Table Of Contents

9.0 EX9000-MTCP Web Server

Table Of Contents

1.0 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 V DC

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: - 10 to 70º C (14 to 158º F)

Storage Temperature: - 25 to 85º C (-13 to 185º F)

Humidity: 5 to 95%, non-condensing

Atmosphere: No corrosive gases

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

1.7 Dimensions

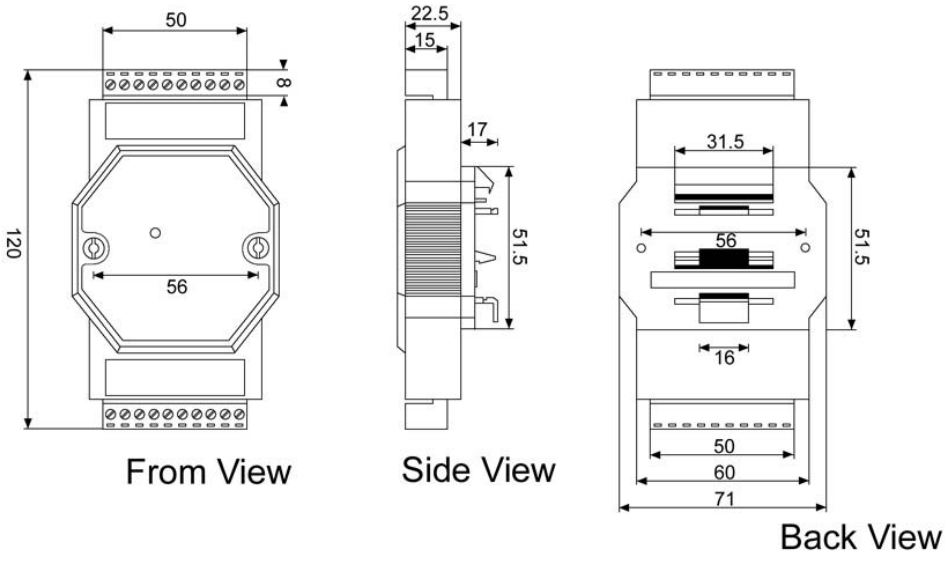
The following diagrams show the dimensions of the EX9000-MTCP I/O module in millimeters.

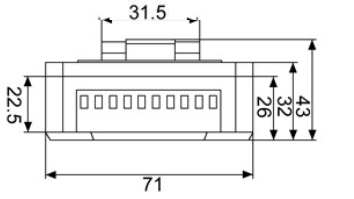
LED Status: Red indicator. This LED is normal on whenever EX9000-MTCP module is running.

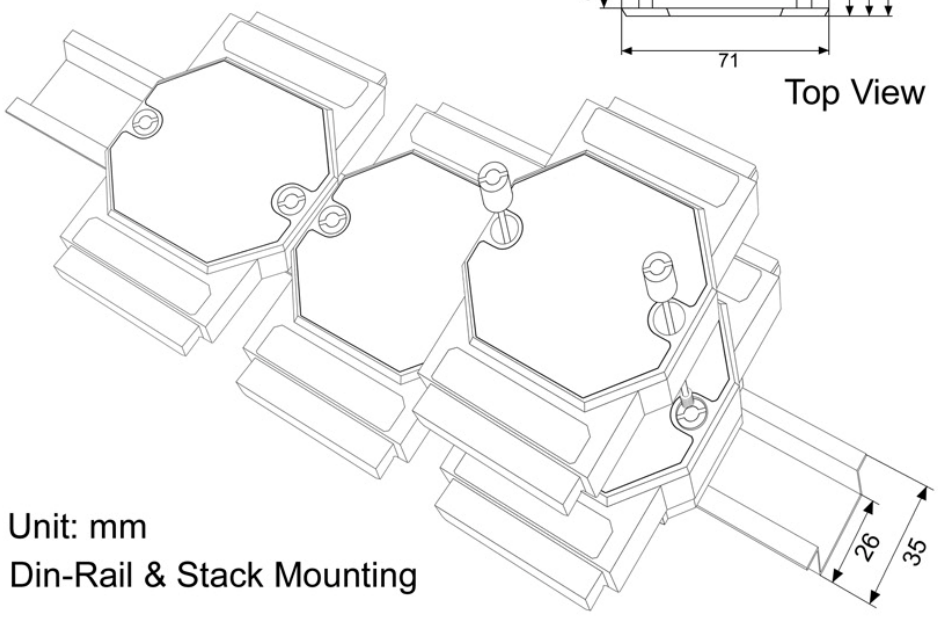
EX9000-MTCP I/O Modules support Din-Rail & Wall Mount.

EX9000-MTCP IO Modules support stack Mounting also.

9000 Dimension







Unit: mm

Din-Rail & Stack Mounting

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

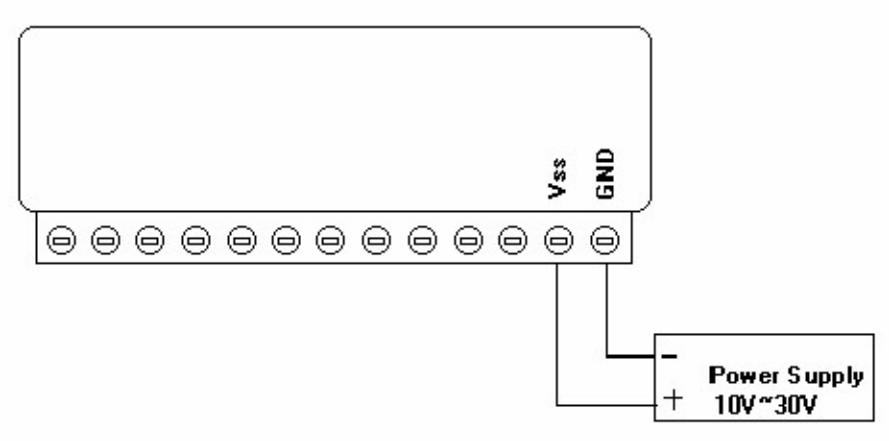
Power supply for EX9000-MTCP (+10 to +30 V unregulated)

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 V DC power supply, they accept any power unit that supplies within the range of +10 to +30 V_{DC}. 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 V_{DC}. 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.

Table Of Contents

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 and trans-former based isolation up to 2,000/2,500/3,000 V_{DC}.

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:

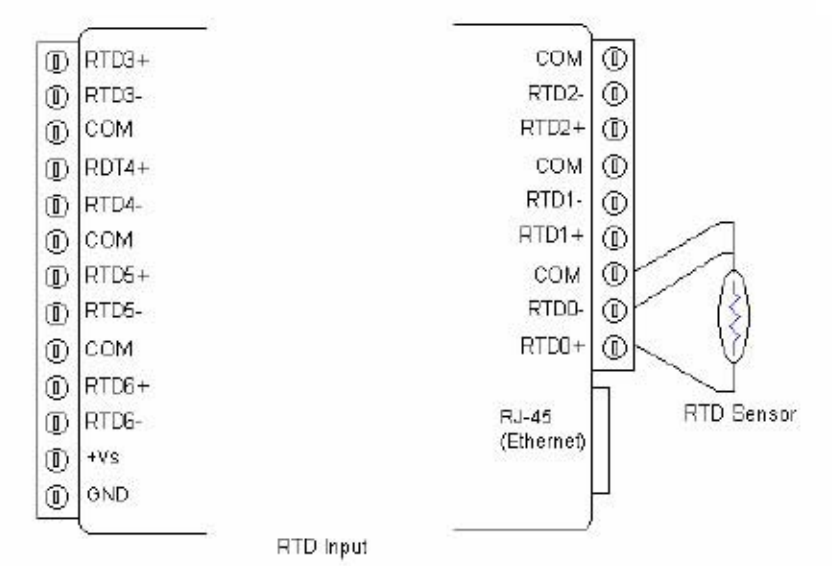
- Effective resolution: 16-bit
- Channels: 7
- Input type: PT100, PT1000, Balco 500 & Ni
- Input range:
  - PT100 -50°C ~ 150°C ,0°C ~ 100°C ,0°C ~ 200°C ,0°C ~ 400°C ,-200°C ~ 200°C ,
  - PT1000 -40°C ~ 160°C
  - Balco 500 -30°C ~ 120°C
  - Ni 604 -80°C ~ 100°C or 0°C ~ 100°C
  - Ni 1000 -60°C ~ 160°C
- Isolation voltage: 2000Vrms
- 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

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: 2500 Vrms
- Fault and overvoltage protection: Withstands overvoltage 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 to 50 V, 500 mA max. load
- Optical Isolation: 2500Vrms

Built-in Watchdog Timer

Power requirements: Unregulated +10 ~ +30 VDC

Power consumption: 2.2 W

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 to Fig 2-1, each analog input channel has built-in a jumper on the PCB for users to set as a voltage mode or current mode.

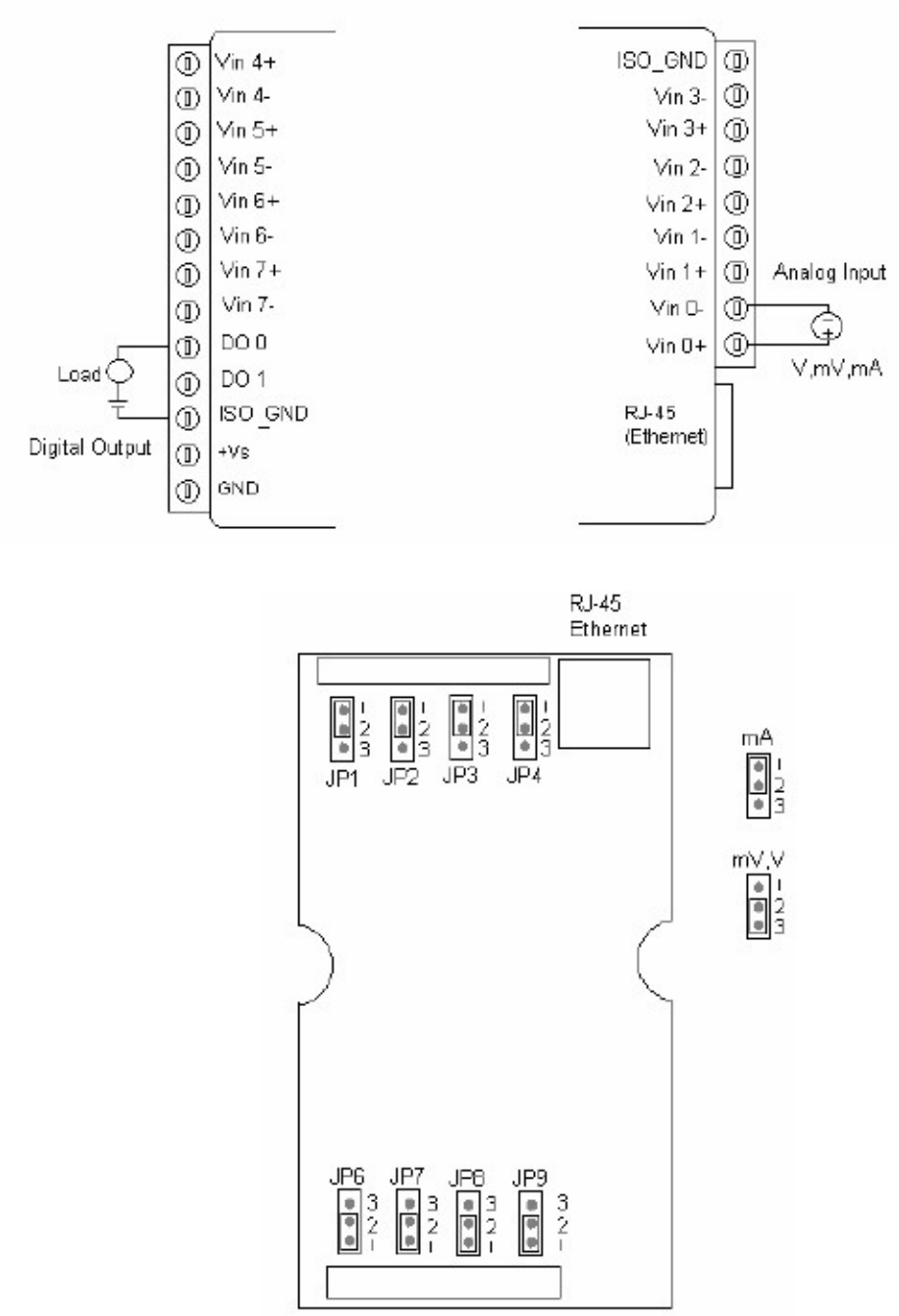


Fig 2-1

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 EX9019-MTCP 8-channel T/C Input with 2/DO Module

The EX9019-MTCP is a 16-bit, 8-channel Thermocouple input module that provides programmable input ranges on all channels. It accepts Various Thermocouple inputs (Type J, K, T, E, R, S, B) 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.

9019MTCP Specification

Analog Input:

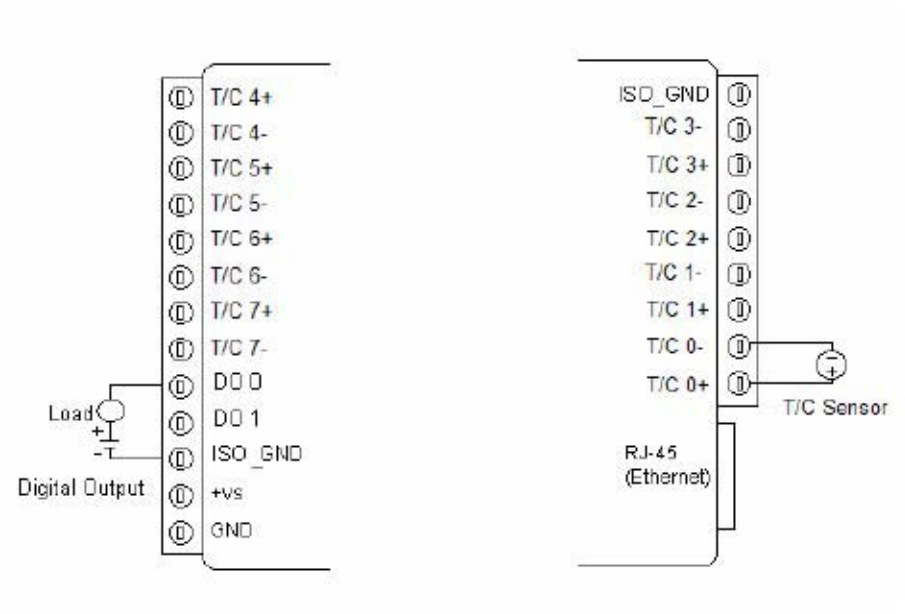
- Effective resolution: 16-bit
- Channels: 8
- Input type: J, K, T, E, R, S, B
- Input range:
  - J type: 0 ~ 760 °C
  - K type: 0 ~ 1370 °C
  - T type: -100 ~ 400 °C
  - E type: 0 ~ 1000 °C
  - R type: 500 ~ 1750 °C
  - S type: 500 ~ 1750 °C
  - B type: 500 ~ 1800 °C
- Output Type: 2 channels, Open Collect to 30Vdc/100mA(max), 400mA(max) for all DO
- Isolation voltage: 2000 Vrms
- Sampling rate: 10 samples / sec.
- Input impedance: 20 MΩ
- Accuracy: ±0.15% or better
- Zero drift: ±6 μV/°C
- Span drift: ±25 ppm/°C
- CMR @ 50/60 Hz: 92 dB

Built-in Watchdog Timer

Power requirements: Unregulated +10 ~ +30 VDC

Power consumption: 2 W/Typical, 3W/max

Application Wiring



Assigning ModBus addresses

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.

2.4 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 2500Vrms 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

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

Dry Contact: Logic level 0: Close to GND or Logic level 1: Open

Wet Contact: Logic level 0: 0V~+2V or Logic level 1: +5V~+30V

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

Digital Output:

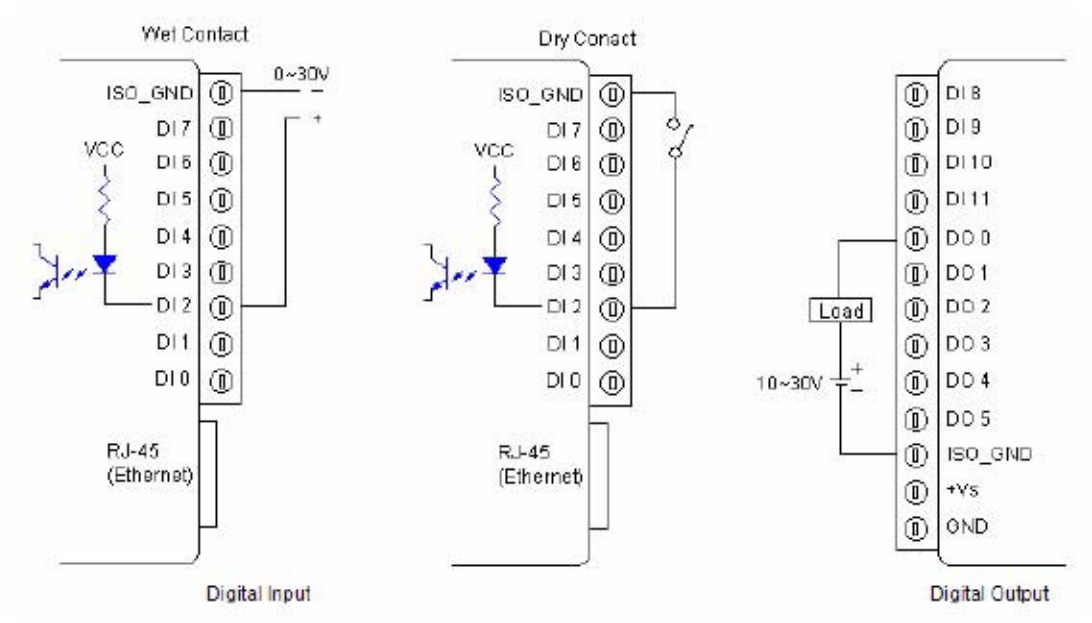
Channel: 6

Open Collector to 50V/500 mA max. load

Optical Isolation: 2500Vrms

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).

2.5 EX9051-MTCP 18-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 (4.5 KHz) channels with 2500Vrms 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

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

Dry Contact: Logic level 0: Close to GND

Logic level 1: Open

Wet Contact: Logic level 0: 0V~+2Vac

Logic level 1: +5Vac~+30Vac

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

Digital Output:

Channel: 2

Open Collector to 50V/ 500 mA max. load

Optical Isolation: 2500Vrms

Counter:

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: 2500Vrms

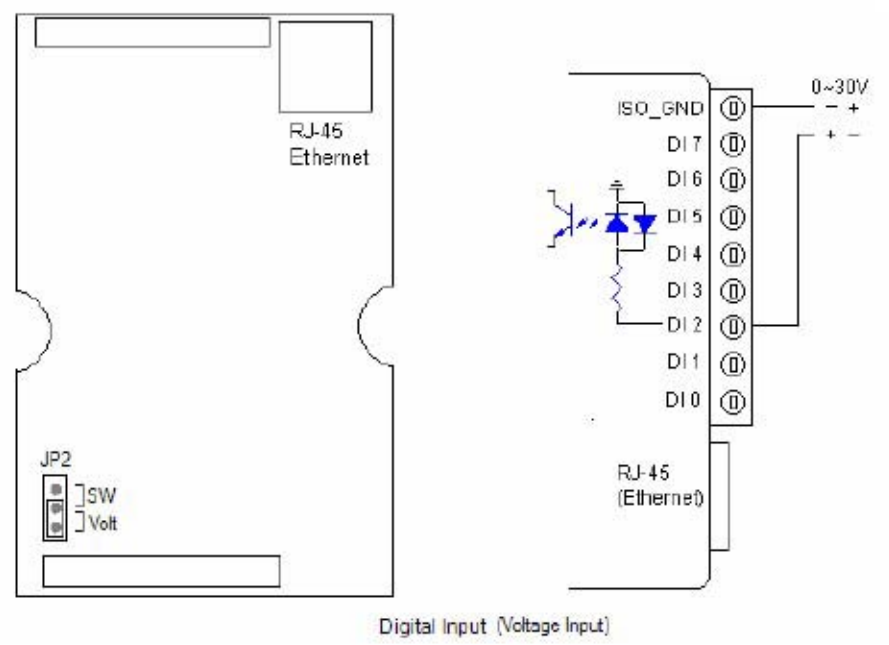
Mode: Counter, Frequency

Power Consumption: 2 W (Typical)

Application Wiring

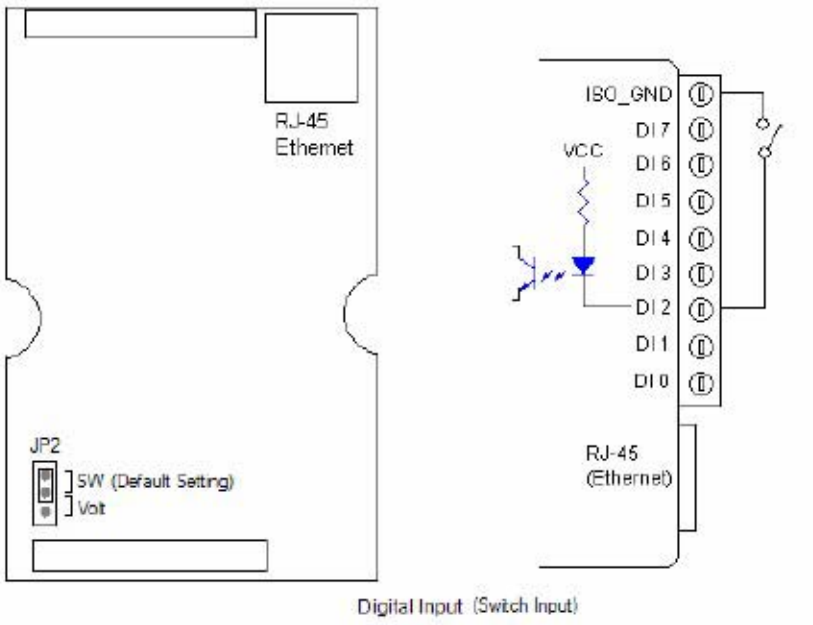
User need to adjust the jumper (JP2) setting to choose the input type as shown bwlows:

Voltage Input(Wet Contact) type:



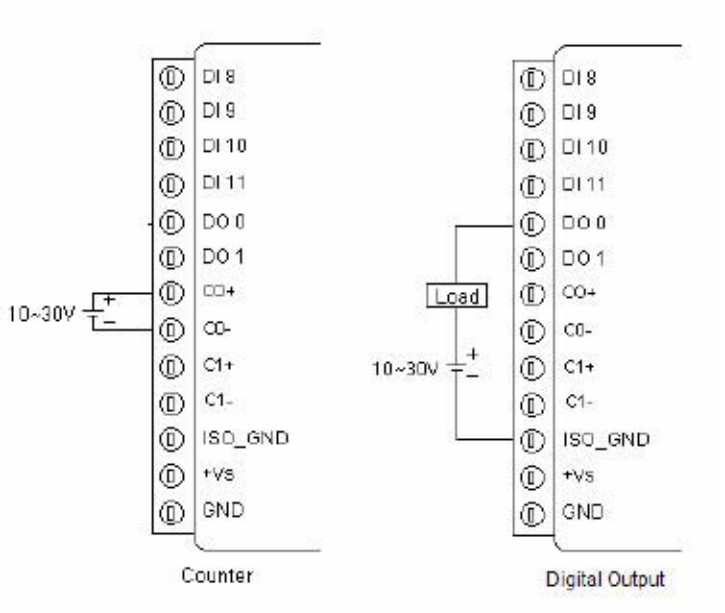
Digital Input (Voltage Input)

Switch Input(Dry Contact) type:



Digital Input (Switch Input)

Digital Counter and Output type:



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. The I/O address will be mapped as Figures

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:

Dry Contact :

Logic level 1 : Close to GND

Logic level 0 : Open

Wet Contact :

Logic level 1 : +2 Vac max

Logic level 0 : +5 to 30 Vac max

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

Digital Output:

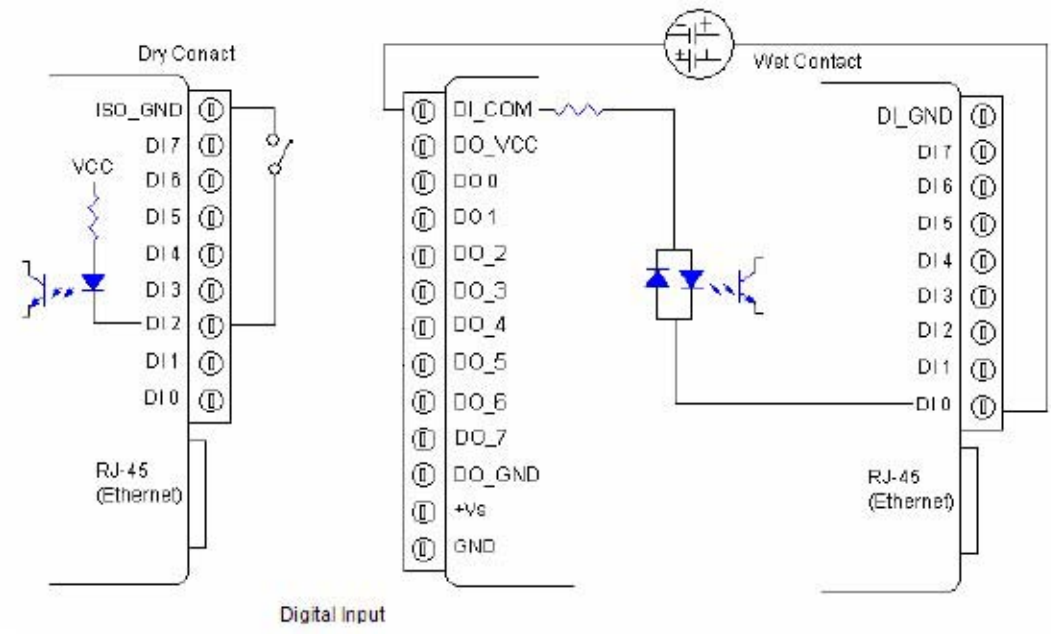
Source Type : 30Vdc, 370mA per channel

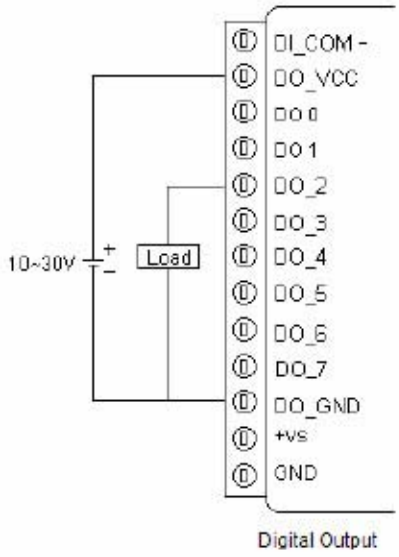
Optical Isolation: 2500 Vrms

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)

Table Of Contents

3.0 EX9000-MTCP Utility Guide

In order to properly configure 9000MTCP 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) 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)

Make sure to prepare all of the items above, then connect the power and network wiring as Figure 3-1 Power wiring.

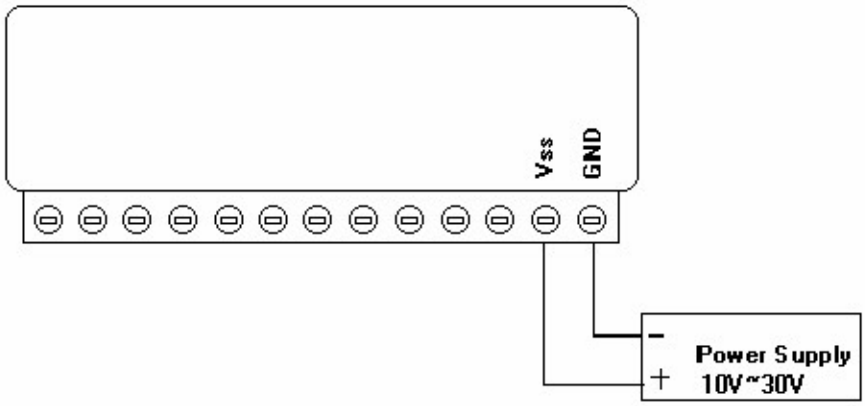


Figure 3-1 Power wiring

3.2 Install Utility Software on Host PC

EXPERTDAQ provide free download Manual and Utility software for EX9000-MTCP modules' operation and configuration. Link to the web site: www.topsccc.com 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 EX9000-MTCP 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 of Event/ Stream

3.3 Main Menu

Double Click the icon of 9050MTCP Ethernet I/O Utility shortcut, the Operation screen will pop up as Figure3-2.

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.

Manual on CD: Contents of CD as user's operation guide, 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

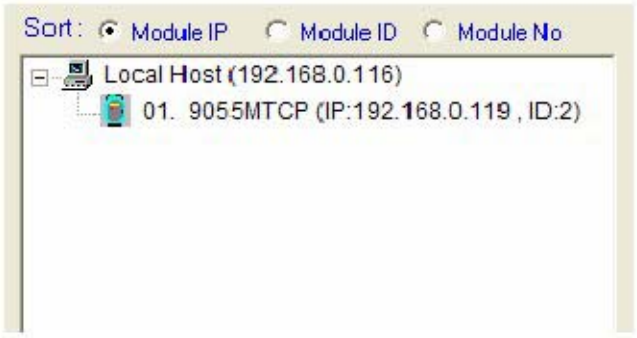
Search: Search 9050MTCP module

Add: Add remote 9050MTCP I/O module

Monitor: Monitor the Stream/Event Data

3.7 List Sort

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



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

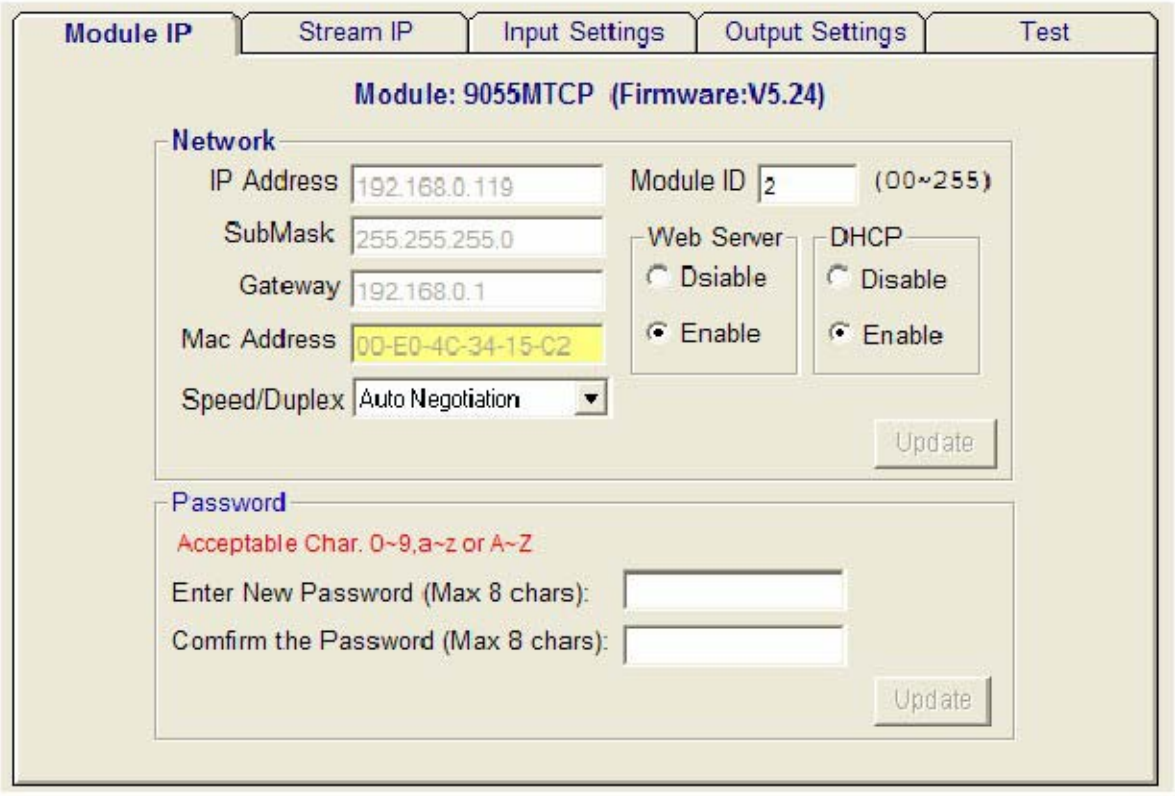


Figure 3-3

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, 255.255.255.0 and 0.0.0.0)

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 9050MTCP 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 is default disabled

Web Server: (default Enabled)

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:

9050MTCP 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

Avtive Stream time inetval: (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. settings 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 9050MTCP Ethernet I/O unit into the tree-structure display area.

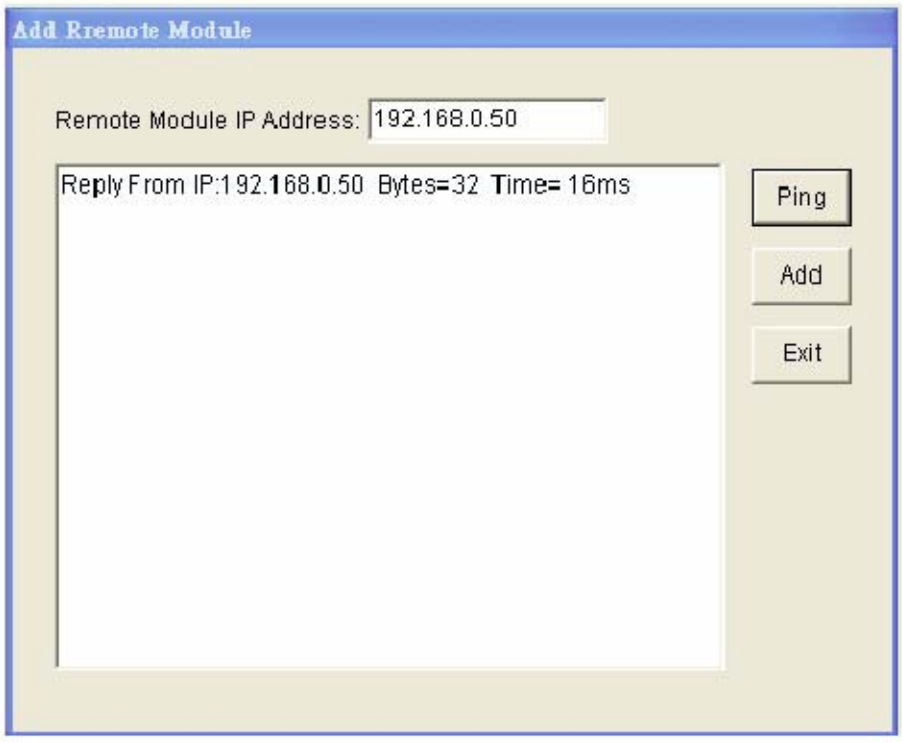


Figure3-4 Add remote module

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 topsccc'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.

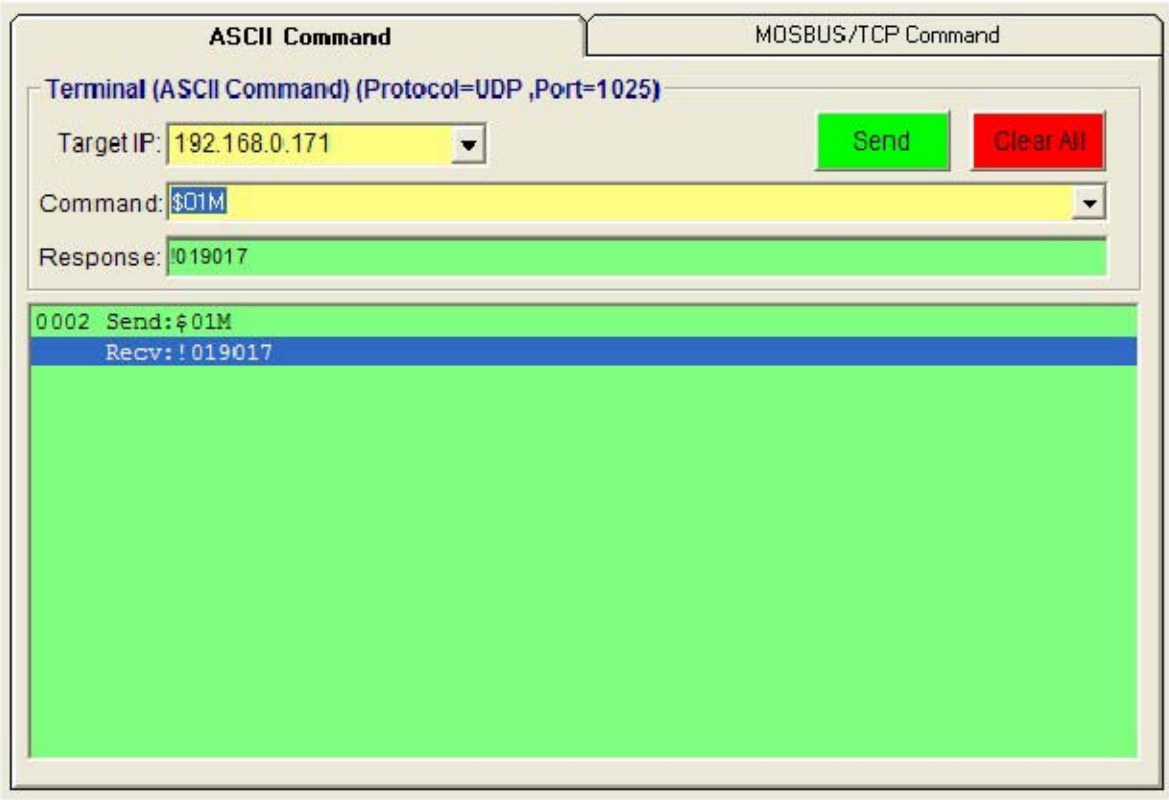


Figure 3-5 ASCII Command Terminal

ModBus Hexadecimal mode: shown as Figure 3-6.

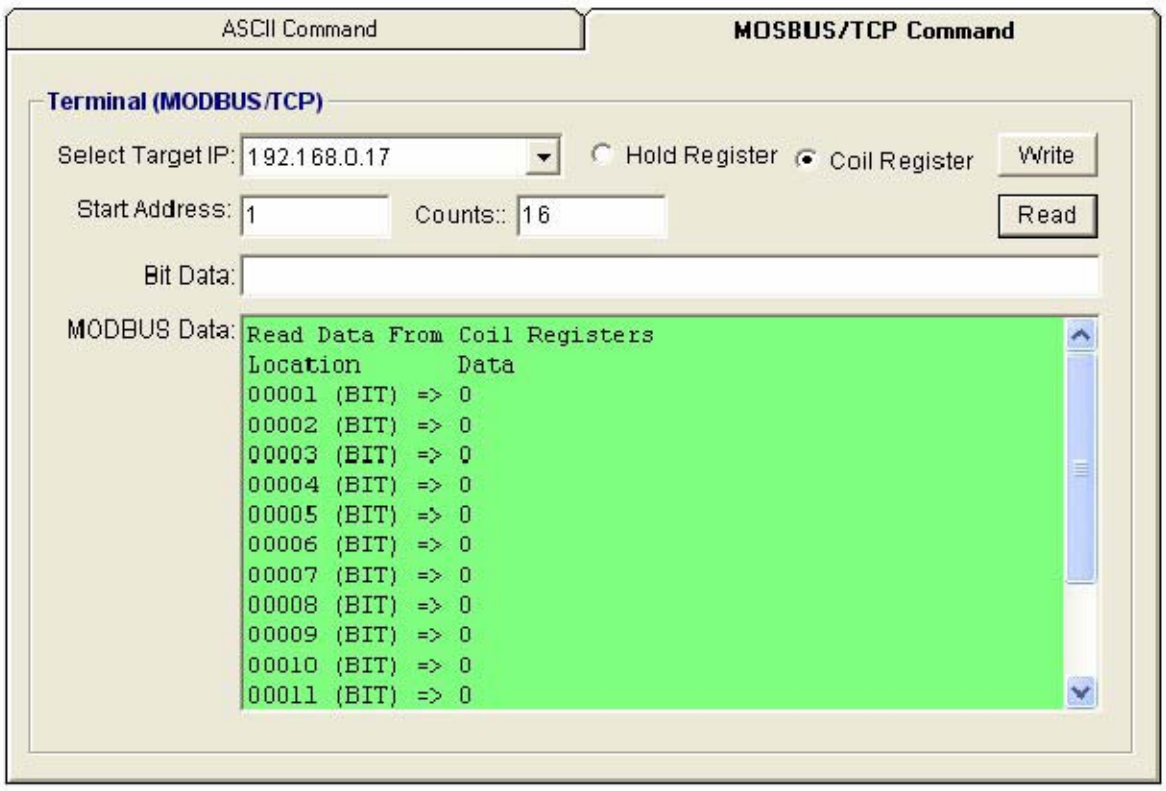


Figure 3-6 ModBus Terminal

3.12 Data of Stream/Even

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 Stream 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 Event Configuration

In addition to TCP/IP communication protocol, EX9000-MTCP supports UDP communication protocol to regularly broadcast even to specific host PCs. Click the tab of Data Event, then configure the broadcasting interval and the specific IP addresses which need to receive Data Event from the specific EX9000-MTCP I/O module. This UDP Data Even 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.

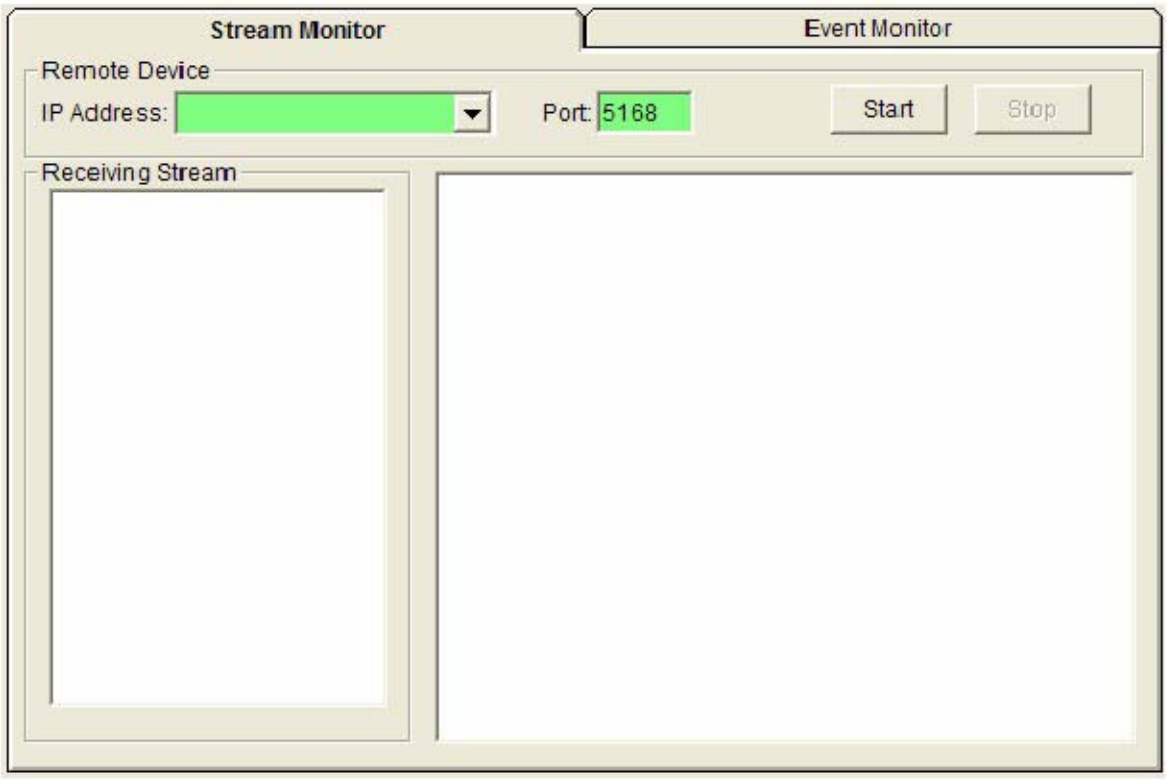


Figure 3-7 Stream display

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 event data on this operation display.

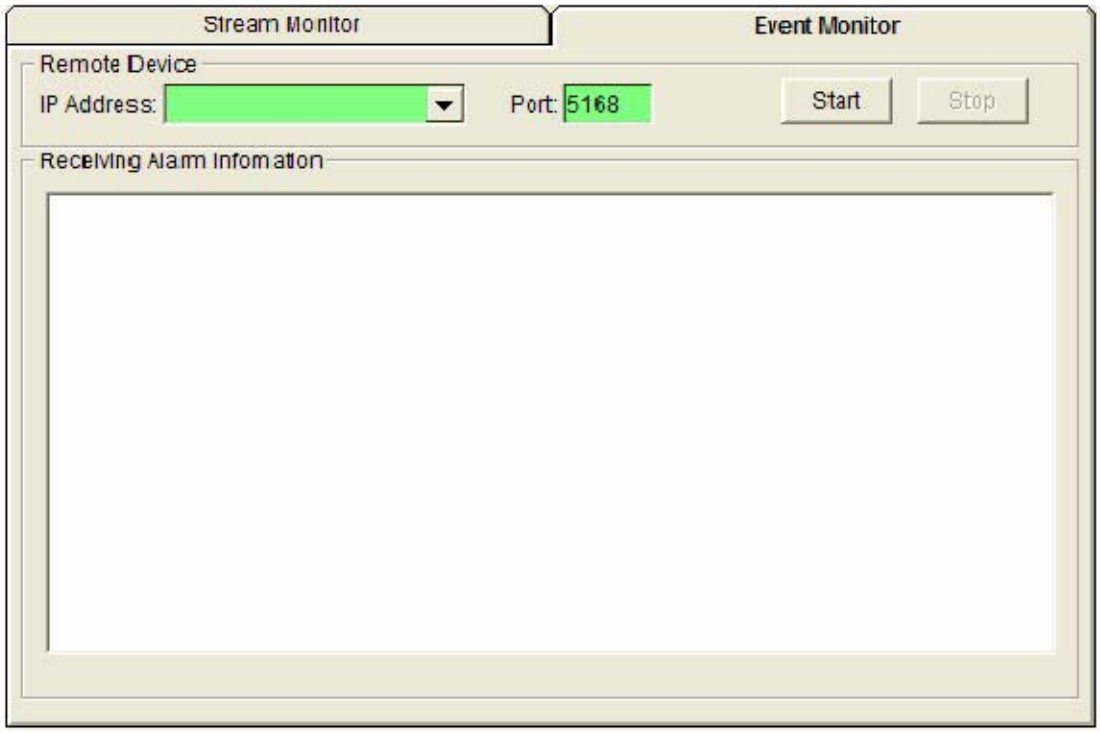
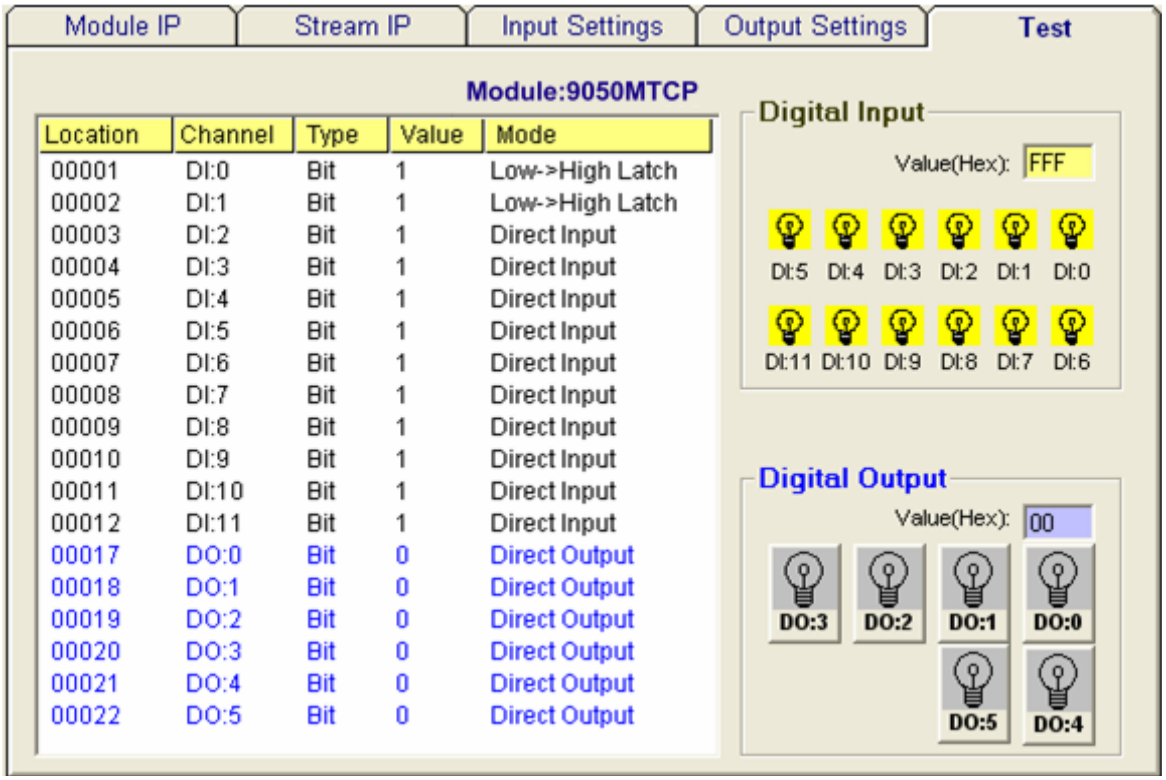


Figure 3-8 Event display

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.



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

Figure 3-9 ModBus location and I/O status

Digital I/O Module Configuration

Location: Standard Modbus address. 9050MTCP Ethernet I/O Utility shows the Modbus mapping address of each I/O channel. (Please refer to EX9000-MTCP.pdf file) 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.

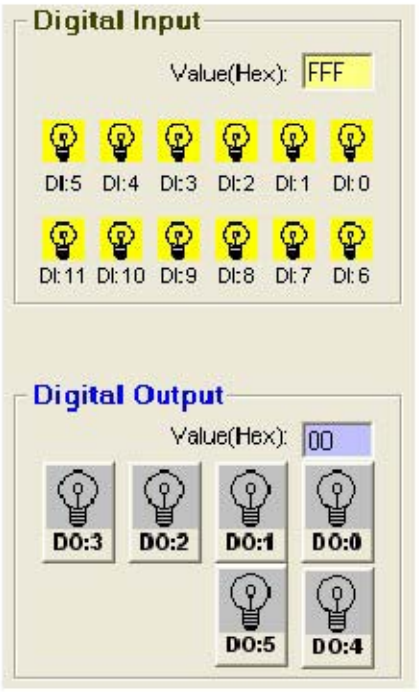


Figure3-10 DI/O status display

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

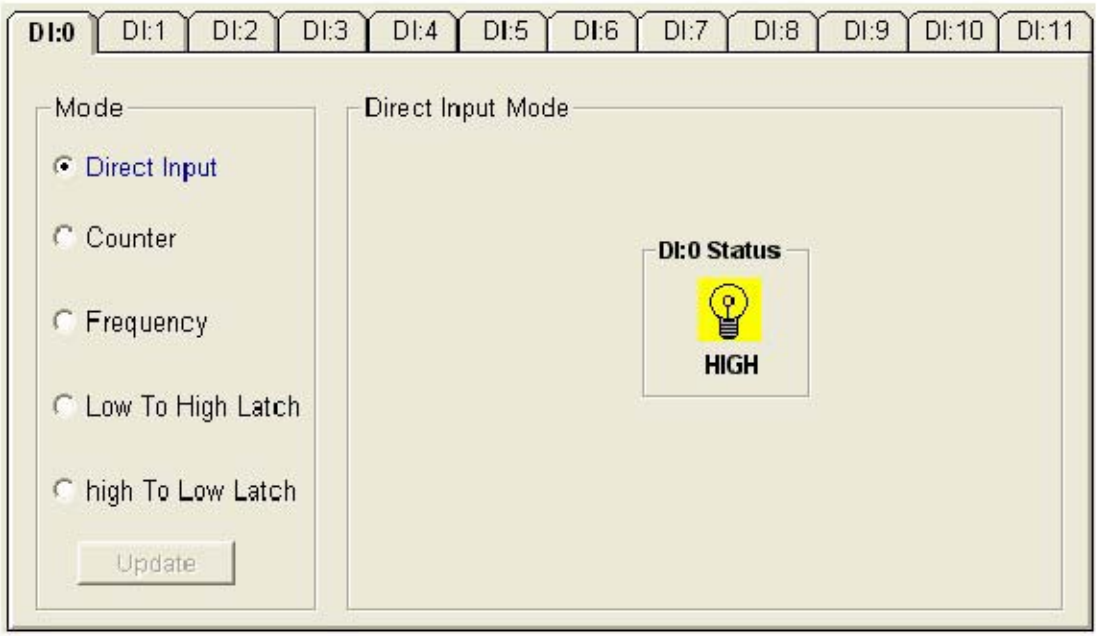


Figure 3-11 Direct input mode

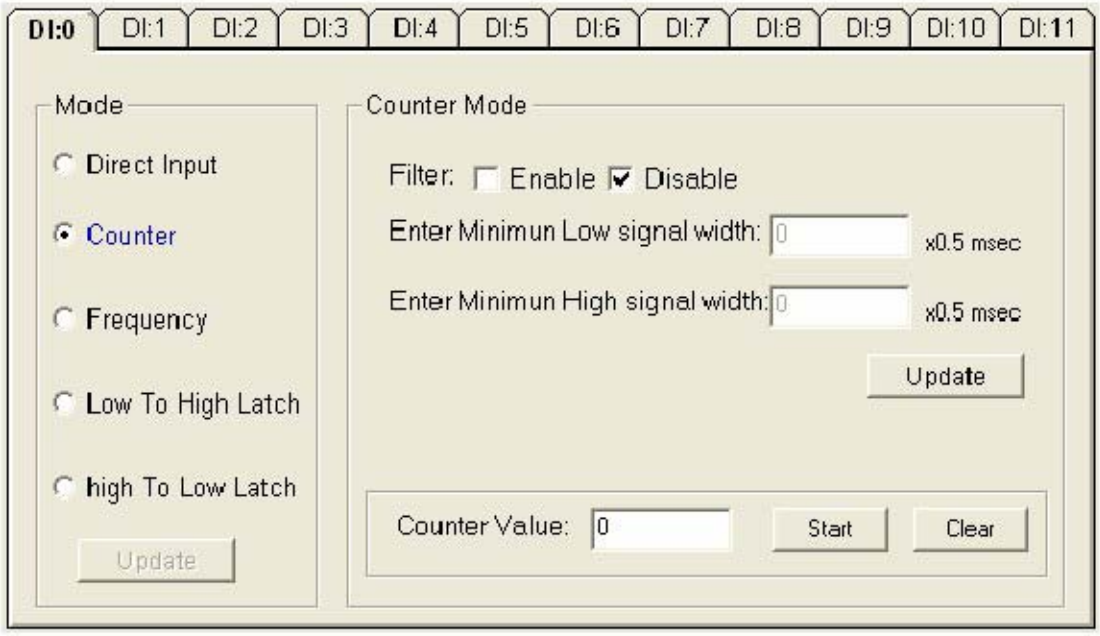


Figure 3-12 Counter setting

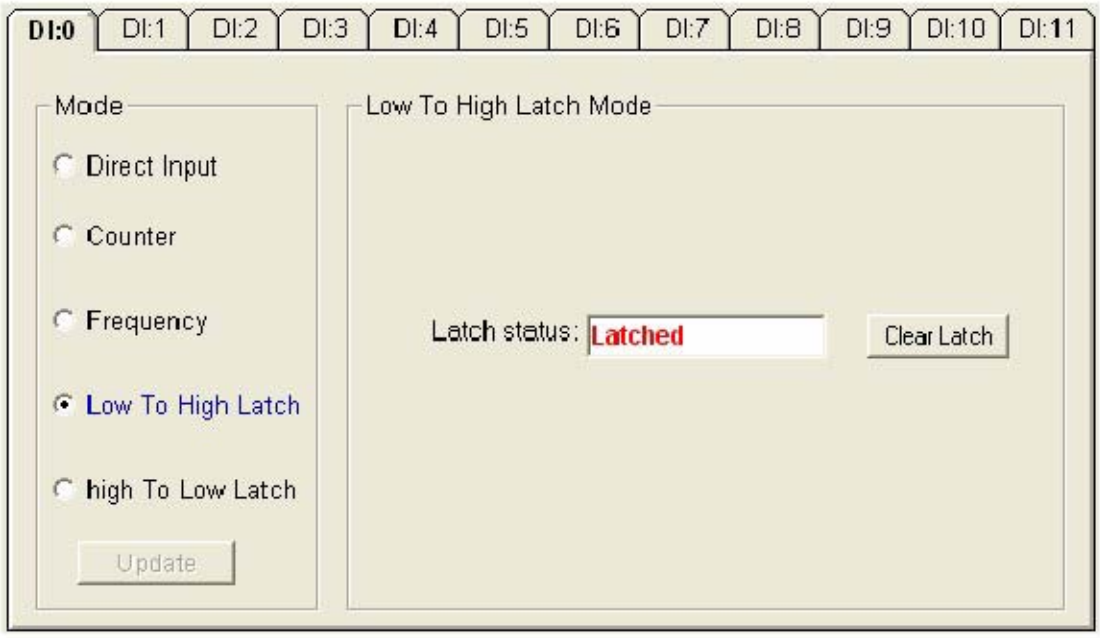


Figure 3-13 Input latch setting

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.

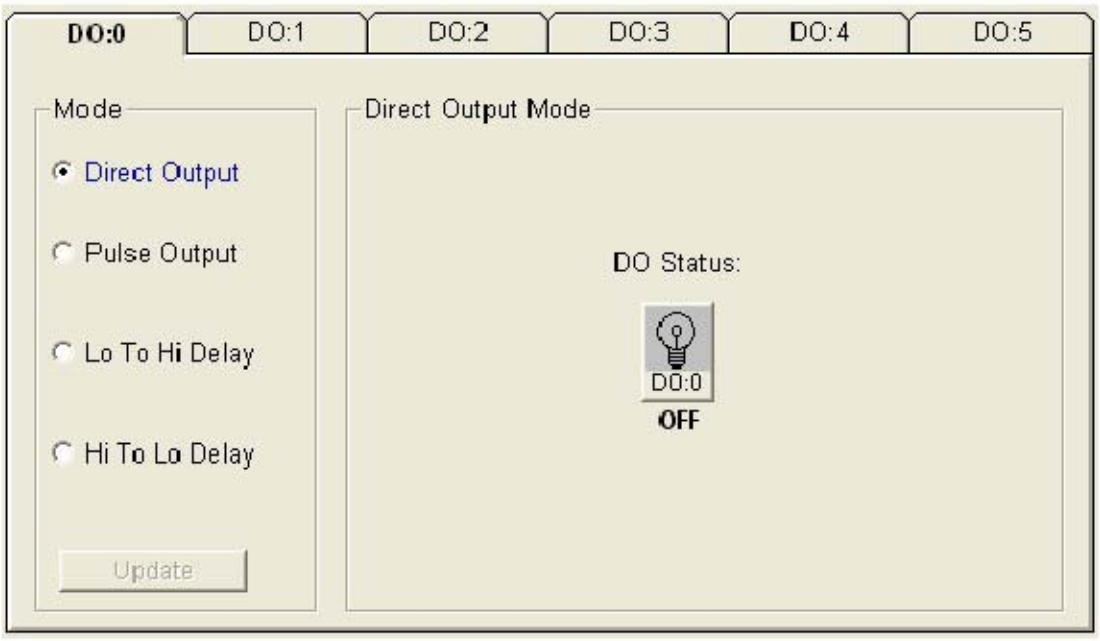


Figure 3-14 direct output setting

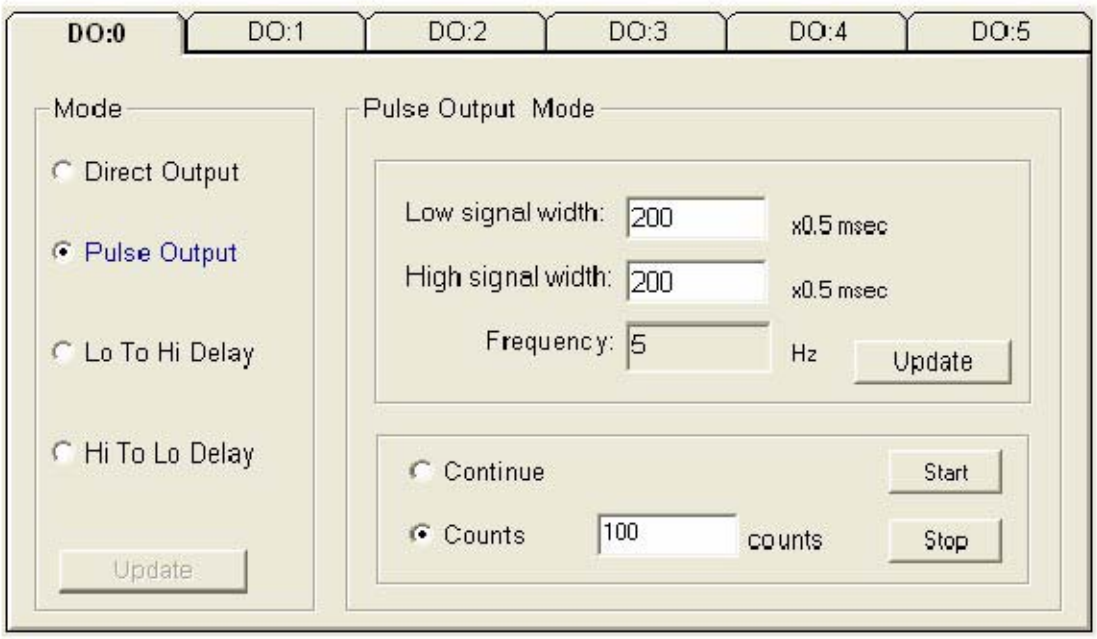


Figure 3-15 Pulse output setting

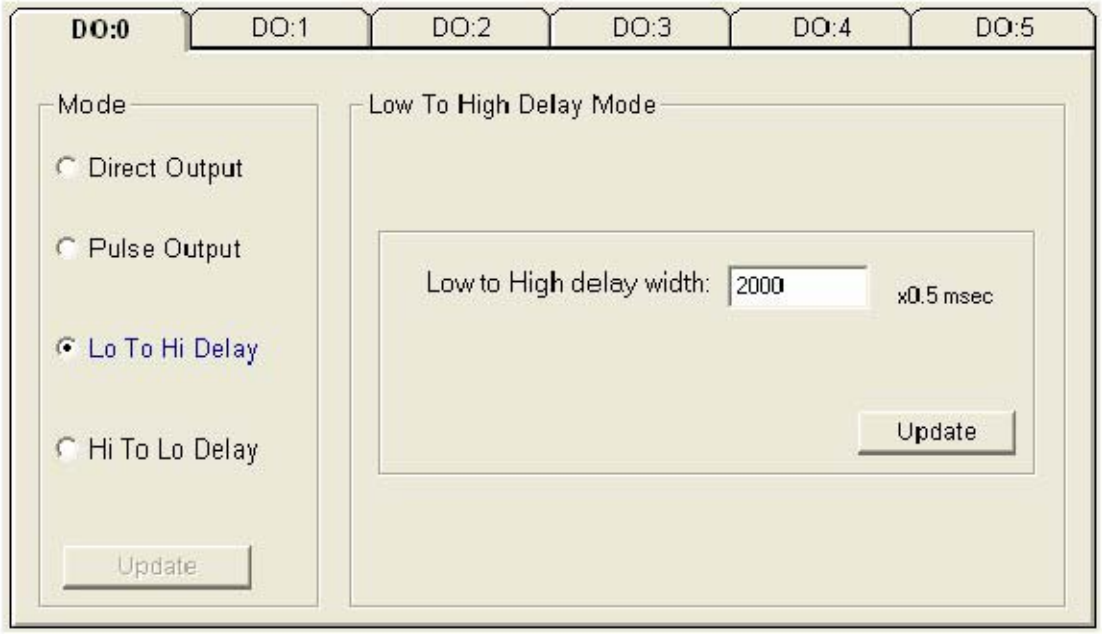


Figure 3-16 Low to High Delay setting

3.13.2 Analog Input Module

Selecting EX9000-MTCP analog input Modules includes 9017MTCP and select "General Settings" tab, user can read following information from the Utility.

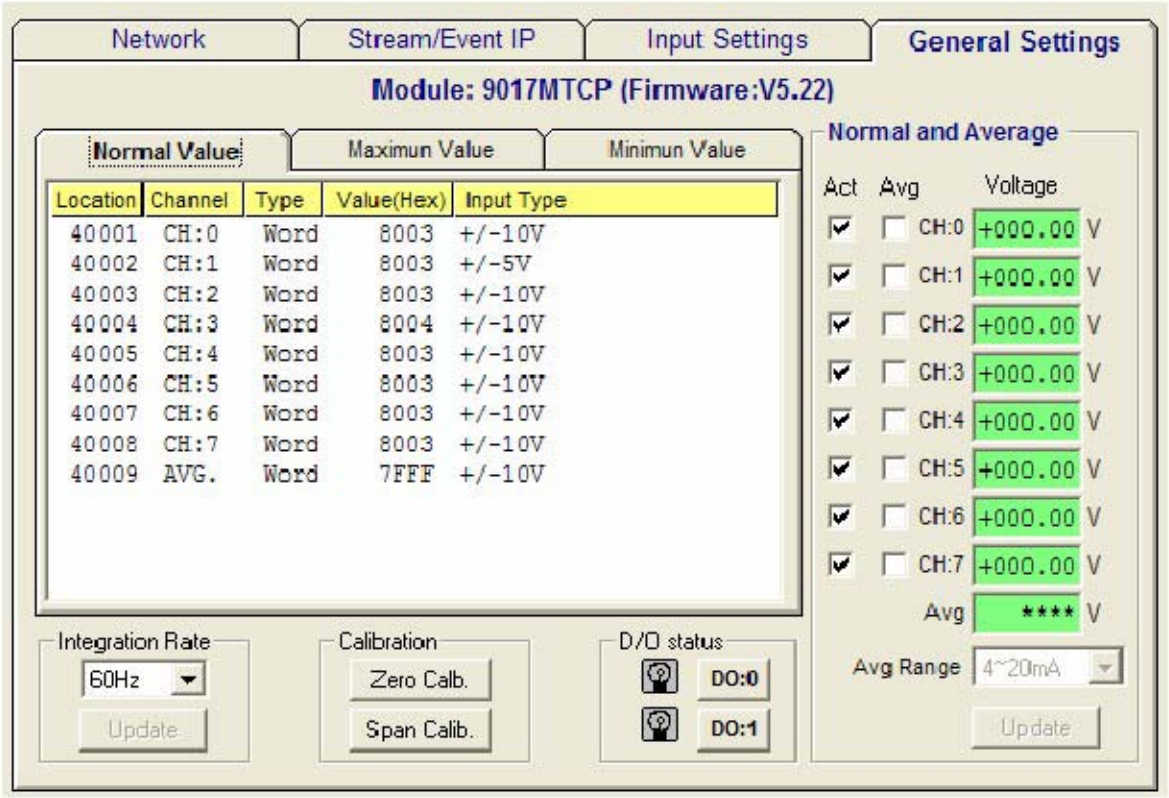


Figure 3-17 ModBus location and analog value

Location: Standard Modbus address. (Refer 4.0 to Assigning address for I/O module)

Channel: the channel number

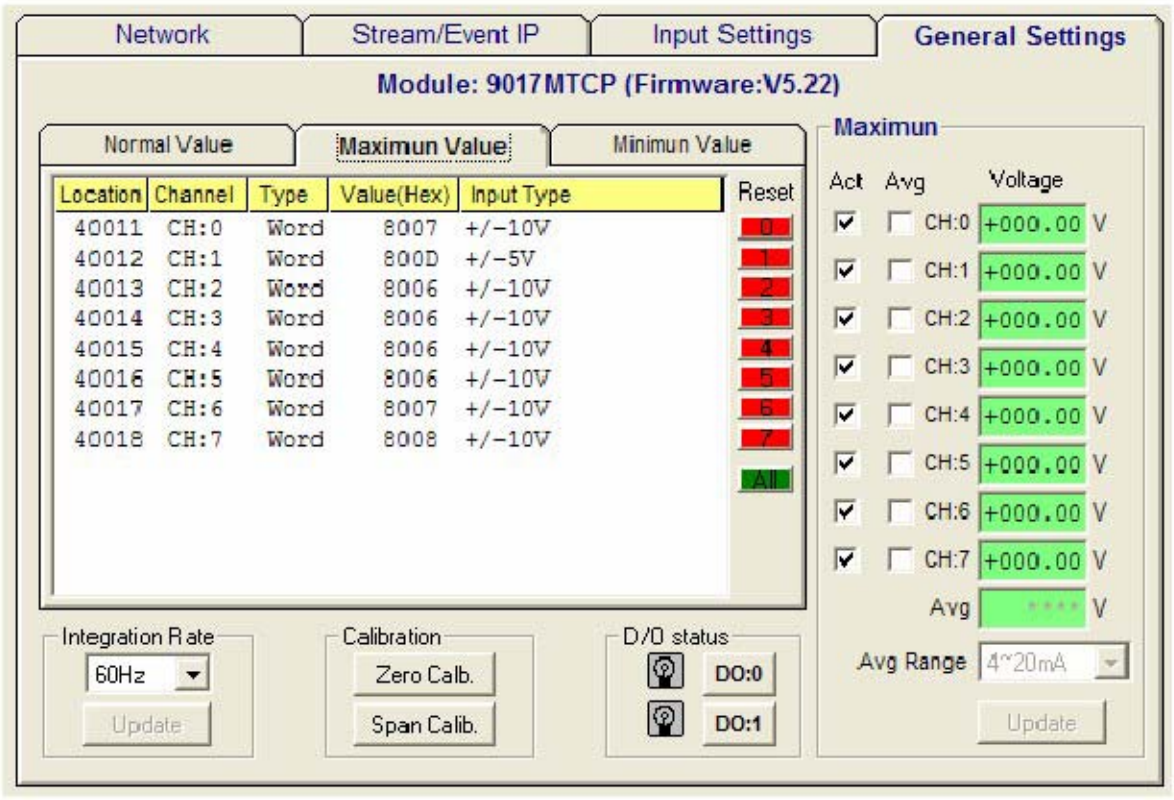
Type: Data type of the I/O channel. The data type of analog Input modules is always "word".

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

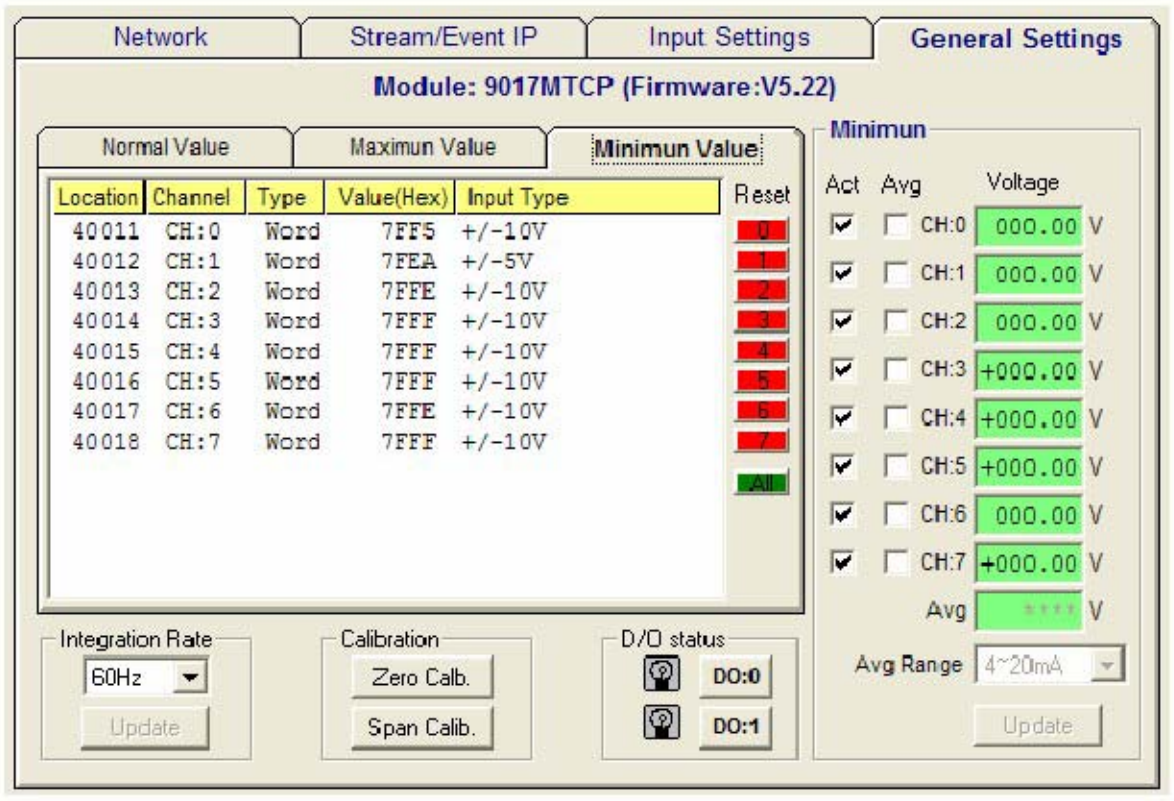
Input Type: Sensor types and measurement range of the specified module.

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.



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



3.14 I/O Module Calibrations

Calibration is to adjust the accuracy of EX9000M-TCP 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 9017MTCP 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.

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 9017MTCP module.
3. Click the Execute button.

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

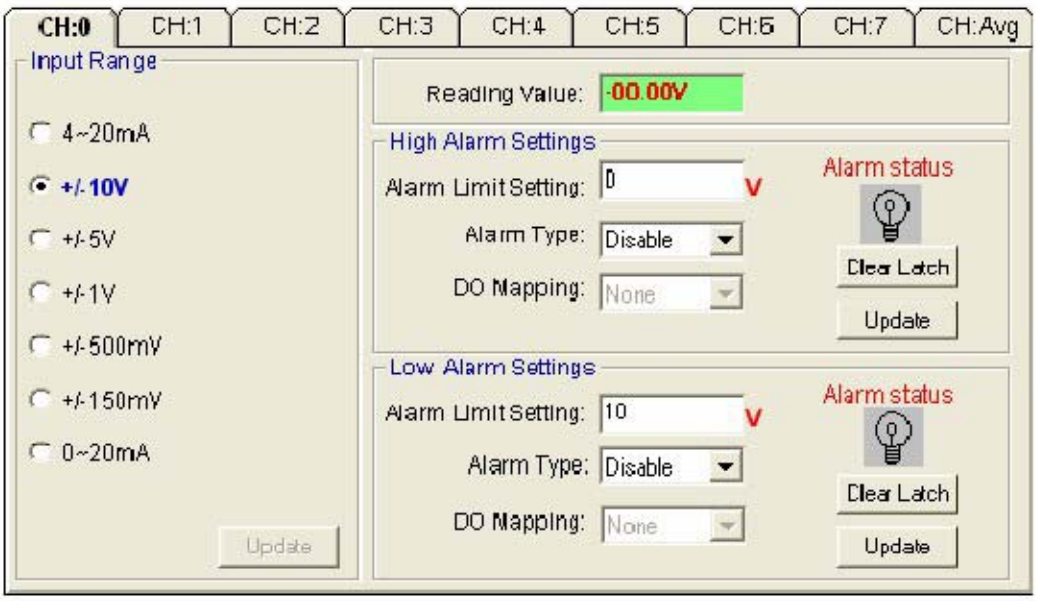


Figure 3-18 Input type setting

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 9017MTCP.

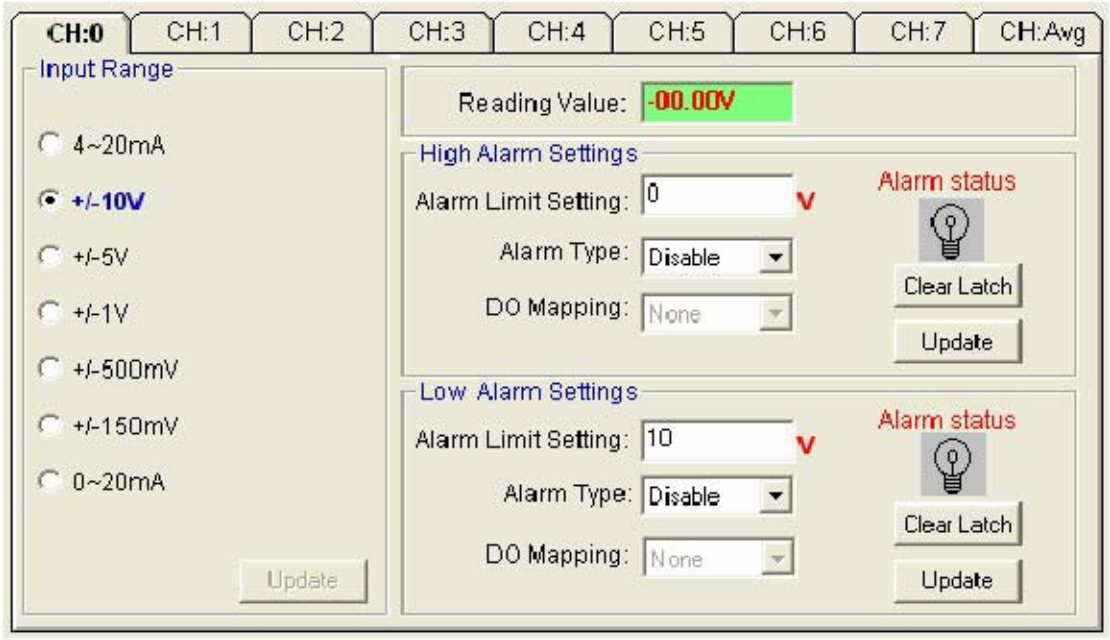


Figure 3-19 Alarm Setting

Table Of Contents

4.0 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 TCPDAQ ActiveX Control and

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

Insert the TCPDAQ installation CD 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 drive):

D: \Setup.exe

4.2 Building TCPDAQ ActiveX Control with Various Tools

This 4.0 of chapter describes how you can use the TCPDAQ ActiveX Control with the following development tools:

Microsoft Visual C++ version 6.0 (SP5)

Microsoft Visual Basic version 6.0 (SP5)

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

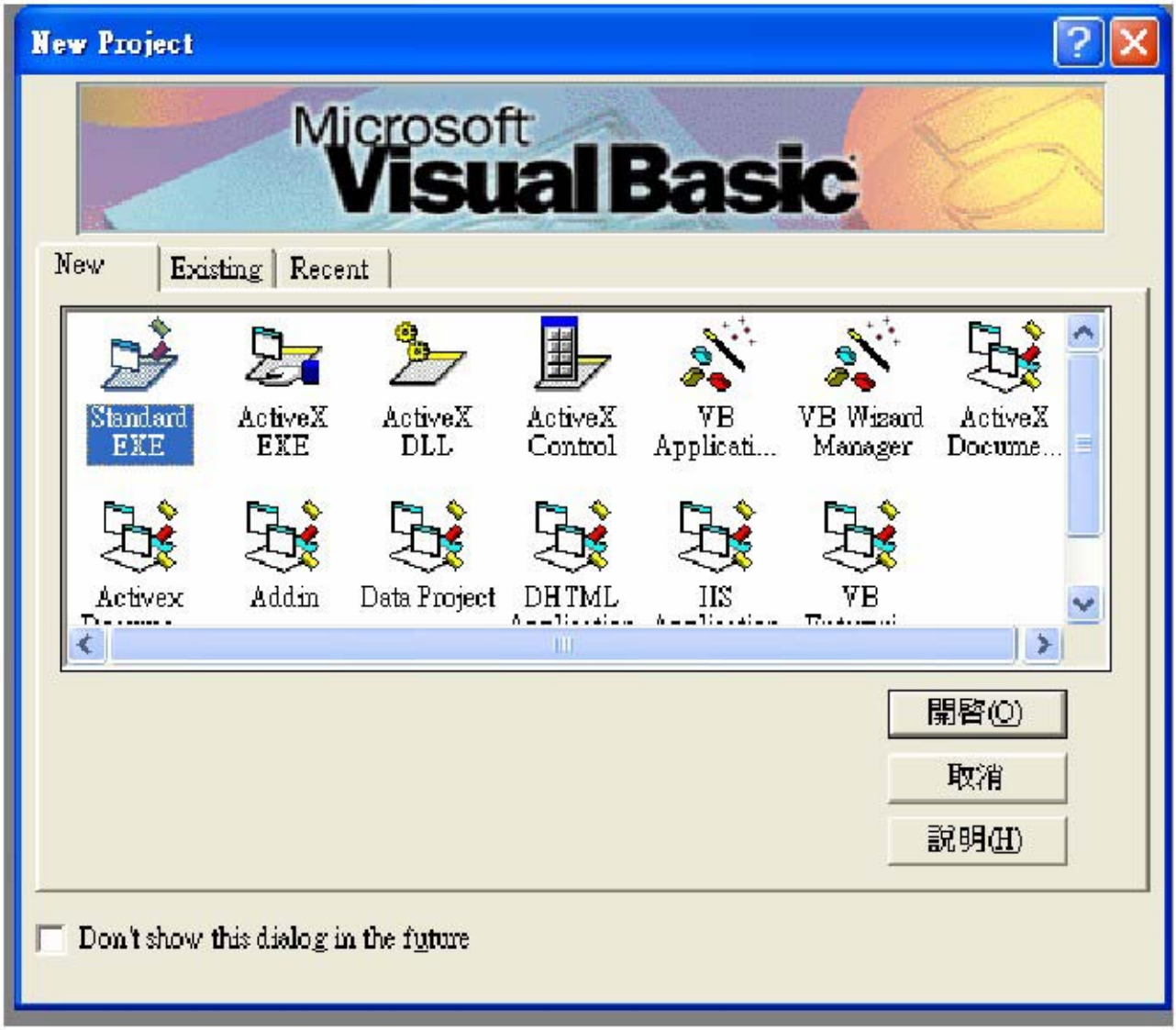
Borland C++ Builder version 5.0

This 4.0 of chapter assumes that you are familiar with the basic concepts of using Visual Basic, Delphi, Borland C++ Builder, and Visual C++, including selecting the type of application, designing the form, placing the control on the form, configuring the properties of the control, creating the code (event handler routines) for this control.

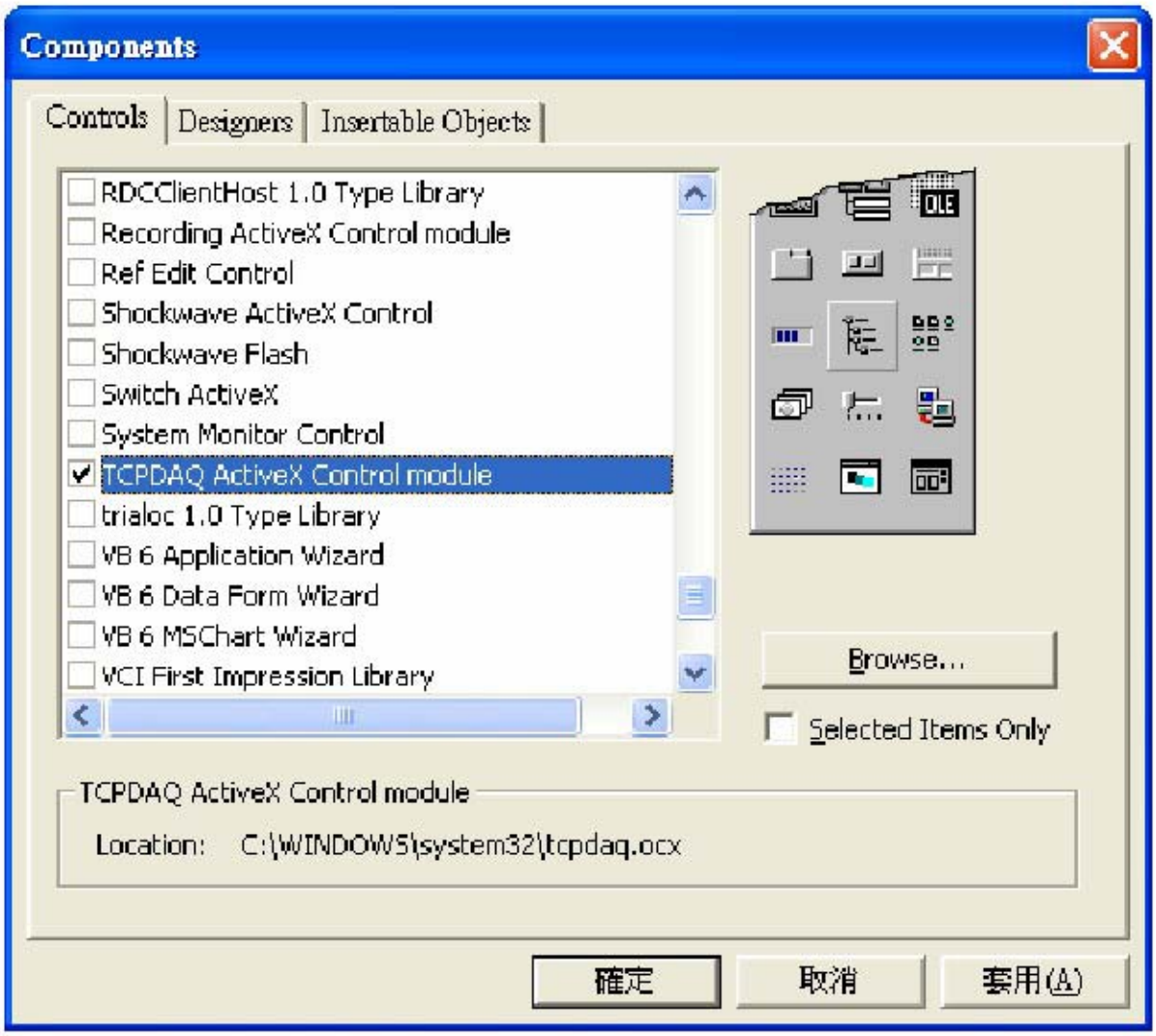
Note: For Borland Delphi 6, the Delphi 6 Update Pack fixes for ActiveX must be installed.

4.2.1 Building TCPDAQ Applications with Visual Basic

Start Visual Basic.

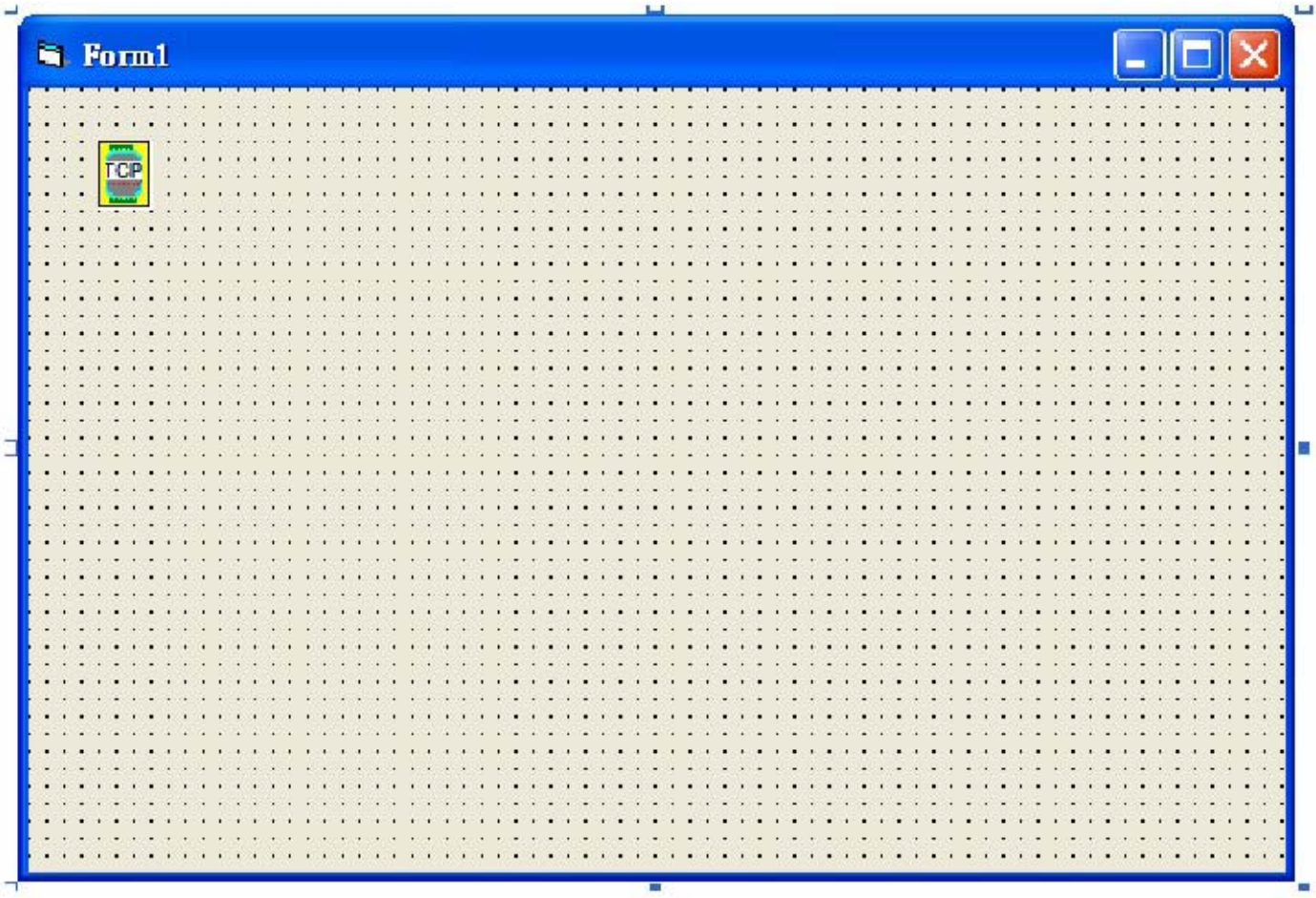


Select Standard EXE icon and press the Open button. A new project is created. Click on Components... from the Project menu. The Components dialog box is loaded as shown below:



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

Your form should look similar to the one shown below:

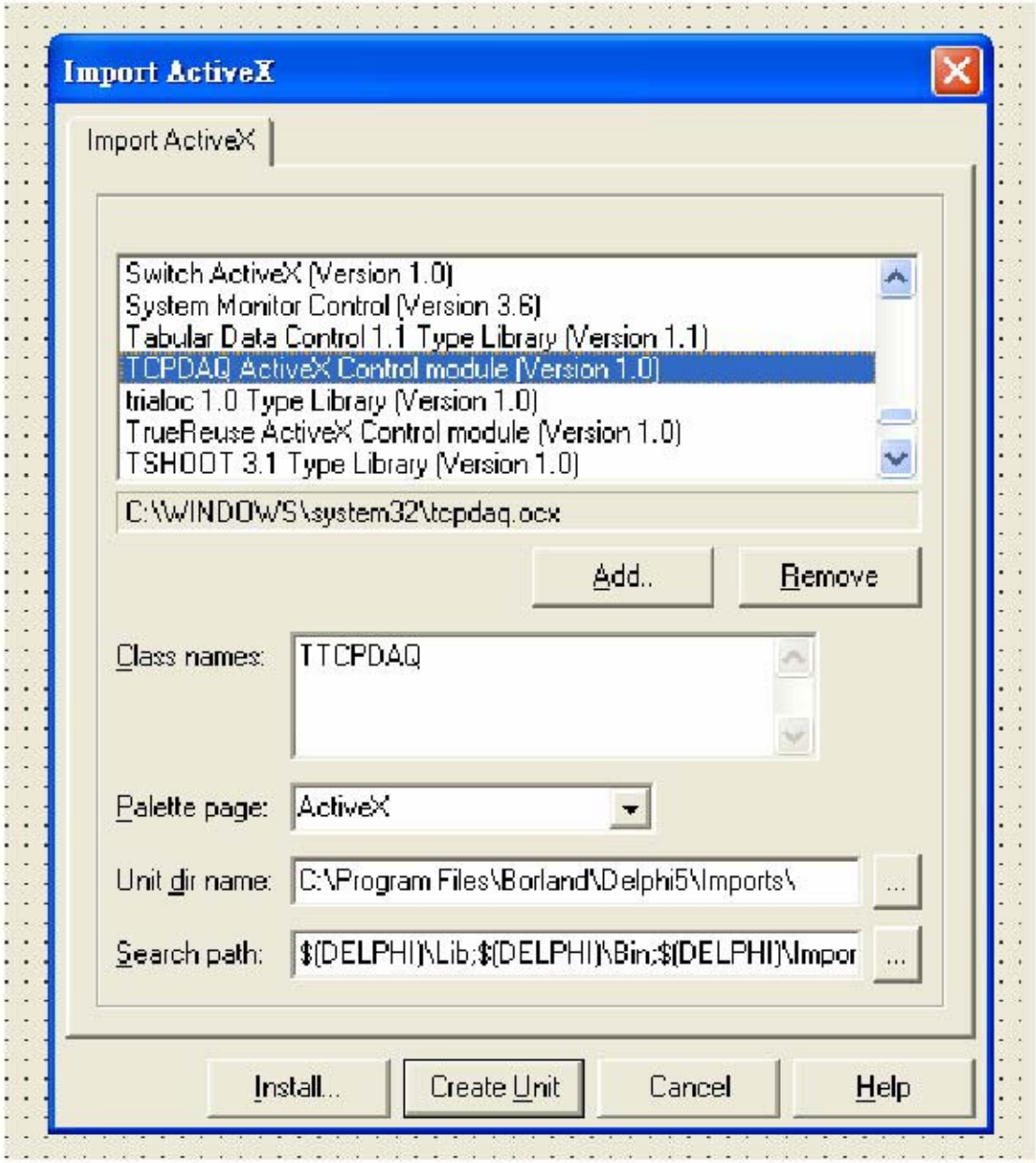


4.2.2 Building TCPDAQ Applications with Delphi

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:

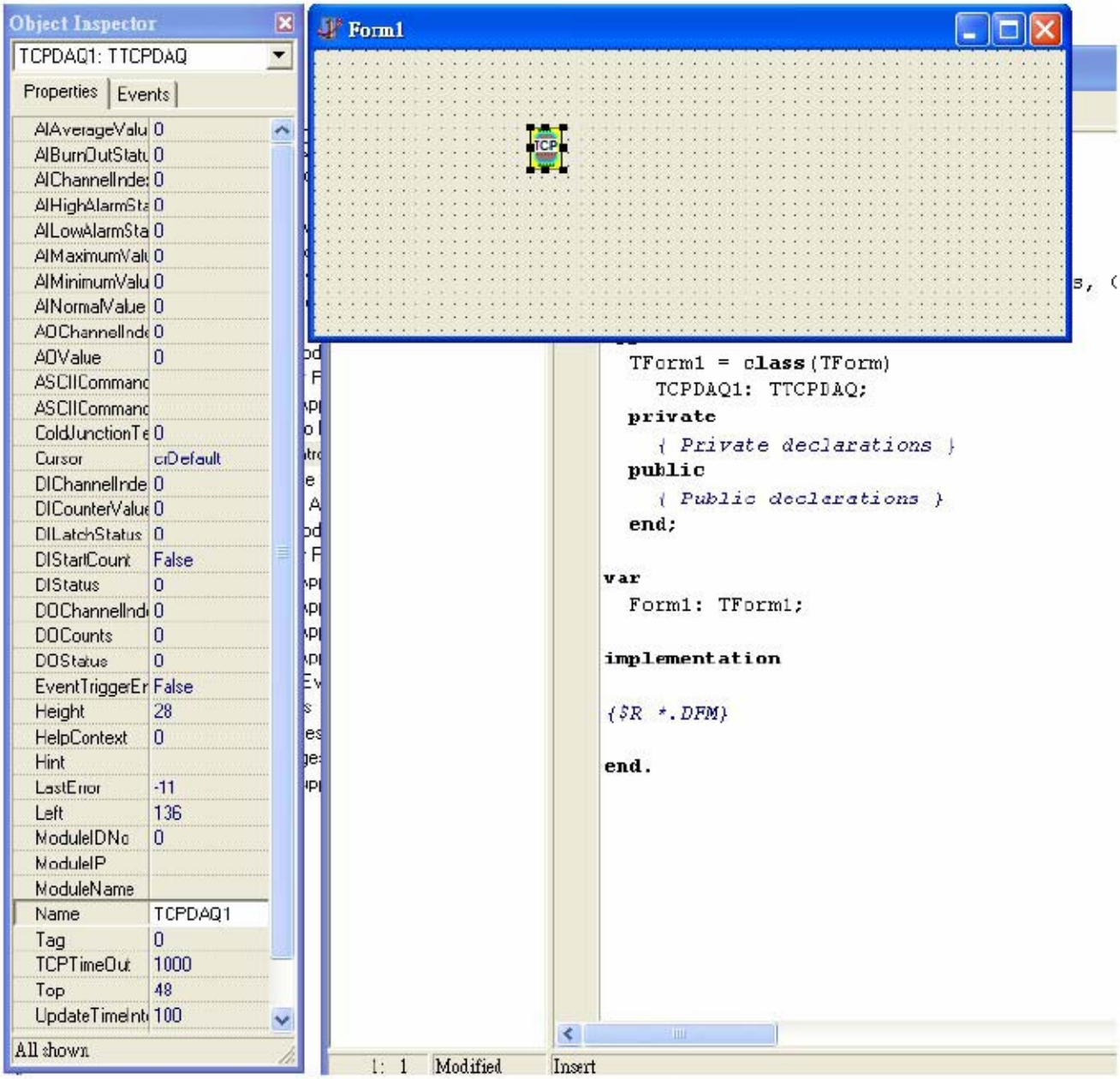


The TCPDAQ control is loaded into the Component Palette. You can check it by clicking on Install Package... from the Component menu. A dialog box is shown as below.

Switch to the form and select the ActiveX tab from the Component Palette.

Place a TCPDAQ control from the Component Palette on the form. Use the default names TCPDAQ1.

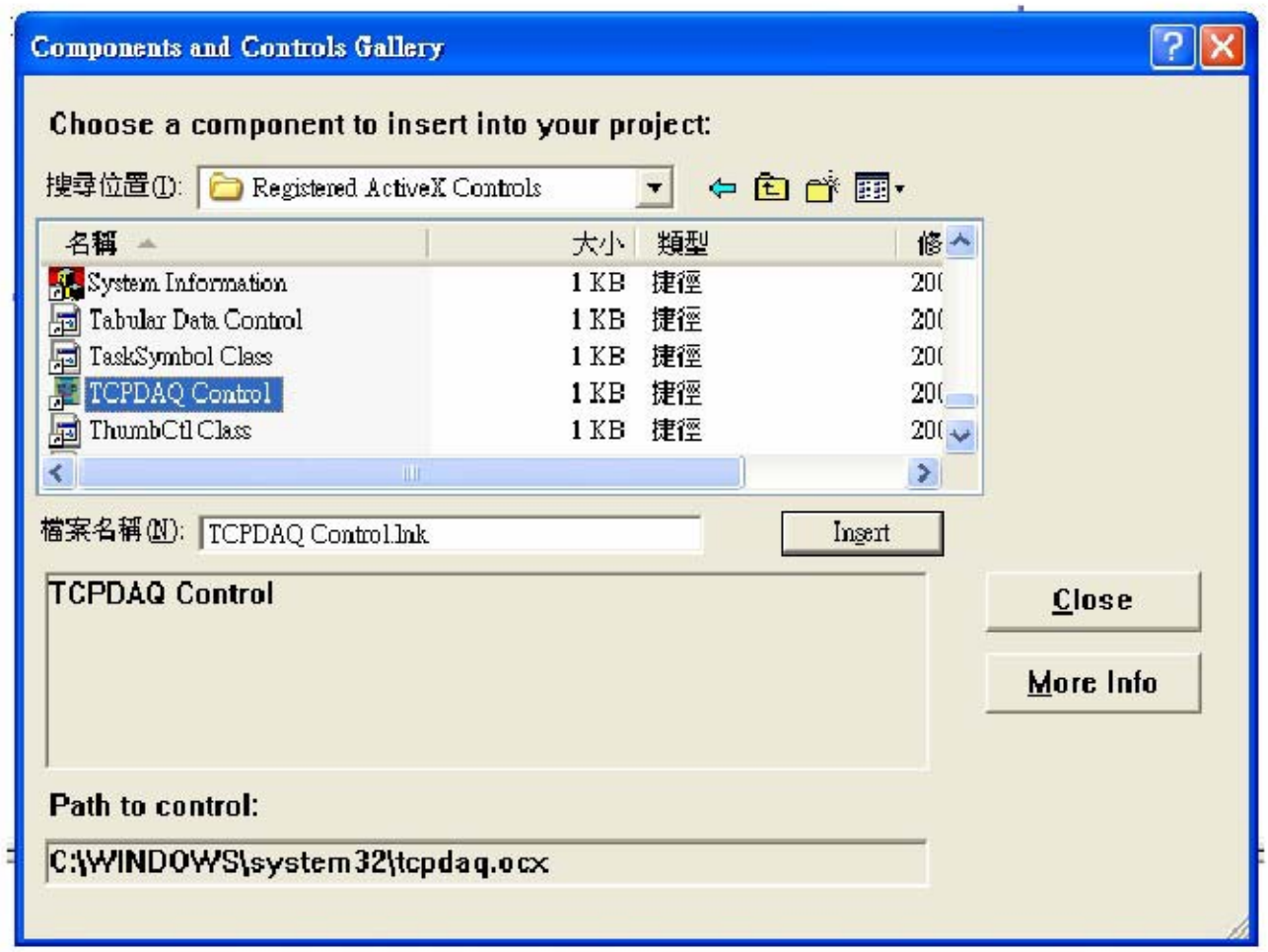
Your form should look similar to the one shown below:



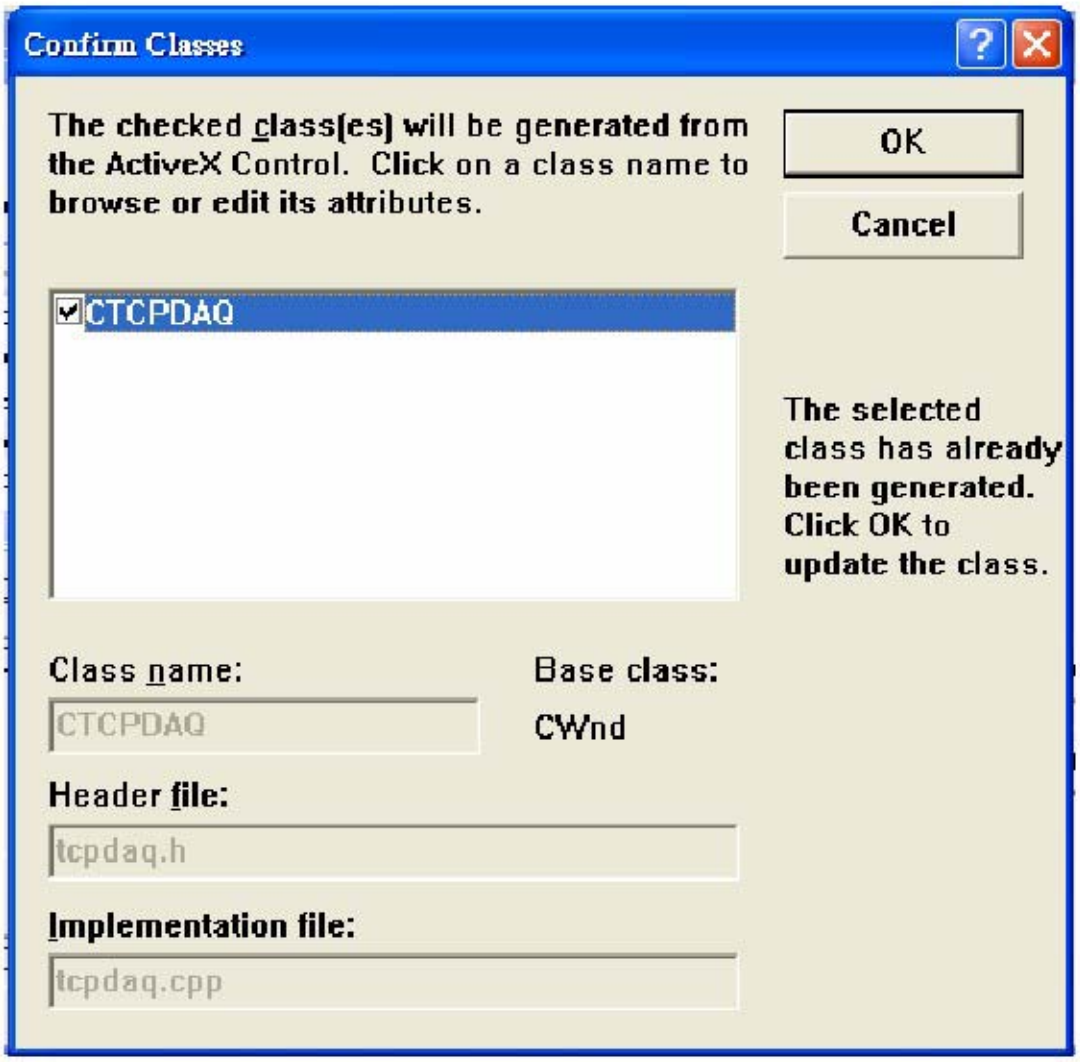
4.2.3 Building TCPDAQ Applications with Visual C++

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:

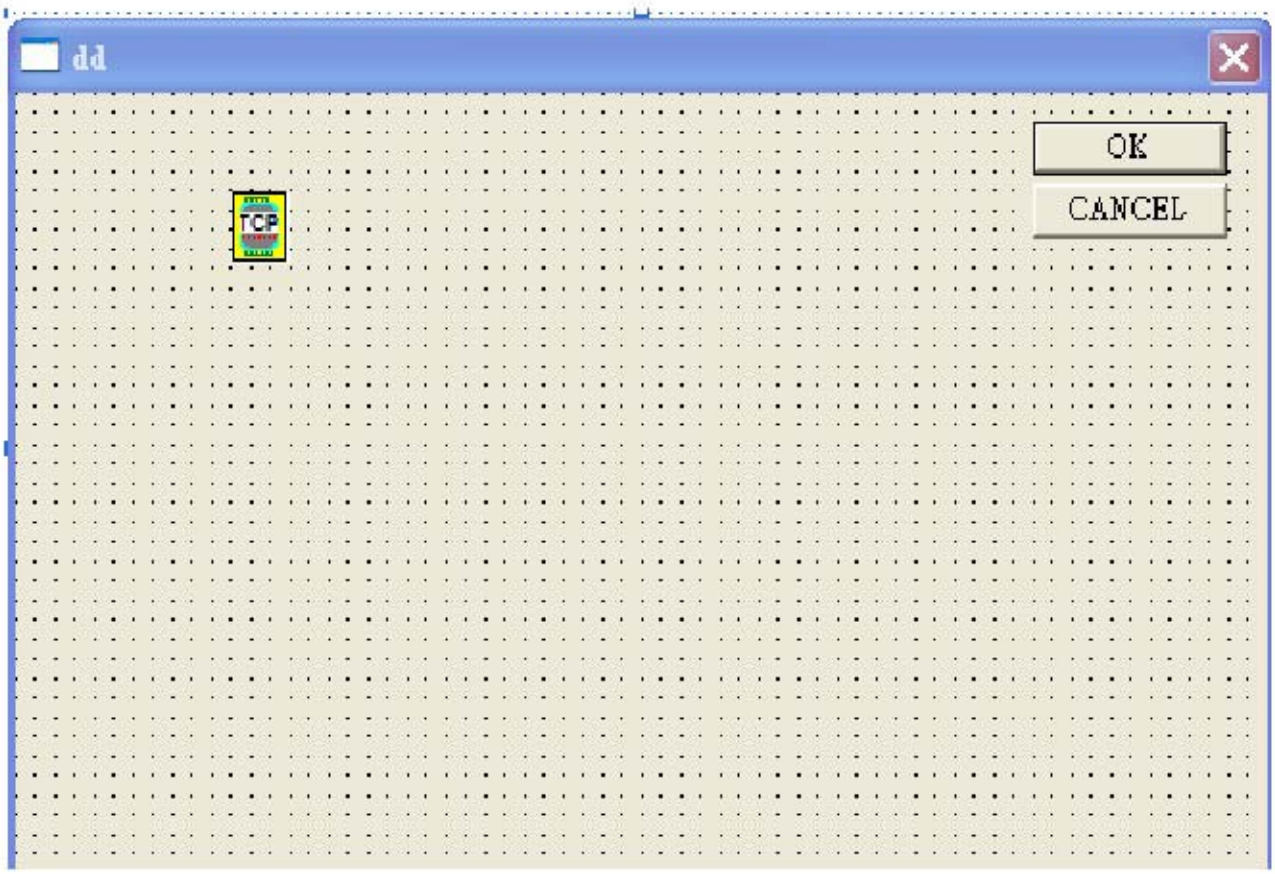


Scroll down to the TCPDAQ Control and press the Insert button. A Class Confirm dialog box is displayed, Press OK button.



The TCPDAQ control will be showed in Visual C++ Toolbar.

Place a TCPDAQ control from the Controls Toolbar on the dialog-based form.

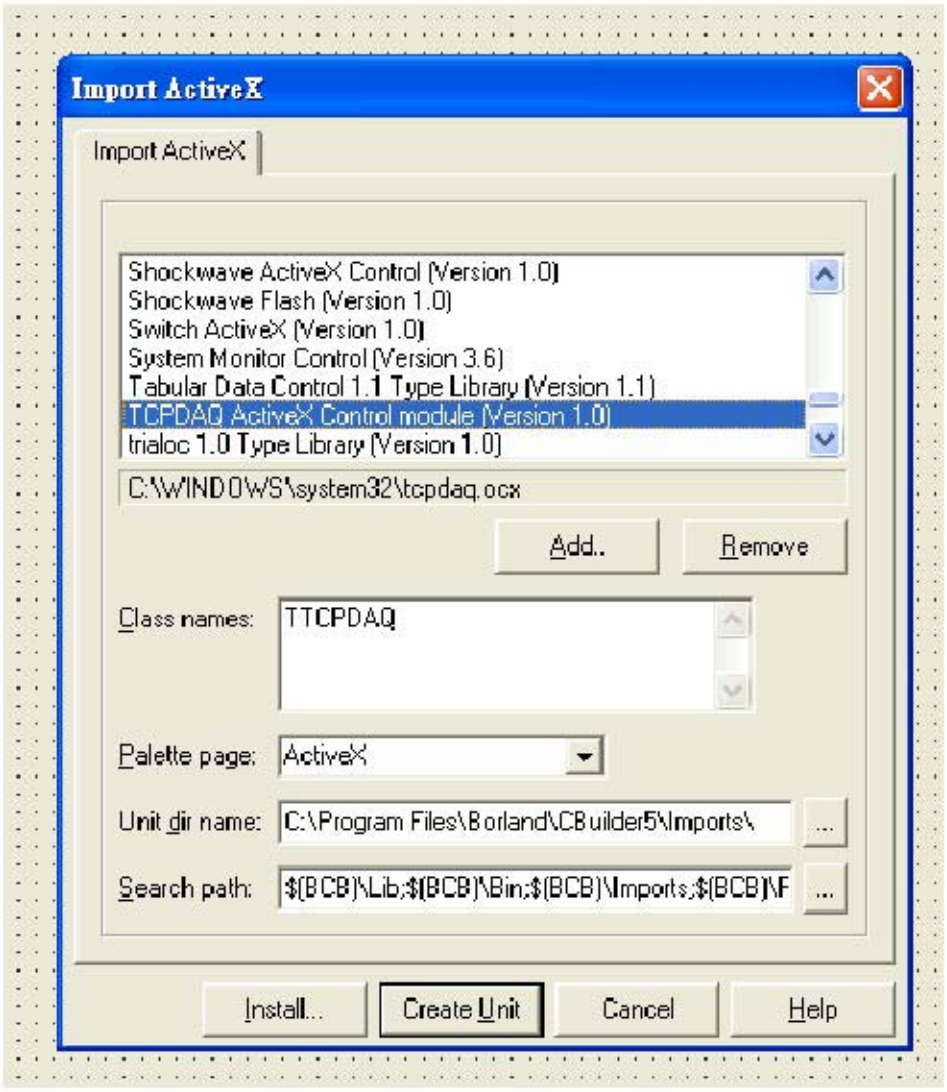


4.2.4 Building TCPDAQ Applications with Borland C++ Builder

Start Borland C++ Builder (BCB), BCB will launch as shown below:

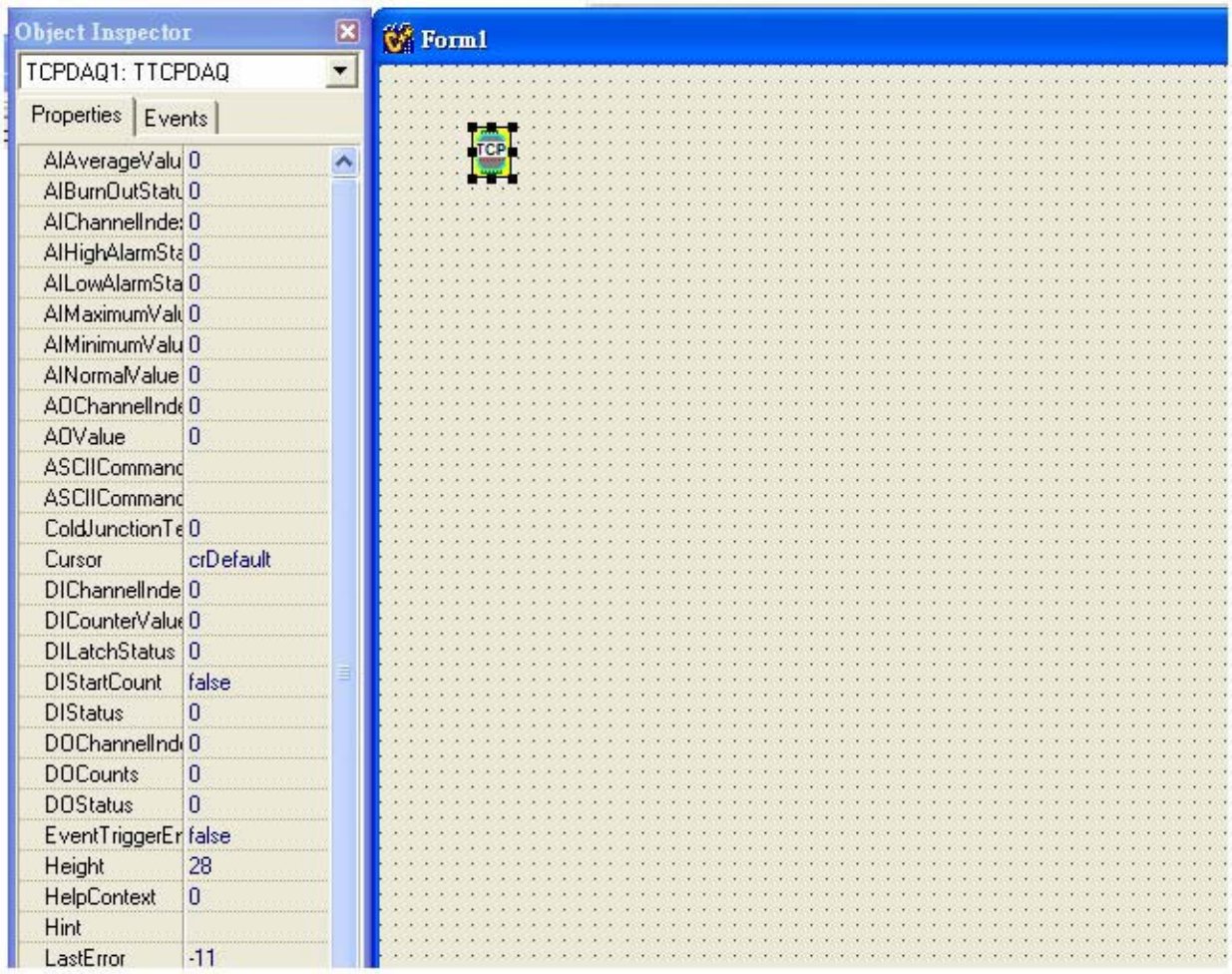
Select Import ActiveX Control... from the Component menu. The Import ActiveX dialog box loads:

Select the TCPDAQ Control and press the Install... button. A dialog box is displayed as follows:



Enter "TCPDAQ" into the File name field under the Into new package tab, and press OK button. A Confirm dialog box is displayed. press Yes button.

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



4.3 Properties of TCPDAQ ActiveX Control

Name	Type	DescriptionAvaliable Model(s)	
AIChannelIndex	short	Specifies the analog input channel to perform other AI properties read/write operation.	9015,9017,9019
AINormalValue	double	Normal voltage of specifies the analog channel	9015,9017,9019
AIAveragevalue	double	Average voltage value of the channels that are in average	9015,9017,9019
AIMaximumValue	double	Maximal voltage of specifies the analog channel	9015,9017,9019
AIMinimumValue	double	Minimal voltage of specifies the analog channel	9015,9017,9019
AILowAlarmStatus	short	Return the low alarm status of specifies the analog channel (1=Alarm occurred, 0=No alarm)	9015,9017,9019
AIHighAlarmStatus	short	Return the high alarm status of specifies the analog channel (1=Alarm occurred, 0=No alarm)	9015,9017,9019
AIBurnOutStatus	short	Return the Burnout status of specifies the analog channel (1=open, 0=normal)	9015 and 9019
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
ColdJunction Temperature	double	Return the cold junction temperature	9019
DIChannelIndex	short	Specifies the digital input channel to perform other DI properties read/write operation.	9050,9051,9055
DIounterValue	long	Return the counting value for the specified DI channel which functions in "Count/Frequency mode"	9050,9051,9055
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)	9050,9051,9055
DIStartCount	boolean	Start/stop counting for the specified DI channel which functions in "Count/Frequency mode" (True=Start, 0=Stop)	9050,9051,9055
DIStatus	short	Return the status for the specified DI channel which functions in "DI mode" (1=Active, 0=Inactive)	9050,9051,9055
DOChannelIndex	short	Specifies the digital output channel to perform other DO properties read/write operation.	9017,9019,9050,9051,9055

DOCount	long	Set the output count value for the specified DO channel which functions in "Pulse output mode"	9050,9051,9055
DOStatus	short	Return/set the status for the specified DO channel which functions in "D/O mode" (1=Active, 0=Inactive)	9017,9019,9050,9051,9055
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) string Datetime(out)	None	be called when error occurred
EventDataArrival	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 Refer 8.0 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 ActiveX control module are included in the provided CD. The Installed folders include the demo programs for various development tools.

Table Of Contents

5.0 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 module	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 module	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(9015MTCP/9019MTCP 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 9019MTCP module	5.6.64
TCP_ReadCJCOffset	To read cold junction offset from a specific 9019MTCPmodule	5.6.65
TCP_ReadCJCTemperature	To read cold junction temperature from a specific 9019MTCP 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 CD.

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: (ref TCPDAQ.bas)

```
Declare Sub TCP_Open Lib "TCPDAQ.dll" Alias "_TCP_Open@0" ()
```

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

```
int TCP_Open();
```

Delphi: (ref TCPDAQ.pas)

```
function TCP_Open(); StdCal;
```

VC++: (ref 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: (ref TCPDAQ.bas)

```
Declare Sub TCP_Close Lib "TCPDAQ.dll" Alias "_TCP_Close@0" ()
```

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

```
int TCP_ Close();
```

Delphi: (ref TCPDAQ.pas)

```
function TCP_ Close(); StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_Connect (szIP: PChar; port: Integer; ConnectionTimeout: Longint;
            SendTimeout: Longint;ReceiveTimeout: Longint): Longint; StdCall;
```

VC++: (ref 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: (ref TCPDAQ.bas)

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

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

```
void TCP_Disconnect(void);
```

Delphi: (ref TCPDAQ.pas)

```
procedure TCP_Disconnect ; StdCall;
```

VC++: (ref 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: (ref TCPDAQ.bas)

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

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

```
Int     TCP_ModuleDisconnect(char szIP[]);
```

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_SendData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_RecvData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_SendReceiveasCcmd (szIP: PChar; Sendbuf: PChar; Recvbuf: PChar): Longint;
    StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function UDP_Connect (szIP: PChar; s_port: word; d_port: word; ConnectionTimeout: Longint;
          SendTimeout: Longint; ReceiveTimeout: Longint): Longint; StdCall;
```

VC++: (ref 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: (ref TCPDAQ.bas)

```
Declare Sub UDP_Disconnect Lib "TCPDAQ.dll" Alias "_UDP_Disconnect@0" ()
```

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

```
void    UDP_Disconnect(void);
```

Delphi: (ref TCPDAQ.pas)

```
procedure UDP_Disconnect ; StdCall;
```

VC++: (ref 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: (ref TCPDAQ.bas)

```
Declare Function UDP_ModuleDisconnect Lib "TCPDAQ.dll" Alias "_UDP_ModuleDisconnect@4"
                (ByVal szIP As String) As Long
```

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

```
int     UDP_ModuleDisconnect(Char szIP[]);
```

Delphi: (ref TCPDAQ.pas)

```
Function UDP_ModuleDisconnect (szIP: PChar): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

```
int     UDP_SendData(char szIP[],char *pData,u_short wDataLen);
```

Delphi: (ref TCPDAQ.pas)

```
Function UDP_SendData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

```
int     UDP_RecvData(char szIP[],char *pData,u_short wDataLen);
```

Delphi: (ref TCPDAQ.pas)

```
Function UDP_RecvData (szIP: PChar; pData: PByte; wDataLen: Integer): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function    UDP_SendReceiveAsCcmd (szIP: PChar; Txdata:PChar; Rxdata: PChar): Longint;
StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function    TCP_GetModuleIPinfo (szIP: PChar; var ModuleIP: TModuleInfo): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function    TCP_GetModuleID(szIP: PChar; ModuleID: PByte): Longint; StdCall;;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function    TCP_GetIPFromID(szID: Byte; szIP: PChar): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function    Scan_OnLineModules (var ModuleIP: TModuleInfo; Sortkey: Byte): Longint; StdCall;
```

VC++: (ref 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: (ref TCPDAQ.bas)

```
Declare Function TCP_GetDLLVersion Lib "TCPDAQ.dll" Alias "_TCP_GetDLLVersion@0" () As
                Long
```

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

```
Int     TCP_GetDLLVersion(void);
```

Delphi: (ref TCPDAQ.pas)

```
Function TCP_GetDLLVersion: Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_GetModuleNo (szIP: PChar; Mname: PByte): Longint; StdCall;
```

VC++: (ref 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: (ref TCPDAQ.bas)

```
Declare Function TCP_GetLastError Lib "TCPDAQ.dll" Alias "_TCP_GetLastError@0" () As Long
```

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

```
Int     TCP_GetLastError(void);
```

Delphi: (ref TCPDAQ.pas)

```
Function TCP_GetLastError: Longint ; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_PingIP(szIP: PChar;PingTimes: Integer): Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref TCPDAQ.bas)

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

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

```
int     TCP_StopStream(void);
```

Delphi: (ref TCPDAQ.pas)

```
Function TCP_StopStream: Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref TCPDAQ.bas)

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

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

```
Int     TCP_StopEvent(void);
```

Delphi: (ref TCPDAQ.pas)

```
Function TCP_StopEvent: Longint; StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function    TCP_ReadDIOMode (szIP: PChar; DImode: PByte; DOmode: PByte): Longint;
            StdCall;
```

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

```
Function TCP_ReadDIO (szIP: PChar; ByDi: PByte; ByDo: PByte): Longint; StdCall;
```

VC++: (ref 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

byDO[out]: an 8 bit array that stored the DO channel status

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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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

Return Code:

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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref TCPDAQ.h)

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

Parameters:

szIP[in]: the IP address for an EX9000-MTCP 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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 (9015MTCP, 9019MTCP only)

Syntax:

Visual Basic: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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 9019MTCP,9015MTCP 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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 9019MTCP module

Syntax:

Visual Basic: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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 9019MTCP module

Syntax:

Visual Basic: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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 9019 module

Syntax:

Visual Basic: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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: (ref 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: (ref TCPDAQ.h)

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

Delphi: (ref TCPDAQ.pas)

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

VC++: (ref 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.

Table Of Contents

6.0 ASCII Commands for EX9000-MTCP Modules

6.1 About ASCII Commands

For users do not familiar to Modbus protocol, EXPERTDAQ 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(MTCP)	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

6.3.2 Analog commands

Command	Command name	Description	Modules(MTCP)	Sec.
#aan	Read single analog Input	Read the input value from the specified analog input channel	9015/9017/9019	6.4.4
#aa	Read all analog Input	Read the input values from all analog input channels	9015/9017/9019	6.4.5
#aaAcctt	Set analog input type	Set type of the specified analog input channel	9015/9017/9019	6.4.6
$aaBhh	Read input type	read input type of the specified analog channel	9015/9017/9019	6.4.7
$aa0	Span Calibration	Calibrate the analog input module to correct the gain error	9015/9017/9019	6.4.8
$aa1	Offset Calibration	Calibrate the analog input module to correct the offset error	9015/9017/9019	6.4.9
$aa6	Read Channel Enable/Disable Status	Read the Enable/Disable status of all analog input channels	9015/9017/9019	6.4.10
$aa5mm	Enable/disable ananlog channel(s)	Enable/disable analog input channels	9015/9017/9019	6.4.11
#aaMH	Read all Max. Data	Read the maximum data from all analog input channels	9015/9017/9019	6.4.12
#aaMHn	Read single Max. Data	Read the maximum date from a specified analog input channel	9015/9017/9019	6.4.13
#aaML	Read all Min. Data	Read the minimum data from all analog input channels	9015/9017/9019	6.4.14
#aaMLn	Read single Min. Data	Read the minimum data from a specified analog input channel	9015/9017/9019	6.4.15
$aaCjAhs	Set Alarm Mode	Set the High/Low alarm in either Momentary or Latching mode	9015/9017/9019	5.6.16
$aaCjAh	Read Alarm Mode	Returns the alarm mode for the specified channels	9015/9017/9019	6.4.17
$aaCjAhEs	Enable/Disable Alarm	Enables/Disables the high/low alarm of the specified channels	9015/9017/9019	6.4.18
$aaCjCh	Clear Latch Alarm	Resets a latched alarm	9015/9017/9019	6.4.19
$aaCjAhCCn	Set Alarm Connection	Connects the High/Low alarm of a specified input channel to interlock with a specified output channel	9017/9019	6.4.20
$aaCjRhC	Read Alarm Connection	Returns the alarm configuration of a specified input channel	9017/9019	6.4.21
$aaCjAhU	Set Alarm Limit	Sets the High/Low alarm limit value to a specified channel	9015/9017/9019	6.4.22
$aaCjRhU	Read Alarm Limit	Returns the High/Low alarm limit value of the specified channel	9015/9017/9019	6.4.23
$aaCjS	Read Alarm Status	Reads whether an alarm occurred in the specified Channel	9015/9017/9019	6.4.24
$aa3	Read cold junction	Return the Cold Junction temperature	9019	6.4.25
$aa9hhhhh	Set CJ offset	Set Cold Junction temperature offset	9019	6.4.26
$aa9	Read CJ offset	Return Cold Junction temperature offset	9019	6.4.27

6.3.3 Digital I/O commands

Commands	Command name	Descritpion	Modules(MTCP)	Sec.
$aa6	Read DI/O Channel Status	read the status of all DI and DO channels	9050/9051/9055	6.4.28
@aa	Read DI/O Status	read the status of all DI and DO channels	9050/9051/9055	6.4.29
#aa7	Read DI latch status	Read DI latch status	9050/9051/9055	6.4.30
#aa00dd	Write All DO channels	Write a value to all digital output channels	9050/9051/9055	6.4.31
#aa1n0d	Set single DO channel	Set the single digital output channel	9050/9051/9055	6.4.32
$aaEcn	Enable/disable DI counter	Start/stop counter of the specified DI channel	9050/9051/9055	6.4.33
$aaCn	Clear DI counter	Clear DI counter of the specified DI channel	9050/9051/9055	6.4.34
#aa	Read DI counter	Read all DI counter values	9050/9051/9055	6.4.35
#aan	Read DI counter	Read DI counter value of the specified DI channel	9050/9051/9055	6.4.36
#aa2npppppppp	Write DO pulse counts	Write DO pulse counts to the specified DO channel	9050/9051/9055	6.4.37
$aaCLS	Clear all latch status	Clear latch status of all DI channels	9050/9051/9055	6.4.38

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: !aa90bb(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.
bb	(range 00-FF) represents the 2-character model numberof an EX9000-MTCP module.
(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 indicating that there is an 9050MTCP 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(version)(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.
(version)	represents the firmware version of the module.
(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.

6.4.3 $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).

6.4.4 #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 9050MTCP analog module at address 01h responds with an input value +01.000.

6.4.5 #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

6.4.6 $aaAcctt Set analog input type (range)

Description: Set the analog input type (range) in EX9000-MTCP analog input module.

Syntax: $aaAnntt(cr)

$	is a delimiter character.
01	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(ref. 6.4.7)
(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

6.4.7 $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 -50°C ~ 150°C
0x21	IEC Pt100 0°C ~ 100°C
0x22	IEC Pt100 0°C ~ 200°C
0x23	IEC Pt100 0°C ~ 400°C
0x24	IEC Pt100 -200°C ~ 200°C
0x25	JIS Pt100 -50°C ~ 150°C
0x26	JIS Pt100 0°C ~ 100°C
0x27	JIS Pt100 0°C ~ 200°C
0x28	JIS Pt100 0°C ~ 400°C
0x29	JIS Pt100 -200°C ~ 200°C
0x2A	Pt1000-40°C ~ 160°C
0x2B	BALCO500 -30°C ~ 120°C
0x2C	Ni -80°C ~ 100°C
0x2D	Ni0°C ~ 100°C

Figure 6-1 Analog input types

6.4.8 $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.

6.4.9 $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 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.

6.4.10 $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).

6.4.11 $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-MTCP 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.

6.4.12 #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.

6.4.13 #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.

6.4.14 #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.

6.4.15 #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.

6.4.16 $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.

6.4.17 $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.

6.4.18 $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).
AhEs	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.

6.4.19 $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.

6.4.20 $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 9050MTCP 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.

6.4.21 $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.

6.4.22 $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.

6.4.23 $aaCjRhU Read Alarm Limit

Description: Returns the High/Low alarm limit value for the specified input channel in the addressed EX9000-MTCPmodule

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.

6.4.24 $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.

6.4.25 $aa3 Read cold junction temperature

Description: Return the Cold Junction temperature of 9019MTCP

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 +17.5°C

6.4.26 $aa9hhhhh Set CJ offset

Description: Set Cold Junction temperature offset of 9019MTCP

Syntax: $aa19hhhhh(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 EX9000-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 2°C , then the actual ASCII value should be 2 *80=160 (hex=000A0). Hence the complete ASCII command string is $019000A0(cr)

6.4.27 $aa9 Read CJ offset

Description: Return Cold Junction temperature offset of 9019MTCP

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 +5.5°C

6.4.28 $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(Do data)(Di data)(Di data)(Di 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 network address of an EX9000-MTCP module.

(Do data) 2-character hexadecimal value representing the values of the digital output module.

(Di data) 3-character hexadecimal value representing the values of the digital input module.

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

Example: command: $016(cr)

response: !0100A35D(cr)

0A: the status of digital output channels.

0A=(00001010) DO channels 1/3 =ON, DO Channel 0/2/4/5/6/7=OFF

35D: the status of digital input channels.

35D=(001101011101) DI channels 0/2/3/4/6/8/9 =Active state, DI Channel 1/5/7=Inactive

6.4.29 @aa Read DIO status

Description: Read digital input and output status.

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.

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

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

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

Example: command: @01(cr)

response: >03004(cr)

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

004 represents DI 2 is ON and DI 0, DI 1, and DI 3~DI 11 are OFF

Note: data2 is always 000 for 9050MTCP

6.4.30 $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: !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	(range 00-FF) represents the 2-character hexadecimal Modbus address of an EX9000-MTCP module.
data	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

6.4.31 #aa00dd Write All Digital Output

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

Syntax: #aa00nn(data)(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

6.4.32 #aa1n0d Set Single Digital Output Channel

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

Syntax: #aa1n0d(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.
d	(range 0-1) represents the status you want to set to the specific channel
(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.

6.4.33 $aaEcn Start/ Stop single DI counter

Description: start/stop single digital input counter

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: !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: $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

6.4.34 $aaCn Clear single DI counter value

Description: clear single digital input counter value

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: !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: $01C2(cr)

response: !01(cr)

2 represents DI counter channel 2

6.4.35 #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: !01(data)(data)(data)….(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.
01	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: !0100000001230230000001000000005230000005001101000004 32…… (cr)

0000000123 represents channel #0 counter value is 123

0230000001 represents channel #1 counter value is 230000001

0000000523 represents channel #2 counter value is 523

0000005001 represents channel #3 counter value is 5001

… so on

Note:

This command is valid for 9050MTCP/9051MTCP/9055MTCP digital I/O modules only

This command is supported for 9050MTCP/9051MTCP/9055MTCP with firmware V2.21 or later

6.4.36 #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: !01(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.
01	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)

2 represents DI counter channel 2

0000000123 represents counter value is 123

6.4.37 #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: !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: #0123001F(cr)

response: !01(cr)

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

6.4.38 $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: !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: $01CLS(cr)

response: !01(cr)

The command clears all DI latch status

Table Of Contents

7.0 MODBUS/TCP Command structure

EX9000-MTCP system accepts a command/response form with the host computer. When systems are not
MODBUS/TCP Command structure

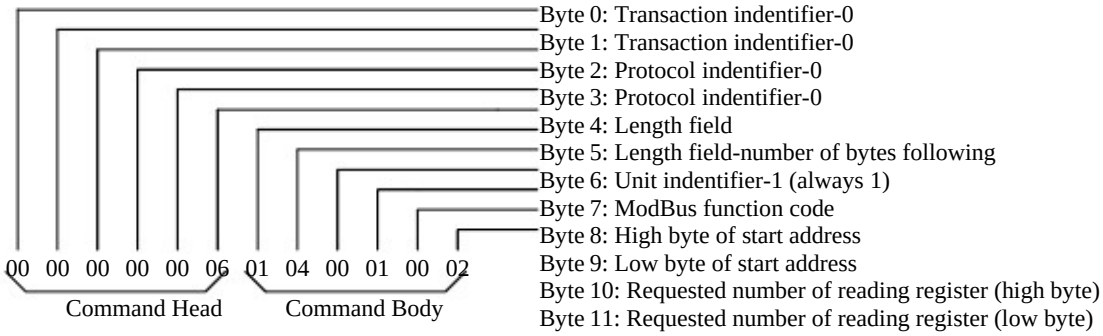
EX9000-MTCP system accepts a command/response form with the host computer. When systems are not transmitting they are in listen mode. The host issues a command to a system with a specified address and waits a certain amount of time for the system to respond. If no response arrives, a time-out aborts 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.

7.1 Command Structure

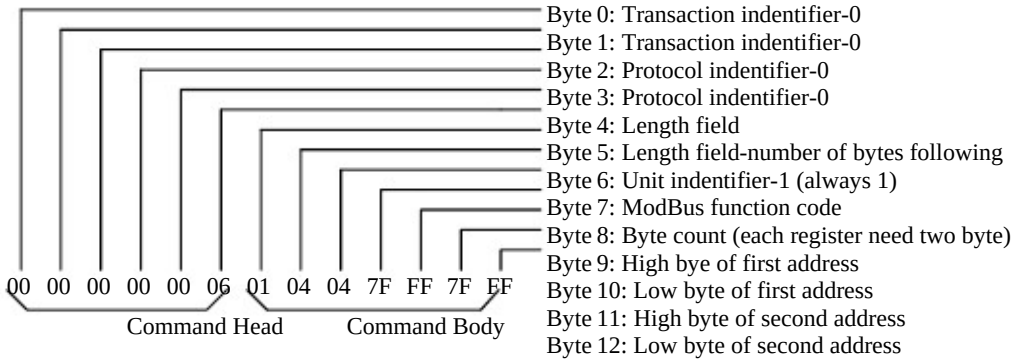
It is important to understand the encapsulation of a Modbus request or response carried on the Modbus/TCP network. A complete command is consisted of command head and command body. The command head is prefixed by six bytes and responded to pack Modbus format; the command body defines target device and requested action. Following example will help you to realize this structure quickly.

Example:

If you want to read the first two values of EX9017-MTCP (address: 40001~40002), the request command should be:



And the response should be:



7.2 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

7.3 9050MTCP 12 Digital Input/6 Digital Output Module

7.3.1 Holding Register Address (Unit:16bits)

Where X=40000 for function 03, function 06, function 16

X=30000 for function 04

Address	Channel	Item
X+0001~X+0024	For Counter	12 Channels, 32 Bits
X+0025~X+0036	For Pulse Output L level, time Unit:0.1ms	6 Channels, 32 Bits
X+0037~X+0048	For Pulse Output H level, time Unit:0.1ms	6 Channels, 32 Bits
X+0049~X+0060	Set Absolute pulse (Set to 0=Continue mode)	6 Channels, 32 Bits
X+0061~X+0073	Set DO pulse value	Channels, 32 Bit

7.3.2 Bit Address (Unit:1Bit)

Where X=00000 for function 01, function 05

X=10000 for function 02

Address	Channel	Item
X+0001~X+0012	For DI	12 Channels, 1 Bit
X+0017~X+0022	For DO	6 Channels, 1 Bit
X+0032	Ch0 (For Counter Mode)	Start(1)/Stop(0)
X+0033	Ch0 (For Counter Mode)	Clear Counter(1)
X+0034	Ch0 (For Counter Mode)	Clear Overflow
X+0035	Ch0 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0036	Ch1 (For Counter Mode)	Start(1)/Stop(0)
X+0037	Ch1 (For Counter Mode)	Clear Counter(1)
X+0038	Ch1 (For Counter Mode)	Clear Overflow
X+0040	Ch1 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0041	Ch2 (For Counter Mode)	Start(1)/Stop(0)
X+0042	Ch2 (For Counter Mode)	Clear Counter(1)
X+0043	Ch2 (For Counter Mode)	Clear Overflow
X+0044	Ch2 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0045	Ch3 (For Counter Mode)	Start(1)/Stop(0)
X+0046	Ch3 (For Counter Mode)	Clear Counter(1)
X+0047	Ch3 (For Counter Mode)	Clear Overflow
X+0048	Ch3 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0049	Ch4 (For Counter Mode)	Start(1)/Stop(0)
X+0050	Ch4 (For Counter Mode)	Clear Counter(1)
X+0051	Ch4 (For Counter Mode)	Clear Overflow
X+0052	Ch4 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0053	Ch5 (For Counter Mode)	Start(1)/Stop(0)
X+0054	Ch5 (For Counter Mode)	Clear Counter(1)
X+0055	Ch5 (For Counter Mode)	Clear Overflow
X+0056	Ch5 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0057	Ch6 (For Counter Mode)	Start(1)/Stop(0)
X+0058	Ch6 (For Counter Mode)	Clear Counter(1)
X+0059	Ch6 (For Counter Mode)	Clear Overflow
X+0060	Ch6 (For Counter Mode)	Latch Status(read)/Clear Status(Write)

X+0061	Ch7 (For Counter Mode)	Start(1)/Stop(0)
X+0062	Ch7 (For Counter Mode)	Clear Counter(1)
X+0063	Ch7 (For Counter Mode)	Clear Overflow
X+0064	Ch7 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0065	Ch8 (For Counter Mode)	Start(1)/Stop(0)
X+0066	Ch8 (For Counter Mode)	Clear Counter(1)
X+0067	Ch8 (For Counter Mode)	Clear Overflow
X+0068	Ch8 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0069	Ch9 (For Counter Mode)	Start(1)/Stop(0)
X+0070	Ch9 (For Counter Mode)	Clear Counter(1)
X+0071	Ch9 (For Counter Mode)	Clear Overflow
X+0072	Ch9 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0073	Ch10 (For Counter Mode)	Start(1)/Stop(0)
X+0074	Ch10 (For Counter Mode)	Clear Counter(1)
X+0075	Ch10 (For Counter Mode)	Clear Overflow
X+0076	Ch10 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0077	Ch11 (For Counter Mode)	Start(1)/Stop(0)
X+0078	Ch11 (For Counter Mode)	Clear Counter(1)
X+0079	Ch11 (For Counter Mode)	Clear Overflow
X+0080	Ch11 (For Counter Mode)	Latch Status(read)/Clear Status(Write)

7.4 9051MTCP 12 Digital Input/2 Counter&Freq /2 Output Module

7.4.1 Register Address (Unit:16bits)

Where X=40000 for function 03, function 06, function 16

X=30000 for function 04

Address	Channel	Item
X+0001~X+0028	For Counter	14 Channels, 32 Bits
X+0029~X+0032	For Pulse Output L level, time Unit:0.1ms	2 Channels, 32 Bits
X+0033~X+0036	For Pulse Output H level, time Unit:0.1ms	2 Channels, 32 Bits
X+0037~X+0040	Set Absolute pulse(Set to 0=Continue mode)	2 Channels, 32 Bits
X+0041~X+0044	Set DO pulse value	2 Channels, 32 Bits

7.4.2 Bit Address (Unit:1Bit)

Where X=00000 for function 01, function 05

X=10000 for function 02

Address	Channel	Item
X+0001~X+0014	For DI 14 Channels, 1 Bit	
X+0017~X+0018	For DO 2 Channels, 1 Bit	
X+0033	Ch0 (For Counter Mode)	Start(1)/Stop(0)
X+0034	Ch0 (For Counter Mode)	Clear Counter(1)
X+0035	Ch0 (For Counter Mode)	Clear Overflow
X+0036	Ch0 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0037	Ch1 (For Counter Mode)	Start(1)/Stop(0)
X+0038	Ch1 (For Counter Mode)	Clear Counter(1)
X+0039	Ch1 (For Counter Mode)	Clear Overflow
X+0040	Ch1 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0041	Ch2 (For Counter Mode)	Start(1)/Stop(0)
X+0042	Ch2 (For Counter Mode)	Clear Counter(1)
X+0043	Ch2 (For Counter Mode)	Clear Overflow
X+0044	Ch2 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0045	Ch3 (For Counter Mode)	Start(1)/Stop(0)
X+0046	Ch3 (For Counter Mode)	Clear Counter(1)
X+0047	Ch3 (For Counter Mode)	Clear Overflow
X+0048	Ch3 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0049	Ch4 (For Counter Mode)	Start(1)/Stop(0)
X+0050	Ch4 (For Counter Mode)	Clear Counter(1)
X+0051	Ch4 (For Counter Mode)	Clear Overflow
X+0052	Ch4 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0053	Ch5 (For Counter Mode)	Start(1)/Stop(0)
X+0054	Ch5 (For Counter Mode)	Clear Counter(1)
X+0055	Ch5 (For Counter Mode)	Clear Overflow
X+0056	Ch5 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0057	Ch6 (For Counter Mode)	Start(1)/Stop(0)
X+0058	Ch6 (For Counter Mode)	Clear Counter(1)
X+0059	Ch6 (For Counter Mode)	Clear Overflow
X+0060	Ch6 (For Counter Mode)	Latch Status(read)/Clear Status(Write)

X+0061	Ch7 (For Counter Mode)	Start(1)/Stop(0)
X+0062	Ch7 (For Counter Mode)	Clear Counter(1)
X+0063	Ch7 (For Counter Mode)	Clear Overflow
X+0064	Ch7 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0065	Ch8 (For Counter Mode)	Start(1)/Stop(0)
X+0066	Ch8 (For Counter Mode)	Clear Counter(1)
X+0067	Ch8 (For Counter Mode)	Clear Overflow
X+0068	Ch8 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0069	Ch9 (For Counter Mode)	Start(1)/Stop(0)
X+0070	Ch9 (For Counter Mode)	Clear Counter(1)
X+0071	Ch9 (For Counter Mode)	Clear Overflow
X+0072	Ch9 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0073	Ch10 (For Counter Mode)	Start(1)/Stop(0)
X+0074	Ch10 (For Counter Mode)	Clear Counter(1)
X+0075	Ch10 (For Counter Mode)	Clear Overflow
X+0076	Ch10 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0077	Ch11 (For Counter Mode)	Start(1)/Stop(0)
X+0078	Ch11 (For Counter Mode)	Clear Counter(1)
X+0079	Ch11 (For Counter Mode)	Clear Overflow
X+0080	Ch11 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0081	Ch12 (For Counter Mode)	Start(1)/Stop(0)
X+0082	Ch12 (For Counter Mode)	Clear Counter(1)
X+0083	Ch12 (For Counter Mode)	Clear Overflow
X+0084	Ch12 (For Counter Mode)	Latch Status(read)/Clear Status(Write)
X+0085	Ch13 (For Counter Mode)	Start(1)/Stop(0)
X+0086	Ch13 (For Counter Mode)	Clear Counter(1)
X+0087	Ch13 (For Counter Mode)	Clear Overflow
X+0088	Ch13 (For Counter Mode)	Latch Status(read)/Clear Status(Writ

7.5 9055MTCP 16 Channel 8 Digital Input/ 8 Digital output Module

7.5.1 Register Address (Unit:16bits)

Where X=40000 for function 03, function 06, function 16

X=30000 for function 04

Address	Channel	Item	Type
X+0001~X+0016	For DI Counter (32 bits/channel)	8 Channels, 32 Bits	R
X+0017~X+0032	For Pulse Output L level, time Unit:0.1ms	8 Channels, 32 Bits	R/W
X+0033~X+0048	For Pulse Output H level, time Unit:0.1ms	8 Channels, 32 Bits	R/W
X+0049~X+0064	Set DO pulse value (Set to 0=Continue mode)	8 Channels, 32 Bits	R/W
X+0065	Digital input status	8 channel,16 Bits	R
X+0066	Digital output status	8 channel,16 Bits	R/W

7.5.2 Bit Address (Unit:1Bit)

Where X=00000 for function 01, function 05

X=10000 for function 02

Address	Channel	Item	Type
X+0001~X+0008	For DI 8 Channels, 1 Bit/channel		R
X+0017~X+0024	For DO 8 Channels, 1 Bit/channel		R/W
X+0033	Ch0 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0034	Ch0 (For Counter Mode)	Clear Counter(1)	R/W
X+0035	Ch0 (For Counter Mode)	Clear Overflow	R/W
X+0036	Ch0 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0037	Ch1 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0038	Ch1 (For Counter Mode)	Clear Counter(1)	R/W
X+0039	Ch1 (For Counter Mode)	Clear Overflow	R/W
X+0040	Ch1 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0041	Ch2 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0042	Ch2 (For Counter Mode)	Clear Counter(1)	R/W
X+0043	Ch2 (For Counter Mode)	Clear Overflow	R/W
X+0044	Ch2 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0045	Ch3 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0046	Ch3 (For Counter Mode)	Clear Counter(1)	R/W
X+0047	Ch3 (For Counter Mode)	Clear Overflow	R/W
X+0048	Ch3 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0049	Ch4 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0050	Ch4 (For Counter Mode)	Clear Counter(1)	R/W
X+0051	Ch4 (For Counter Mode)	Clear Overflow	R/W
X+0052	Ch4 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0053	Ch5 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0054	Ch5 (For Counter Mode)	Clear Counter(1)	R/W
X+0055	Ch5 (For Counter Mode)	Clear Overflow	R/W
X+0056	Ch5 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0057	Ch6 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0058	Ch6 (For Counter Mode)	Clear Counter(1)	R/W
X+0059	Ch6 (For Counter Mode)	Clear Overflow	R/W
X+0060	Ch6 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W
X+0061	Ch7 (For Counter Mode)	Start(1)/Stop(0)	R/W
X+0062	Ch7 (For Counter Mode)	Clear Counter(1)	R/W
X+0063	Ch7 (For Counter Mode)	Clear Overflow	R/W
X+0064	Ch7 (For Counter Mode)	Latch Status(read)/Clear Status(Write)	R/W

7.6 9015MTCP 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 9017MTCP 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+0109~0110	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 9019MTCP 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

Table Of Contents

8.0 Data Structure of TCPDAQ

```
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
                                            //0:Alarm ON to OFF, 1:Alarm OFF to ON
                                            //Alarm value.For DIO, this value could be "0" or "1" means that "ON" or "OFF"
    u_short     byAlarmStatus;              //For high or low alarm, this is the AIO value.
    u_short     wValue;                     //For connection lost, this value is '0'.


} _AlarmInfo;

typedef struct _StreamData                  //Stream Event data structure
{
     u_char     szIP[4];                    //The IP address which send the stream data
     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;


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;


typedef struct ModuleData                   //Used for function TCP_ReadAllDataFromModule (..)
{
    u_char      Din[16];                    //Digital input data (DI#0~DI#15),avaliable for 9050/51/55 MTCP
    u_char      Dout[16];                   //Digital output data (DO#0~DO#15),avaliable for 9050/51/55/17/19 MTCP
    u_char      DiLatch[16];                //Digital input latch status (DI#0~DI#15),avaliable for 9050/51/55 MTCP
    long        DiCounter[16];              //Digital input counter value (DI#0~DI#15),avaliable for 9050/51/55 MTCP
    double      AiNormalValue[16];          //Analog Input value(AI#0~AI#15),avaliable for 9015/17/19 MTCP
    double      AiMaxValue[16];             //Analog maximum value(AI#0~AI#15),avaliable for 9015/17/19 MTCP
    double      AiMinValue[16];             //Analog minimum value(AI#0~AI#15),avaliable for 9015/17/19 MTCP
    u_char      AiHighAlarm[16];            //Analog high alarm status(AI#0~AI#15),avaliable for 9015/17/19 MTCP
    u_char      AiLowAlarm[16];             //Analog low alarm status(AI#0~AI#15),avaliable for 9015/17/19 MTCP
    u_char      AiChannelType[16];          //Analog channel Type, avaliable for 9015/17/19 MTCP
    u_char      AiBurnOut[16] ;             //Analog channel burn out status,avaliable for 9019/15 MTCP only
    double      CJCTemperature ;            //Cold junction temperature,avaliable for 9019 MTCP only
} ModuleData;
```

Table Of Contents

9.0 EX9000-MTCP Web Server

9.1 TCPDAQ(Ethernet IO) 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)

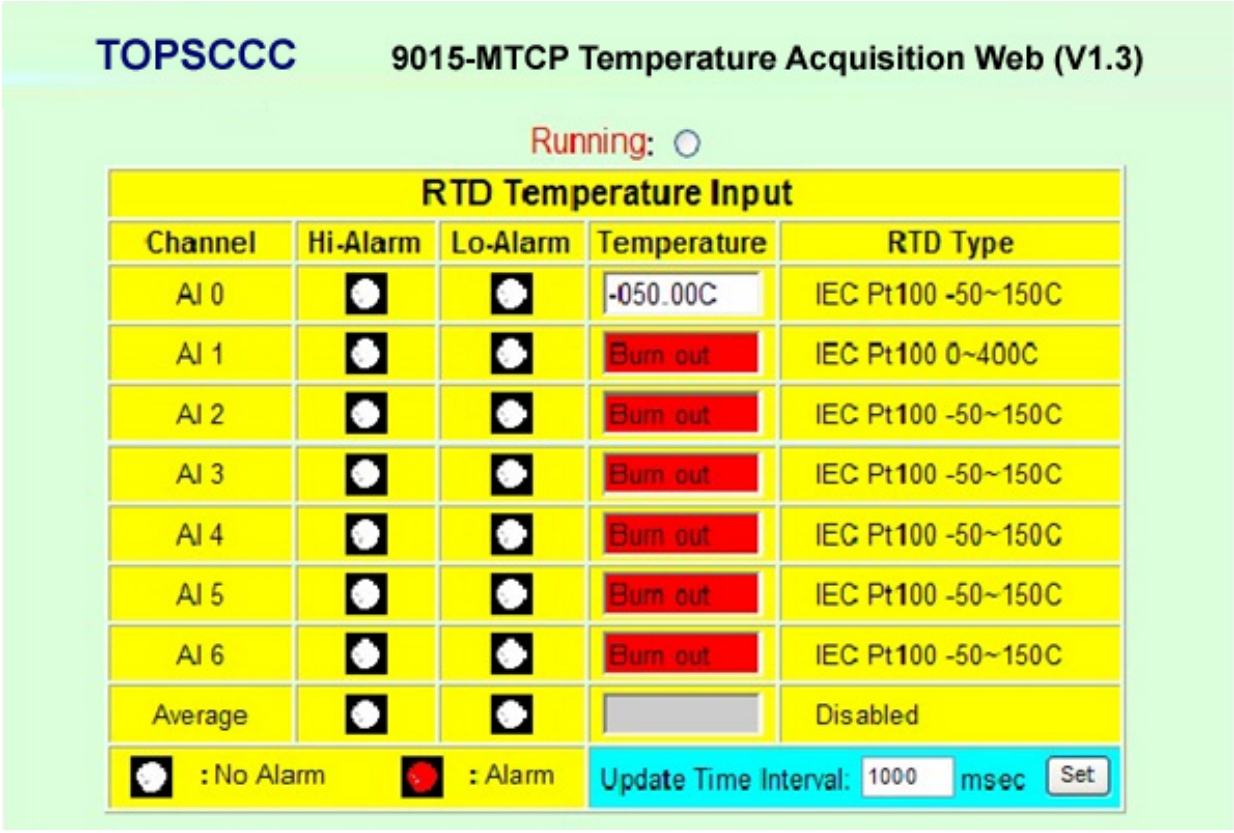
The home page will pop-up in the browser window to ask you to enter the password

Enter the correct password and click send button to verify the password. If the password is not correct, a Warning 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 9015MTCP monitoring page



Channel: Channel number of RTD input

Hi-Alarm: Analog channel High alarm status

Lo-Alarm: Analog channel low alarm status

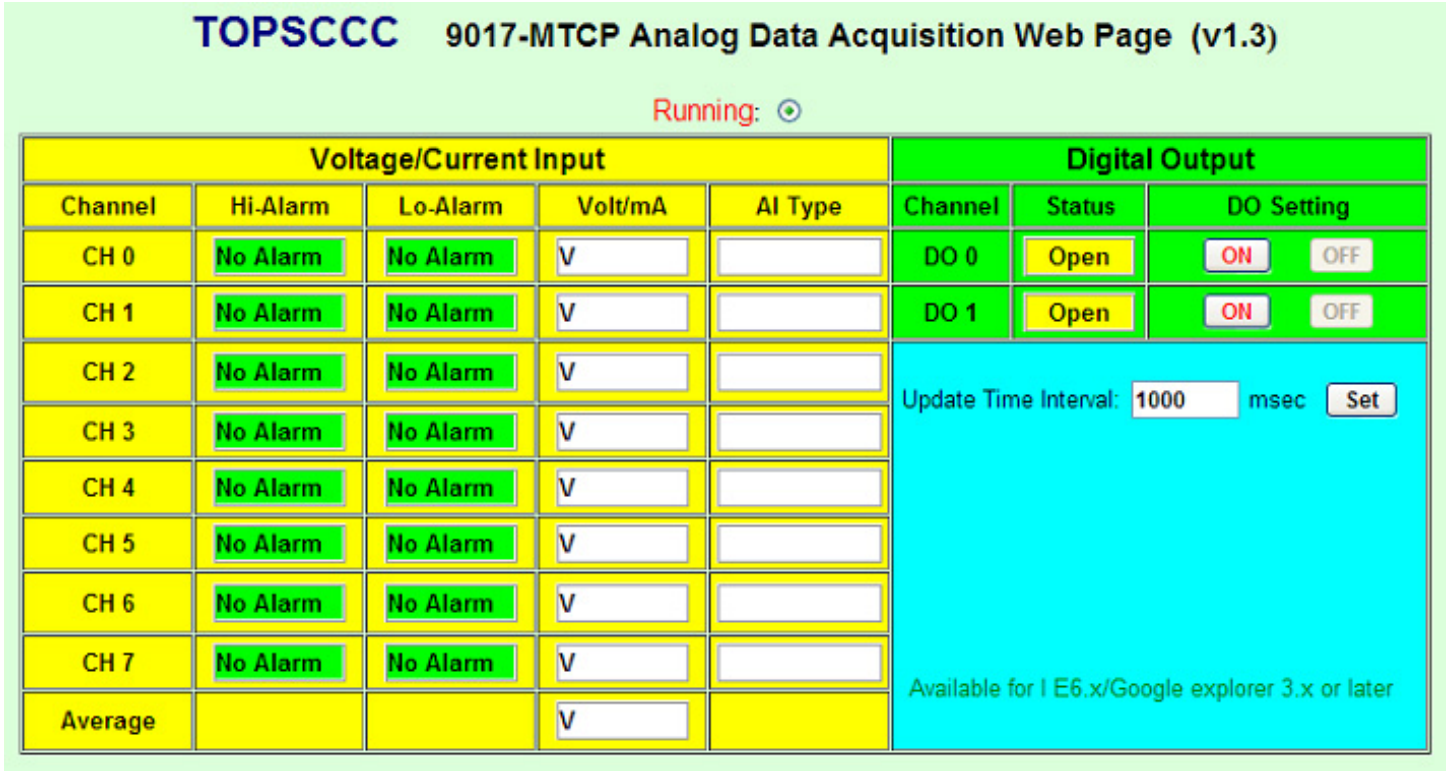
Temperature: Temperature value of RTD input channel

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 9017MTCP monitoring page



Channel: Channel number of analog input or digital output

Hi-Alarm: Analog channel High alarm status

Lo-Alarm: Analog channel low alarm status

Voltage/Current: Voltage/ Current value of analog input channel

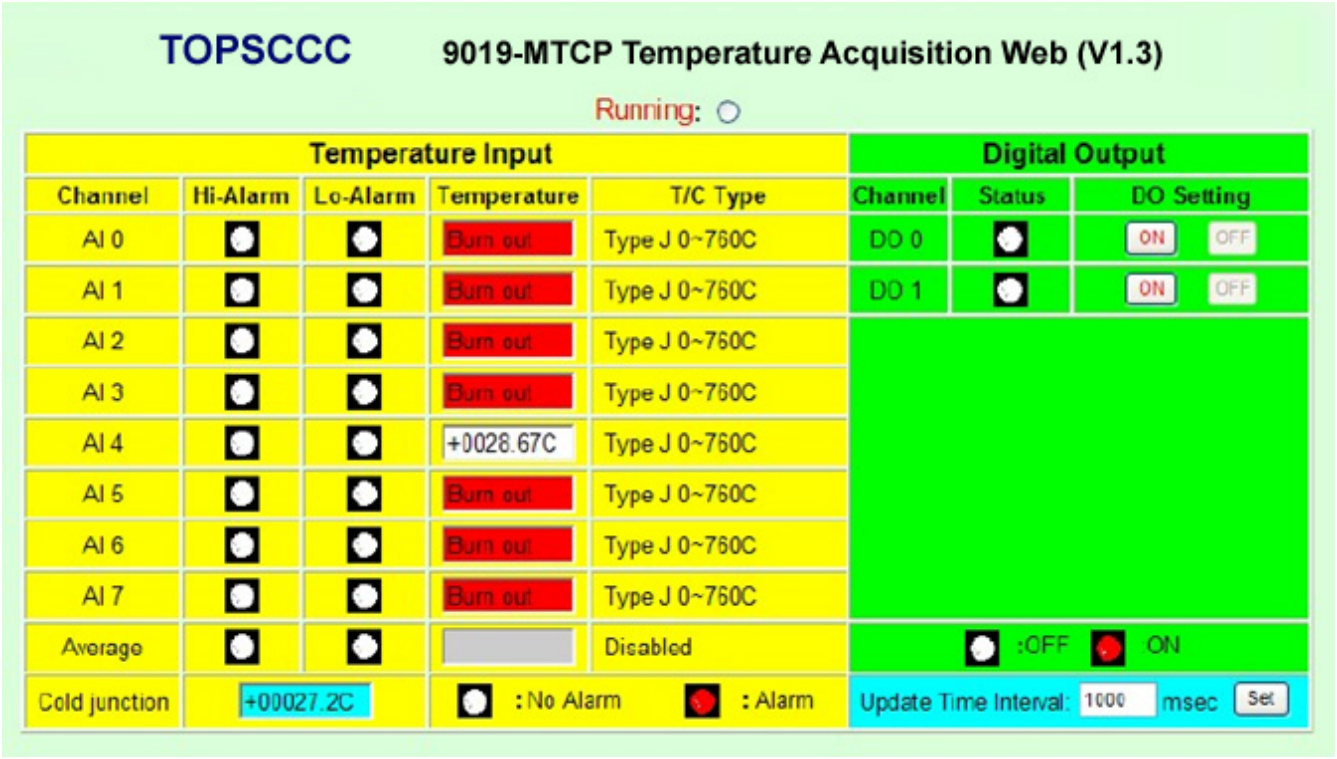
Input Range: Range of analog input channel

Status: Digital output status

DO Setting: Set digital output on or off

Time interval: I/O status update time interval

9.3.3 9019MTCP monitoring page



Channel : Channel number of analog input or digital

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 9050MTCP monitoring page

Channel : Channel number of digital input or output

Status : Current input or output status

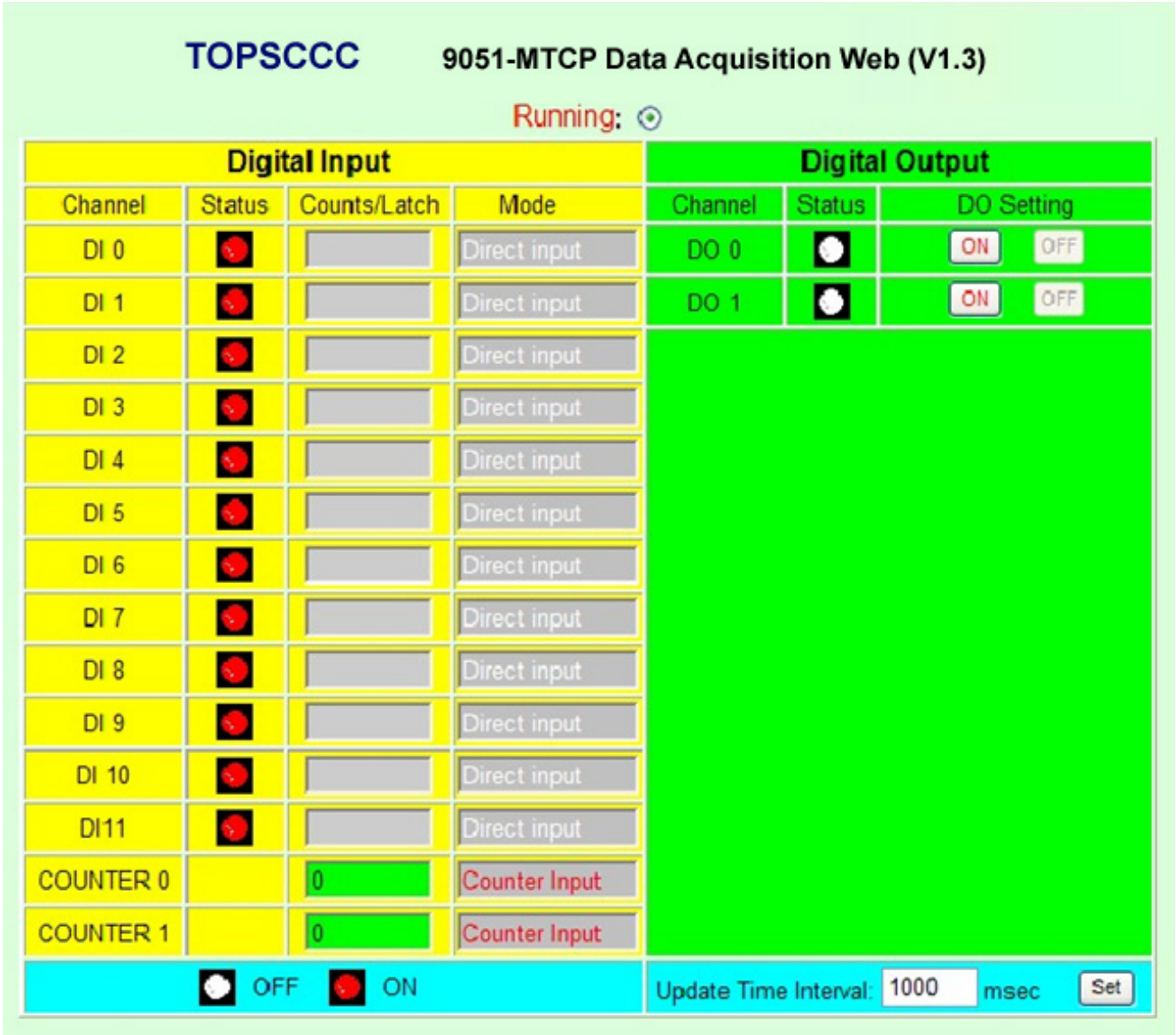
Count/Latch : Counter value or latch status of digital input which functions at "Counter" mode or "Latch" mode

Mode : Channel operating mode

DO Setting : Set digital output on or off

Time interval : I/O status update time interval

9.3.5 9051MTCP monitoring page



Channel : Channel number of digital input or output

Status : Current input or output status

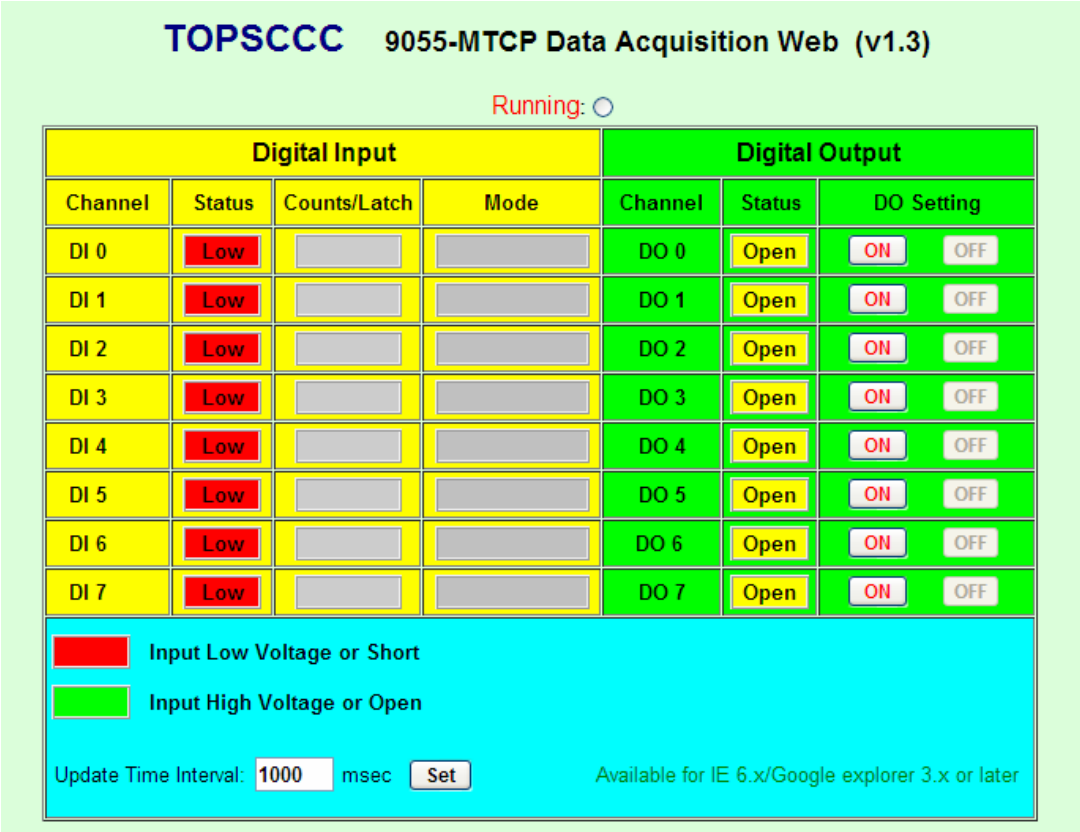
Count/Latch : Counter value or latch status of digital input which functions at "Counter" mode or "Latch" mode

Mode : Channel operating mode

DO Setting : Set digital output on or off

Time interval : I/O status update time interval

9.3.6 9055MTCP monitoring page



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