels, 32 Bits/channel	R/W
X+0049~X+0064	Set DO pulse count value. (Set to 0= Continue mode & 1= stop)	8 Channels, 32 Bits/channel	R/W

7.4.4 For ALL EX9000-MTCP(new ver.) Digital Input/Output Modules **(recommend)(Firmware Ver: 6.000 or later)**

Address	Channel	Item	type
X+1001~ X+1064	Read DI Counter Count value, (For DI Counter Mode).	32 Channels, **32Bits per channel** (x+1001) for DI ch-0 bit 0~15 (x+1002) for DI ch-0 bit 16~31	R
X+1065~X+1128	For Pulse Output L level, (max. 13170). time **Unit: 0.5ms**	32 Channels, 32 Bits/channel	R/W
X+1129~X+1192	For Pulse Output H level, (max. 13170). time **Unit: 0.5ms**	32 Channels, 32 Bits/channel	R/W
X+1193~X+1256	DO Pulse output mode. (0= continue, 1= stop, 2= start)	32 Channels, 32 Bits per channel	R/W
X+1257~X+1320	DO pulse output count value,	32 Channels, 32 Bit per channel.	R/W
X+1321~X+1322	For DI(0~31) Channels, 1 Bit/channel.	32 Channels, 32 Bits. (x+1321) for DI ch(0~15), (x+1322) for DI ch(16~31)	R
X+1323~X+1324	For DO(0~31) Channels, 1 Bit/channel.	32 Channels, 32 Bits. (x+1323) for DOI ch(0~15), (x+1324) for DO ch(16~31)	R/W
X+1325~X+1388	For Pulse Output L level, time, Unit: 0.1ms	32 Channels, 32 Bit per channel.	R/W
X+1389~X+1452	For Pulse Output H level, time, Unit: 0.1ms	32 Channels, 32 Bit per channel.	R/W
X+1453~X+1484	DO mode setting: = 0 - Direct DO output, (default) = 1 - Pluse output mode, = 2 - low to high delay, = 3 - high to low delay	**16 Bits per channel.**	R/W
X+1485~X+1526	DI mode setting: = 0 - Direct DI input, (default) = 1 - Counter Mode, = 2 - low to high latch = 3 - high to low latch = 4 - Input frequency mode (0.3 ~ 4500 Hz max)	**16 Bits per channel.**	R/W
x+5678	Informs all modules that the host is OK. (ref. ~AA**, or ~**)	(No reply to modbus response)	R

Printed Date: June 1, 2016

x+5601	host communication timeout value (unit: 0.1sec)	ref. (x+5604) Host watchdog timeout status	R/W
x+5602~x+5603	Host wdt timeout Safe DO value, DO(0~31). **Note:** After timeout the all of D/O commands are disabled.	(x5602) for DO0~DO15 & (x5603) for DO16~DO31	R/W
x+5604	Host watchdog timeout status,	0xFF00 = host wdt timeout bit is set, Write (0xFF00) to clear host watchdog timeout.	R/W
x+5605	host wdt Enable(0xFF00) and Disable(0x0000)	**ref.** Ascii command: "~AA3EVVV"	R/W
x+5606	Firmware version (ex: ver: 0x6901)		R
x+5607	read factory Module name		R
x+5608	set/read power-on delay time (0.1sec)		R/W
(x+5609)~(x+5610)	power-on DO value,	(x+5609) for DO0~DO15 & (x+5610) for DO16~DO31	R/W
x+5612	Reads the Reset Status of a module (ref. $AA5)	the Reset Status of a module, the module is been reseted (0xFF00)	R
x+5613	user defined module ID number	(ref. $AAIDFF, $AAID)	R/W
x+5614	DI active status. = 0xFF00 - Input value 1 for input active(ON) .	(ref. ~AADSMN, ~AADS)	R/W
x+5615	DO active status. = 0xFF00 - Output value 1 for output inactive(OFF)	(ref. ~AADSMN, ~AADS)	R/W

Where X = 00000 for function 01, function 05, function 15
X = 10000 for function 02

7.5.1 For ALL EX9000-MTCP Digital Input/Output Modules

Address	Channel	Item	Type
X+0001~X+0012	For Digital Input (DI)	16 Channels, 1 Bit/channel	R
X+0017~X+0032	For Digital Output (DO)	16 Channels, 1 Bit /channel	R/W
X+0033	**Ch0** (For Counter Mode)	Start(0xFF00) , Stop(0) DI Counter	R/W
X+0034	**Ch0** (For Counter Mode)	Clear DI Counter (0xFF00)	W
X+0035	**Ch0** (For Counter Mode)	Read & Clear Overflow : = 1 - overflow has occurred = 0 - no overflow has occurred)	R
X+0036	**Ch0** (For DI H/L latch Mode)	Read DI Latch(1) Status / Clear(0) Status. **Example:** # read ch0 Latch Status request: 01 02 27 33 00 01 response: 01 02 01 01 ;ch0 is latched # clear ch0 Latch Status request: 01 0F 00 23 00 01 FF FE response: 01 0F 00 23 00 01 # clear ch0 Latch Status request: 01 05 00 23 00 00 response: 01 05 00 23 00 00 # read ch0 Latch Status request: 01 02 27 33 00 01 response: 01 02 01 00 ;ch0 not latched	R/W
X+0037 ~ X+0160	**Ch1 ~ Ch31** (For DI H/L latch Mode)	Read DI Latch(1) Status / Clear(0) Status.	R/W
X+0161~x+0192 (V: 6.000 or later)	For DI(0~31) Channels, 1 Bit per channel.	32 Channels, 32 Bits, 1 Bit per channel.	R
X+0193~x+0224 (V: 6.000 or later)	For DO(0~31) Channels, 1 Bit per channel.	32 Channels, 32 Bits, 1 Bit per channel.	R/W

7.6 EX9015-MTCP 7-Channel RTD Input Module

7.6.1 Register Address (unit:16 bits)

Where X=40000 for function 03, function 06, function 16

X=30000 for function 04

Address	Channel	Item	Attribute
X+0001	0	Current value	R
X+0002	1	Current value	R
X+0003	2	Current value	R
X+0004	3	Current value	R
X+0005	4	Current value	R
X+0006	5	Current value	R
X+0007	6	Current value	R
X+0008		Reserved	R
X+0009	8	Average ch0~ch6	R
X+0010	-	Reserved	R
X+0011	0	Max value	R
X+0012	1	Max value	R
X+0013	2	Max value	R
X+0014	3	Max value	R
X+0015	4	Max value	R
X+0016	5	Max value	R
X+0017	6	Max value	R
X+0018		Reserved	
X+0019~X+0020		Reserved	
X+0021	0	Min value	R
X+0022	1	Min value	R
X+0023	2	Min value	R
X+0024	3	Min value	R
X+0025	4	Min value	R
X+0026	5	Min value	R
X+0027	6	Min value	R
X+0028~X+0030		Reserved	

7.6.2 Bit Address (unit:1 bit)

Where X=00000 for function 01, function 05

X=10000 for function 02

Address	Channel	Item	Attribute
X+0101	0	Reset Max. value	R/W
X+0102	1	Reset Max. value	R/W
X+0103	2	Reset Max. value	R/W
X+0104	3	Reset Max. value	R/W
X+0105	4	Reset Max. value	R/W
X+0106	5	Reset Max. value	R/W
X+0107	6	Reset Max. value	R/W
X+0108~X+0110		Reserved	
X+0111	0	Reset Min. value	R/W
X+0112	1	Reset Min. value	R/W
X+0113	2	Reset Min. value	R/W
X+0114	3	Reset Min. value	R/W
X+0115	4	Reset Min. value	R/W
X+0116	5	Reset Min. value	R/W
X+0117	6	Reset Min. value	R/W
X+0118~X+0120	--	Reserved	
X+0121	0	Burnout flag	R
X+0122	1	Burnout flag	R
X+0123	2	Burnout flag	R
X+0124	3	Burnout flag	R
X+0125	4	Burnout flag	R
X+0126	5	Burnout flag	R
X+0127	6	Burnout flag	R
X+0128~X+0130	--	Reserved	
X+0131	0	High alarm flag	R
X+0132	1	High alarm flag	R
X+0133	2	High alarm flag	R
X+0134	3	High alarm flag	R
X+0135	4	High alarm flag	R
X+0136	5	High alarm flag	R
X+0137	6	High alarm flag	R
X+0138~X+0140	--	Reserved	
X+0141	0	Low alarm flag	R
X+0142	1	Low alarm flag	R
X+0143	2	Low alarm flag	R
X+0144	3	Low alarm flag	R
X+0145	4	Low alarm flag	R
X+0146	5	Low alarm flag	R
X+0147	6	Low alarm flag	R

7.7 EX9017-MTCP 8-Channel Voltage/Current Input Module

7.7.1 Register Address (unit:16 bits)

Where X=40000 for function 03, function 06, function 16

X=30000 for function 04

Address	Channel	Item	Attribute
X+0001	0	Current value	R
X+0002	1	Current value	R
X+0003	2	Current value	R
X+0004	3	Current value	R
X+0005	4	Current value	R
X+0006	5	Current value	R
X+0007	6	Current value	R
X+0008	7	Current Value	R
X+0009	8	Average ch0~ch7	R
X+0010	-	Reserved	R
X+0011	0	Max value	R
X+0012	1	Max value	R
X+0013	2	Max value	R
X+0014	3	Max value	R
X+0015	4	Max value	R
X+0016	5	Max value	R
X+0017	6	Max value	R
X+0018	7	Max value	R
X+0019~X+0020		Reserved	
X+0021	0	Min value	R
X+0022	1	Min value	R
X+0023	2	Min value	R
X+0024	3	Min value	R
X+0025	4	Min value	R
X+0026	5	Min value	R
X+0027	6	Min value	R
X+0028	7	Min value	R
X+0029 ~X+0030		Reserved	

7.7.2 Bit Address (unit:1 bit)

Where X=00000 for function 01, function 05

X=10000 for function 02

Address	Channel	Item	Attribute
X+0017	0	DO value	R/W
X+0018	1	DO value	R/W
X+0101	0	Reset Max. value	R/W
X+0102	1	Reset Max. value	R/W
X+0103	2	Reset Max. value	R/W
X+0104	3	Reset Max. value	R/W
X+0105	4	Reset Max. value	R/W
X+0106	5	Reset Max. value	R/W
X+0107	6	Reset Max. value	R/W
X+0108	7	Reset Max. value	R/W
X+0109~X+0110	8	Reserved	
X+0111	0	Reset Min. value	R/W
X+0112	1	Reset Min. value	R/W
X+0113	2	Reset Min. value	R/W
X+0114	3	Reset Min. value	R/W
X+0115	4	Reset Min. value	R/W
X+0116	5	Reset Min. value	R/W
X+0117	6	Reset Min. value	R/W
X+0118	7	Reset Min. value	R/W
X+0119~X+0130	--	Reserved	
X+0131	0	High alarm flag	R
X+0132	1	High alarm flag	R
X+0133	2	High alarm flag	R
X+0134	3	High alarm flag	R
X+0135	4	High alarm flag	R
X+0136	5	High alarm flag	R
X+0137	6	High alarm flag	R
X+0138	7	High alarm flag	R
X+0139~X+0140	--	Reserved	
X+0141	0	Low alarm flag	R
X+0142	1	Low alarm flag	R
X+0143	2	Low alarm flag	R
X+0144	3	Low alarm flag	R
X+0145	4	Low alarm flag	R
X+0146	5	Low alarm flag	R
X+0147	6	Low alarm flag	R
X+0148	7	Low alarm flag	R

7.8 EX9019-MTCP 8-Channel T/C Input Module

7.8.1 Register Address (unit:16 bits)

Where X=40000 for function 03, function 06, function 16

X=30000 for function 04

Address	Channel	Item	Attribute
X+0001	0	Current value	R
X+0002	1	Current value	R
X+0003	2	Current value	R
X+0004	3	Current value	R
X+0005	4	Current value	R
X+0006	5	Current value	R
X+0007	6	Current value	R
X+0008		Current value	R
X+0009	8	Average ch0~ch7	R
X+0010	-	Reserved	R
X+0011	0	Max value	R
X+0012	1	Max value	R
X+0013	2	Max value	R
X+0014	3	Max value	R
X+0015	4	Max value	R
X+0016	5	Max value	R
X+0017	6	Max value	R
X+0018	7	Max value	
X+0019~X+0020		Reserved	
X+0021	0	Min value	R
X+0022	1	Min value	R
X+0023	2	Min value	R
X+0024	3	Min value	R
X+0025	4	Min value	R
X+0026	5	Min value	R
X+0027	6	Min value	R
X+0028~X+0030		Reserved	

7.8.2 Bit Address (unit:1 bit)

Where X=00000 for function 01, function 05
X=10000 for function 02

Address	**Channel**	**Item**	**Attribute**
X+0017	0	DO value	R/W
X+0018	1	DO value	R/W
X+0101	0	Reset Max. value	R/W
X+0102	1	Reset Max. value	R/W
X+0103	2	Reset Max. value	R/W
X+0104	3	Reset Max. value	R/W
X+0105	4	Reset Max. value	R/W
X+0106	5	Reset Max. value	R/W
X+0107	6	Reset Max. value	R/W
X+0108	7	Reset Max. value	R/W
X+0109~X+0110		Reserved	
X+0111	0	Reset Min. value	R/W
X+0112	1	Reset Min. value	R/W
X+0113	2	Reset Min. value	R/W
X+0114	3	Reset Min. value	R/W
X+0115	4	Reset Min. value	R/W
X+0116	5	Reset Min. value	R/W
X+0117	6	Reset Min. value	R/W
X+0118	7	Reset Min. value	R/W
X+0119~X+0120	--	Reserved	
X+0121	0	Burnout flag	R
X+0122	1	Burnout flag	R
X+0123	2	Burnout flag	R
X+0124	3	Burnout flag	R
X+0125	4	Burnout flag	R
X+0126	5	Burnout flag	R
X+0127	6	Burnout flag	R
X+0128	7	Burnout flag	R
X+0129~X+0130	--	Reserved	
X+0131	0	High alarm flag	R
X+0132	1	High alarm flag	R
X+0133	2	High alarm flag	R
X+0134	3	High alarm flag	R
X+0135	4	High alarm flag	R
X+0136	5	High alarm flag	R
X+0137	6	High alarm flag	R
X+0138	7	High alarm flag	R
X+0139~X+0140	--	Reserved	
X+0141	0	Low alarm flag	R
X+0142	1	Low alarm flag	R
X+0143	2	Low alarm flag	R
X+0144	3	Low alarm flag	R
X+0145	4	Low alarm flag	R
X+0146	5	Low alarm flag	R

X+0147	6	Low alarm flag	R
X+0148	7	Low alarm flag	R

Chapter 8 TCPDAQ Data Structure

8.1Typedef 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 10:12:34:567 (Year/Month/Day
  Hour:Minute:Second:mSecond)
    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;
```

8.2Typedef 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;
```

8.3Typedef 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;
```

8.4Typedef struct ModuleData

```
typedef struct ModuleData              //Used for function TCP_ReadAllDataFromModule (..)
{  u_char    Din[16];                  //Digital input data (DI#0~DI#15),avaliable for EX9050/51/55-MTCP
   u_char    Dout[16];                 //Digital output data (DO#0~DO#15),avaliable for EX9050/51/55/17/19 –MTCP
   u_char    DiLatch[16];              //Digital input latch status (DI#0~DI#15),avaliable for EX9050/51/55-MTCP
   long      DiCounter[16];            //Digital input counter value (DI#0~DI#15),avaliable for EX9050/51/55-MTCP
   double    AiNormalValue[16];        //Analog Input value(AI#0~AI#15),avaliable for EX9015/17/19-MTCP
   double    AiMaxValue[16];           //Analog maximum value(AI#0~AI#15),avaliable forEX9015/17/19-MTCP
   double    AiMinValue[16];           //Analog minimum value(AI#0~AI#15),avaliable for EX9015/17/19-MTCP
   u_char    AiHighAlarm[16];          //Analog high alarm status(AI#0~AI#15),avaliable for EX9015/17/19-MTCP
   u_char    AiLowAlarm[16];           //Analog low alarm status(AI#0~AI#15),avaliable for EX9015/17/19-MTCP
   u_char    AiChannelType[16];        //Analog channel Type, avaliable for EX9015/17/19-MTCP
   u_char    AiBurnOut[16] ;           //Analog channel burn out status,avaliable for EX9019/15-MTCP only
   double    CJCTemperature ;          //Cold junction temperature,avaliable for EX9019-MTCP only
} ModuleData;
```

Chapter 9 EX9000-MTCP Web Server

9.1 What is TCPDAQ Web Server?

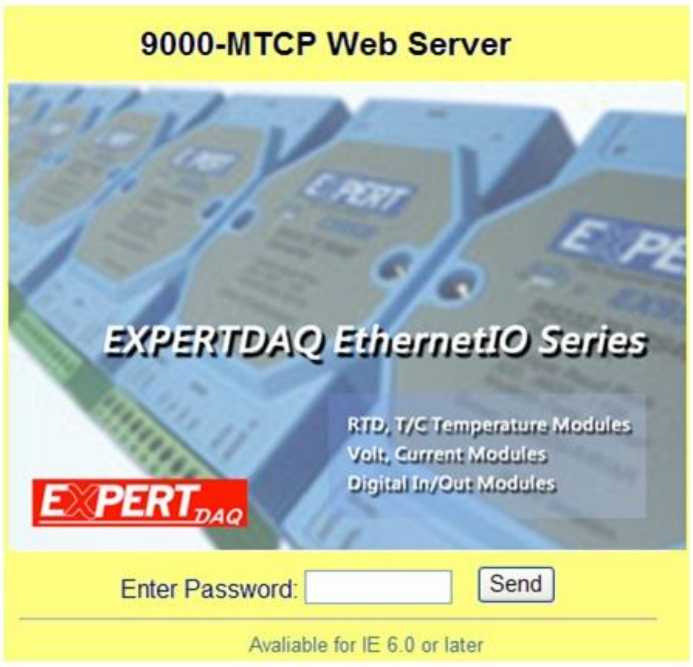
EX9000-MTCP I/O modules all features built-in web server. Remote computer or devices can monitor and control I/O status on EX9000-MTCP modules remotely through web browser. There is default built-in web page on EX9000-MTCP modules.

To use your computer to browse the web page on EX9000-MTCP module, you can simply type the IP address to connect to your EX9000-MTCP 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 EX9000-MTCP modules.

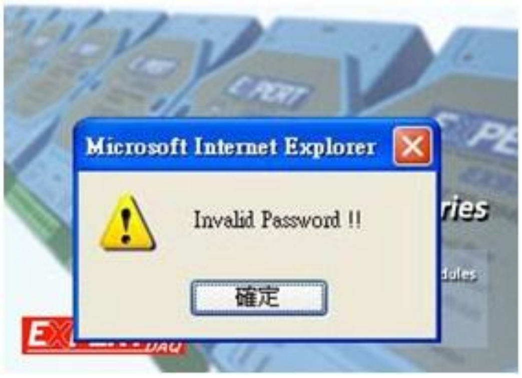
Notice: Please use Windows Internet Explorer 5.5 (IE 5.5 or later version)

9.2 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
-



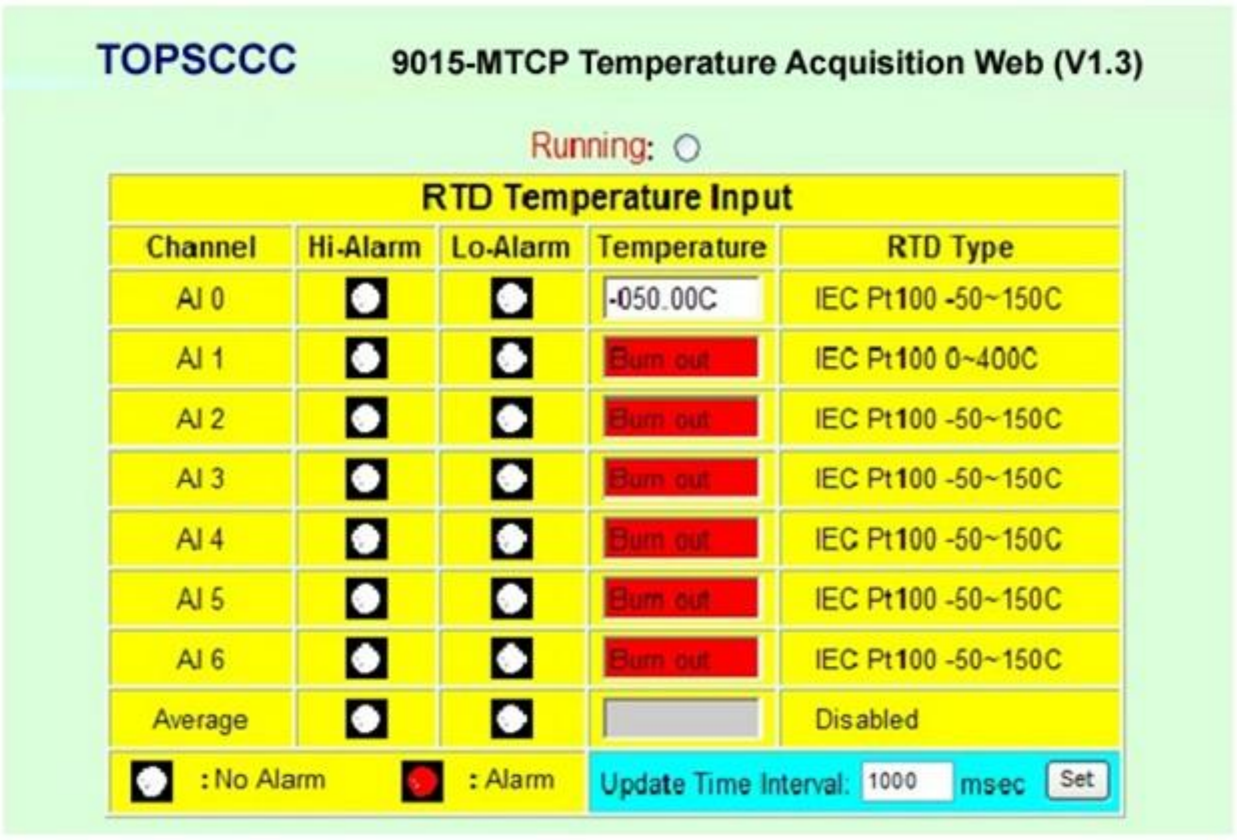
- Enter the correct password and click send button to verify the password. If the password is not correct, a warming message box will show up to remain you to reenter the password



- If the password is correct, the module monitoring page will pop up in the web browser.

9.3 Remote IO Module monitoring page

9.3.1 EX9015-MTCP monitoring page



TOPSCCC 9015-MTCP Temperature Acquisition Web (V1.3)

Running: ○

RTD Temperature Input				
Channel	Hi-Alarm	Lo-Alarm	Temperature	RTD Type
AI 0	○	○	-050.00C	IEC Pt100 -50~150C
AI 1	○	○	Burn out	IEC Pt100 0~400C
AI 2	○	○	Burn out	IEC Pt100 -50~150C
AI 3	○	○	Burn out	IEC Pt100 -50~150C
AI 4	○	○	Burn out	IEC Pt100 -50~150C
AI 5	○	○	Burn out	IEC Pt100 -50~150C
AI 6	○	○	Burn out	IEC Pt100 -50~150C
Average	○	○		Disabled

○ : No Alarm ● : Alarm Update Time Interval: 1000 msec Set

Channel : Channel number of RTD input

Hi-Alarm : Analog channel High alarm status

Lo-Alarm: Analog channel low alarm status

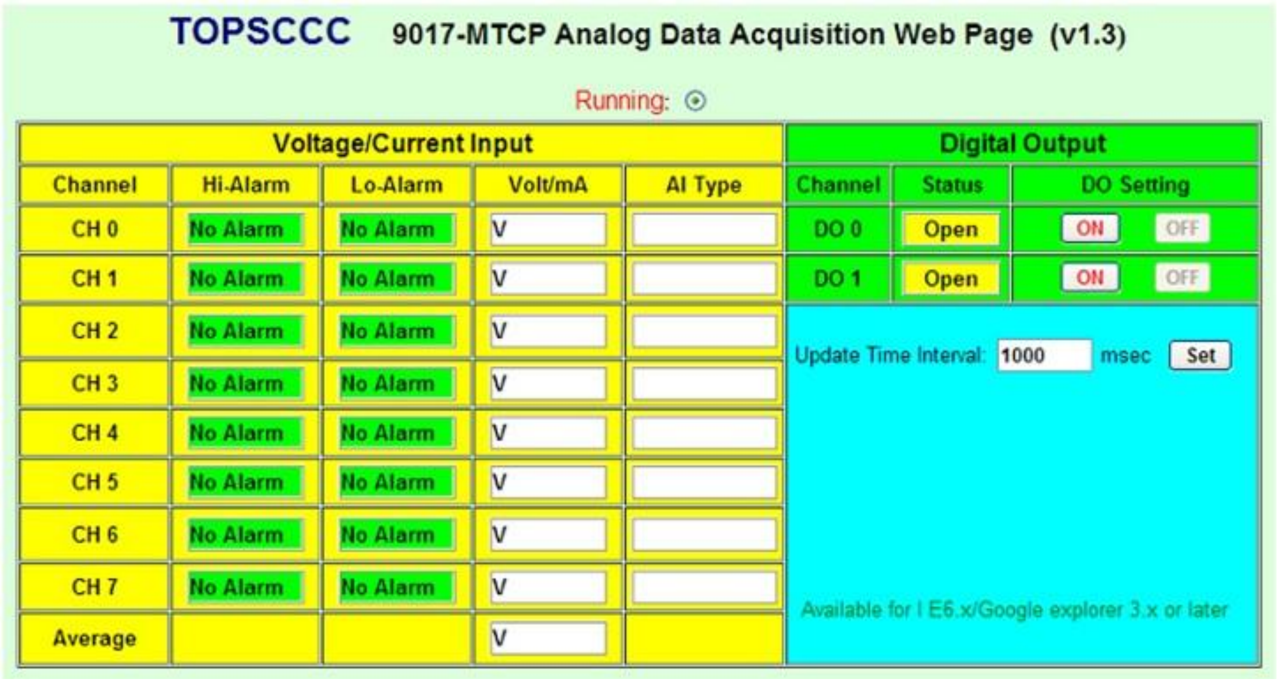
Temperature: Temperature value of RTD input channel

RTD type : RTD type of input channel

Average : Average value of channels which functions in average

Time interval: I/O status update time interval

9.3.2 EX9017-MTCP monitoring page



TOPSCCC 9017-MTCP Analog Data Acquisition Web Page (v1.3)

Running: ⊙

Voltage/Current Input				
Channel	Hi-Alarm	Lo-Alarm	Volt/mA	AI Type
CH 0	No Alarm	No Alarm	V	
CH 1	No Alarm	No Alarm	V	
CH 2	No Alarm	No Alarm	V	
CH 3	No Alarm	No Alarm	V	
CH 4	No Alarm	No Alarm	V	
CH 5	No Alarm	No Alarm	V	
CH 6	No Alarm	No Alarm	V	
CH 7	No Alarm	No Alarm	V	
Average			V	

Digital Output		
Channel	Status	DO Setting
DO 0	Open	ON OFF
DO 1	Open	ON OFF

Update Time Interval: 1000 msec Set

Available for I E6.x/Google explorer 3.x or later

Channel : Channel number of analog input or digital output

Hi-Alarm : Analog channel High alarm status

Lo-Alarm: Analog channel low alarm status

Voltage : Voltage value of analog input channel

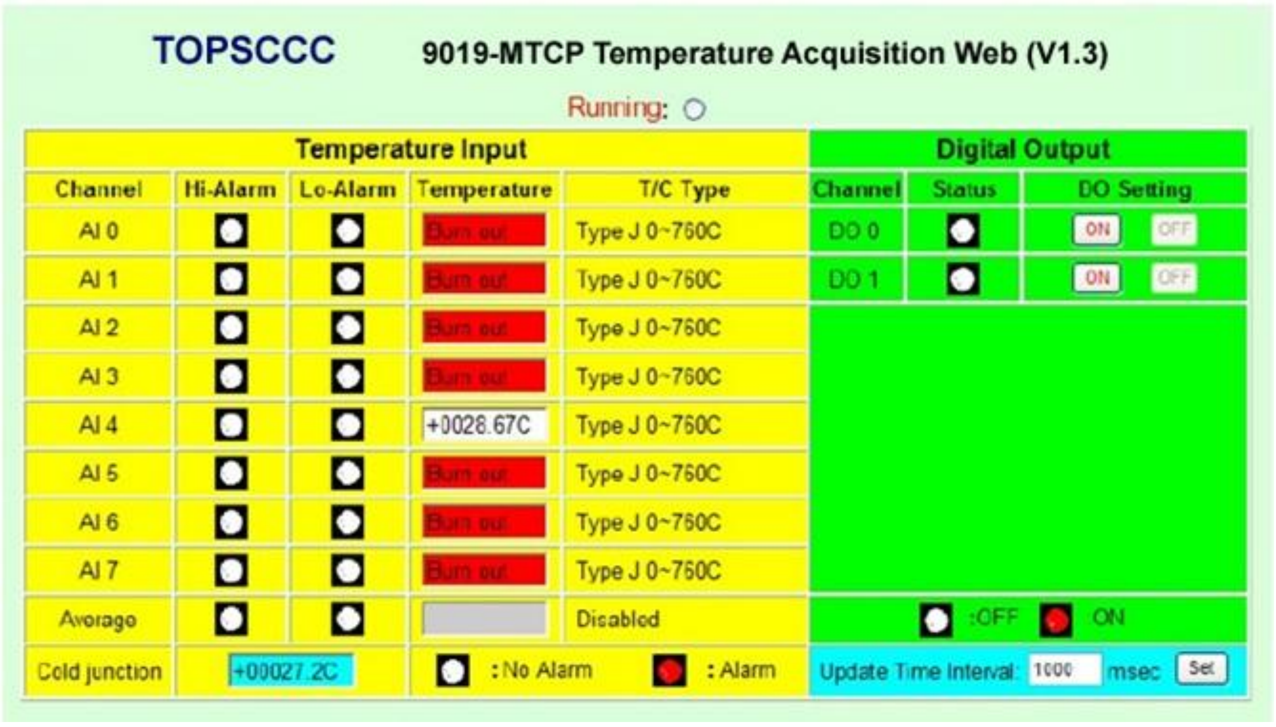
Input Range: Range of analog input channel

Status : Digital output status

DO Setting: Set digital output on or off

Time interval: I/O status update time interval

9.3.3 EX9019-MTCP monitoring page



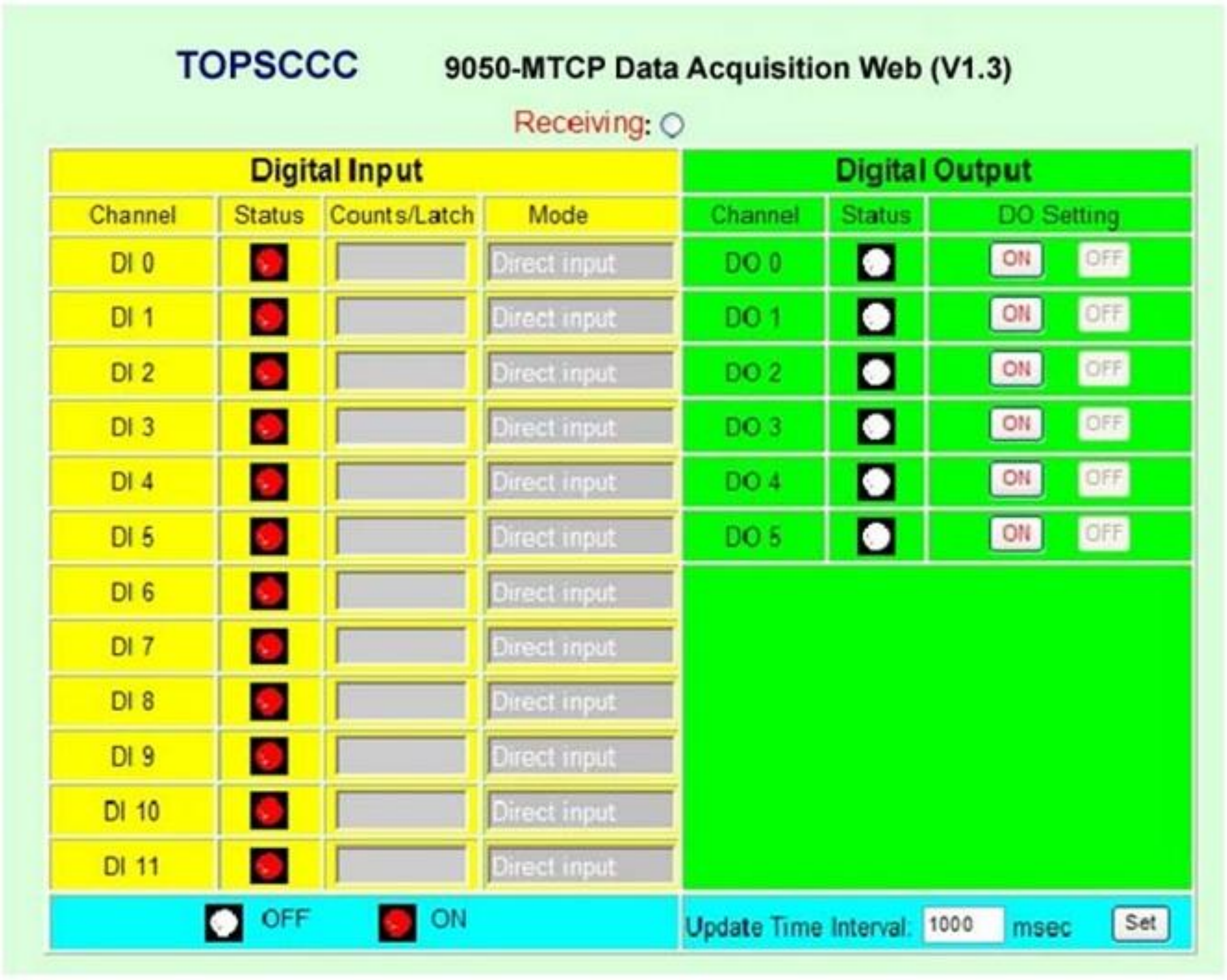
TOPSCCC 9019-MTCP Temperature Acquisition Web (V1.3)

Running: ○

Temperature Input					Digital Output		
Channel	Hi-Alarm	Lo-Alarm	Temperature	T/C Type	Channel	Status	DO Setting
AI 0	○	○	Burn out	Type J 0~760C	DO 0	○	ON OFF
AI 1	○	○	Burn out	Type J 0~760C	DO 1	○	ON OFF
AI 2	○	○	Burn out	Type J 0~760C			
AI 3	○	○	Burn out	Type J 0~760C			
AI 4	○	○	+0028.67C	Type J 0~760C			
AI 5	○	○	Burn out	Type J 0~760C			
AI 6	○	○	Burn out	Type J 0~760C			
AI 7	○	○	Burn out	Type J 0~760C			
Average	○	○		Disabled	○ :OFF ● ON		
Cold junction	+00027.2C	○ : No Alarm ● : Alarm			Update Time Interval: 1000 msec Set		

Channel	: Channel number of analog input or digital output
Hi-Alarm	: Analog channel High alarm status
Lo-Alarm	: Analog channel low alarm status
Temperature	: Temperature value of T/C input channel
T/C type	: Thermal Couple type of input channel
Cold junction	: Temperature of T/C cold junction
Average	: Average value of channels which functions in average
Status	: Digital output status
DO Setting	: Set digital output on or off
Time interval	: I/O status update time interval

9.3.4 EX9050-MTCP monitoring page


TOPSCCC 9050-MTCP Data Acquisition Web (V1.3)

Receiving: ○

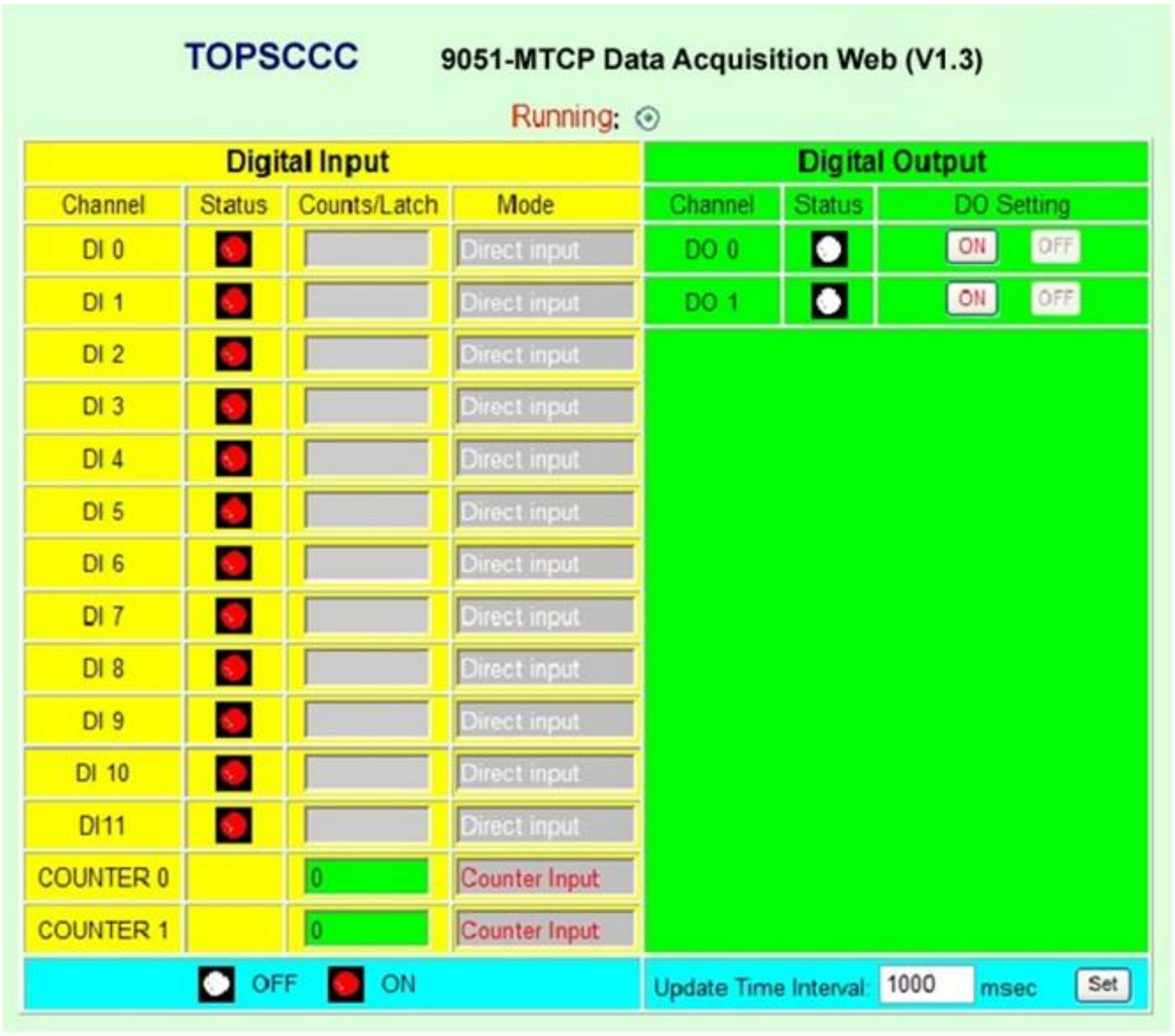
Digital Input				Digital Output		
Channel	Status	Counts/Latch	Mode	Channel	Status	DO Setting
DI 0	🔴		Direct input	DO 0	⚪	ON OFF
DI 1	🔴		Direct input	DO 1	⚪	ON OFF
DI 2	🔴		Direct input	DO 2	⚪	ON OFF
DI 3	🔴		Direct input	DO 3	⚪	ON OFF
DI 4	🔴		Direct input	DO 4	⚪	ON OFF
DI 5	🔴		Direct input	DO 5	⚪	ON OFF
DI 6	🔴		Direct input			
DI 7	🔴		Direct input			
DI 8	🔴		Direct input			
DI 9	🔴		Direct input			
DI 10	🔴		Direct input			
DI 11	🔴		Direct input			

⚪ OFF 🔴 ON

Update Time Interval: 1000 msec Set

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

9.3.5 EX9051-MTCP monitoring page



TOPSCCC 9051-MTCP Data Acquisition Web (V1.3)

Running: ⊙

Digital Input				Digital Output		
Channel	Status	Counts/Latch	Mode	Channel	Status	DO Setting
DI 0	🔴		Direct input	DO 0	⚪	ON OFF
DI 1	🔴		Direct input	DO 1	⚪	ON OFF
DI 2	🔴		Direct input			
DI 3	🔴		Direct input			
DI 4	🔴		Direct input			
DI 5	🔴		Direct input			
DI 6	🔴		Direct input			
DI 7	🔴		Direct input			
DI 8	🔴		Direct input			
DI 9	🔴		Direct input			
DI 10	🔴		Direct input			
DI11	🔴		Direct input			
COUNTER 0		0	Counter Input			
COUNTER 1		0	Counter Input			

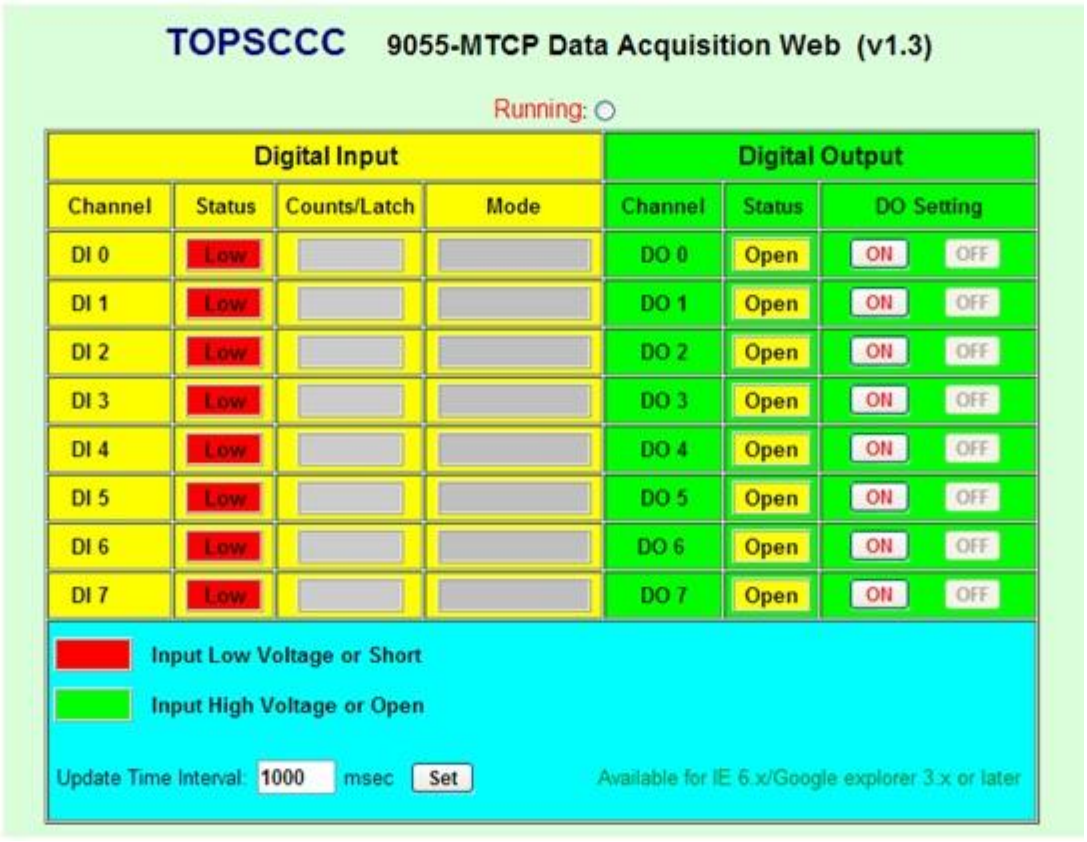
⚪ OFF 🔴 ON

Update Time Interval: 1000 msec Set

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

Printed Date: June 1, 2016

9.3.6 EX9055 monitoring page



TOPSCCC 9055-MTCP Data Acquisition Web (v1.3)

Running: ○

Digital Input				Digital Output		
Channel	Status	Counts/Latch	Mode	Channel	Status	DO Setting
DI 0	Low			DO 0	Open	ON OFF
DI 1	Low			DO 1	Open	ON OFF
DI 2	Low			DO 2	Open	ON OFF
DI 3	Low			DO 3	Open	ON OFF
DI 4	Low			DO 4	Open	ON OFF
DI 5	Low			DO 5	Open	ON OFF
DI 6	Low			DO 6	Open	ON OFF
DI 7	Low			DO 7	Open	ON OFF

🟥 Input Low Voltage or Short

🟩 Input High Voltage or Open

Update Time Interval: 1000 msec Set

Available for IE 6.x/Google explorer 3.x or later

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

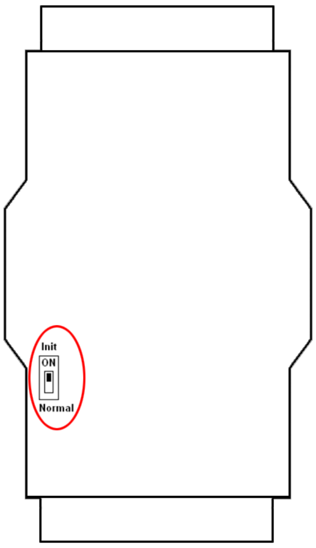
Appendix A INIT* switch operation (Only for EX9000MTCP)

The EX9000-MTCP "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 EXPERTDAQ 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, 255.0.0.0 and 10.0.0.1)

■ Originally, the INIT mode is accessed by connecting the INIT* terminal to the GND terminal. New EX9000-MTCP 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 EX9000-MTCP 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:**

➢ IP Address : 10.0.0.1
➢ Subnet Mask:
255.0.0.0 ➢ Gateway:
10.0.0.1 ➢ DHCP: Disabled
➢ Web Server:
Disabled ➢ Module ID:
00
➢ Password: 00000000

Appendix B Module Status (for version 6.000 or later)

Power-On Reset or Module Watchdog Reset will let all output goto Power-On Value. And the module may accept the host's command to change the output value. Host Watchdog Timeout will let all digital output goto Safe Value.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.

Appendix C Dual Watchdog Operation (for version 6.000 or later)

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 EX9000 series module with Dual Watchdog may let the control system more reliable and stable.

Appendix D Reset Status (for version 6.000 or later)

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.

Appendix E 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. 500Hz / 1KHz). 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.

Appendix F Power-on & Safe value (for version 6.000 or later)

Power-on value:

Power-on value are 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 specified 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 specified 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 safetystate.