EDAM-9000(A)

Analog & DIO series

Data Acquisition Modules
User's Manual

Web site: www.inlog.com.tw

Table of Contents

Chapter 1 Ethernet-enabled DA&C I/O Modules 12
1.1 Intelligent I/O Modules 12
1.2 Mixed I/O in One Module to fit all applications 12
1.3 Industrial standard Modbus/TCP protocol supported for open connectivity 12
1.4 Software Support 12
1.5 Common technical specification of EDAM-9000(A)/9400 13
1.6 Product Warranty (1 years) 13
1.7 Dimensions 14
1.8 Summary of DIO modules 17
Chapter 2 Block diagram of DIO modules 18
2.1 EDAM-9050/9050A 18
2.1.1 Block diagram 18
2.1.2 Wire connection 18
2.2 EDAM-9050AB 19
2.2.1 Block diagram 19
2.2.2 Wire connection 19
2.3 EDAM-9051/9051A 20
2.3.1 Block diagram 20
2.3.2 Wire connection 20
2.4 EDAM-9051AB 21
2.4.1 Block diagram 21
2.4.2 Wire connection 21
2.5 EDAM-9052/9052A 22
2.5.1 Block diagram 22
2.5.2 Wire connection 22
2.6 EDAM-9052AB 23
2.6.1 Block diagram 23
2.6.2 Wire connection 23
2.7 EDAM-9053A 24
2.7.1 Block diagram 24
2.7.2 Wire connection 24
2.8 EDAM-9054A 25
2.8.1 Block diagram 25
2.8.2 Wire connection 25
2.9 EDAM-9060A 26
2.9.1 Block diagram 26
2.9.2 Wire connection 26
2.10 EDAM-9061A / 9461(PoE) / 9066A / 9466(PoE) 27
2.10.1 Block diagram 27

2.10.2 Wire connection ... 27
2.11 EDAM-9063A ... 28
2.11.1 Block diagram ... 28
2.11.2 Wire connection ... 28
Chapter 3 System Requirements ... 29
3.1 Wiring and Connections ... 29
3.2 Power supply wiring ... 29
3.3 Industrial PoE Solution ... 29
3.4 Status LED indicator for EDAM-9000A/9400 I/O modules ... 30
3.5 Status LED indicator of EDAM-9000 ... 30
3.6 I/O modules wiring ... 30
3.7 Initializing a Module ... 31
3.7.1 Factory default settings: ... 31
3.7.2 **INIT* State settings:** (only for EDAM-9000A/9400 series) ... 31
Chapter 4 Specification and wiring ... 32
4.1 EDAM-9015 7-channel RTD Input Module ... 32
4.1.1 EDAM-9015 Specification ... 32
4.1.2 Application Wiring ... 33
4.2 EDAM-9017 8-channel Analog Input with 2/DO Module ... 34
4.2.1 EDAM-9017 Specification ... 34
4.2.2 Application Wiring ... 35
4.3 EDAM-9019 8-channel T/C Input with 2/DO Module ... 36
4.3.1 EDAM-9019 Specification ... 36
4.3.2 Application Wiring ... 36
4.4 EDAM-9050(A) 12 D*I* and 6 DO channels Digital I/O Module ... 37
4.4.1 Specification ... 37
4.4.2 Application Wiring ... 37
4.5 EDAM-9050AB 12 DI and 6 DO channels DIO Module ... 38
4.5.1 Specification ... 38
4.5.2 Application Wiring ... 38
4.6 EDAM-9051(A) 12 DI , 2 DO and 2 Counter chs DIO Module ... 39
4.6.1 Specification ... 39
4.6.2 Application Wiring ... 39
4.7 EDAM-9051AB 12 DI , 2 DO and 2 Counter chs DIO Module ... 40
4.7.1 Specification ... 40
4.7.2 Application Wiring ... 40
4.8 EDAM-9052(A) 8 D*I* and 8 DO channels Digital I/O Module ... 41
4.8.1 Specification ... 41
4.8.2 Application Wiring ... 41
4.9 EDAM-9052AB 8 DI and 8 DO channels DIO Module ... 42

4.9.1 Specification 42
4.9.2 Application Wiring 42
4.10 EDAM-9053A 12 DI and 4 DO channels Digital I/O Module 44
4.10.1 Specification 44
4.10.2 Application Wiring 44
4.11 EDAM-9054A 8 differential DI and 2 DO channel Digital I/O Module 46
4.11.1 Specification 46
4.11.2 Application Wiring 46
4.12 EDAM-9054AB 6 differential DI ch. 2 DO ch. and 2 ch. counter module 47
4.12.1 Specification 47
4.12.2 Application Wiring 47
4.13 EDAM-9060A 5-channel Digital Input and 3 RELAY output Module 48
4.13.1 Specification 48
4.13.2 Application Wiring 48
4.14 EDAM-9061A / 9461 6-channel Digital Input and 6 RELAY output Module 50
4.14.1 Specification 50
4.14.2 Application Wiring 50
4.15 EDAM-9063A 7-channel Digital Input and 3 RELAY output Module 52
4.15.1 Specification 52
4.15.2 Application Wiring 52
4.16 EDAM-9066A / 9466 6-channel Digital Input and 6 RELAY output Module 54
4.16.1 Specification 54
4.16.2 Application Wiring 54
Chapter 5 EDAM-9x00(A) Utility Guide 56
5.1 System Requirement 56
5.2 Install Utility Software on Host PC 56
5.3 EDAM Ethernet I/O Utility Overview 57
5.4 Main Menu 57
5.4.1 Function Menu 57
5.4.2 Tool Bar 58
5.4.3 List Sort 58
5.5 Network Setting 59
5.5.1 Module IP 59
5.5.2 TCP/IP port: 60
5.5.3 Stream/Alarm IP 60
5.6 Add Remote Stations 61
5.7 Security Setting 61
5.8 Terminal Emulations 62
5.9 Data /Event Stream 63
5.10 Digital I/O Module Settings 65

5.10.1 Digital Test Tab 65

5.10.2 Digital Input Settings Tab 66

5.10.3 Digital Output Settings Tab 68

5.11 Analog Input Module 70

5.11.1 Analog Input Test Tab 70

5.11.2 Analog Input Setting Tab 72

5.11.3 Analog Module Calibrations 73

Chapter 6 What is TCPDAQ ActiveX Control? 74

6.1 Installing the TCPDAQ ActiveX Controls 74

6.2 Building TCPDAQ ActiveX Control with Various Tools 75

6.3 Building TCPDAQ Applications with Visual Basic 76

6.4 Building TCPDAQ Applications with Delphi 78

6.5 Building TCPDAQ Applications with Visual C++ 80

6.6 Building TCPDAQ Applications with Borland C++ Builder 83

6.7 Properties of TCPDAQ ActiveX Control 85

6.8 Methods of TCPDAQ ActiveX Control 86

6.9 Events of TCPDAQ ActiveX Control 86

6.10 Building TCPDAQ ActiveX Applications with Various Development Tools 86

Chapter 7 TCPDAQ DLL API 87

7.1 Common Functions 87

7.2 Stream/Event Functions 88

7.3 Digital I/O Functions 89

7.4 Analog I/O Functions 90

7.5 MODBUS/TCP Functions 91

7.6 Function Description 92

7.6.1 TCP_Open 92

7.6.2 TCP_Close 92

7.6.3 TCP_Connect 93

7.6.4 TCP_Disconnect 93

7.6.5 TCP_ModuleDisconnect 94

7.6.6 TCP_SendData 94

7.6.7 TCP_RecvData 94

7.6.8 TCP_SendReceiveASCcmd 95

7.6.9 UDP_Connect 95

7.6.10 UDP_Disconnect 96

7.6.11 UDP_ModuleDisconnect 96

7.6.12 UDP_SendData 96

7.6.13 UDP_RecvData 97

7.6.14 UDP_SendReceiveASCcmd 97

7.6.15 TCP_GetModuleIPinfo 98

7.6.16 TCP_GetModuleID 98
7.6.17 TCP_GetIPFromID 99
7.6.18 TCP_ScanOnLineModules 99
7.6.19 TCP_GetDLLVersion 100
7.6.20 TCP_GetModuleNo 100
7.6.21 TCP_GetLastError........ 101
7.6.22 TCP_PingIP 101
7.6.23 TCP_StartStream........ 102
7.6.24 TCP_StopStream 102
7.6.25 TCP_ReadStreamData........ 103
7.6.26 TCP_StartEvent 103
7.6.27 TCP_StopEvent........ 104
7.6.28 TCP_ReadEventData 104
7.6.29 TCP_ReadDIOMode 105
7.6.30 TCP_ReadDIO........ 105
7.6.31 TCP_ReadDISignalWidth 106
7.6.32 TCP_WriteDISignalWidth 106
7.6.33 TCP_ReadDICounter 107
7.6.34 TCP_ClearDICounter 107
7.6.35 TCP_StartDICounter........ 108
7.6.36 TCP_StopDICounter 108
7.6.37 TCP_ClearDILatch........ 109
7.6.38 TCP_ReadDILatch........ 109
7.6.39 TCP_WriteDO........ 110
7.6.40 TCP_WriteDOPulseCount........ 110
7.6.41 TCP_WriteDODelayWidth........ 111
7.6.42 TCP_ReadDODelayWidth........ 111
7.6.43 TCP_ReadAIAlarmTypes........ 112
7.6.44 TCP_WriteAIAlarmType 112
7.6.45 TCP_ReadAITypes 113
7.6.46 TCP_ReadAIValue........ 113
7.6.47 TCP_ReadAIMaxVal........ 114
7.6.48 TCP_ReadAIMinVal 114
7.6.49 TCP_ReadAIMultiplexChannel 115
7.6.50 TCP_WriteAIMultiplexChannel 115
7.6.51 TCP_ReadAIAverageChannel 116
7.6.52 TCP_WriteAIAverageChannel 116
7.6.53 TCP_ReadAIAlarmDOConnection........ 117
7.6.54 TCP_WriteAIAlarmDOConnection........ 117
7.6.55 TCP_ReadAIAlarmStatus........ 118

7.6.56 TCP_ClearAILatchAlarm 118
7.6.57 TCP_ClearAIMaxVal 119
7.6.58 TCP_ClearAIMinVal 119
7.6.59 TCP_ReadAIBurnOutStatus 120
7.6.60 TCP_ReadAIAlarmLimit 120
7.6.61 TCP_WriteAIAlarmLimit 121
7.6.62 TCP_StartAIAlarm 121
7.6.63 TCP_StopAIAlarm 122
7.6.64 TCP_WriteCJCOffset 122
7.6.65 TCP_ReadCJCOffset 123
7.6.66 TCP_ReadCJCTemperature 123
7.6.67 TCP_MODBUS_ReadCoil 124
7.6.68 TCP_MODBUS_WriteCoil 124
7.6.69 TCP_MODBUS_ReadReg 125
7.6.70 TCP_MODBUS_WriteReg 126

Chapter 8 ASCII Commands for EDAM-9000(A) Modules 127

8.1 About ASCII Commands 127
8.2 Syntax of ASCII 127
8.3 ASCII Command Set 128
8.3.1 Common commands 128
8.3.2 Digital I/O commands 128
8.3.3 DI Counter Commands 130
8.3.4 Watchdog Commands (All command for firmware version 6.000 or later) 130
8.3.5 Analog commands 131
8.4 **ASCII Command Description** 133
8.4.1 **$AAM Read Module Name** 133
8.4.2 **$AAF Read Firmware Version** 134
8.4.3 **~AADnnnnn Set timout to search DHCP** 135
8.4.4 **~AAD Read timout to search DHCP** 136
8.4.5 **$AAID Read module ID number** 137
8.4.6 **$AAIDFF Set module ID number** 138
8.4.7 **$AAMD(data) Set module description** 139
8.4.8 **$AAMD Read module description** 140
8.4.9 **^AAMAC Read MAC address** 141
8.4.10 **$AARS Reboot the module to the power-on state** 142
8.4.11 **$AAS1 Reloads the module factory default** 143
8.4.12 **~AAI Set the Soft INIT*** 144
8.4.13 **~AATNN Sets the soft INIT* timeout value.** 145
8.4.14 **$AA5 Reads the Reset Status of a module** 146
8.4.15 1.1.9 $AA6 Read DI /DO Channel Status 147

8.4.16 **@AA Read DIO status** 148
8.4.17 **$AA7 Read DI latch status** 149
8.4.18 **#AA00dd Write All Digital Output** 150
8.4.19 **#AA1n0dd Set Single Digital Output Channel** 151
8.4.20 **$AA0Mcc Read DI counter debounce time** 152
8.4.21 **$AA0MCC(data) Set DI counter debounce time** 153
8.4.22 **$AAEcn Start/stop single DI counter** 154
8.4.23 **$AACn Clear single DI counter value and overflow flag** 155
8.4.24 **#AA Read all DI counter value** 156
8.4.25 **#AAn Read single DI counter value** 157
8.4.26 **#AArn Read single DI counter value and overflow flag** 158
8.4.27 **#AA2npppppppp Write DO pulse counts** 159
8.4.28 **$AACLS Clear DI latch status** 160
8.4.29 **~AADSMN Set DI/O active state** 161
8.4.30 **~AADS Read DI/O active state** 162
8.4.31 **$AACONNDD Set a single DO channel mode** 163
8.4.32 **$AACONN Read DO mode** 164
8.4.33 **$AACINNDD Set a single DI channel mode** 165
8.4.34 **$AACINN Read a single DI channel mode** 166
8.4.35 **@AA6 Read DIO status** 167
8.4.36 **@AA6DDDD Write DO channels (0~15)** 168
8.4.37 **@AA6Onss Set/reset a single digital output channel** 169
8.4.38 **@AA6ON Read a single digital output channel** 170
8.4.39 **@AA6IN Read a single digital input channel** 171
8.4.40 **$AA9DNNLLLLHHHH Set DO low/high delay time** 172
8.4.41 **$AA9PNNLLLLHHHH Set DO pluse high/low width of channel N** 173
8.4.42 **$AA9TN Generate a output pulse of DO channel N** 174
8.4.43 **$AAYM1CPSHHHHLLLL(data) Set DIO Sync. DI match/mismatch DO Toggle Mode** 175
8.4.44 **$AAYM2CPSTTTT (data) Set DIO Sync. DI match DO latch Mode** 176
8.4.45 **$AAYM3CPSTTTT (data) Set DIO Sync. DI mismatch DO latch Mode** 177
8.4.46 **$AAYMRCS Start(Run)/Stop *DIO Synchronization* operation** 178
8.4.47 **$AAYMC Read DIO Synchronization Mode parameters** 179
8.4.48 **$AAYMS Read DO current status during DIO Synchronization operation** 180
8.4.49 **$AA9NN Read DO pulse and DO low/high delay width for channel N** 181
8.4.50 **~** Send "Host OK" to all modules via broadcast** 182
8.4.51 **~AA** Send Host OK to the specific module** 183
8.4.52 **~AA0 Read watchdog timeout status** 184
8.4.53 **~AA1 Reset host watchdog timeout status** 185
8.4.54 **~AA2 Read host communication Timeout value.** 186
8.4.55 **~AA3EVVV Set Host watchdog timeout interval** 187

8.4.56 **~AA4V Read power-on or safe DO value of module** 188
8.4.57 **~AA3PPP Set module Power-on delay time.** 189
8.4.58 **~AA5V Sets the current DO value as power-on or safe value** 190
8.4.59 **~AA3P Read module Power-on delay time.** 191
8.4.60 **#AAn Read value of analog Input channel N** 192
8.4.61 **#AA Read value of all Analog Input Channels** 193
8.4.62 **$AAAcctt Set analog input type (range)** 194
8.4.63 **$AABhh Read analog input type** 195
8.4.64 **$AA0 Span Calibration** 197
8.4.65 **$AA1 Zero Calibration** 198
8.4.66 **$AA6 Read AI channel enable/disable Status** 199
8.4.67 **$AA5mm Enable/disable all AI channels** 200
8.4.68 **#AAMH Read Maximum Value of AI channels** 201
8.4.69 **#AAMHn Read Maximum Value of AI channel N** 202
8.4.70 **#AAML Read Minimum Value of all AI channels** 203
8.4.71 **#AAMLn Read Minimum Value of AI channel N** 204
8.4.72 **$AACjAhs Set Alarm Mode** 205
8.4.73 **$AACjAh Read Alarm Mode** 206
8.4.74 **$AACjAhEs Enable/disable alarm** 207
8.4.75 **$AACjCh Clear latch alarm** 208
8.4.76 **$AACjAhCCn Set Alarm Connection** 209
8.4.77 **$AACjRhC Read Alarm Connection** 210
8.4.78 **$AACjAhU Set Alarm Limit** 211
8.4.79 **$AACjRhU Read Alarm Limit** 212
8.4.80 **$AACjS Read Alarm Status** 213
8.4.81 **$AA3 Read cold junction temperature** 214
8.4.82 **$AA9hhhhh Set CJC offset** 215
8.4.83 **$AA9 Read CJC offset** 216
Chapter 9 MODBUS/TCP Command structure **217**
9.1 Command Structure 217
9.2 ModBus Function code introductions 218
9.3 For DIO Modules: Register Address (Unit: 16bits) 218
9.4 **EDAM-9050:** 12 Digital Input/6 Digital Output Module 218
9.4.1 Holding Register Address (Unit: 16bits) 218
9.4.2 Bit Address (Unit: 1Bit) 218
9.5 **EDAM-9051:** 12 Digital Input/2 Counter/2 Output Module 219
9.5.1 Holding Register Address (Unit: 16bits) 219
9.5.2 Bit Address (Unit: 1Bit) 220
9.6 **EDAM-9052:** 8 channel digital Input /digital out Module 221
9.6.1 Register Address (Unit: 16bits) 221

9.6.2 Bit Address (Unit: 1Bit) 221
9.7 **EDAM-9015** 7-Channel RTD Input Module 222
9.7.1 Register Address (unit:16 bits) 222
9.7.2 Bit Address (unit:1 bit) 223
9.8 **EDAM-9017** 8-Channel Voltage/Current Input Module 224
9.8.1 Register Address (unit:16 bits) 224
9.8.2 Bit Address (unit:1 bit) 224
9.9 **EDAM-9019** 8-Channel T/C Input Module 225
9.9.1 Register Address (unit:16 bits) 225
9.9.2 Bit Address (unit:1 bit) 227
Chapter 10 Advanced EDAM-9000A/9400 MODBUS/TCP address Mapping 228
10.1 **All EDAM-9000A/9400** Digital Input/Output Modules 228
10.1.1 Register Address (Unit: 16bits) (Firmware Ver: 6.000 or later) 228
10.1.2 Bit Address (Unit:1Bit) 228
10.1.3 Bit Address (Unit:1Bit) 230
Chapter 11 TCPDAQ Data Structure 232
11.1 Typedef struct _AlarmInfo 232
11.2 Typedef struct _StreamData 232
11.3 Typedef struct ModuleInfo 232
11.4 Typedef struct ModuleData 232
Chapter 12 EDAM-9000 Web Server 234
12.1 What is TCPDAQ Web Server? 234
12.2 Home Page 234
12.3 Module monitoring page 235
12.3.1 EDAM-9015 monitoring page 235
12.3.2 EDAM-9017 monitoring page 236
12.3.3 EDAM-9019 monitoring page 237
12.3.4 EDAM-9050 monitoring page 238
12.3.5 EDAM-9051 monitoring page 239
12.3.6 EDAM-9052 monitoring page 240
Chapter 13 Appendix 241
13.1 **INIT* switch operation** (EDAM-9000A/9400 only) 241
13.2 **Module Status** (version 6.000 or later) 242
13.3 **Dual Watchdog Operation** (version 6.000 or later) 242
13.4 **Reset Status** (version 6.000 or later) 242
13.5 Input counter and Input latch 242
13.6 **Power-on & Safe value** (version 6.000 or later) 242
13.7 **DIO Synchronization** (Mirror Local DI to DO) 243
13.7.1 The *DIO Synchronization* is divided into three modes: 243
□ Using ASCII command: 243

□ Using Modbus-TCP commands: 244

13.7.2 DIO Synchronization –DI match/mismatch DO Toggle Mode 244

13.7.3 DIO Synchronization –DI match DO latch mode 245

13.7.4 DIO Synchronization –DI mismatch DO latch mode 247

13.8 High-to-Low and Low-to-High delay output 248

13.8.1 Low to High Delay output 248

13.8.2 High to Low Delay output 248

13.8.3 DO pulse output 249

Chapter 1 Ethernet-enabled DA&C I/O Modules

EDAM-9000(A) is based on the popular Ethernet networking standards used today in most business environments. Users can easily add EDAM-9000(A) I/O modules to existing Ethernet networks or use EDAM-9000(A) modules in new Ethernet-enabled Manufacturing networks. EDAM-9000(A) module features a 10/100 Mbps Ethernet chip and supports industrial popular Modus/TCP protocol over TCP/IP for data connection. EDAM-9000 also supports UDP protocol over Ethernet networking. With UDP/IP, EDAM-9000(A) I/O modules can actively send I/O data stream to 8 Ethernet nodes. Through Ethernet networking HMI/SCADA system and controller can access or gather real-time data from EDAM-9000(A) Ethernet enabled DA&C modules. And, these real-time data can be integrated with business system to create valuable, competitive business information immediately.

1.1 Intelligent I/O Modules

Enhancing from traditional I/O modules, EDAM-9000(A) I/O modules have pre-built intelligent mathematic functions to empower the system capacity. The Digital Input modules provide Counter, Totalizer functions; the Digital Output modules provide pulse output and *DIO Synchronization (Mirror Local DI to DO)*; the Analog Input modules provide the Max./Min./Average data calculation; the Analog Output modules provide the PID loop control function.

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

EDAM-9000(A) mixed I/O module design concept provides the most cost-effective I/O usage for application system. The most common used I/O type for single function unit are collected in ONE module. This design concept not only save I/O usage and spare modules cost but also speed up I/O relative operations. For small DA&C system or standalone control unit in a middle or large scale, EDAM-9000(A) mixed I/O design can easily fit application needs by one or two modules only. With additional embedded control modules, EDAM-9000(A) can easily create a localized, less complex, and more distributed I/O architecture.

1.3 Industrial standard Modbus/TCP protocol supported for open connectivity

EDAM-9000(A) modules support the popular industrial standard, Modbus/TCP protocol, to connect with Ethernet Controller or HMI/SCADA software built with Modbus/TCP driver. Inlog also provides OPC server for Modbus/TCP to integrate EDAM-9000(A) I/O real-time data value with OPC client enabled software. Users don't need to take care of special driver's development.

1.4 Software Support

Based on the Modbus/TCP standard, the EDAM-9000(A) firmware is a built-in Modbus/TCP server. Therefore, Inlog provides the necessary DLL drivers, and Windows Utility for users for client data for the EDAM-9000. Users can configure this DA&C system via Windows Utility; integrate with HMI software package via Modbus/TCP driver or Modbus/TCP OPC Server. Even more, you can use the DLL driver and ActiveX to develop your own applications.

1.5 Common technical specification of EDAM-9000(A)/9400

- Ethernet: 10 BASE-T IEEE 802.3 100 BASE-TX IEEE 802.3u
- Wiring: UTP, category 5 or greater
- Bus Connection: RJ45 modular jack
- Comm. Protocol: Modbus/TCP on TCP/IP and UDP
- Data Transfer Rate: Up to 100 Mbps
- Unregulated 10 to 30VDC
- Protection: Over-voltage and power reversal
- Status Indicator: Power, CPU, Communication (Link, Collide, 10/100 Mbps, Tx, Rx)
- 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.

1.6 Product Warranty (1 years)

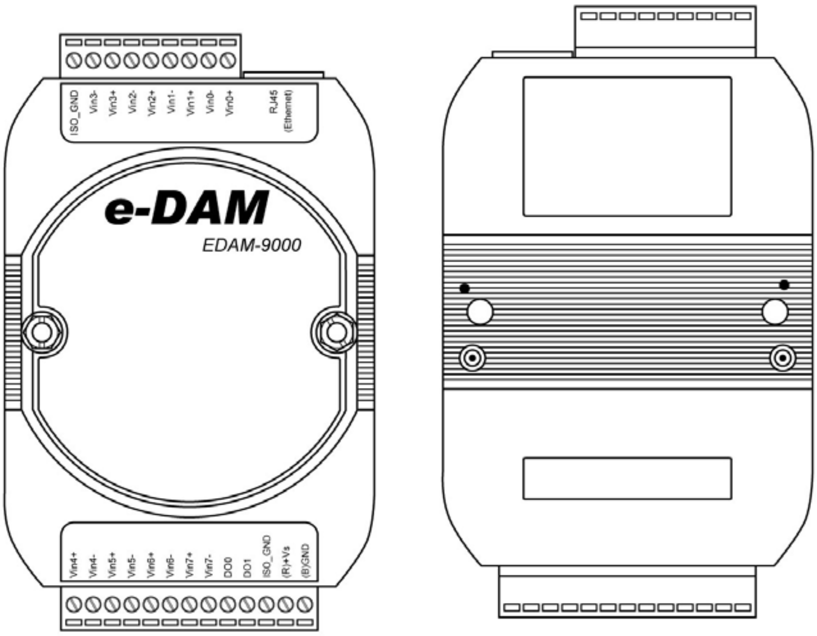
Inlog warrants to you, the original purchaser, that each of its products will be free from defects in materials and workmanship for one year from the date of purchase. This warranty does not apply to any products which have been repaired or altered by persons other than repair personnel authorized by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

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

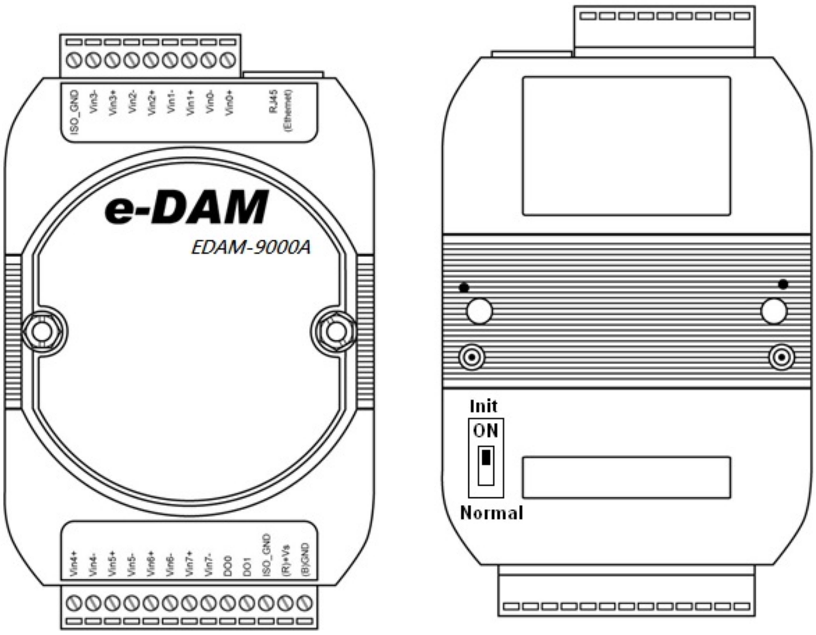
1.7 Dimensions

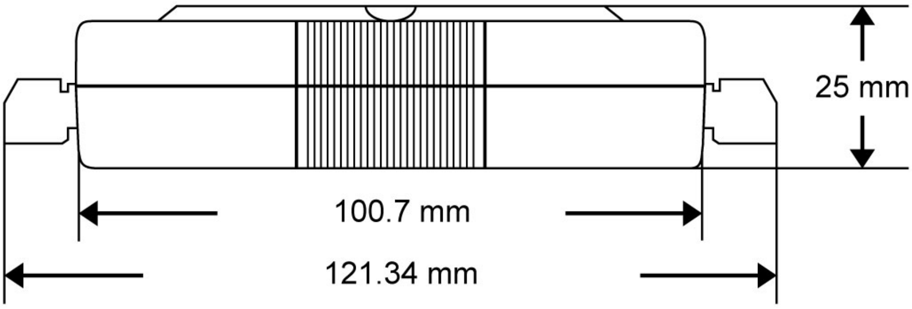
The following diagrams show the dimensions of the EDAM-9000(A)/9400 I/O module in millimeters.

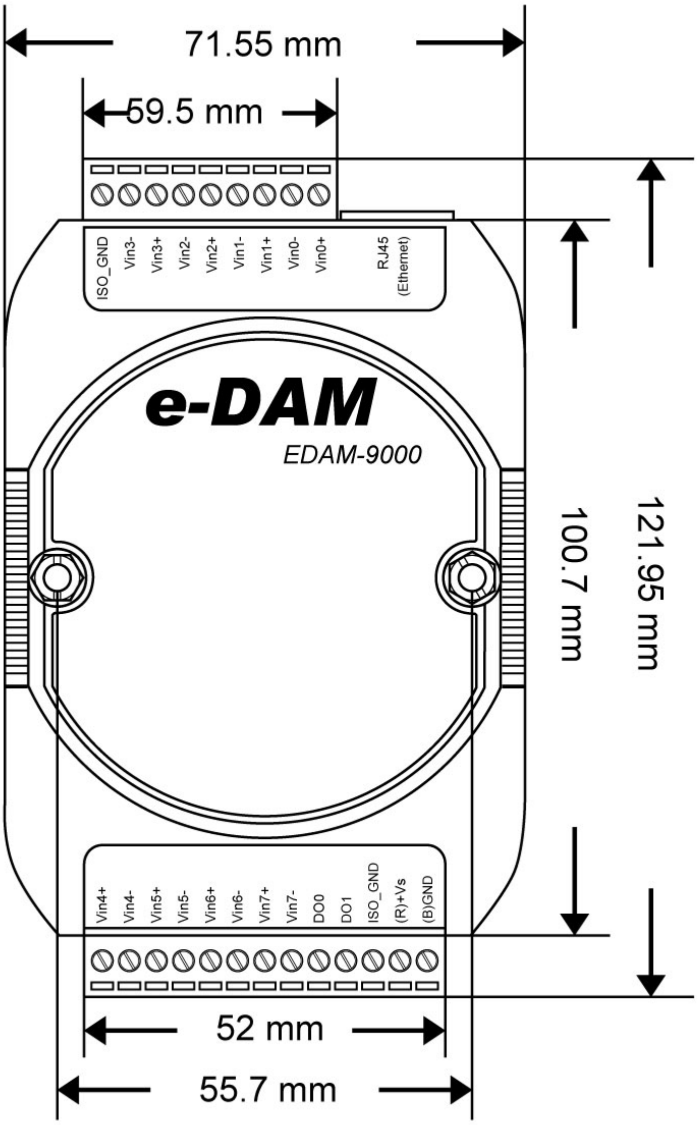
- ♦ **EDAM-9000 series**

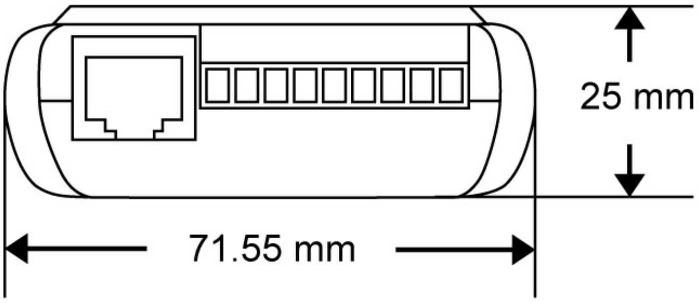


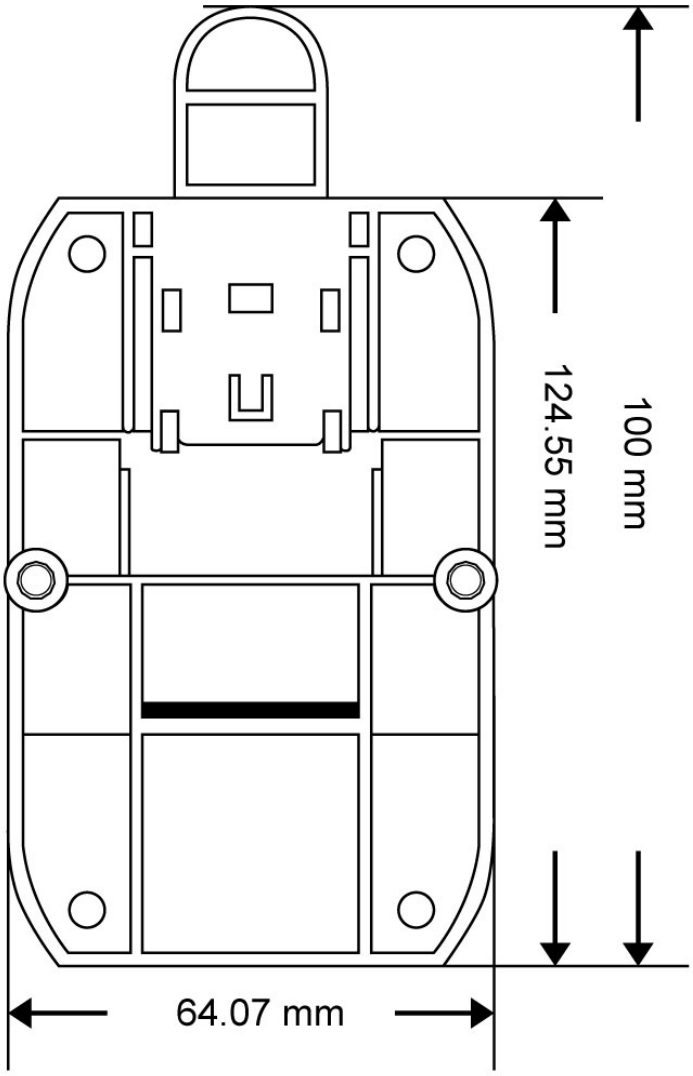
- ♦ **EDAM-9000A series**




25 mm
100.7 mm
121.34 mm


71.55 mm
59.5 mm
ISO_GND
Vin3-
Vin3+
Vin2-
Vin2+
Vin1-
Vin1+
Vin0-
Vin0+
RJ45
(Ethernet)
e-DAM
EDAM-9000
100.7 mm
121.95 mm
Vin4+
Vin4-
Vin5+
Vin5-
Vin6+
Vin6-
Vin7+
Vin7-
DO0
DO1
ISO_GND
(R)+Vs
(B)GND
52 mm
55.7 mm


25 mm
71.55 mm


124.55 mm
100 mm
64.07 mm

1.8 Summary of DIO modules

The EDAM-9000/9000A/9400A provides a series of digital input or output modules to sense the digital signal or to control the remote devices.

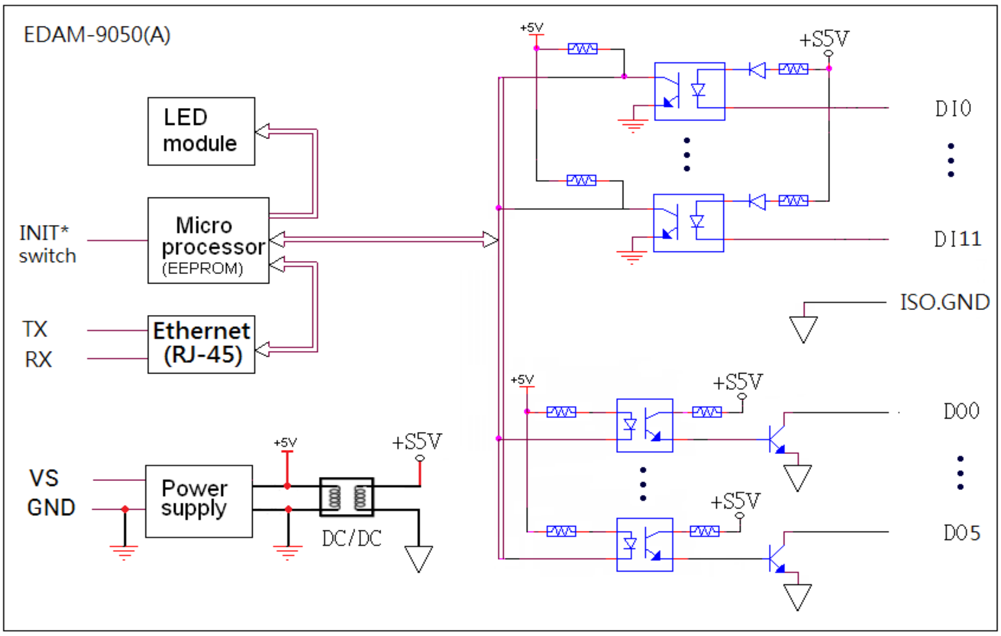
DC Input and DC Output modules				
Module	**DI ch.**	**Input type**	**DO ch.**	**Output type**
9050 9050A	12	Isolated single ended with Dry Contact *(common source)*.	6	Isolation with Open collector (NPN)
9050AB	12	Isolated single ended with Wet Contact (*common source or ground*)	6	Isolation with Open collector (NPN)
9051 9051A	12	Isolated single ended with Dry Contact *(common source)*	2	Isolation with Open collector (NPN)
	2	Iso. with differential counter input		
9051AB	12	Isolated single ended with Wet Contact *(common source)*	2	Isolation with Open collector (NPN)
	2	Iso. with differential counter input		
9052 9052A	8	Isolated single ended with Dry Contact *(common ground)*.	8	Isolated with open drain (P-MOSFET), (3A/per channel)
9052AB	8	Isolated single ended with Wet Contact*(common source or ground)*	8	Isolated with open drain (P-MOSFET), (3A/per channel)
9053A	12	Isolated single ended with Dry/Wet Contact*(common source or ground)*	4	Isolation with Open collector (NPN)
9054A	8	Isolated differential input channels *(sink/source)*	2	Isolated with open drain (P-MOSFET), (3A/per channel)
9054AB	6	Isolated differential input channels (sink/source)	2	Isolated with open drain (P-MOSFET), (3A/per channel)
	2	Isolated with differential counter input		
9060A	5	Isolated single ended with common source or ground.	3	Relay output 0.6A@125VAC/2A@30VDC. RL1 Form A, RL2,RL3 Form C
9061A 9461	6	Isolated single ended with Dry Contact (common source).	6	Relay output 0.6A@125VAC/2A@30VDC (Form A)
9063A	7	Isolated single ended with common source or ground.	3	Relay output 5A@250VAC/5A@30VDC. (Form A)
9066A 9466	6	Isolated single ended with Dry Contact (common source).	6	5A@250VAC/5A@30VDC. (Form A)

Notice: (EDAM-94xx =EDAM-90xxA with built-in Power-over-Ethernet (**PoE**) function)

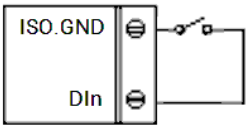
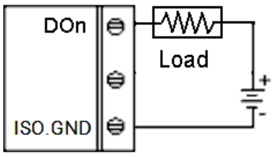
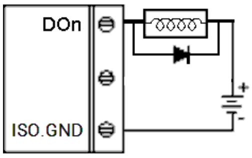
Chapter 2 Block diagram of DIO modules

2.1 EDAM-9050/9050A

2.1.1 Block diagram

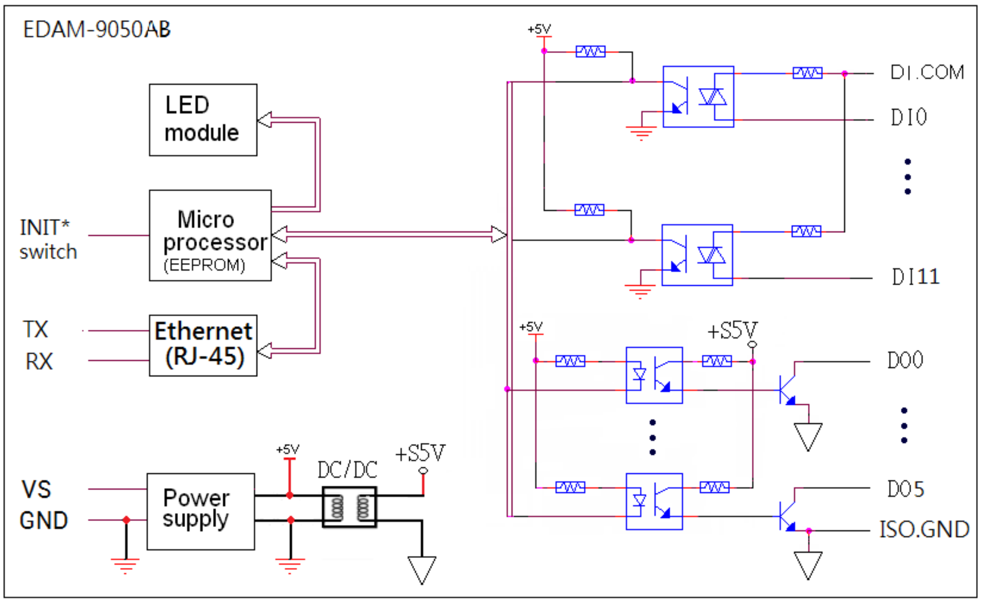


2.1.2 Wire connection

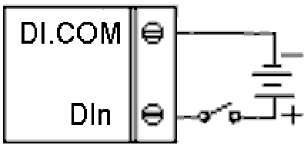
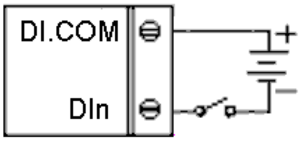
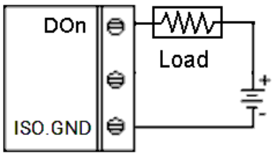
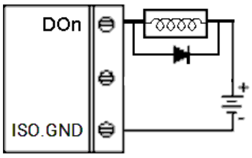
Dry Contact	
	
Open collector output	
	

2.2 EDAM-9050AB

2.2.1 Block diagram

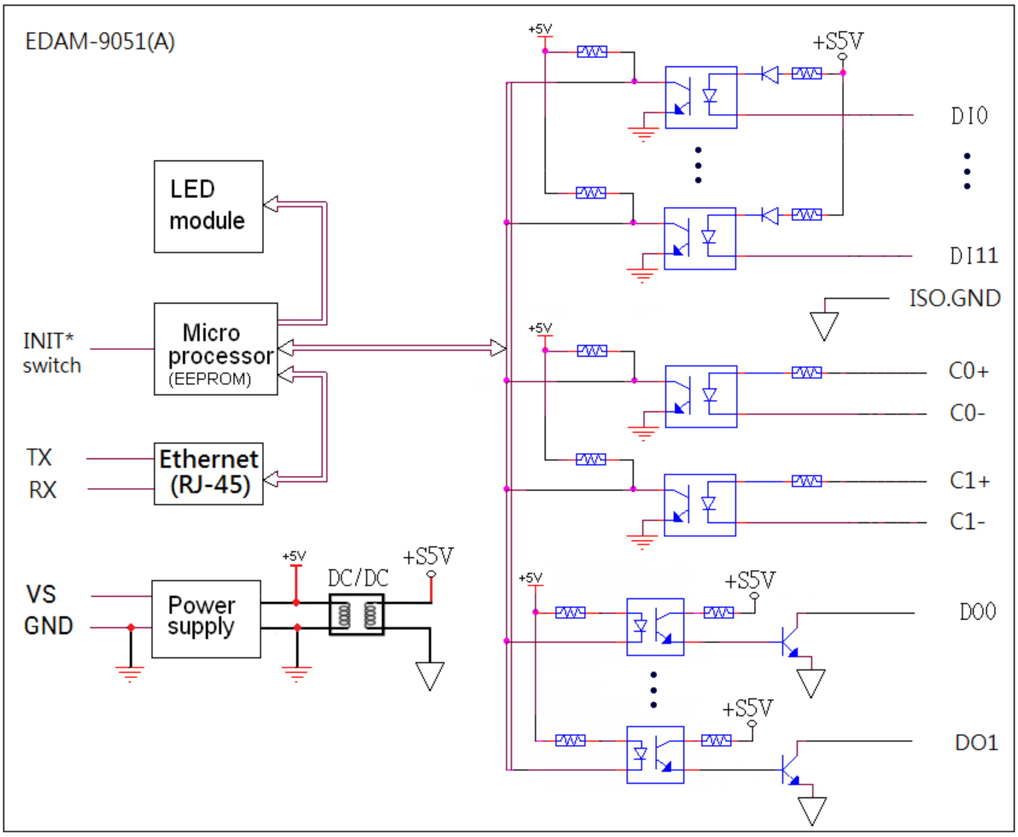


2.2.2 Wire connection

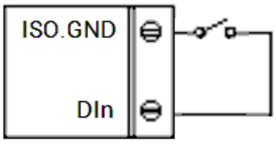
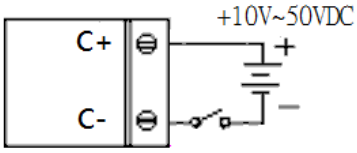
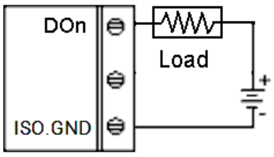
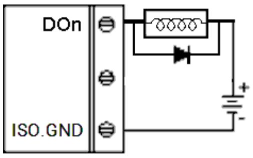
Wet Contact	
	
Open collector output	
	

2.3 EDAM-9051/9051A

2.3.1 Block diagram

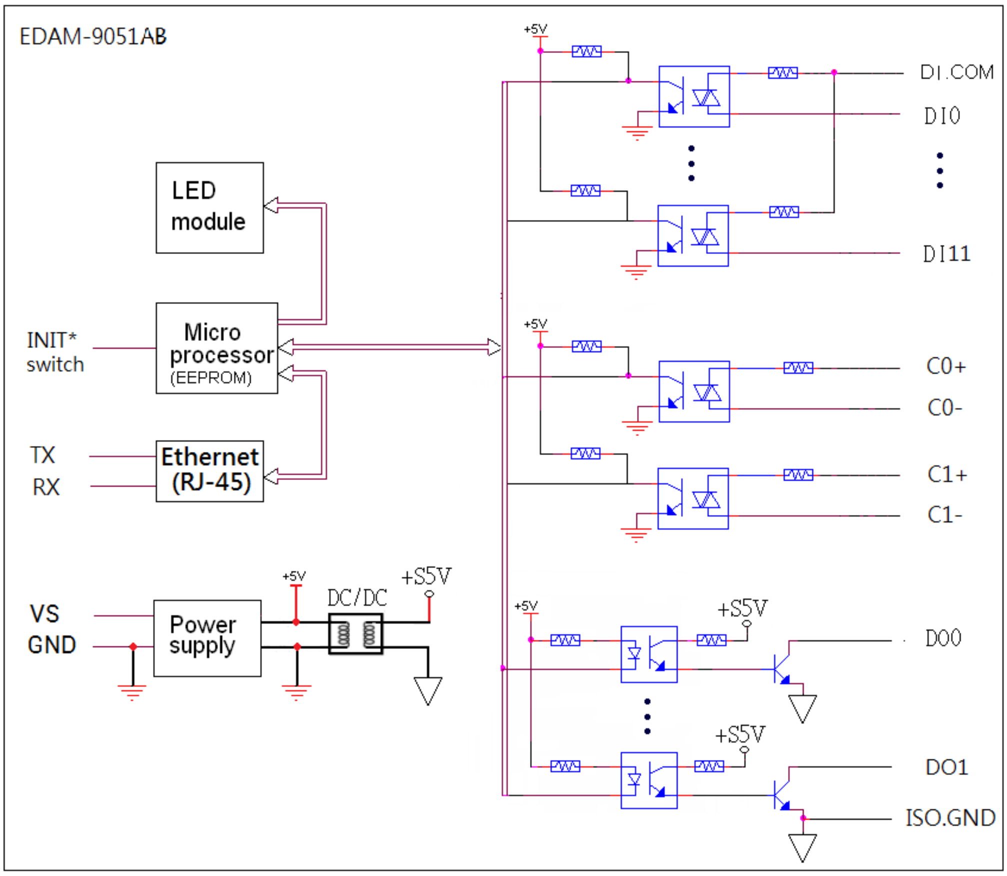


2.3.2 Wire connection

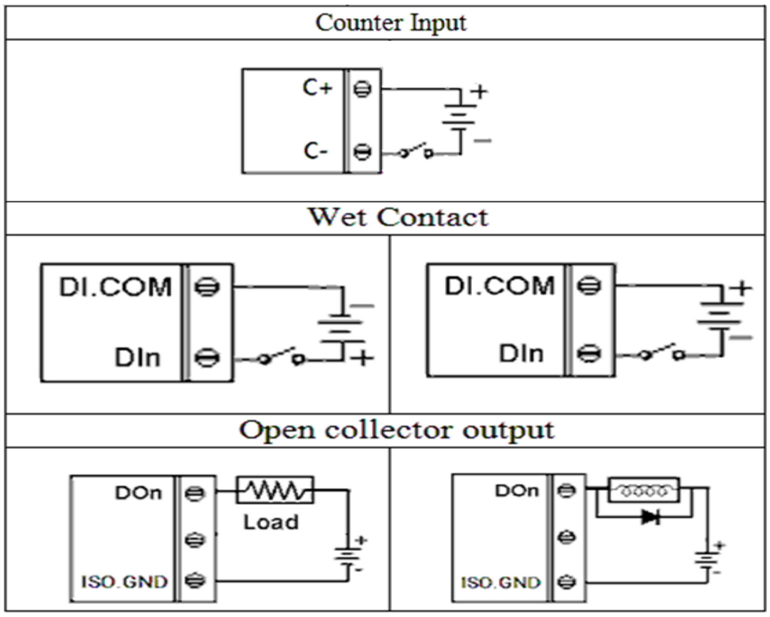
Dry Contact	Counter Input
	
Open collector output	
	

2.4 EDAM-9051AB

2.4.1 Block diagram

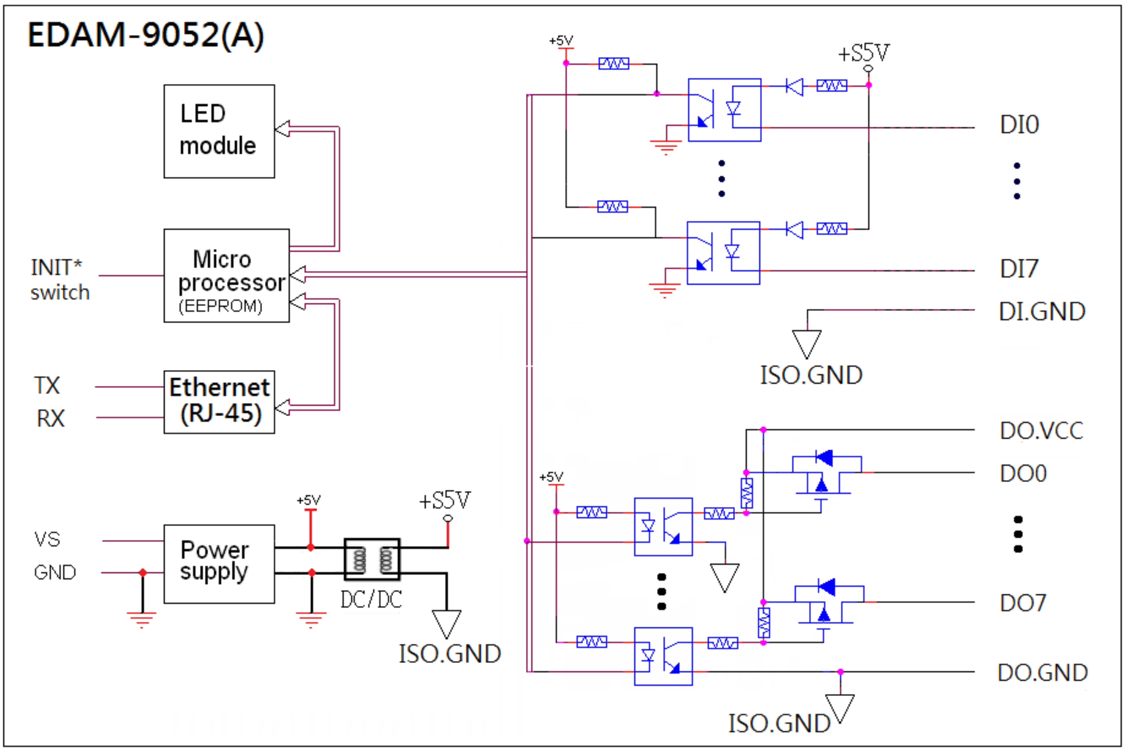


2.4.2 Wire connection

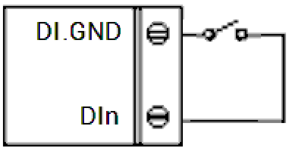
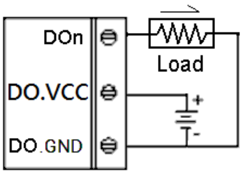
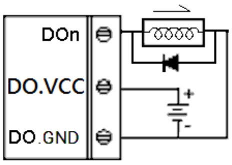


2.5 EDAM-9052/9052A

2.5.1 Block diagram

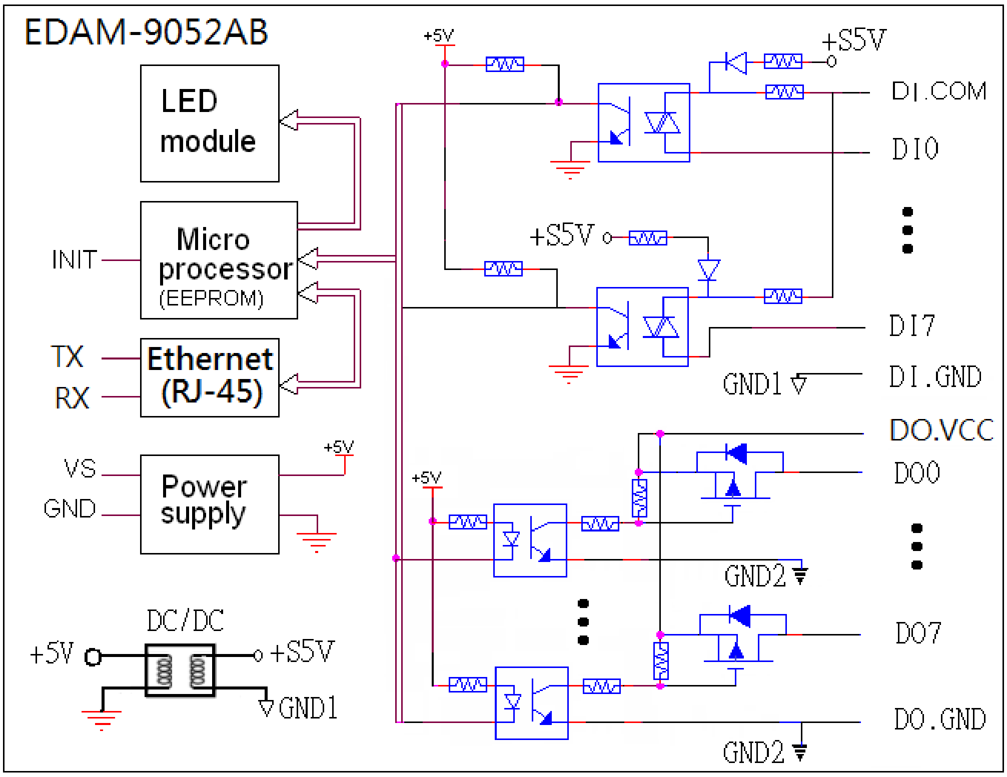


2.5.2 Wire connection

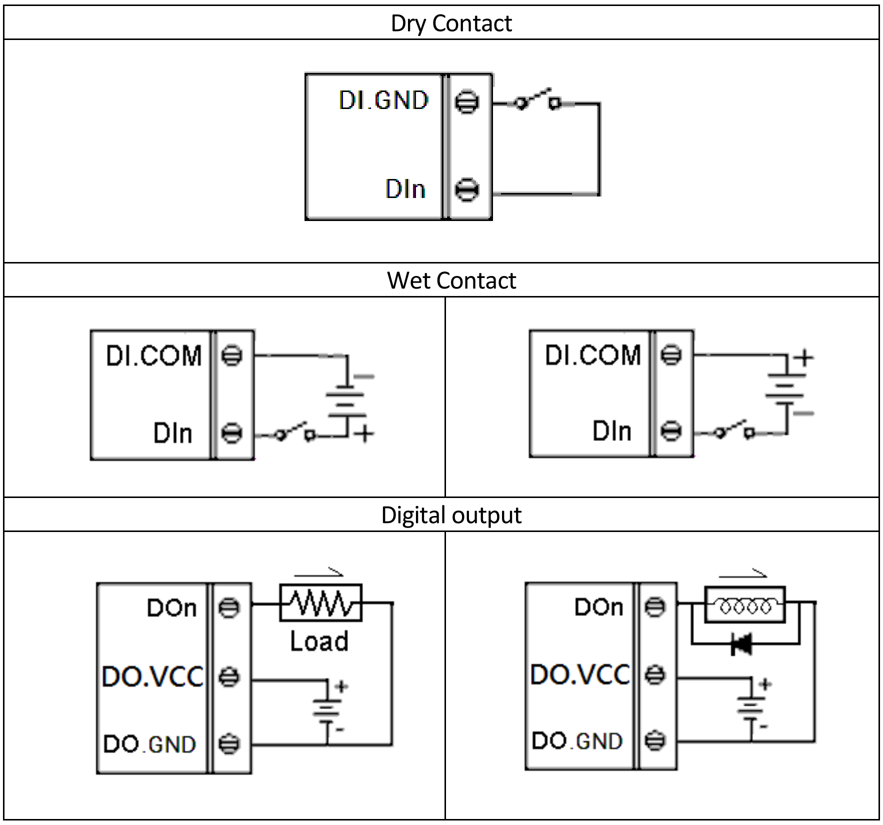
Dry Contact	
	
Open collector output	
	

2.6 EDAM-9052AB

2.6.1 Block diagram

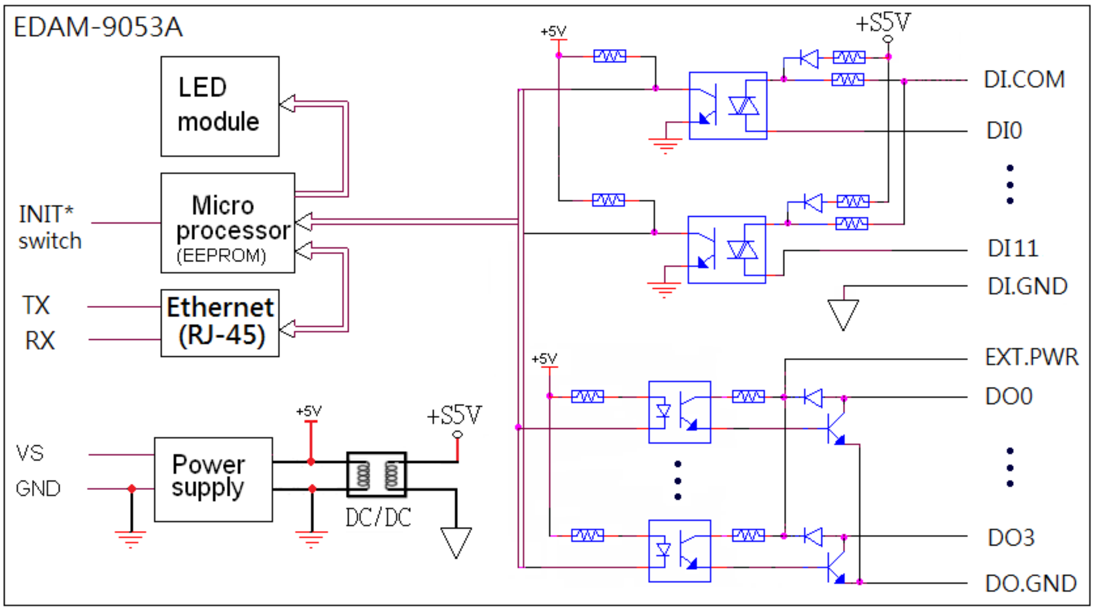


2.6.2 Wire connection

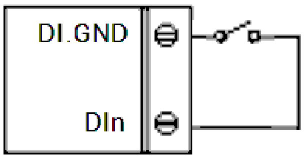
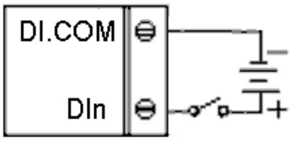
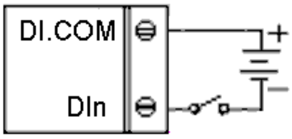
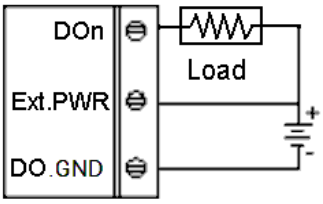
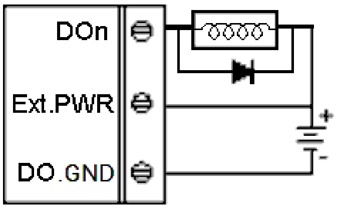


2.7 EDAM-9053A

2.7.1 Block diagram

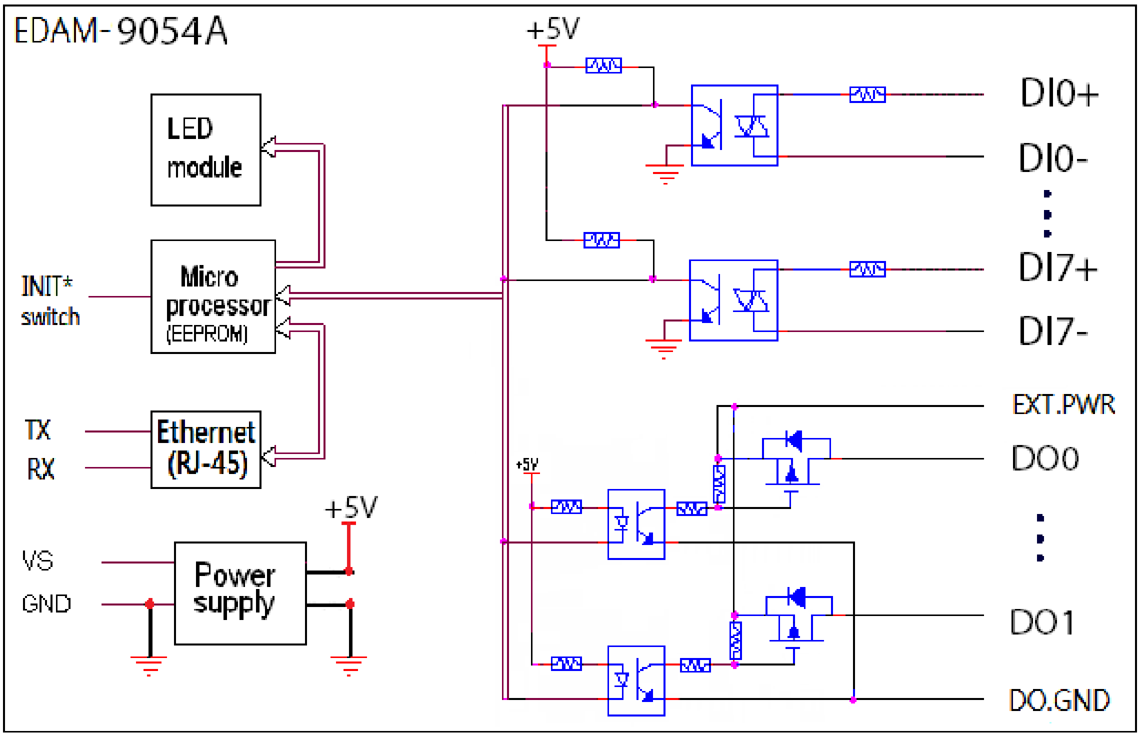


2.7.2 Wire connection

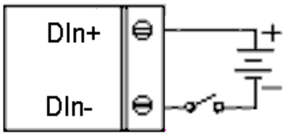
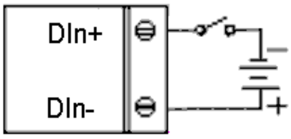
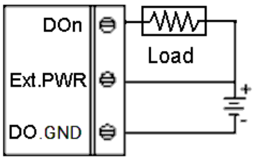
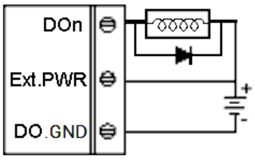
<table>
<tr><th colspan="2">Dry Contact</th></tr>
<tr><td colspan="2"> </td></tr>
<tr><th colspan="2">Wet Contact</th></tr>
<tr><td> </td><td> </td></tr>
<tr><th colspan="2">Open collector output</th></tr>
<tr><td> </td><td> </td></tr>
</table>

2.8 EDAM-9054A

2.8.1 Block diagram

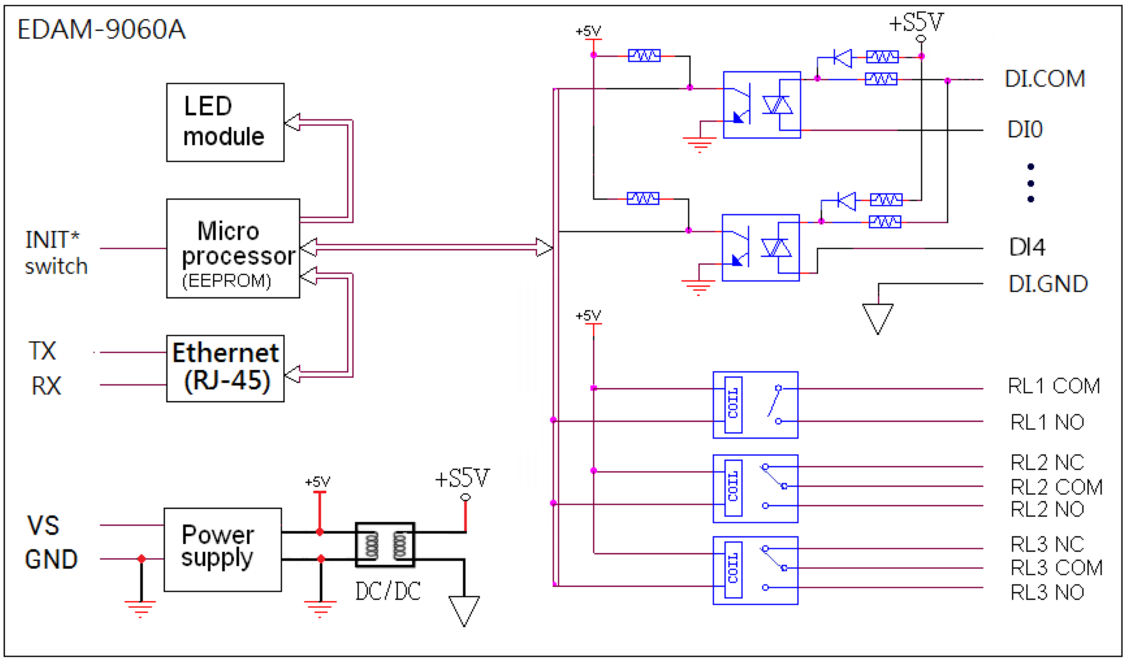


2.8.2 Wire connection

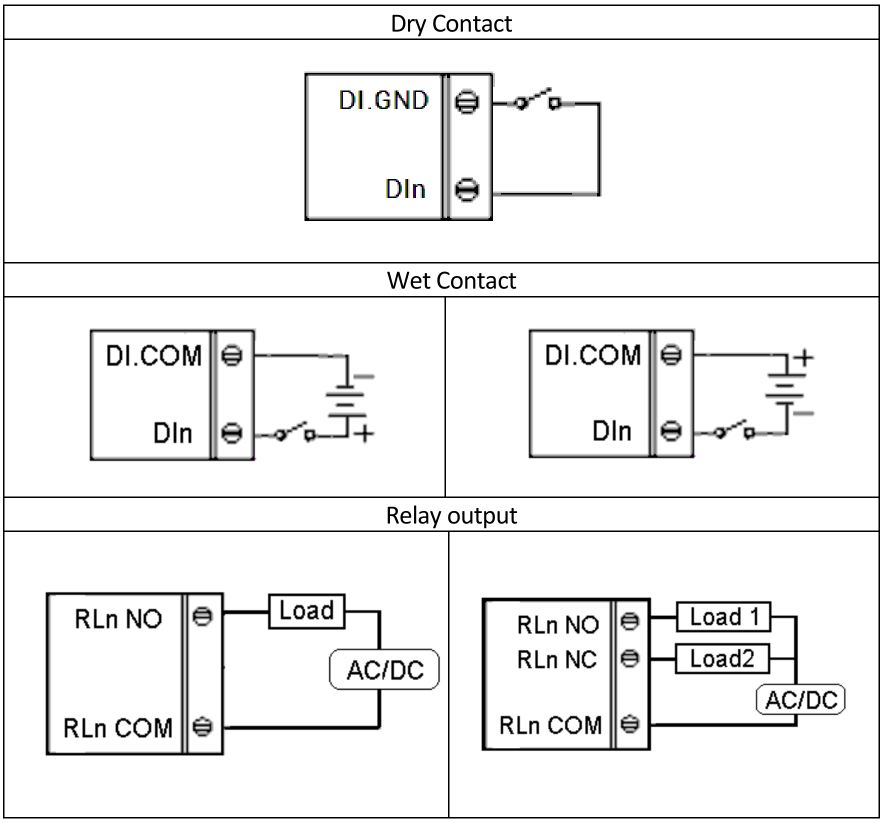
Wet Contact	
	
Digital output	
	

2.9 EDAM-9060A

2.9.1 Block diagram

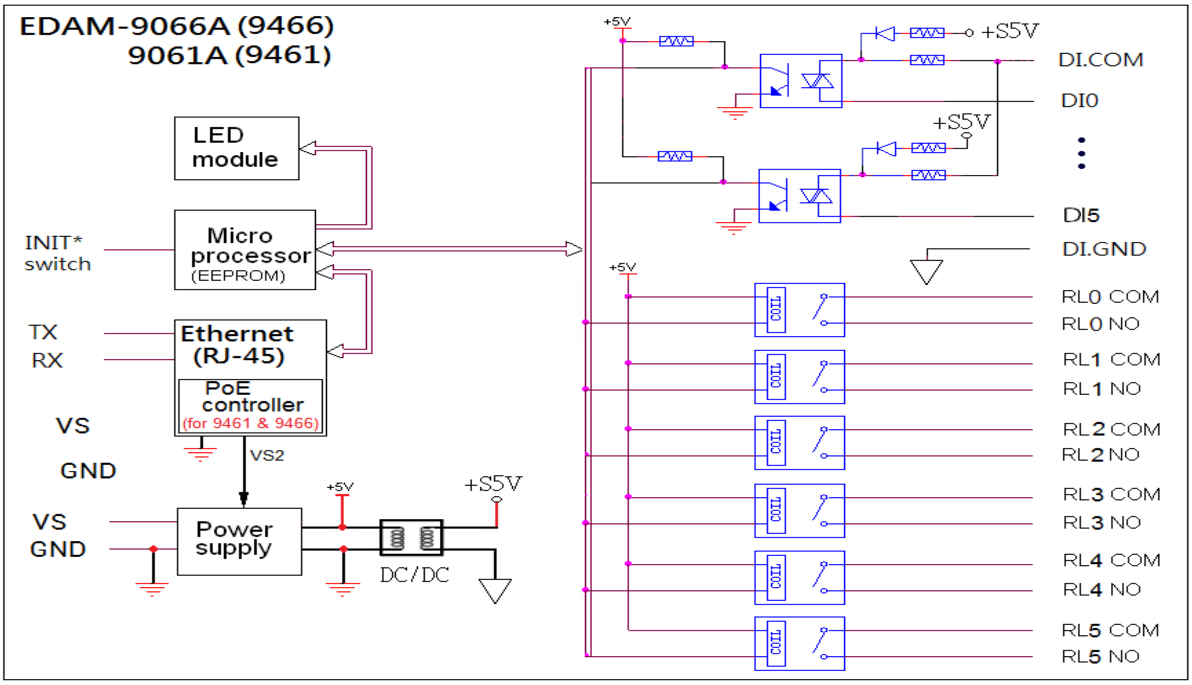


2.9.2 Wire connection

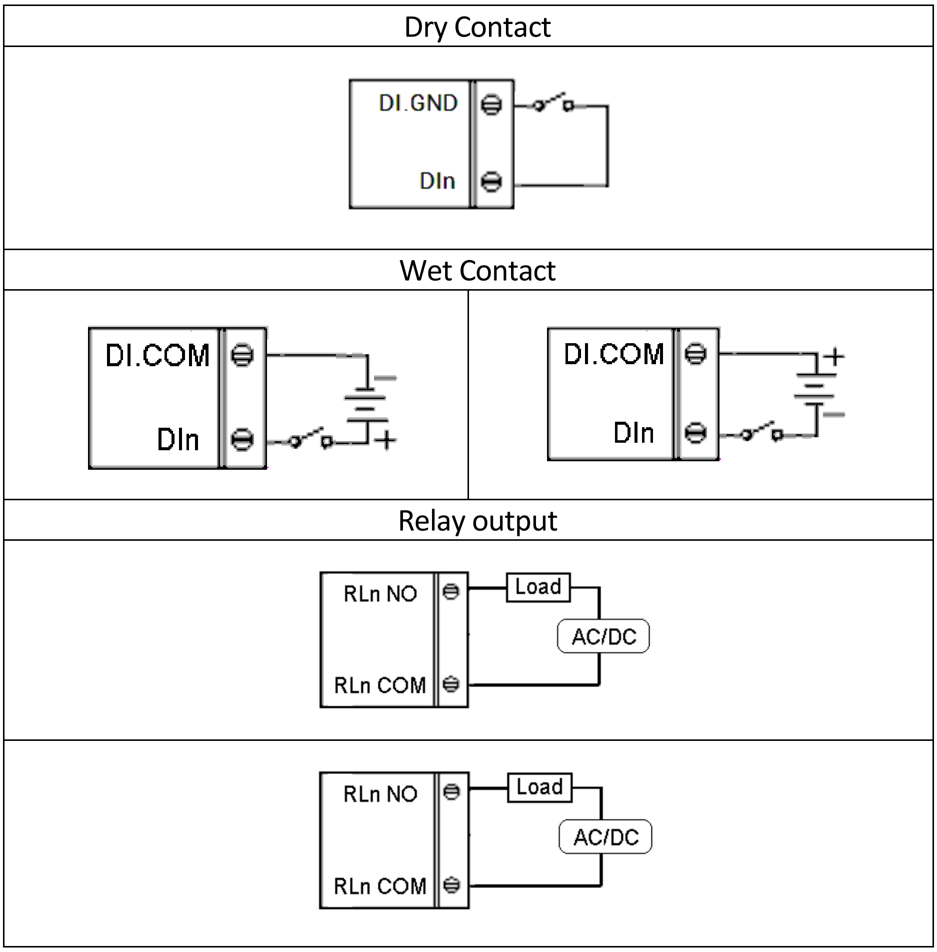


2.10 EDAM-9061A / 9461(PoE) / 9066A / 9466(PoE)

2.10.1 Block diagram

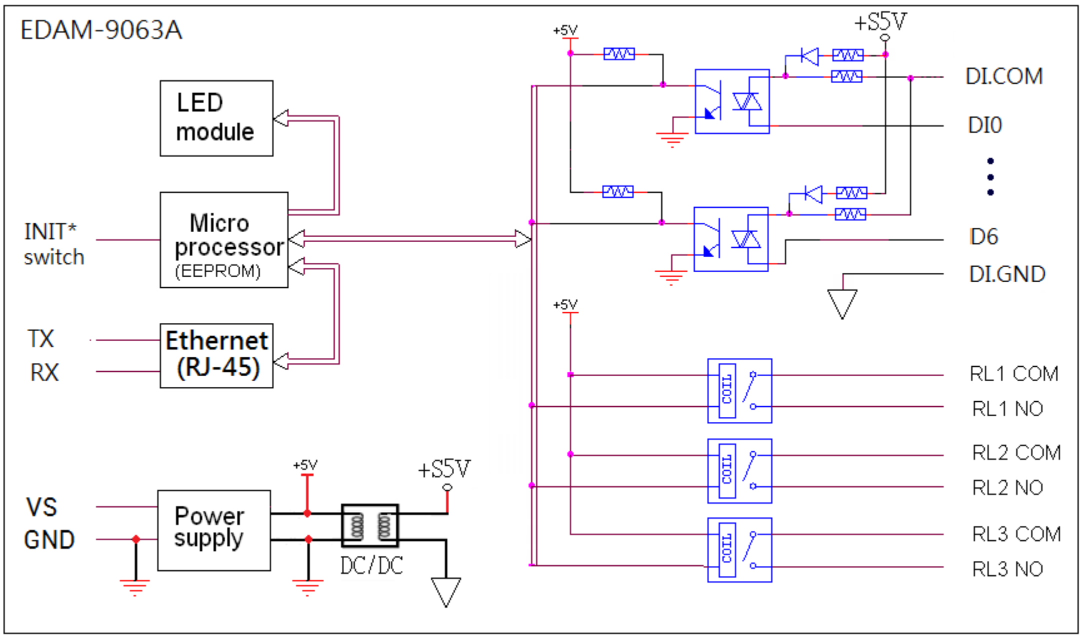


2.10.2 Wire connection

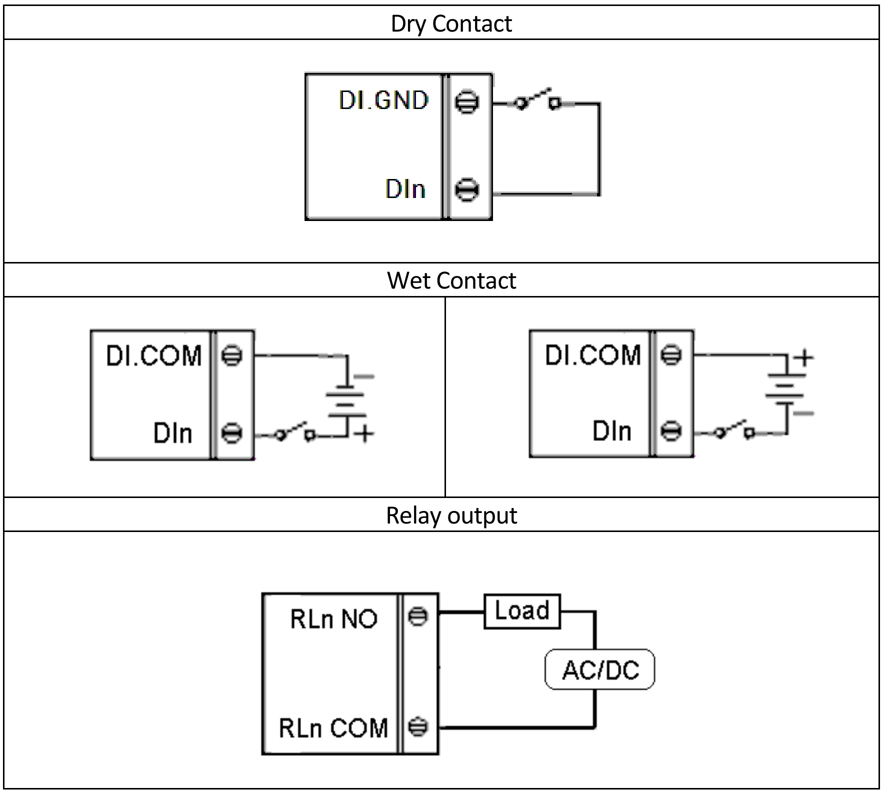


2.11 EDAM-9063A

2.11.1 Block diagram



2.11.2 Wire connection



Chapter 3 System Requirements

- IBM PC compatible computer with 486 CPU (Pentium is recommended)
- Microsoft 95/98/2000/NT 4.0 (SP3 or SP4)/XP/Win 7,8 or higher versions
- At least 32 MB RAM
- 20 MB of hard disk space available
- VGA color monitor
- 2x or higher speed CD-ROM
- Mouse or other pointing devices
- 10 or 100 Mbps Ethernet Card
- 10 or 100 Mbps Ethernet Hub (at least 2 ports)
- Two Ethernet Cable with RJ-45 connector
- Power supply for EDAM-9000 (+10 to +30 V unregulated), (EDAM-9400 option)

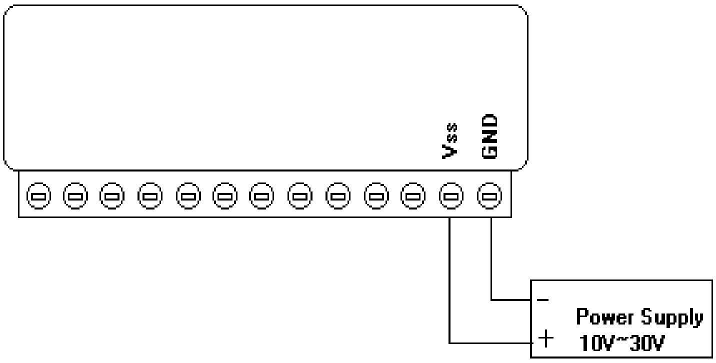
3.1 Wiring and Connections

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

3.2 Power supply wiring

Although the EDAM-9000/TCP systems are designed for a standard industrial unregulated 24 V DC power supply, they accept any power unit that supplies within the range of +10 to +30 VDC. The power supply ripple must be limited to 200 mV peak-to-peak, and the immediate ripple voltage should be maintained between +10 and +30 VDC. Screw terminals +Vs and GND are for power supply wiring.

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



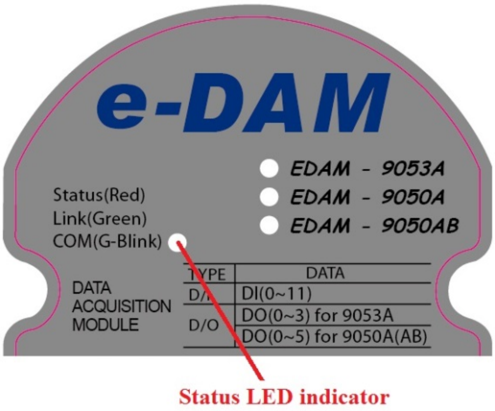
3.3 Industrial PoE Solution

(EDAM-94xx =EDAM-90xxA with built-in Power-over-Ethernet)

The EDAM-9400 features true IEEE 802.3af-compliant (classification, Class 1) PoE using both Ethernet pairs (Category 5 Ethernet cable). The EDAM-9400 can also receive power from auxiliary power sources such as DC adapters and external battery packs. The PoE switch is the ideal power source when using the EDAM-9400 module. The PoE switch automatically detects whether the connected devices are PoE-enabled or not, which ensures that the PoE switch will function in conjunction with both PoE and non-PoE devices simultaneously. There are two ways for EDAM-9400 series devices to obtain power. The first is through the Ethernet via a PoE switch; the second one is the usual method through wiring from an external power source.

3.4 Status LED indicator for EDAM-9000A/9400 I/O modules

There are four flash types of the Power-LED(Status LED indicator) on the front panel of EDAM-9000A/9400 series.



No.	Color	LED Status	Definition
1	RED+Green	On	(Status) EDAM-9000A module is initializing.
2	RED	On	(Status) EDAM-9000A module is running.
3	RED	Blinking	(Status) Host WtachDog timeout.
4	Green	On	(LINK) On whenever the Ethernet is connected
5	Green	Blinking	(COM) Blinks whenever EDAM-9000A module is transmitting or receiving data via Ethernet.

3.5 Status LED indicator of EDAM-9000

There are two LEDs on the EDAM-9000 I/O Modules front panel. Each LEDs built with two indicators to represent the EDAM-9000 system status,
as explained below:

- ♦ Full : Red indicator. This LED is blanking when EDAM-9000 module is running.
- ♦ Link : Red indicator. This LED is normal on whenever the EDAM-9000 module's Ethernet wiring is connected

3.6 I/O modules wiring

The system uses a plug-in screw terminal block for the interface between I/O modules and field devices. The following information must be considered when connecting electrical devices to I/O modules. The terminal block accepts wires from 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.

3.7 Initializing a Module

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

3.7.1 Factory default settings:

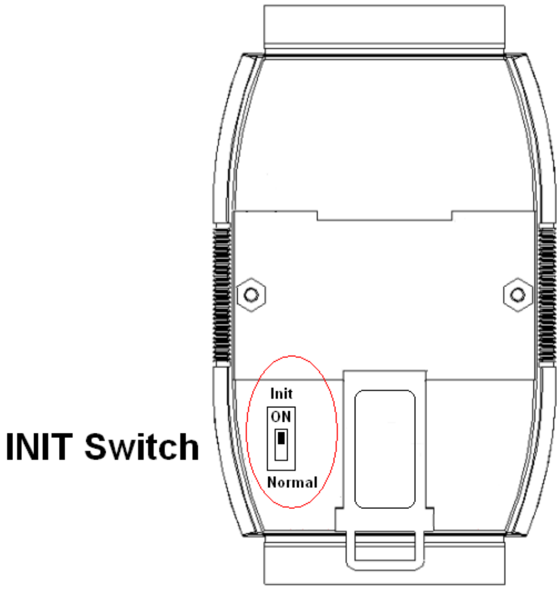
♦ IP Address :	10.0.0.1
♦ Subnet Mask:	255.0.0.0
♦ Gateway:	10.0.0.1
♦ DHCP:	Disabled
♦ Web Server:	Disabled
♦ Module ID:	00
♦ Password:	00000000

3.7.2 **INIT* State settings:** (only for EDAM-9000A/9400 series)

♦ The EDAM-9000A I/O modules must be set at "***INIT" State*** when you want to change the default settings, such as the *IP address, DIO mode status* etc. All EDAM-9000A I/O modules have an special slide-switch as **INIT-SWITCH *(ref. 14.1)***. The module will be in *Default State* if the INIT switch set to "INIT" mode when power ON. Under this state, the default configuration is set as following :

♦ IP Address :	10.0.0.1
♦ Subnet Mask:	255.0.0.0
♦ Gateway:	10.0.0.1
♦ DHCP:	Disabled
♦ Web Server:	Disabled
♦ Module ID:	00
♦ Password:	00000000

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



(EDAM-9000A only)

Chapter 4 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 3,000 VDC .

4.1 EDAM-9015 7-channel RTD Input Module

The EDAM-9015 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 (oC). In order to satisfy various temperature requirements in one module, each analog channel is allowed to configure an individual range for several applications.

4.1.1 EDAM-9015 Specification

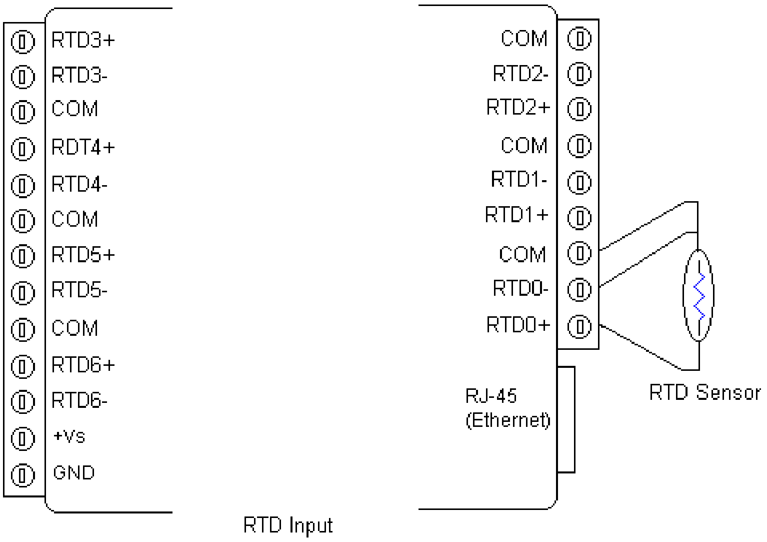
Analog Input:

- ♦ 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: 2000V
- ♦ 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

4.1.2 Application Wiring



4.2 EDAM-9017 8-channel Analog Input with 2/DO Module

The EDAM-9017 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, EDAM-9017 has designed with 8 analog inputs and 2 digital outputs. Each analog channel is allowed to configure an individual range for variety of applications.

4.2.1 EDAM-9017 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

4.2.2 Application Wiring

EDAM-9017 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 Figure 4-1, each analog input channel has built-in a jumper on the PCB for users to set as a voltage mode or current mode.

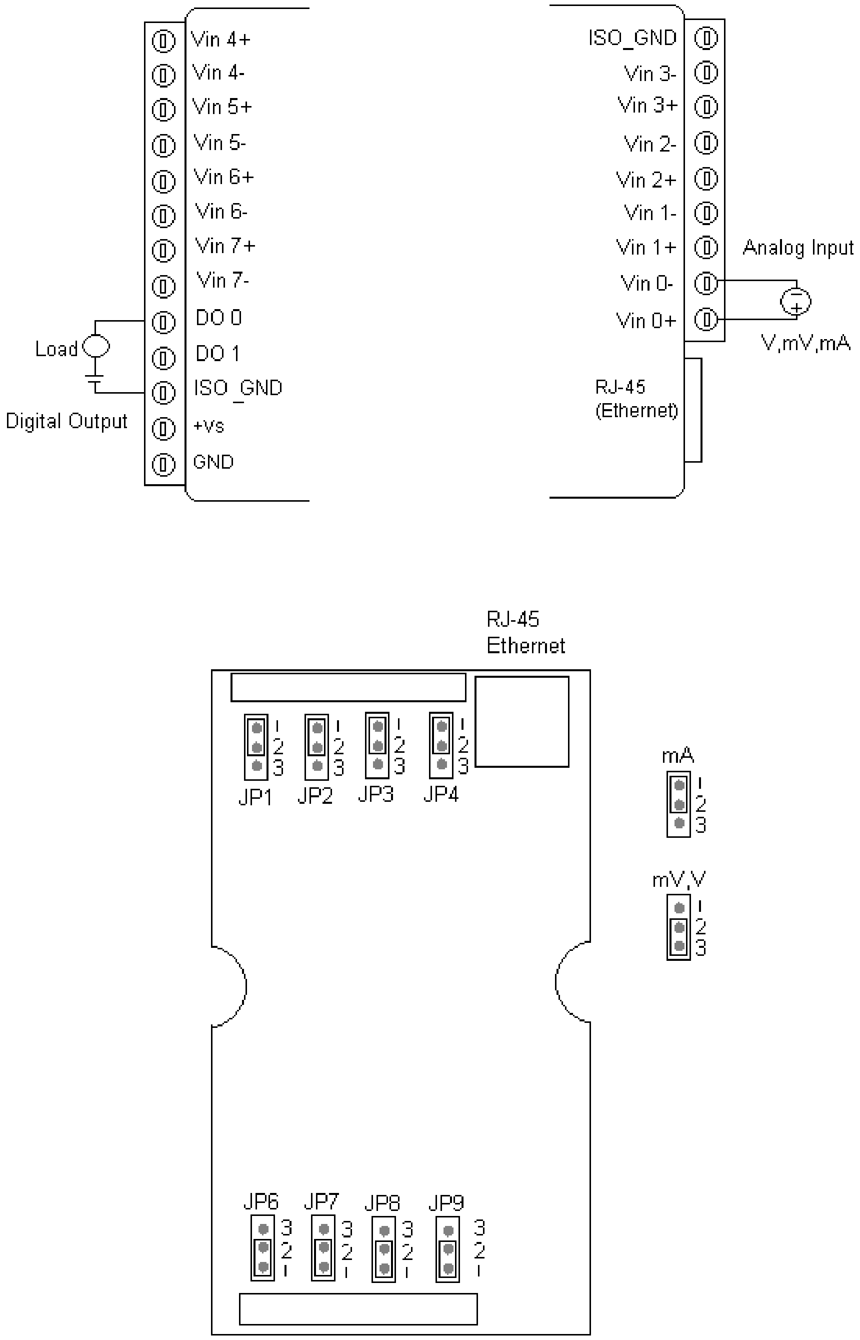


Figure 4-1

4.3 EDAM-9019 8-channel T/C Input with 2/DO Module

The EDAM-9019 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 (o C). In order to satisfy various temperature requirements in one module, each analog channel is allowed to configure an individual range for several applications.

4.3.1 EDAM-9019 Specification

Analog Input:

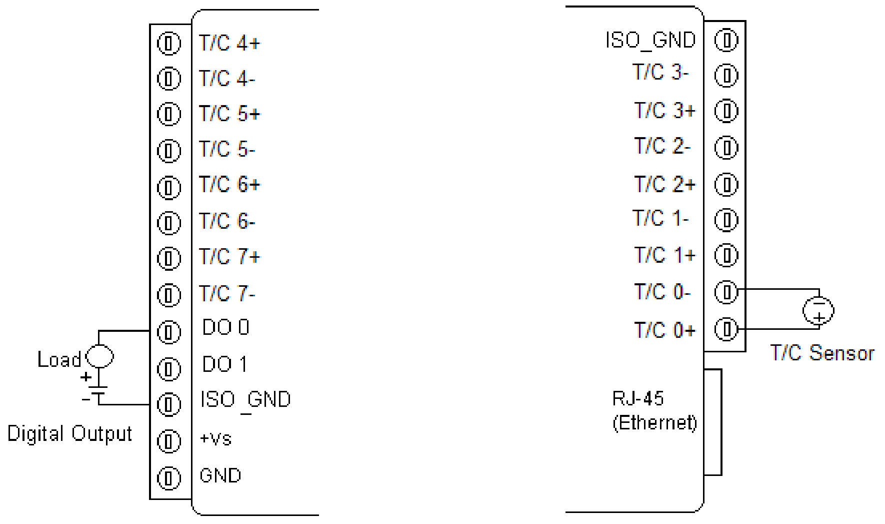
- Effective resolution: 16-bit
- Channels: 8
- Input type: J, K, T, E, R, S, B
- Input range:
- J type: 0 ~ 760 o C
- K type: 0 ~ 1370 o C
- T type: -100 ~ 400 o C
- E type: 0 ~ 1000 o C
- R type: 500 ~ 1750 o C
- S type: 500 ~ 1750 o C
- B type: 500 ~ 1800 o C
- Output Type: 8 channels, Open Collect to 30Vdc/100mA(max), 400mA(max) for all DO.
- Isolation voltage: 2000 VDC
- 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

4.3.2 Application Wiring



4.4 EDAM-9050(A) 12 D*I* and 6 DO channels Digital I/O Module

The EDAM-9050(A) 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 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9050A Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.4.1 Specification

Digital input : Isolated single ended with common source (Dry Contact*)*.

- ♦ Channel : 12 channels (DI0~DI11).
- ♦ Input Level : Logic level status can be inversed via ASCII/Modbus command.
- ♦ Dry Contact : Logic level 0 (active), Close to GND
  Logic level 1 (inactive), Open
- ♦ Counter mode : Supports 500Hz counter(by software,32-bit + 1-bit overflow)
- ♦ Optical Isolation Voltage: 3750Vrms

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

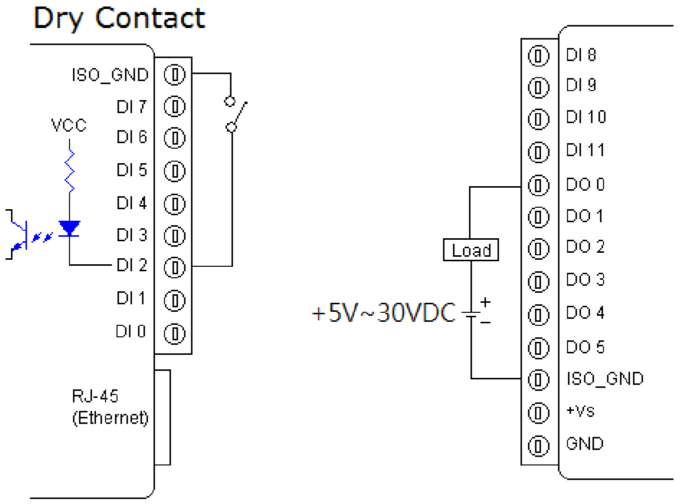
- ♦ Channel : 6 channels (DO0~DO5) .
- ♦ Logical level : Logic level status can be inversed via ASCII/Modbus command.
- ♦ Open Collector : +5V~30V/500 mA max. load
- ♦ Pulse Output : Each channel supports 1KHz pulse output
- ♦ Optical Isolation Voltage: 3750Vrms

Display : 12 digital inputs & 6 digital output status LED

Power requirements : Unregulated +10 ~ +30 VDC

Power Consumption : 1.8 W (Typical)

4.4.2 Application Wiring



Digital Input Digital Output

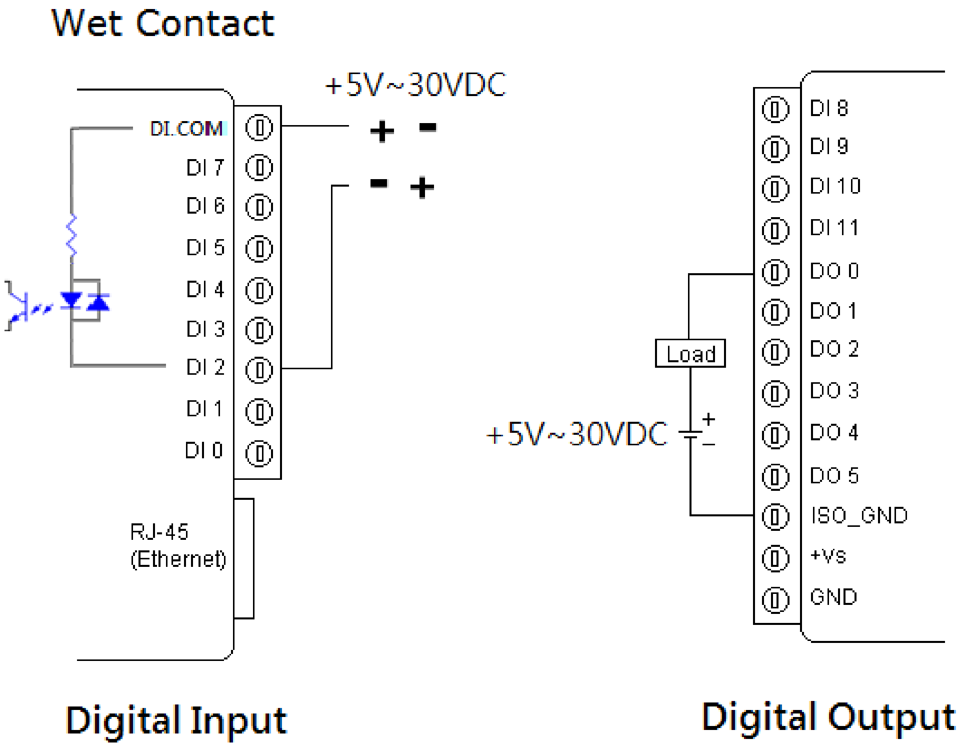
4.5 EDAM-9050AB 12 DI and 6 DO channels DIO Module

The EDAM-9050AB 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 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9050BA Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.5.1 Specification

Digital Input	: Isolated single ended with common source/ground (Web Contact).
♦ Channel	: 12 channels (DI0~DI11).
♦ Input level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Wet Contact	: Logic level 0 (active), +2 Vac max Logic level 1 (inactive), +5V to +30VDC max.
♦ Input Impedance	: 2K ohm (Wet Contact)
♦ Counter mode	: Supports 500Hz soft counter (by software, 32-bit + 1-bit overflow)
♦ Optical Isolation Voltage	: 3750 Vrms
Digital Output	: isolated Open collector (NPN) output channels.
♦ Channel	: 6 channels (DO0~DO5) .
♦ Logical level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Open Collector	: +5V~30V/500 mA max. load
♦ Pulse Output	: Each channel supports 1KHz pulse output
♦ Optical Isolation Voltage	: 3750Vrms
Display	: 12 digital inputs & 6 digital output status LED
Power requirements	: Unregulated +10 ~ +30 VDC
Power Consumption	: 1.8 W (Typical)

4.5.2 Application Wiring



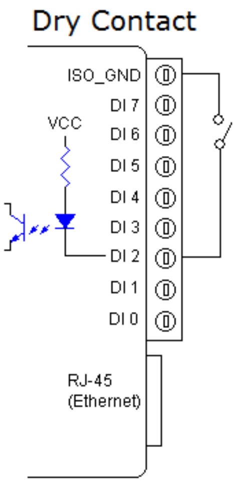
4.6 EDAM-9051(A) 12 DI , 2 DO and 2 Counter chs DIO Module

The EDAM-9051(A) provides ***12 digital input(Dry contact), 2 digital output, and 2 counter*** channels with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9051A Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

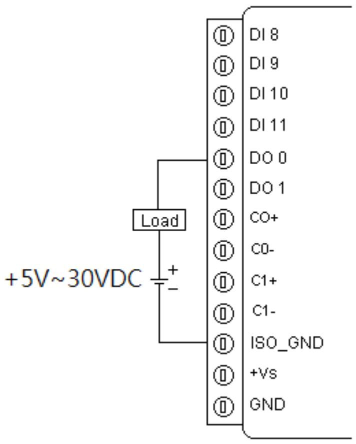
4.6.1 Specification

Digital Input	: Isolated single ended with common source (Dry Contact).
♦ Channel	: 12 channels (DI0~DI11)
♦ Input level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Dry contact	: Logic level 0 (active) ,Close to GND Logic level 1 (inactive) ,Open
♦ Counter	: 500Hz software counter(32-bit + 1-bit overflow)
♦ Impedance	: 2K ohm(Wet Contact)
♦ Optical Isolation Voltage	: 3750Vrms
Digital Output	: Isolated Open collector (NPN) output channels.
♦ Channel	: 2 channels (DO0~DO1) .
♦ Logical level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Open Collector	: +5V~30V / 500 mA max. load
♦ Pulse Output	: Each channel supports 1 kHz pulse output
♦ Optical Isolation Voltage	: 3750Vrms
Counter	: 2 channel hardware input counter
♦ Channel	: 2 (C0=DI12, C1=DI13)
♦ Input logic level	: Logic level 1 (active),+5V to 30VDC max Logic level 0 (inactive) ,+2 Vac max
♦ Maximum Count	: 4,294,967,285 (32-bit + 1-bit overflow)
♦ Input Impedance	: 2K ohm(Wet Contact)
♦ Input frequency	: 4500 Hz max.
♦ Optical Isolation Voltage	: 3750Vrms
Display	: 12 digital inputs, 2 Counter & 2 digital output status LED
Power requirements	: Unregulated +10 ~ +30 VDC
Power Consumption	: 1.8 W (Typical)

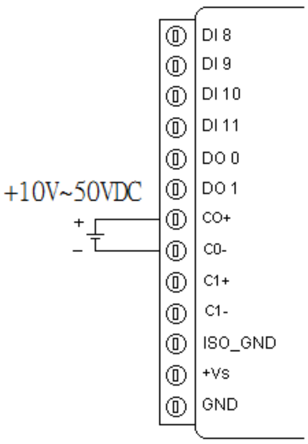
4.6.2 Application Wiring



Digital Input



Digital Output



Counter

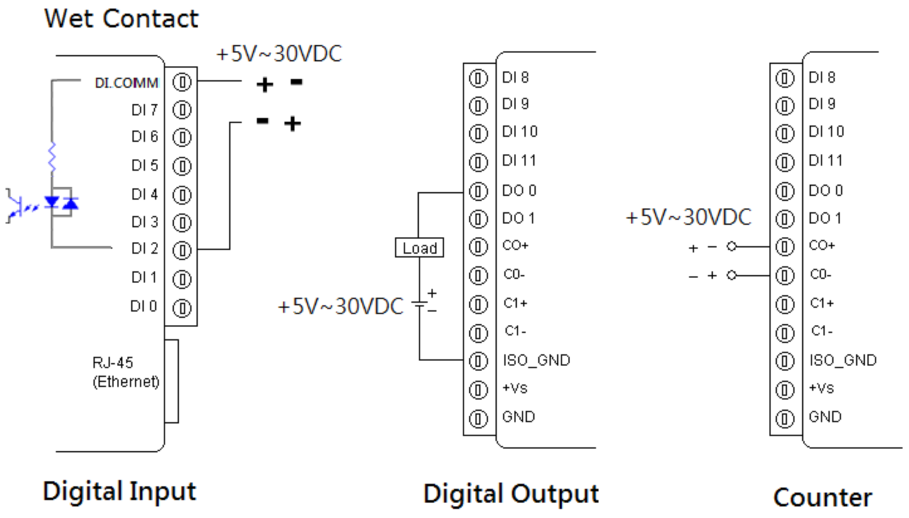
4.7 EDAM-9051AB 12 DI , 2 DO and 2 Counter chs DIO Module

The EDAM-9051AB provides ***12 digital input(Wet contact), 2 digital output, and 2 counter*** channels with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9051AB Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.7.1 Specification

Digital Input : Isolated single ended digital input with common source
♦ Channel : 12 channels (DI0~DI11)
♦ Input level : Logic level status can be inversed via ASCII/Modbus command.
♦ Wet Contact : Logic level 0 (active) , +2 Vdc max
Logic level 1 (inactive) , +5V to +30VDC max.
♦ Counter : 500Hz software counter (32-bit + 1-bit overflow)
♦ Impedance : 2K ohm (Wet Contact)
♦ Optical Isolation Voltage: 3750Vrms
Digital Output : isolated Open collector (NPN) output channels.
♦ Channel : 2 channels (DO0~DO1) .
♦ Logic level : Logic level status can be inversed via ASCII/Modbus command.
♦ Open Collector : +5V~30V / 500 mA max. load (NPN)
♦ Pulse Output : Each channel supports 1 kHz pulse output
♦ Optical Isolation Voltage: 3750Vrms
Counter : 2 channel hardware counter input
♦ Channel : 2 (C0=DI12, C1=DI13)
♦ Input level : Logic level 1 (active),+5V to 30VDC max
Logic level 0 (inactive) ,+2 Vac max
♦ Maximum Count : 4,294,967,285 (32-bit + 1-bit overflow)
♦ Input Impedance : 2K ohm(Wet Contact)
♦ Input frequency : 4500 Hz max.
♦ Optical Isolation Voltage: 3750Vrms
Display : 12 digital inputs, 2 Counter & 2 digital output status LED
Power requirements : Unregulated +10 ~ +30 VDC
Power Consumption : 1.8 W (Typical)

4.7.2 Application Wiring



4.8 EDAM-9052(A) 8 D*I* and 8 DO channels Digital I/O Module

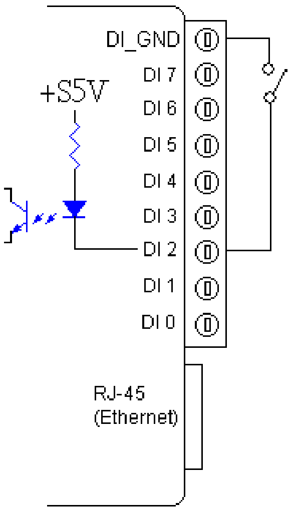
The EDAM-9052(A) is a high-density digital I/O module designed with a 10/100 based-T interface for seamless Ethernet connectivity. It provides ***8 digital input channels(Dry Contact), and 8 digital output*** **channels.** All of the digital input channels support the input latch function for important signal handling. The digital output channels support source type output. Opposite to the intelligent DI functions, the EDAM-9052A Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.8.1 Specification

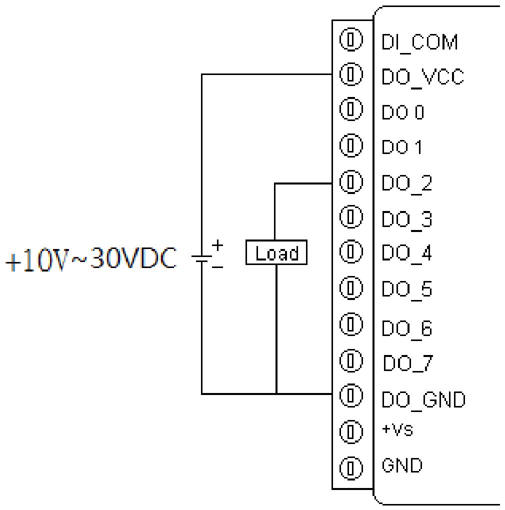
Digital input	: Isolated single ended with common source (Dry Contact).
♦ Channel	: 8 channels (DI0~DI7).
♦ Input Level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Dry Contact	: Logic level 1(active) : Close to DI.GND Logic level 0 (inactive): Open
♦ Counter	: 500Hz software counter(32-bit + 1-bit overflow)
♦ Optical Isolation Voltage	: 3750Vrms
Digital Output	: isolatedopen drain (P-MOSFET) output channels.
♦ Channel	: 8 channels (DO0~DO7) .
♦ Logical level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Load voltage	: +5V ~ +30Vdc
♦ Load current	: 3 A/ channel Max.(total output channels maximum 8A).
♦ Pulse Output	: Each channel supports 1 kHz pulse output
♦ Optical Isolation Voltage	: 3750Vrms
Display	: 8 digital inputs & 8 digital outputs status LED
Power requirements	: Unregulated +10 ~ +30 VDC
Power Consumption	: 1.8 W (Typical)

4.8.2 Application Wiring

Dry Contact



Digital Input



Digital Output

4.9 EDAM-9052AB 8 DI and 8 DO channels DIO Module

The EDAM-9052AB 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(Dry/Wet Contact), and 8 digital output channels.*** All of the digital input channels support the input latch function for important signal handling. The digital output channels support source type output. Opposite to the intelligent DI functions, the EDAM-9052AB Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.9.1 Specification

Digital Input : Isolated single ended with common source/ground (Dry/Wet Contact).

- ♦ Channel : 8 channels (DI0~DI7).
- ♦ Input level : Logic level status can be inversed via ASCII/Modbus command.
- ♦ Dry Contact : Logic level 1(active) : Close to DI.GND
  Logic level 0 (inactive): Open
- ♦ Wet Contact : Logic level 1 (active), +10V to 50VDC max
  Logic level 0 (inactive), +2 Vac max
- ♦ Input Impedance : 10K ohm(Wet Contact)
- ♦ Counter mode : Supports 500Hz counter(by software,32-bit + 1-bit overflow)
- ♦ Optical Isolation Voltage: 3750 Vrms

Digital Output : isolated Open drain(P-MOSFET) output channels.

- ♦ Channel : 8 channels (DO0~DO7) .
- ♦ Output logical level : Logic level status can be inversed via ASCII/Modbus command.
- ♦ Output load voltage : +5V ~ +30Vdc
- ♦ Max load current : 3 A/ channel (total amount of DO channels max. 8A).
- ♦ Pulse Output : Each channel supports 1KHz pulse output
- ♦ Optical Isolation Voltage: 3750 Vrms

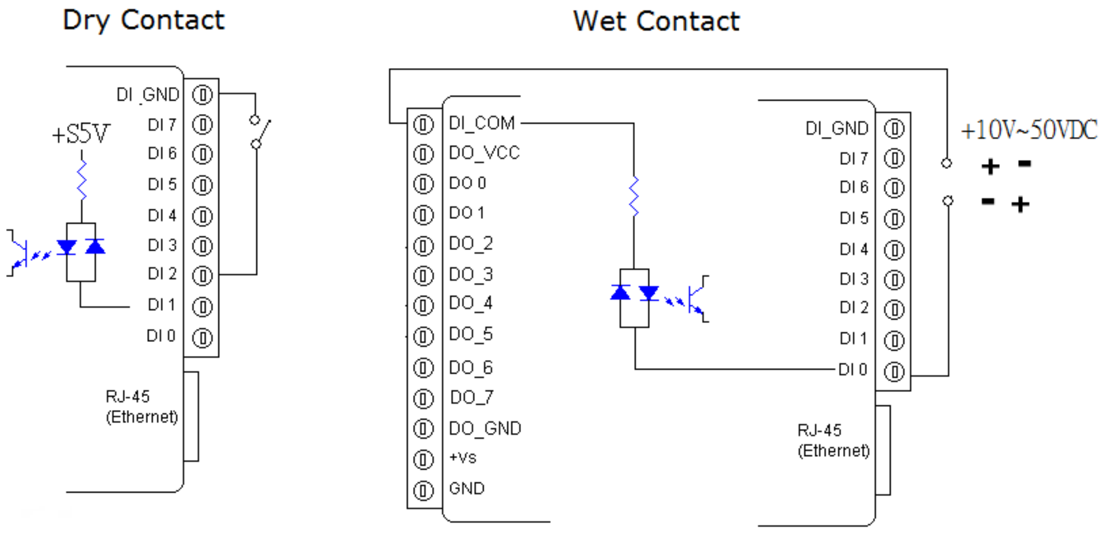
Display : 8 digital inputs & 8 digital outputs status LED

Power requirements : Unregulated, +10V ~ +30 VDC

Power consumption : 1.8 W

4.9.2 Application Wiring

- ♦ **Digital Input type:**

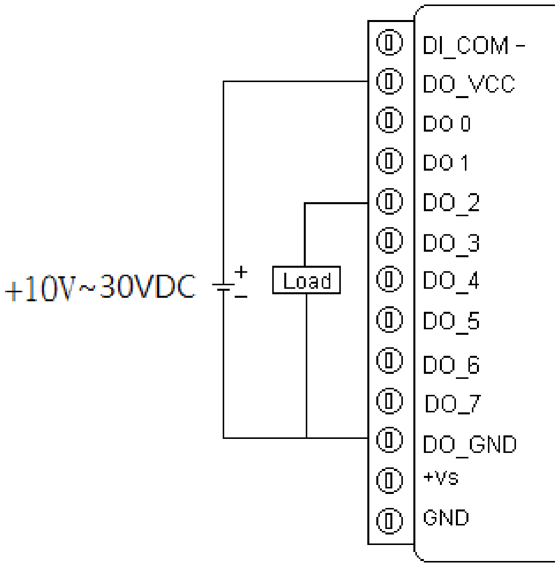


Dry Contact

Wet Contact

Digital Input

♦ **Digital Output type:**



Digital Output

4.10 EDAM-9053A 12 DI and 4 DO channels Digital I/O Module

The EDAM-9053A is a high-density I/O module. It provides ***12 digital input** and **4 digital output channels*** with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Meanwhile, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9053A Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.10.1 Specification

Digital input : Isolated single ended with common source/ground (Dry/Wet Contact).

♦ Channel : 12 channels (DI0~DI11).

♦ Input Level : Logic level status can be inversed via ASCII/Modbus command.

♦ Dry Contact : Logic level 1(active) : Close to GND
Logic level 0 (inactive): Open

♦ Wet Contact : Logic level 1 (active), *+10V to +50VDC max*
Logic level 0 (inactive), +2VDC max

♦ Input Impedance : 10K ohm(Wet Contact)

♦ Counter mode :Supports 500Hz counter(by software,32-bit + 1-bit overflow)

♦ Optical Isolation Voltage: 3750Vrms

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

♦ Channel : 4 channels (DO0~DO3) .

♦ Output logical level : Logic level status can be inversed via ASCII/Modbus command.

♦ Open Collector : +5V ~ 30V / 500 mA max. load

♦ Pulse Output : Supports up to 500Hz

♦ Optical Isolation Voltage: 3750Vrms

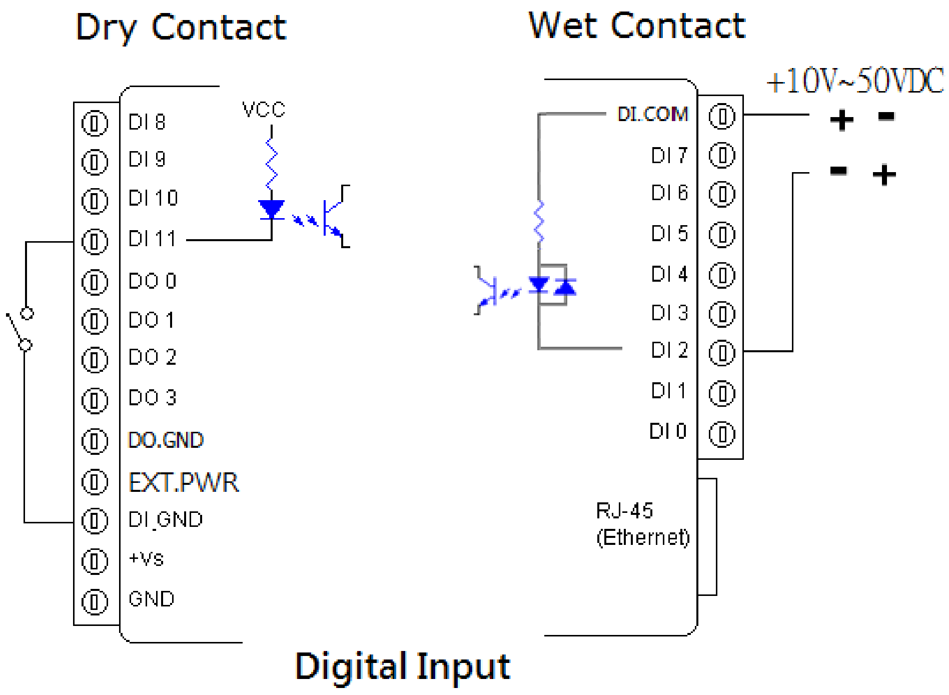
Display : 12 digital input & 4 digital output status LED

Power requirements : Unregulated, +10V ~ +30 VDC

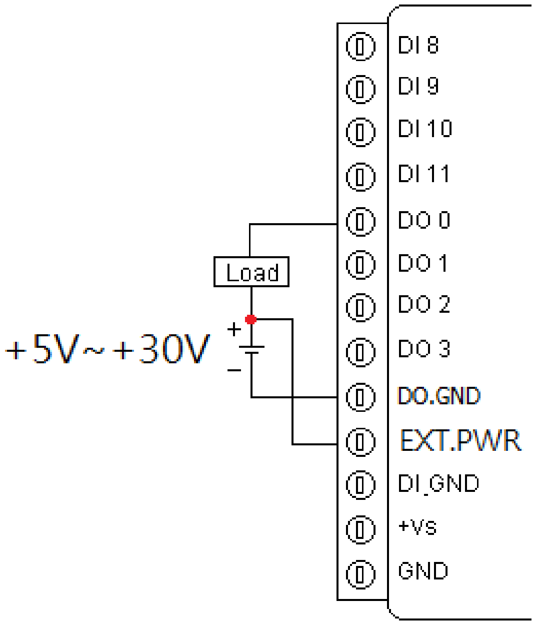
Power Consumption : 1.8 W (Typical)

4.10.2 Application Wiring

♦ **Digital Input type:**



- **Digital Output type:**



Digital Output

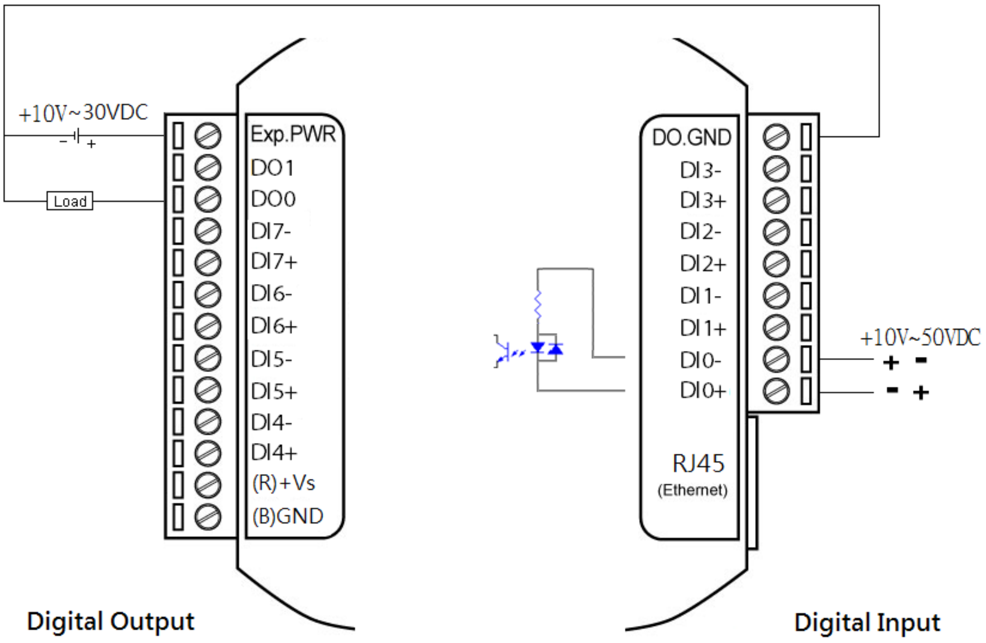
4.11 EDAM-9054A 8 differential DI and 2 DO channel Digital I/O Module

The EDAM-9054A is a high-density I/O module. It provides *8 differential digital input and 2 digital output channels* with 3750VRMS Isolating protection. The EDAM-9054A Digital Input channels support 8 isolated differential digital input (sink/source) channels and All of the Digital Input channels support input latch function for important signal handling. Meanwhile, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9054A Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.11.1 Specification

Digital input	: Isolated differential digital inputs (sink/source).
♦ Channel	: 8 (DI0~DI7) isolated differential input channels (sink/source).
♦ Input Level	: Logic level status can be inversed via ASCII/Modbus command.
	Logical level 0, +:0V ~ +1Vdc Max.
	Logical level 1, +10V ~ +50Vdc
♦ Input Impedance	: 10K ohm
♦ Counter mode	: Supports 500Hz counter(by software,32-bit + 1-bit overflow)
♦ Optical Isolation Voltage	: 3750Vrms
Digital Output	: Isolated open drain (P-MOSFET) outputs.
♦ Channel	: 2 channels (DO0~DO1) .
♦ Output logical level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Output load voltage	: +5V ~ +30Vdc
♦ Max load current	: 3 A/per channel.
♦ Pulse Output	: Supports up to 500Hz pulse output
♦ Optical Isolation Voltage	: 3750 Vrms
Display	: 8 digital inputs & 2 digital outputs status LED
Power requirements	: Unregulated, +10V ~ +30 VDC
Power Consumption	: 2.0 W (Typical)

4.11.2 Application Wiring



4.12 EDAM-9054AB 6 differential DI ch. 2 DO ch. and 2 ch. counter module

The EDAM-9054AB is a high-density I/O module built-in a 10/100 based-T interface for seamless Ethernet connectivity. It provides 6 isolated differential input, 2 digital output, and 2 counter channels with 3750VRMS Isolating protection. All of the Digital Input channels support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. Opposite to the intelligent DI functions, the EDAM-9054AB Digital Output channels also support pulse output function, *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.12.1 Specification

Digital input : Isolated differential digital inputs (no parity)

♦ Channel : 6 (DI0~DI5)

♦ Input Level : Logic level status can be inversed via ASCII/Modbus command.
Logical level 0, 0V ~ +1Vdc Max.
Logical level 1, +10V ~ +50Vdc

♦ Input Impedance : 10K ohm

♦ Counter mode : Supports 500Hz counter(by software,32-bit + 1-bit overflow)

♦ Optical Isolation Voltage: 3750Vrms

Counter : Hardware digital counter

♦ Channel : 2 (C0=DI6, C1=DI7)

♦ Input level : Logic level 1 (active), +10V ~ +50Vdc
Logic level 0 (inactive), 0V ~ +1Vdc Max.

♦ Maximum Count : 4,294,967,285 (32-bit + 1-bit overflow)

♦ Input Impedance : 10K ohm(Wet Contact)

♦ Input frequency : 4500 Hz max.

♦ Optical Isolation Voltage: 3750Vrms

Digital Output : Isolated open drain (P-MOSFET) output channels.

♦ Channel : 2 channels (DO0~DO1) .

♦ Output logical level : Logic level status can be inversed via ASCII/Modbus command.

♦ Output load voltage : +5V ~ +30Vdc

♦ Max load current : 3 A/per channel.

♦ Pulse Output : Supports up to 500Hz pulse output

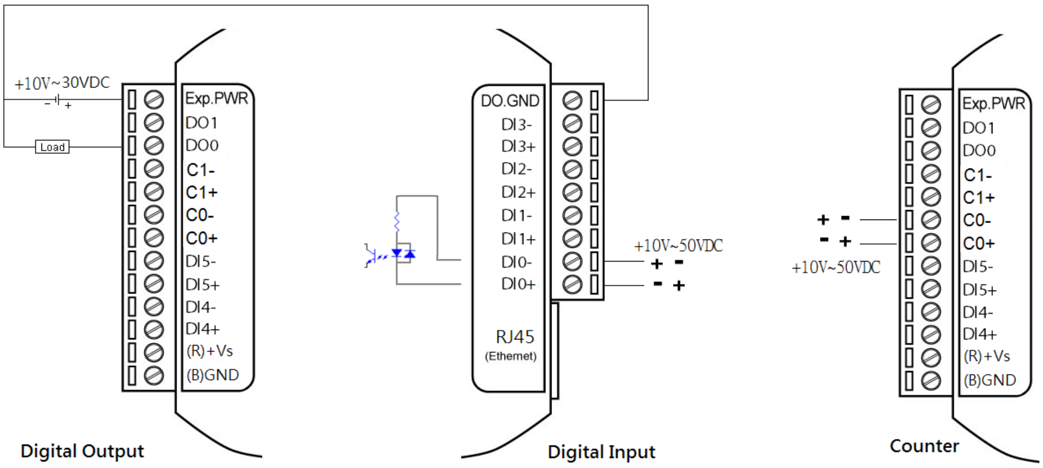
♦ Optical Isolation Voltage: 3750 Vrms

Display : 6 digital input & 3 digital output status LED

Power requirements : Unregulated, +10V ~ +30 VDC

Power Consumption : 2.0 W (Typical)

4.12.2 Application Wiring



Digital Output

Digital Input

Counter

4.13 EDAM-9060A 5-channel Digital Input and 3 RELAY output Module

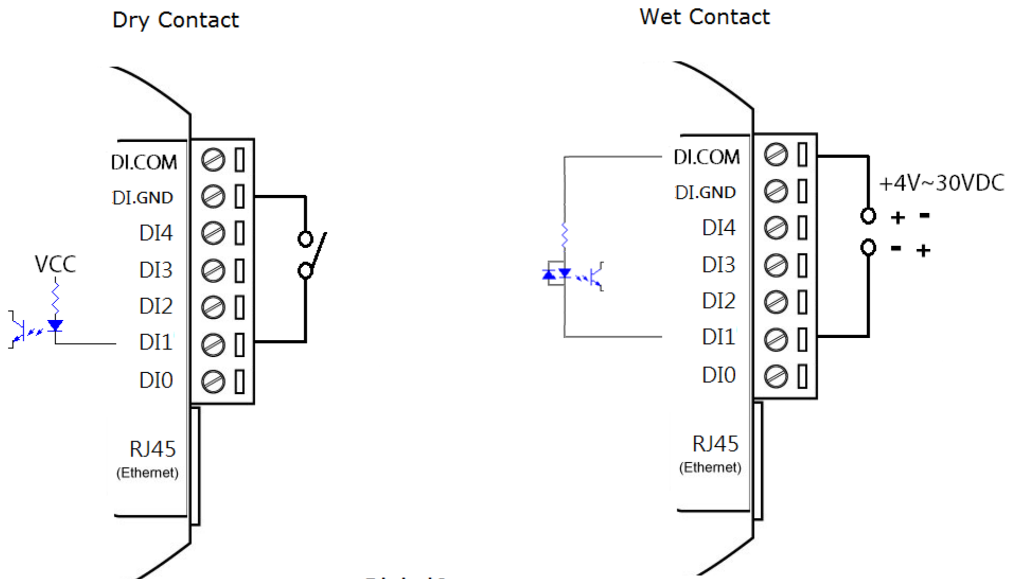
EDAM-9060A provides 5 isolated digital input channels and 3 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common . Opposite to the intelligent DI functions, the EDAM-9060A Digital Output channels also support *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.13.1 Specification

Digital input : Isolated single ended with common source/ground.
♦ Channel : 5 channels (DI0~DI4).
♦ Input Level : Logic level status can be inversed via ASCII/Modbus command.
♦ Dry Contact : Logic level 1(active), Close to GND
Logic level 0 (inactive), Open
♦ Wet Contact : Logic level 1 (active), +4V to +30VDC max
Logic level 0 (inactive), +2VDC max
♦ Input Impedance : 3K ohm(Wet Contact)
♦ Counter mode : Supports up to 500Hz counter(by software,32-bit + 1-bit overflow)
♦ Optical Isolation Voltage: 3750Vrms
Relay Output : RL1, RL2, RL3
♦ Output channels : 3 relay output channels (RL1: Form A, RL2, RL3 Form C).
♦ Surge strength : 500V
♦ Relay contact rating : 0.6A/125Vac, 2A/30Vdc
♦ Operate Time : 3mS max.
♦ Release Time : 2mS max.
♦ Min Life : $5*10^5$ ops
♦ Pulse Output : Each channel supports 500Hz pulse output
Display : 5 digital input & 3 Relay output status LED
Power requirements : Unregulated, +10V ~ +30 VDC
Power Consumption : 1.8 W (Typical)

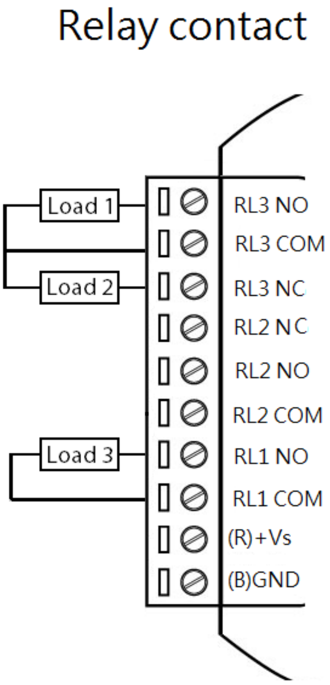
4.13.2 Application Wiring

♦ **Digital Input type:**



Digital Input

- **Relay Output type:**



NO: Normal open, **NC: Normal Close**

4.14 EDAM-9061A / 9461 6-channel Digital Input and 6 RELAY output Module

EDAM-9061A/9461(**PoE**) provides 6 isolated digital input channels and 6 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Mean while, these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common . Opposite to the intelligent DI functions, the EDAM-9061A Digital Output channels also support *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.14.1 Specification

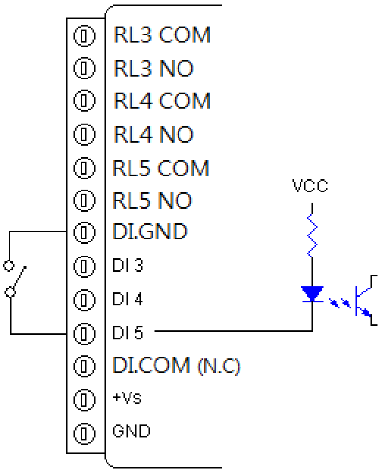
Digital input	: Isolated single ended with common source (Dry Contact).
♦ Channel	: 6 channels (DI0~DI5).
♦ Input Level	: Logic level status can be inversed via ASCII/Modbus command.
♦ Dry Contact	: Logic level 1(active) : Close to GND Logic level 0 (inactive): Open
♦ Wet Contact	: Logic level 1 (active), *+10V to +50VDC max* Logic level 0 (inactive), +2VDC max
♦ Input Impedance	: 3K ohm(Wet Contact)
♦ Counter mode	: Supports up to 500Hz counter(by software, 32-bit + 1-bit overflow)
♦ Optical Isolation Voltage	: 3750Vrms
Relay Output: RL0~RL5	
♦ Output channels	: 6 relay output channels (Form A).
♦ Surge strength	: 500V
♦ Relay contact rating	: 0.6A/125Vac, 2A/30Vdc
♦ Operate Time	: 3mS max.
♦ Release Time	: 2mS max.
♦ Min Life	: 5*105 ops
♦ Pulse Output	: Each channel supports 500Hz pulse output
Display	: 6 digital input & 6 Relay output status LED
Power requirements	: Unregulated, +10V ~ +30 VDC
Power Consumption	: 2.5 W (Typical)

Power Over Ethernet (**PoE**): only for 9461 (**class 1**)

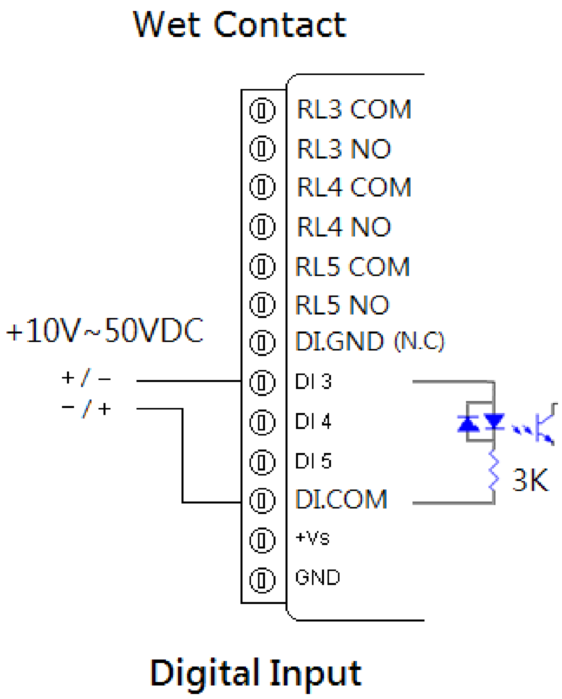
4.14.2 Application Wiring

♦ **Digital Input type:**

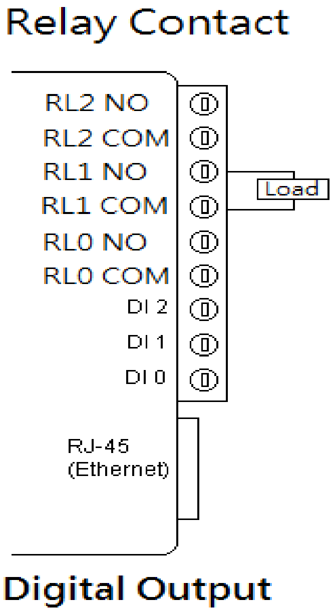
Dry Contact



Digital Input



- **Relay Output type:**



NO: Normal open, NC: Normal Close

4.15 EDAM-9063A 7-channel Digital Input and 3 RELAY output Module

EDAM-9063A provides 7 isolated digital input channels and 3 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Meanwhile, these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common . the EDAM-9063A Digital Output channels also support *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.15.1 Specification

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

- ♦ Channel : 7 channels (DI0~DI6).
- ♦ Input Level : Logic level status can be inversed via ASCII/Modbus command.
- ♦ Dry Contact : Logic level 1(active) : Close to GND
  Logic level 0 (inactive): Open
- ♦ Wet Contact : Logic level 1 (active), +10V to +50VDC max
  Logic level 0 (inactive), +2VDC max
- ♦ Input Impedance : 10K ohm(Wet Contact)
- ♦ Counter mode :Supports up to 500Hz counter(by software,32-bit + 1-bit overflow)
- ♦ Optical Isolation Voltage:3750Vrms

Relay Output: RL1, RL2, RL3 (DO0~DO2)

- ♦ Output channels : 3 relay output channels (Form A).
- ♦ Surge strength : 4000V
- ♦ Relay contact rating : 5A/250Vac, 5A/30Vdc
- ♦ Operate Time : 6mS max.
- ♦ Release Time : 3mS max.
- ♦ Min Life : 105 ops.
- ♦ Pulse Output : Supports up to 500Hz pulse output

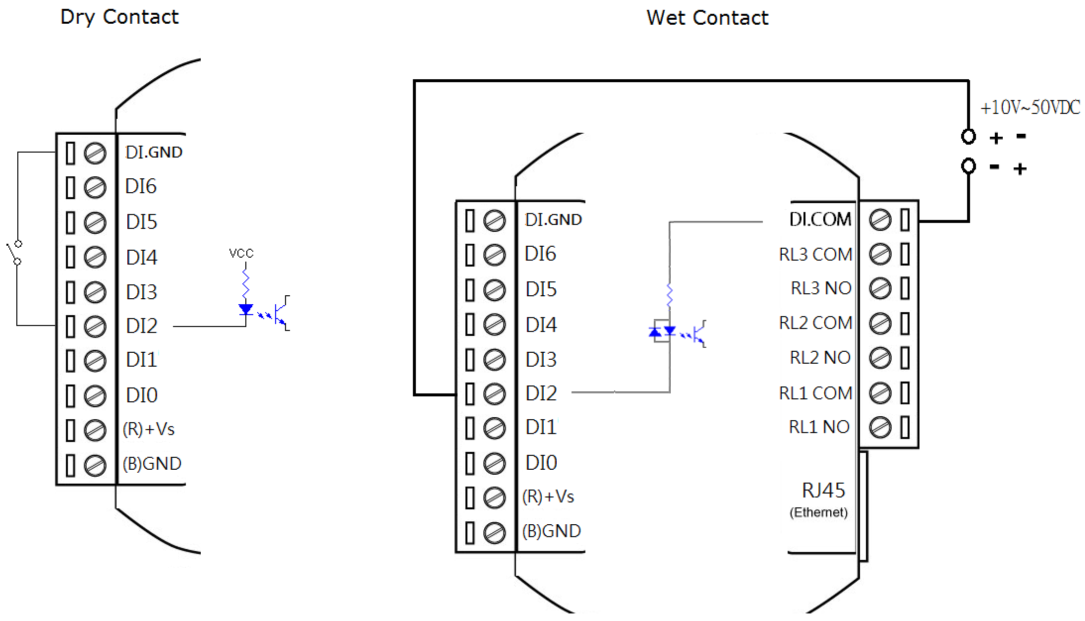
Display : 7 digital input & 3 Relay output status LED

Power requirements : Unregulated, +10V ~ +30 VDC

Power Consumption : 2.4 W (Typical)

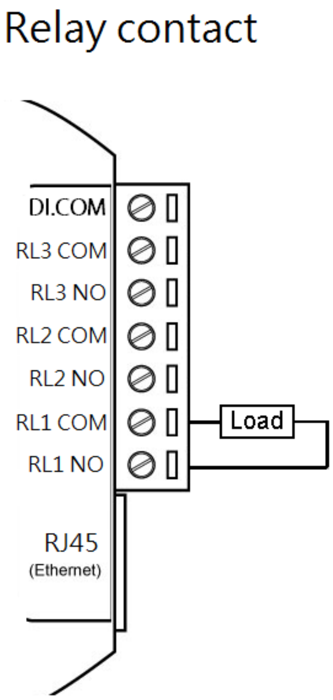
4.15.2 Application Wiring

- ♦ **Digital Input type:**



Digital Input

♦ **Relay Output type:**



NO: Normal open, NC: Normal Close

4.16 EDAM-9066A / 9466 6-channel Digital Input and 6 RELAY output Module

EDAM-9066A/9466(**PoE**) provides 6 isolated digital input channels and 6 relay output channels. All input channels are single ended with common source/ground and support input latch function for important signal handling. Meanwhile, these DI channels allow to be used as 500Hz counter. All relay output channels are differential with individually common . the EDAM-9066A/9466 Digital Output channels also support *Auto-Off Time of digital output* and *DIO Synchronization* function(ref to 14.7).

4.16.1 Specification

Digital input : Isolated single ended with common source (Dry Contact).

♦ Channel : 6 channels (DI0~DI5).

♦ Input Level : Logic level status can be inversed via ASCII/Modbus command.

♦ Dry Contact : Logic level 1(active) : Close to GND
Logic level 0 (inactive): Open

♦ Wet Contact : Logic level 1 (active), *+10V to +50VDC max*
Logic level 0 (inactive), +2VDC max

♦ Input Impedance : 3K ohm(Wet Contact)

♦ Counter mode : Supports up to 500Hz counter(by software, 32-bit + 1-bit overflow)

♦ Optical Isolation Voltage: 3750Vrms

Relay Output: RL0~, RL5 (DO0~DO5)

♦ Output channels : 6 relay output channels (Form A).

♦ Surge strength : 4000V

♦ Relay contact rating : 5A/250Vac, 5A/30Vdc

♦ Operate Time : 6mS max.

♦ Release Time : 3mS max.

♦ Min Life : 10^5 ops.

♦ Pulse Output : Supports up to 500Hz pulse output

Display : Power-LED only, No DIO status display LEDs,

Power requirements : Unregulated,+10V ~ +30 VDC

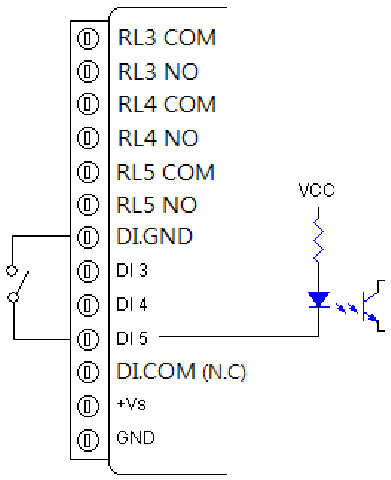
Power Consumption : 2.4 W (Typical)

Power Over Ethernet(**PoE**): only for 9466 (**class 1**)

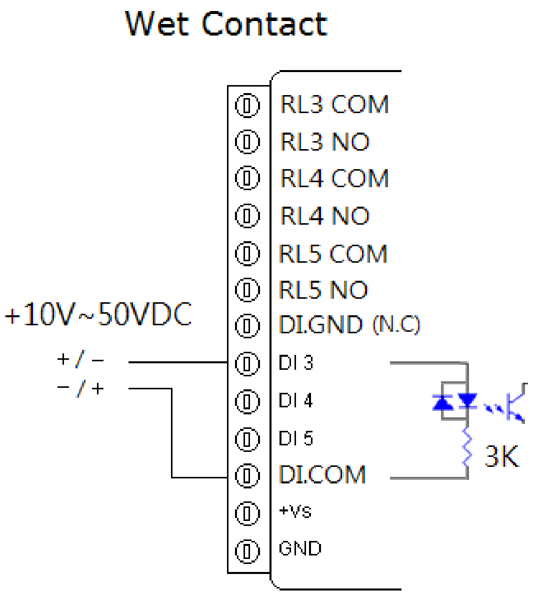
4.16.2 Application Wiring

♦ **Digital Input type:**

Dry Contact

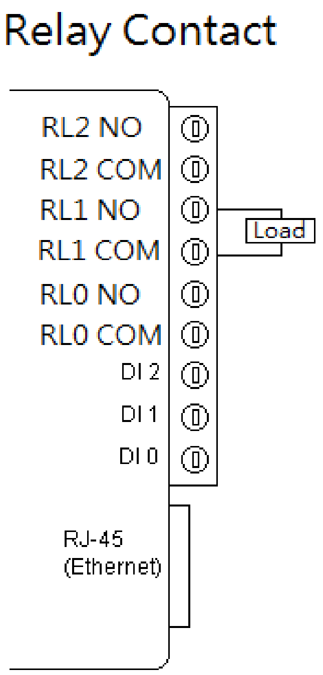


Digital Input



Digital Input

♦ **Relay Output type:**



Digital Output

NO: Normal open, NC: Normal Close

Chapter 5 EDAM-9x00(A) Utility Guide

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

5.1 System Requirement

Host computer

- ♦ IBM PC compatible computer with 486 CPU (Pentium is recommended)
- ♦ Microsoft 95/98/2000/NT 4.0 (SP3 or SP4)/Win 7,8 or higher versions
- ♦ At least 32 MB RAM
- ♦ 20 MB of hard disk space available
- ♦ VGA color monitor
- ♦ 2x or higher speed CD-ROM
- ♦ Mouse or other pointing devices
- ♦ 10 or 100 Mbps Ethernet Card
- ♦ 10 or 100 Mbps Ethernet Hub (at least 2 ports)
- ♦ Two Ethernet Cable with RJ-45 connector
- ♦ Power supply for EDAM-9000 (+10 to +30 V unregulated), (for 94xx: option).
- ♦ Make sure to prepare all of the items above, then connect the power and network wiring as Figure 5-1

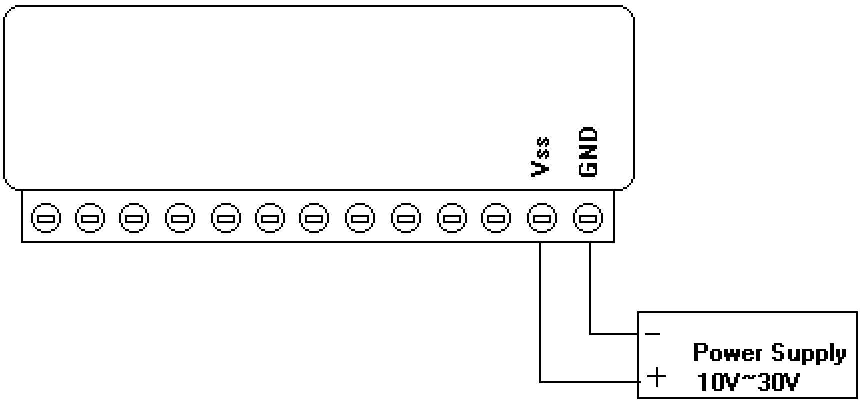


Figure 5-1 Power wiring

5.2 Install Utility Software on Host PC

Inlog provide free download Manual and Utility software for EDAM-9000(A) modules' operation and configuration. Link to the web site: www.inlog.com.tw and click into the "Download Area" to get the latest version EDAM-9000 manual and Ethernet I/O Utility. Once you download and setup the Utility software, there will be a shortcut of the Utility executive program on Windows' desktop after completing the installation.

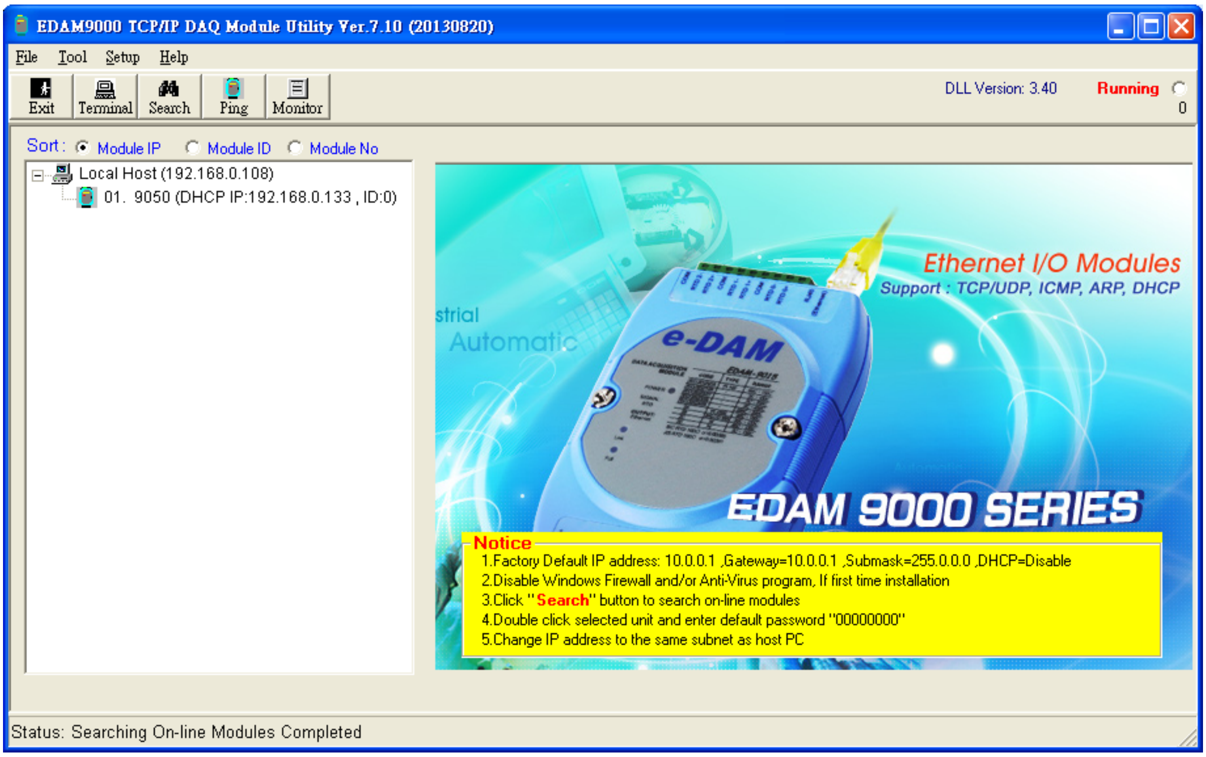
5.3 EDAM Ethernet I/O Utility Overview

The Utility software offers a graphical interface that helps you configure the EDAM-9000(A) modules. It is also very convenient to test and monitor your remote DAQ system. The following guidelines will give you some brief instructions on how to use this Utility.

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

5.4 Main Menu

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



EDAM9000 TCP/IP DAQ Module Utility Ver.7.10 (20130820)

DLL Version: 3.40 Running ○ 0

Sort: ● Module IP ○ Module ID ○ Module No

Local Host (192.168.0.108)
01. 9050 (DHCP IP:192.168.0.133 , ID:0)

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

Status: Searching On-line Modules Completed

Figure 5-2 main window

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

5.4.1 Function Menu

- File contents “Exit” Function, using to exit this Utility program.
- Tool contents functions as below:

Search : for Ethernet Device Search all EDAM-9000 units in the specific Ethernet domination. (The same with host PC’s Ethernet domination)

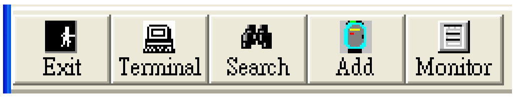
Add Remote Ethernet Device: Create a new EDAM-9000 module located in other Ethernet domination, both available to local LAN and Internet application.

Monitor Stream/Event Data : Comes from the remote I/O module

- Terminal : Call up the operation screen of Terminal emulation to do the request / response command execution.
- Setup : Contents Timeout and Scan Rate setting functions. Please be aware of the time setting for other Ethernet domination usually longer than local network.
- Help : Contents on-line help function as user's operation guide; the item "About" contents information about software version, released date, and support modules.

5.4.2 Tool Bar

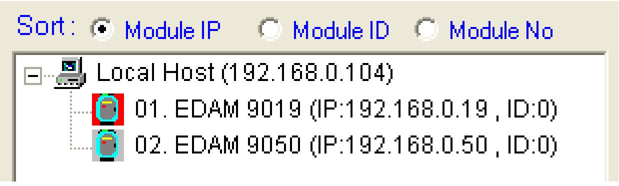
There are five push buttons in the tool bar.



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

5.4.3 List Sort

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



Sort : ● Module IP ○ Module ID ○ Module No

Local Host (192.168.0.104)
01. EDAM 9019 (IP:192.168.0.19 , ID:0)
02. EDAM 9050 (IP:192.168.0.50 , ID:0)

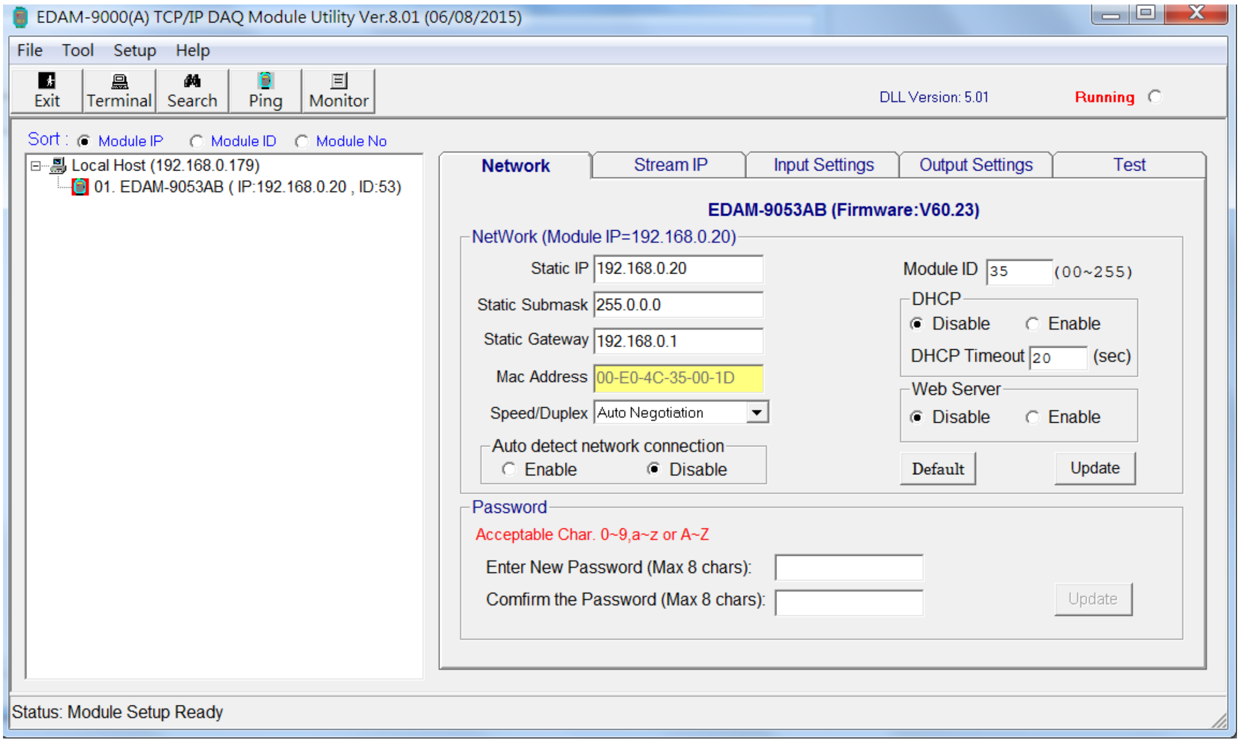
- Module IP : Sort by module IP
- Module ID : Sort by module ID
- Module No : Sort by module name

5.5 Network Setting

As the moment you start up this Windows Utility, it will search all EDAM-9000(A) I/O modules on the host PC's domination Ethernet network automatically. Then the tree-structure display area will appeal with the searched units and the relative IP address.

Since Utility software detects the EDAM-9000(A) 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



Sort : ● Module IP ○ Module ID ○ Module No

Local Host (192.168.0.179)
01. EDAM-9053AB (IP:192.168.0.20 , ID:53)

EDAM-9053AB (Firmware:V60.23)

NetWork (Module IP=192.168.0.20)

Static IP 192.168.0.20

Static Submask 255.0.0.0

Static Gateway 192.168.0.1

Mac Address 00-E0-4C-35-00-1D

Speed/Duplex Auto Negotiation

Auto detect network connection ○ Enable ● Disable

Module ID 35 (00~255)

DHCP ● Disable ○ Enable

DHCP Timeout 20 (sec)

Web Server ● Disable ○ Enable

Password

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

Enter New Password (Max 8 chars):

Comfirm the Password (Max 8 chars):

Status: Module Setup Ready

Figure 5-3

5.5.1 Module IP

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

IP Address, Subnet Mask, and Default Gateway: (default 10.0.0.1, 255.0.0.0 and 10.0.0.1)

The IP address identifies your EDAM-9000(A) devices on the global network. Each EDAM-9000(A) has same default IP address 10.0.0.1. Therefore, *please do not initial many EDAM-9000(A) at the same time to avoid the Ethernet collision*. If you want to configure the EDAM-9000(A) in the host PC's dominating network, only the IP address and Subnet Mask will need to set (The host PC and EDAM Ethernet I/O must belong to same subnet Mask).

If you want to configure the EDAM-9000(A) via Internet or other network domination, you have to ask your network administrator to obtain a specific IP and Gateway addresses, and then configure each EDAM-9000 with the individual setting.

DHCP **:** (default Disabled)

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

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

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

Web Server **:** (default Disabled)

Allow you monitor and control I/O status on EDAM-9000 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() in TCPDAQ.DLL) to get correct IP address for each ID number

Password **:** (default 00000000)

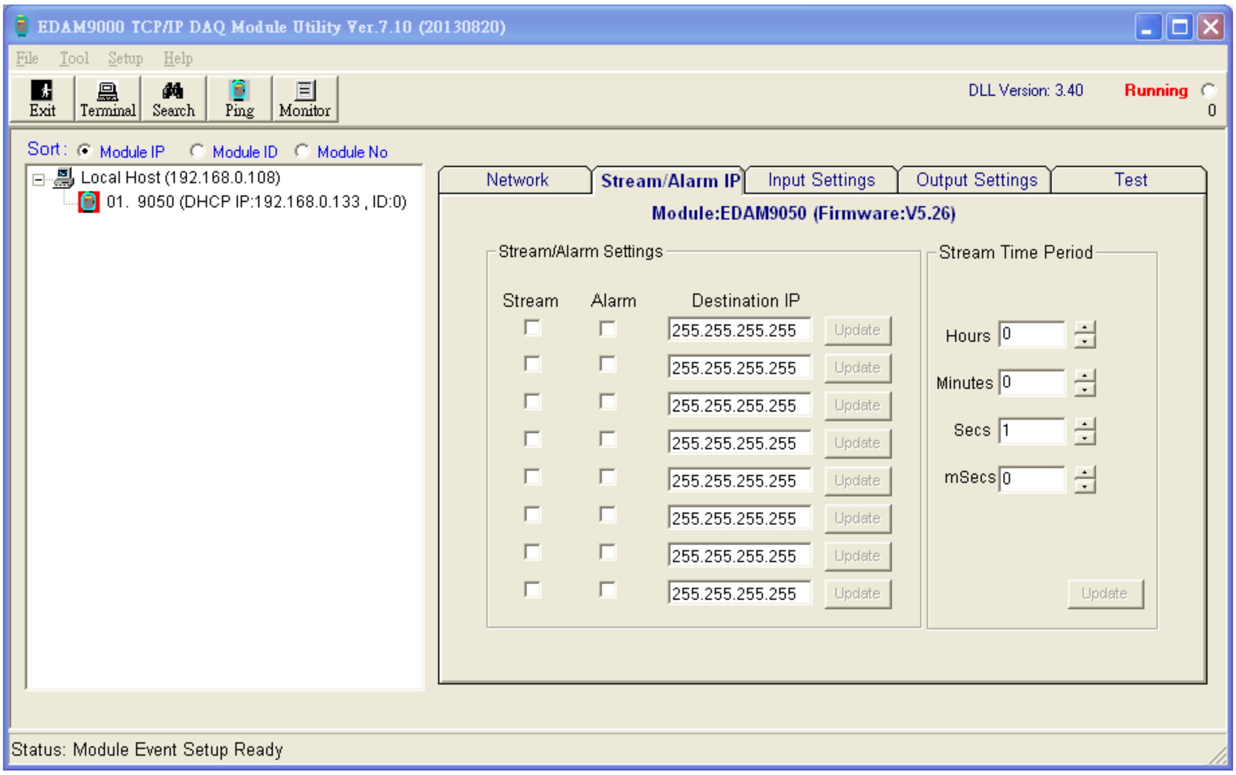
Allow you to change the password of the module

5.5.2 TCP/IP port:

EDAM-9000(A) series use four ports to communication with Host as shown below table

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

5.5.3 Stream/Alarm IP



Sort: ● Module IP ○ Module ID ○ Module No

Local Host (192.168.0.108)

01. 9050 (DHCP IP:192.168.0.133 , ID:0)

Module:EDAM9050 (Firmware:V5.26)

Stream/Alarm Settings

Stream	Alarm	Destination IP
□	□	255.255.255.255
□	□	255.255.255.255
□	□	255.255.255.255
□	□	255.255.255.255
□	□	255.255.255.255
□	□	255.255.255.255
□	□	255.255.255.255
□	□	255.255.255.255

Stream Time Period

Hours 0

Minutes 0

Secs 1

mSecs 0

Status: Module Event Setup Ready

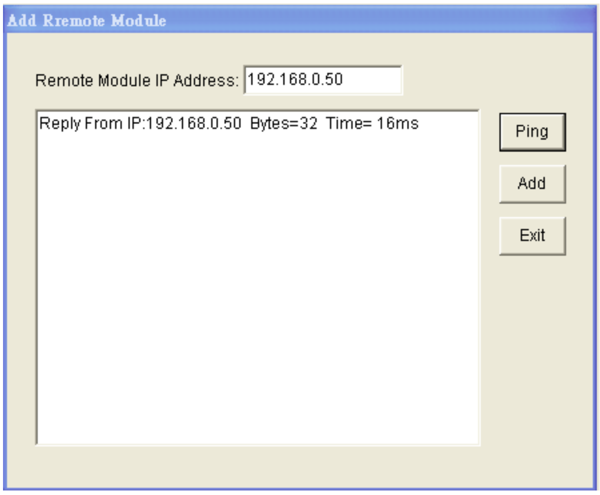
Stream/Alarm event Enable Setting : Set Stream /Event data Destination IP (default all disabled),

Active Stream time period **:** Set time interval for sending stream data (default 1 sec)

5.6 Add Remote Stations

To meet the remote monitoring and maintenance requirements, The EDAM-9000 system does not only available to operate in local LAN, but also allowed to access from Internet or Intranet. Thus users would able to configure an EDAM-9000 easily no matter how far it is.

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



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

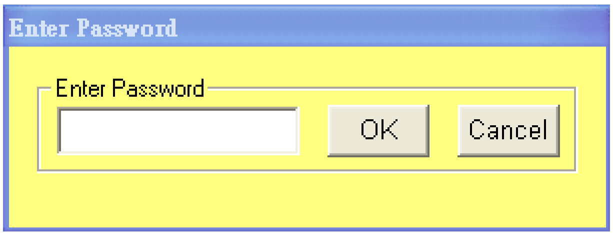
Figure5-4 Add remote module

Note:

- ♦ There is several conditions need to be sure before adding a remote EDAM-9000 system in the Window Utility.
- ♦ Be sure the specific IP is existed and available.
- ♦ Be sure to complete the network linkage for both sides.
- ♦ Be sure to adjust the best timing of timeout setting.
- ♦ Even you are not sure whether the communication is workable or not, there is also a **“Ping”** function for testing the network connection.

5.7 Security Setting

Though the technology of Ethernet discovered with great benefits in speed and integration, there also exist risk about network invading form anywhere. For the reason, the security protection design has built-in EDAM-9000 I/O modules. Once user setting the password into the EDAM-9000 firmware, the important system configurations (Network, Firmware, Password) are only allowed to be changed by password verification.



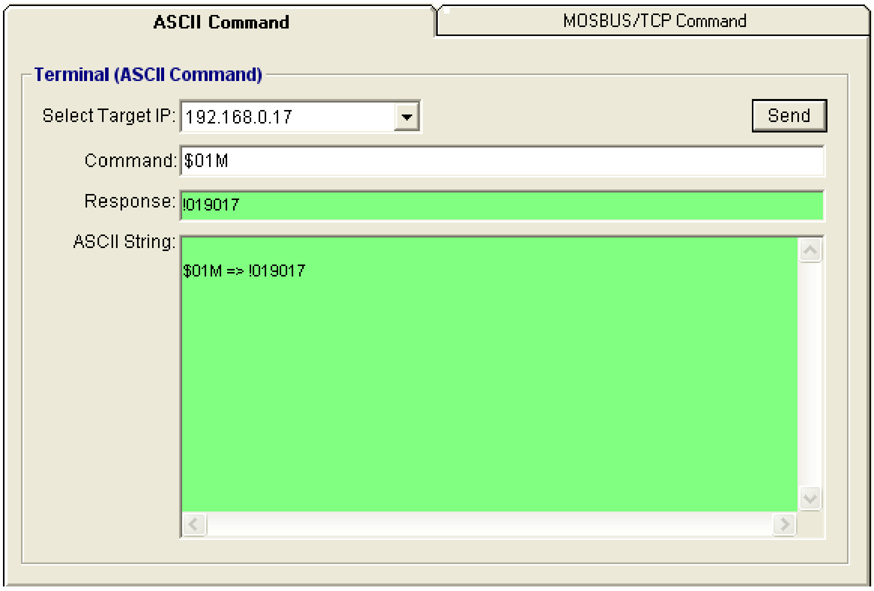
Note:

The default password of EDAM-9000 is “**00000000**”. Please make sure to keep the correct password by yourself. If you lose it, please contact to Inlog’s technical support center for help.

5.8 Terminal Emulations

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

ASCII Command mode: Shown as ASCII Command Terminal



Terminal (ASCII Command)

Select Target IP: 192.168.0.17

Command: $01M

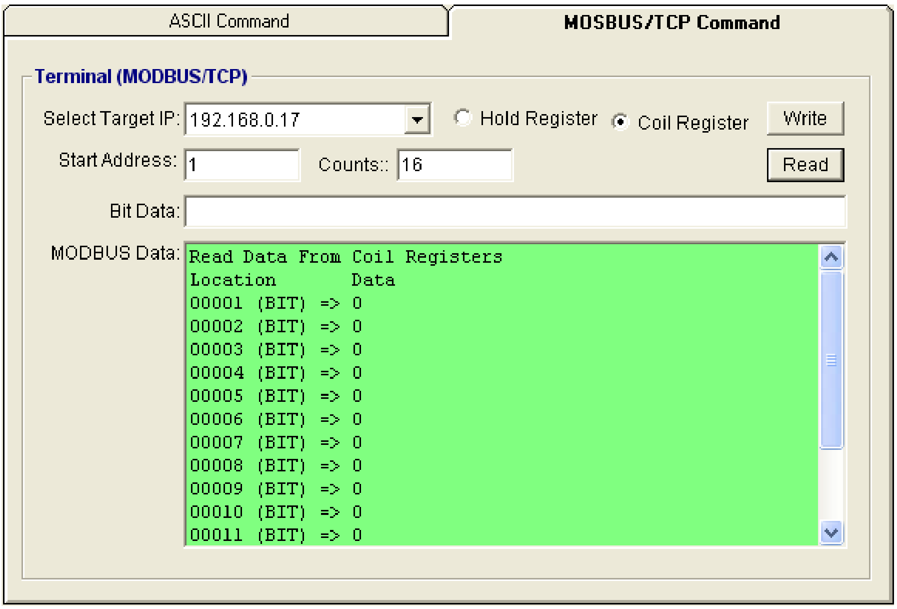
Response: !019017

ASCII String:

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

Figure 5-5 ASCII Command Terminal

ModBus Hexadecimal mode: shown as Chapter 9



Terminal (MODBUS/TCP)

Select Target IP: 192.168.0.17 ○ Hold Register ● Coil Register

Start Address: 1 Counts:: 16

Bit Data:

MODBUS Data:

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

Figure 5-6 ModBus Terminal

5.9 Data /Event Stream

Data Stream Configuration:

In addition to TCP/IP communication protocol, EDAM-9000(A) supports UDP communication protocol to regularly broadcast data to specific host PCs. Click the tab of Data Stream, then configure the broadcasting interval and the specific IP addresses which need to receive data from the specific EDAM-9000 I/O module. This UDP Data Stream function broadcasts up to 8 host PCs simultaneously, and the interval is user-defined from 50ms to 7 Days.

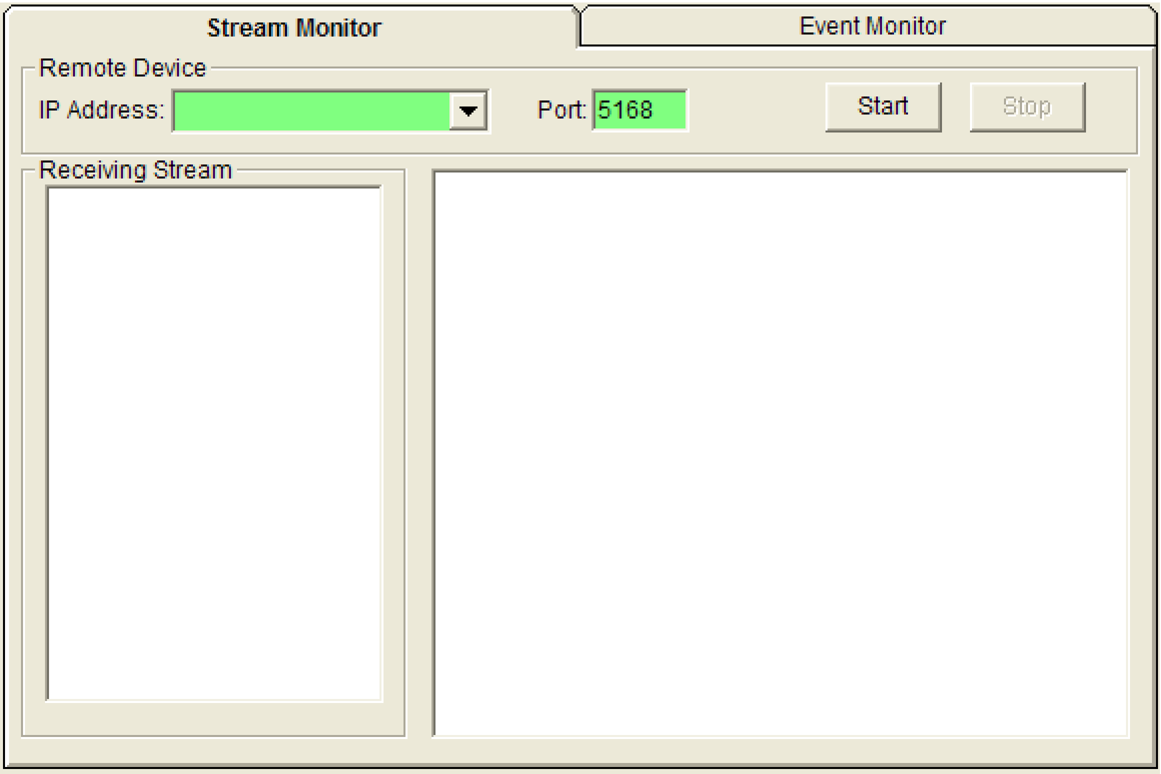
Event Stream Configuration:

In addition to TCP/IP communication protocol, EDAM-9000(A) supports UDP communication protocol to regularly broadcast data to specific host PCs. Click the tab of Data Stream, then configure the broadcasting interval and the specific IP addresses which need to receive data from the specific EDAM-9000 I/O module. This UDP Data Stream function broadcasts up to 8 host PCs simultaneously, and the interval is user-defined from 50ms to 7 Days.

Data Stream Monitoring:

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

Select the IP address of the EDAM-9000 you want to read data, then click “**Start** ” button. The Utility software will begin to receive the stream data on this operation display.



Port: 5168

Figure 5-7 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 1 8 Event display.

Select the IP address of the EDAM-9000 you want to read data, then click "**Start**" button. The Utility software will begin to receive the stream data on this operation display.

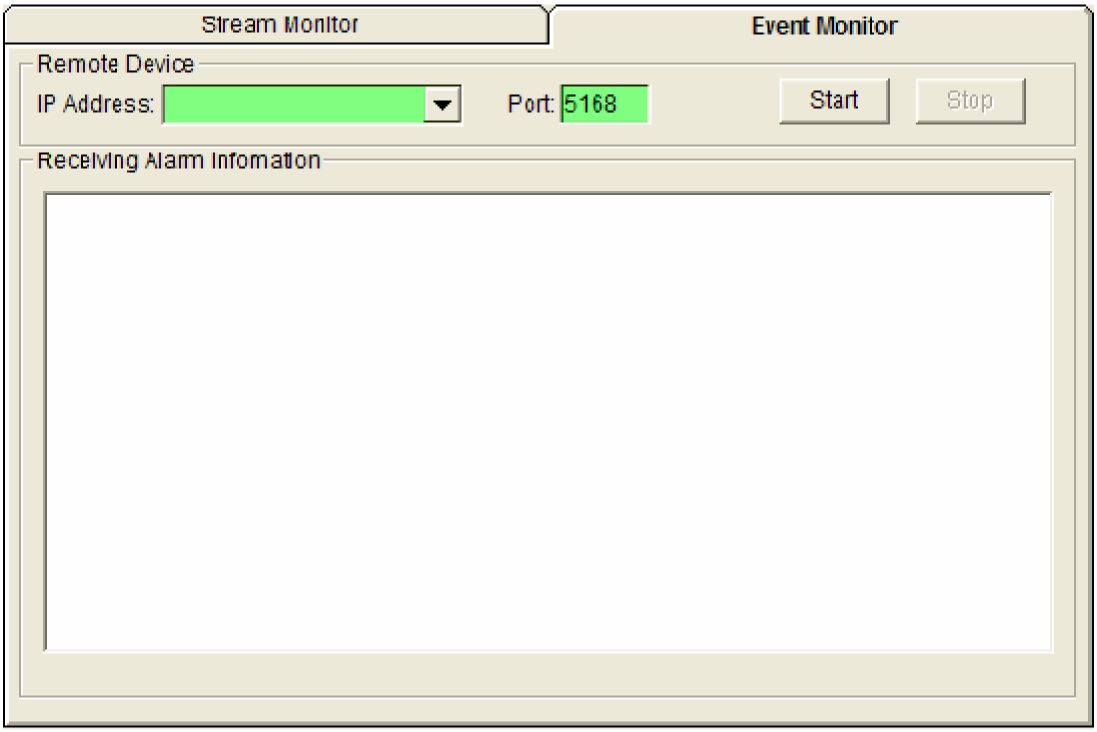
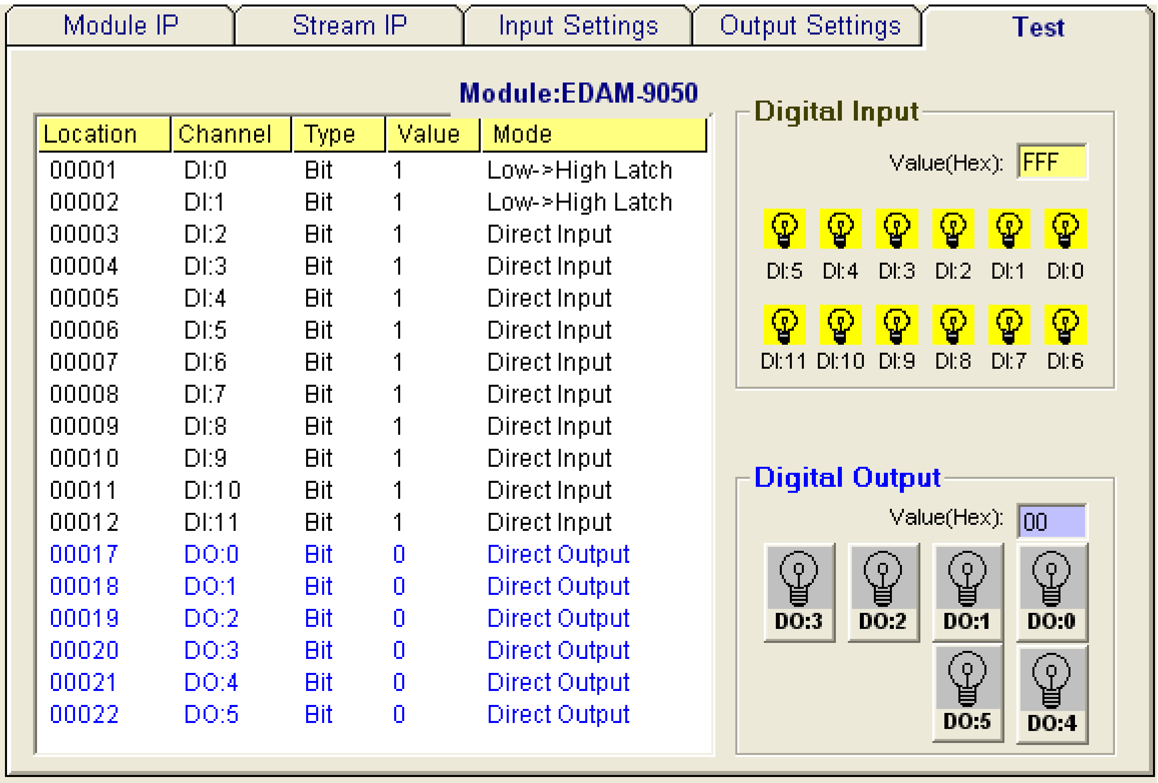


Figure 5-8 Event display

5.10 Digital I/O Module Settings

Selecting EDAM-9000 Digital Modules and select "**Test**" tab, user can read following information from the Utility.

5.10.1 Digital Test Tab



Module:EDAM-9050

Digital Input Value(Hex): FFF

Digital Output Value(Hex): 00

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 5-9 ModBus location and I/O status

Digital I/O Module Test tab"

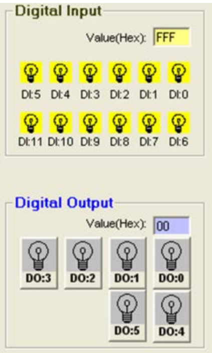
Location : Standard Modbus address. EDAM Ethernet I/O Utility shows the Modbus mapping address of each I/O channel. (Please refer to E9K_Modbus.pdf file) And the addresses will be the indexes for applying into the database of HMI or OPC Server.

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 Figure1 12 DI/O status display. You can read the Digital input status through the change of the indicator icons. Oppositely, you can write the digital output status through clicking the indicator icons.


Digital Input
DI:5 DI:4 DI:3 DI:2 DI:1 DI:0
DI:11 DI:10 DI:9 DI:8 DI:7 DI:6
Digital Output
DO:3 DO:2 DO:1 DO:0
DO:5 DO:4

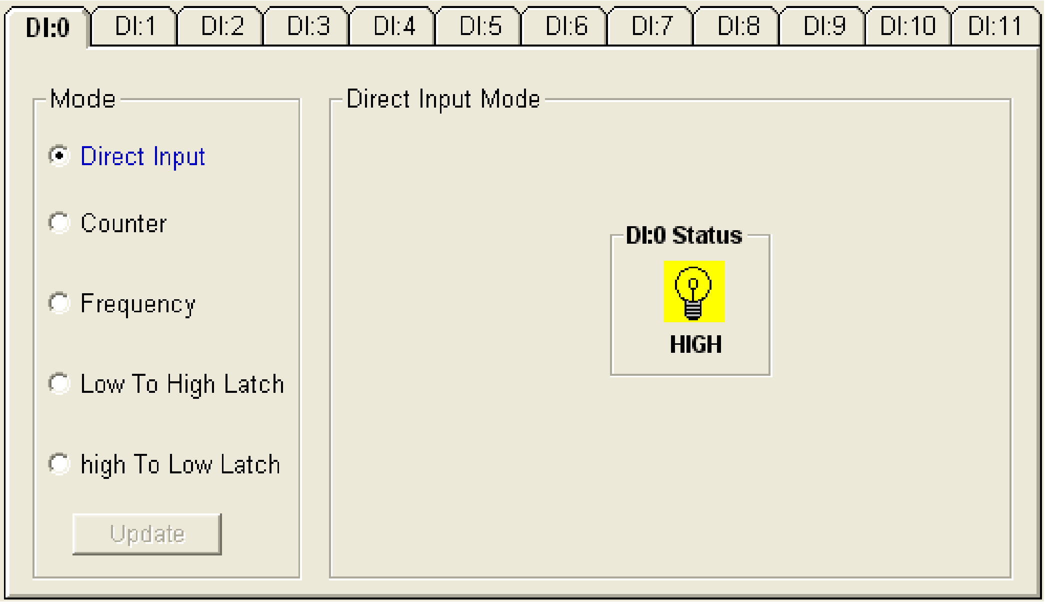
Value(Hex): FFF

Value(Hex): 00

Figure 5-10 DI/O status display

5.10.2 Digital Input Settings Tab

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



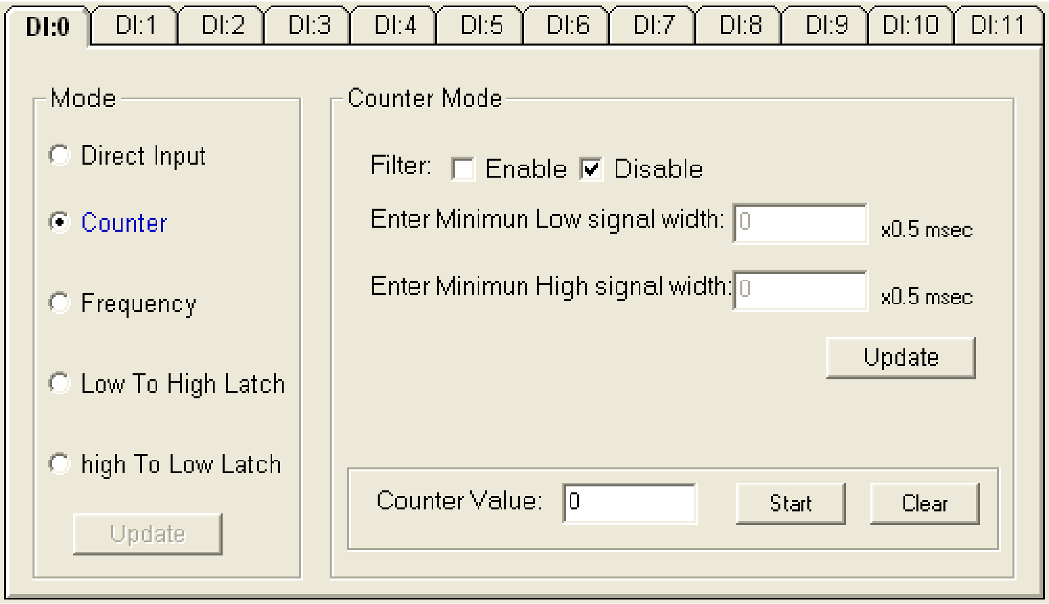
Mode

● Direct Input
○ Counter
○ Frequency
○ Low To High Latch
○ high To Low Latch

Direct Input Mode

DI:0 Status
HIGH

Figure 5-11 Direct input mode



Mode

○ Direct Input
● Counter
○ Frequency
○ Low To High Latch
○ high To Low Latch

Counter Mode

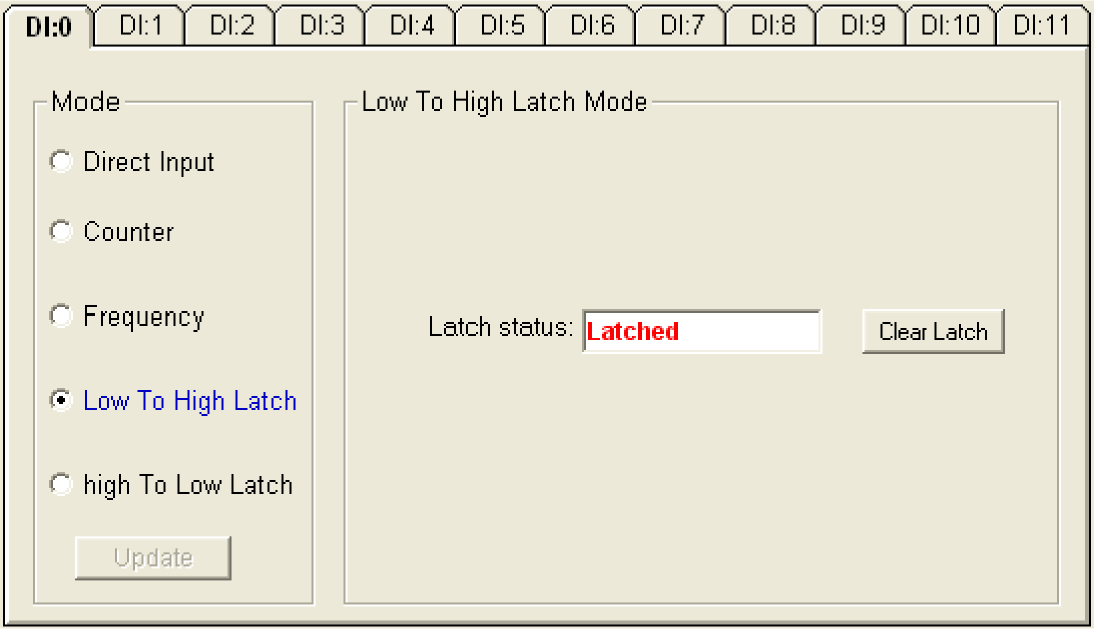
Filter: □ Enable ☑ Disable

Enter Minimun Low signal width: 0 x0.5 msec

Enter Minimun High signal width: 0 x0.5 msec

Counter Value: 0

Figure 5-12 Counter mode



Mode

○ Direct Input

○ Counter

○ Frequency

● Low To High Latch

○ high To Low Latch

Low To High Latch Mode

Latch status: Latched

Figure 5-13 Input latch mode

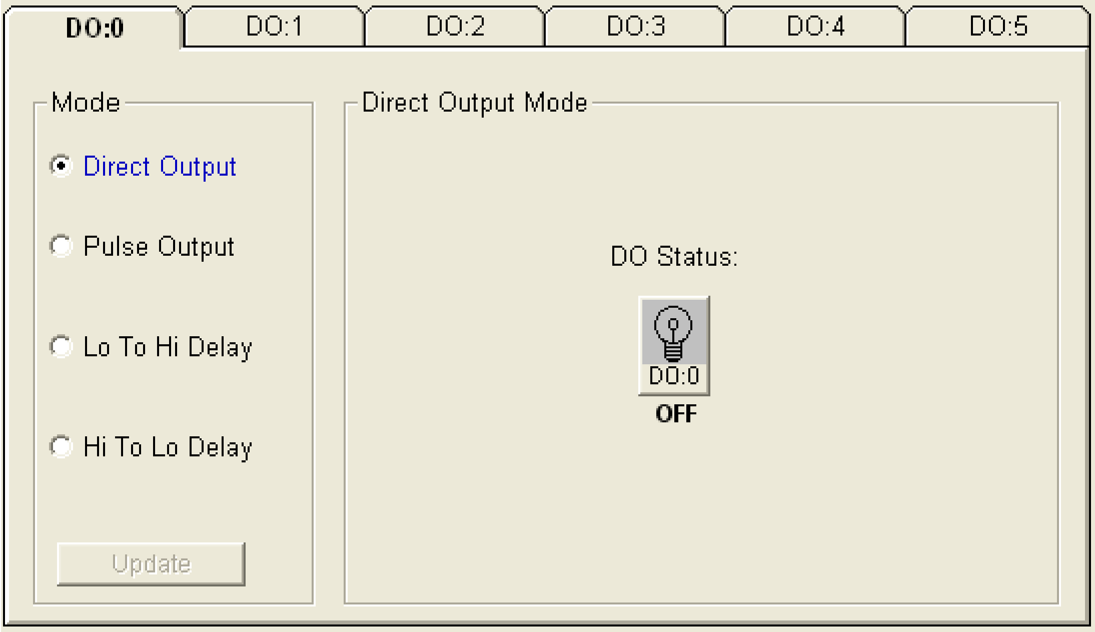
Note:

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

If necessary, users could invert the original single for flexible operation needs.

5.10.3 Digital Output Settings Tab

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



Mode

● Direct Output

○ Pulse Output

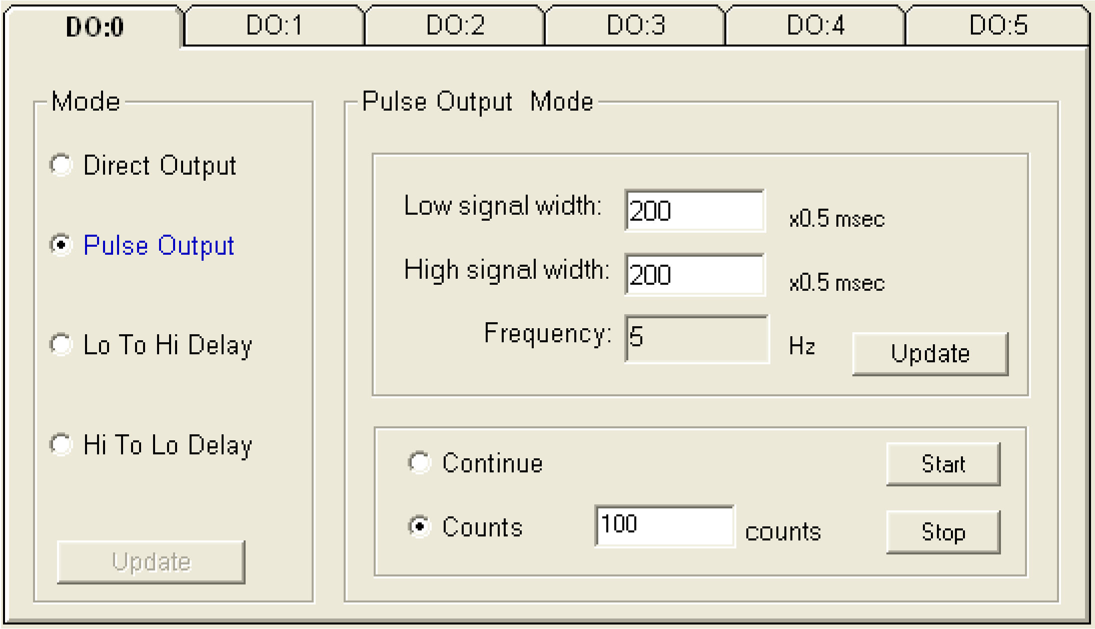
○ Lo To Hi Delay

○ Hi To Lo Delay

Direct Output Mode

DO Status: DO:0 OFF

Figure 5-14 direct output mode



Mode

○ Direct Output

● Pulse Output

○ Lo To Hi Delay

○ Hi To Lo Delay

Pulse Output Mode

Low signal width: 200 x0.5 msec

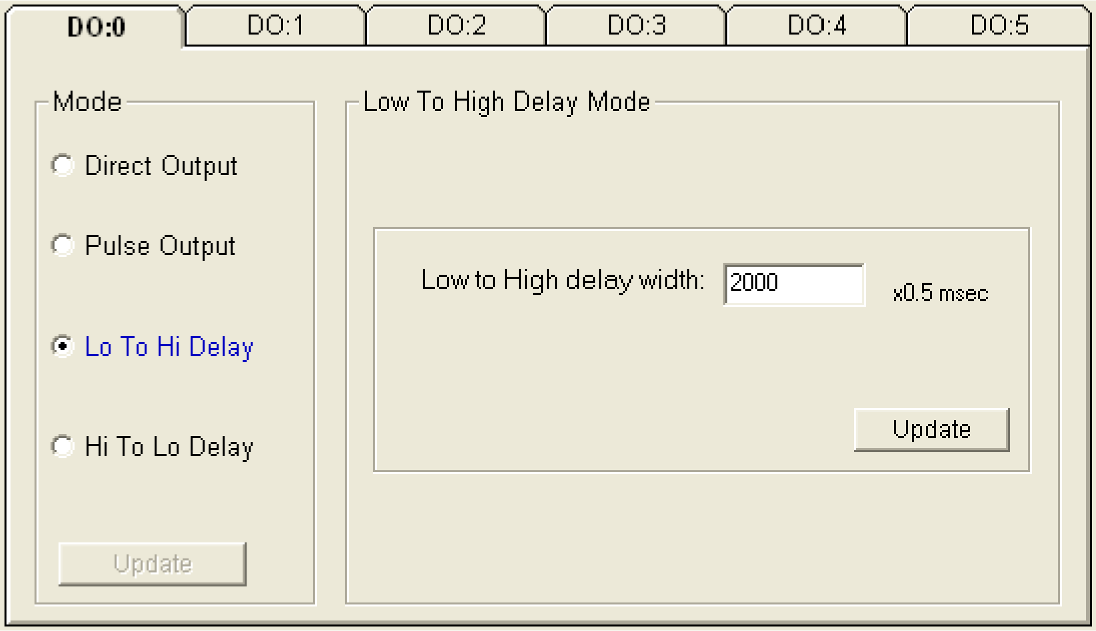
High signal width: 200 x0.5 msec

Frequency: 5 Hz

○ Continue

● Counts 100 counts

Figure 5-15 Pulse output mode



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

Mode

- ○ Direct Output
- ○ Pulse Output
- ● Lo To Hi Delay
- ○ Hi To Lo Delay

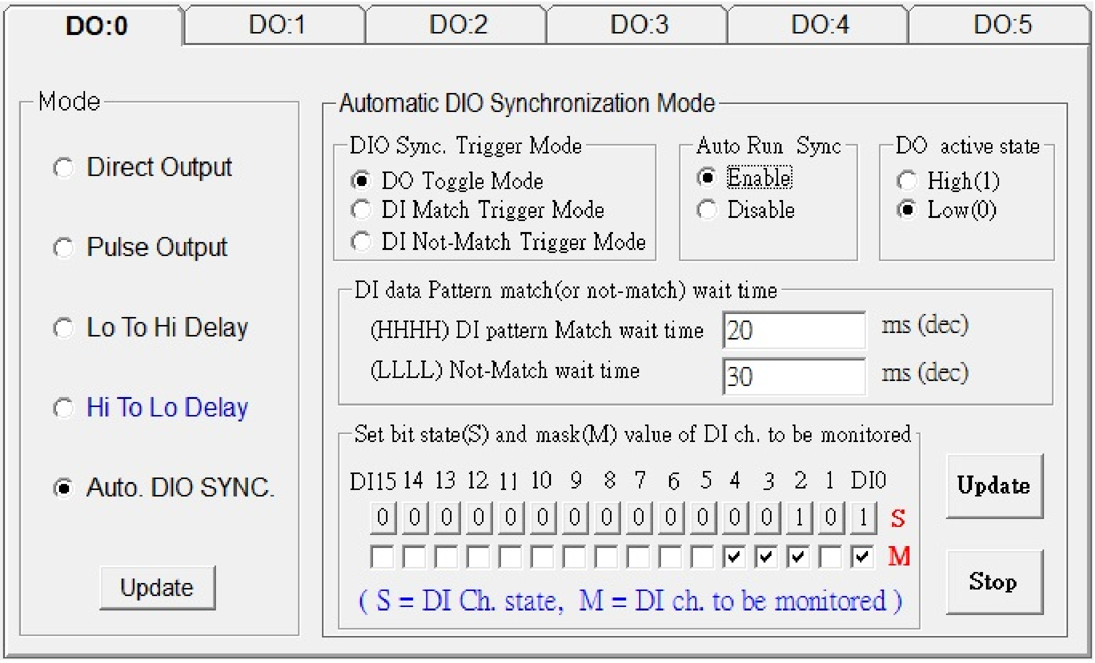
Update

Low To High Delay Mode

Low to High delay width: 2000 x0.5 msec

Update

Figure 5-16 Low to High Delay mode



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

Mode

- ○ Direct Output
- ○ Pulse Output
- ○ Lo To Hi Delay
- ○ Hi To Lo Delay
- ● Auto. DIO SYNC.

Update

Automatic DIO Synchronization Mode

DIO Sync. Trigger Mode

- ● DO Toggle Mode
- ○ DI Match Trigger Mode
- ○ DI Not-Match Trigger Mode

Auto Run Sync

- ● Enable
- ○ Disable

DO active state

- ○ High(1)
- ● Low(0)

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

(HHHH) DI pattern Match wait time 20 ms (dec)

(LLLL) Not-Match wait time 30 ms (dec)

Set bit state(S) and mask(M) value of DI ch. to be monitored

DI15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	DI0	
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	S
□	□	□	□	□	□	□	□	□	□	□	☑	☑	☑	□	☑	M

(S = DI Ch. state, M = DI ch. to be monitored)

Update

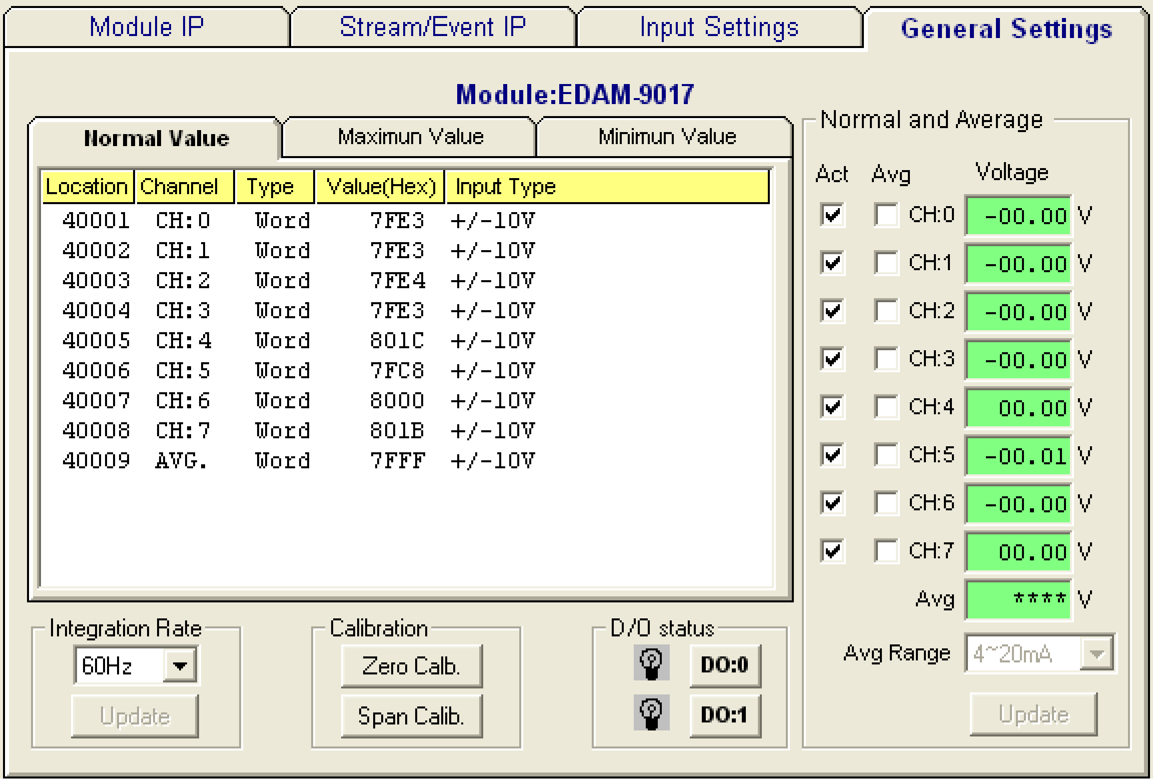
Stop

Figure 5-17 DIO SYNC. mode (EDAM9000A only and BIOS version: 6.070 or later)

5.11 Analog Input Module

5.11.1 Analog Input Test Tab

Selecting EDAM-9000 analog input Modules includes EDAM-9017 and select "**General Settings"** tab, user can read following information from the Utility.



Module:EDAM-9017

Location	Channel	Type	Value(Hex)	Input Type
40001	CH:0	Word	7FE3	+/-10V
40002	CH:1	Word	7FE3	+/-10V
40003	CH:2	Word	7FE4	+/-10V
40004	CH:3	Word	7FE3	+/-10V
40005	CH:4	Word	801C	+/-10V
40006	CH:5	Word	7FC8	+/-10V
40007	CH:6	Word	8000	+/-10V
40008	CH:7	Word	801B	+/-10V
40009	AVG.	Word	7FFF	+/-10V

Normal and Average

Act	Avg		Voltage	
☑	☐	CH:0	-00.00	V
☑	☐	CH:1	-00.00	V
☑	☐	CH:2	-00.00	V
☑	☐	CH:3	-00.00	V
☑	☐	CH:4	00.00	V
☑	☐	CH:5	-00.01	V
☑	☐	CH:6	-00.00	V
☑	☐	CH:7	00.00	V
		Avg	****	V

Avg Range: 4~20mA

Integration Rate: 60Hz

Calibration

Figure 5-18 ModBus location and analog value

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

Channel : The channel number

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

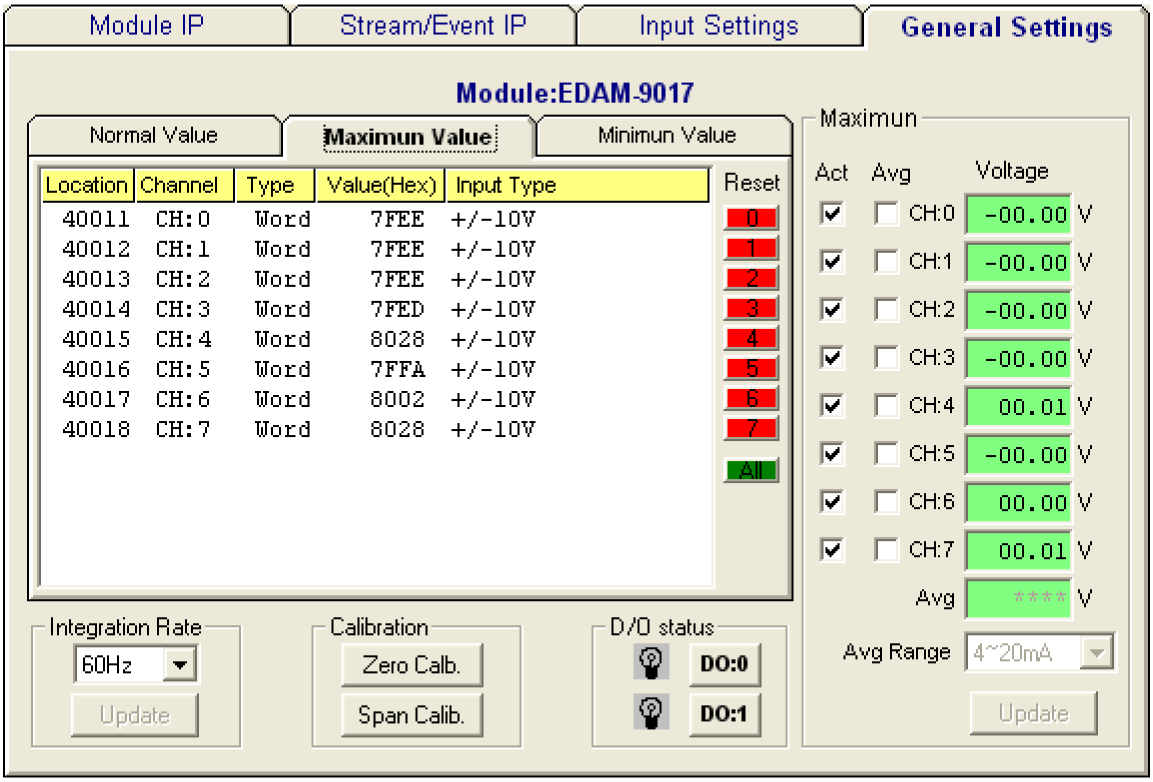
Value : The value of each channel. Windows Utility provides both decimal and hexadecimal values used for different applications.

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

Note:

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 specific range with engineer unit.

To provide users more valuable information, the EDAM-9000 analog modules have designed with calculation functions, includes Maximum, Minimum, and Average values of individual channels. Click the Maximum value tab, you will see the historical maximum values in each channel unless to press the against "**Reset**" buttons.



Module:EDAM-9017

Location	Channel	Type	Value(Hex)	Input Type
40011	CH:0	Word	7FEE	+/-10V
40012	CH:1	Word	7FEE	+/-10V
40013	CH:2	Word	7FEE	+/-10V
40014	CH:3	Word	7FED	+/-10V
40015	CH:4	Word	8028	+/-10V
40016	CH:5	Word	7FFA	+/-10V
40017	CH:6	Word	8002	+/-10V
40018	CH:7	Word	8028	+/-10V

Integration Rate: 60Hz

Calibration

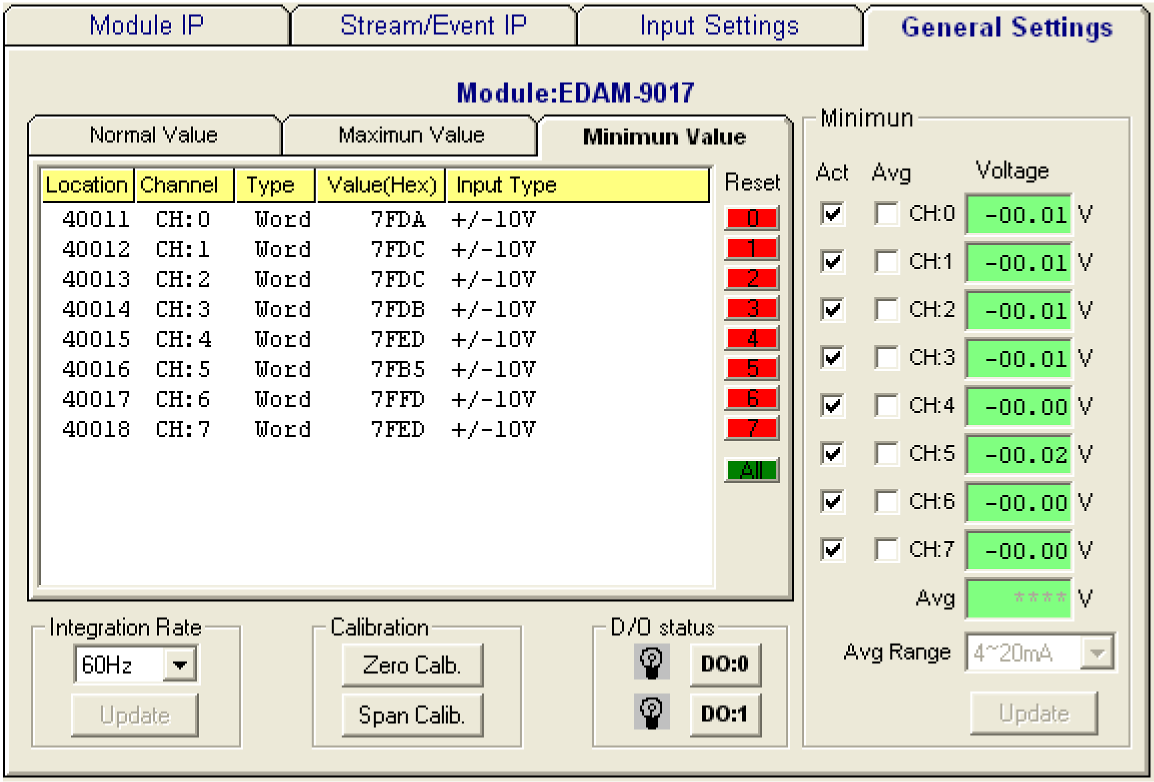
D/O status

Maximun

Act	Avg		Voltage	
☑	□	CH:0	-00.00	V
☑	□	CH:1	-00.00	V
☑	□	CH:2	-00.00	V
☑	□	CH:3	-00.00	V
☑	□	CH:4	00.01	V
☑	□	CH:5	-00.00	V
☑	□	CH:6	00.00	V
☑	□	CH:7	00.01	V
		Avg	****	V

Avg Range: 4~20mA

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



Module:EDAM-9017

Location	Channel	Type	Value(Hex)	Input Type
40011	CH:0	Word	7FDA	+/-10V
40012	CH:1	Word	7FDC	+/-10V
40013	CH:2	Word	7FDC	+/-10V
40014	CH:3	Word	7FDB	+/-10V
40015	CH:4	Word	7FED	+/-10V
40016	CH:5	Word	7FB5	+/-10V
40017	CH:6	Word	7FFD	+/-10V
40018	CH:7	Word	7FED	+/-10V

Integration Rate: 60Hz

Calibration

D/O status

Minimun

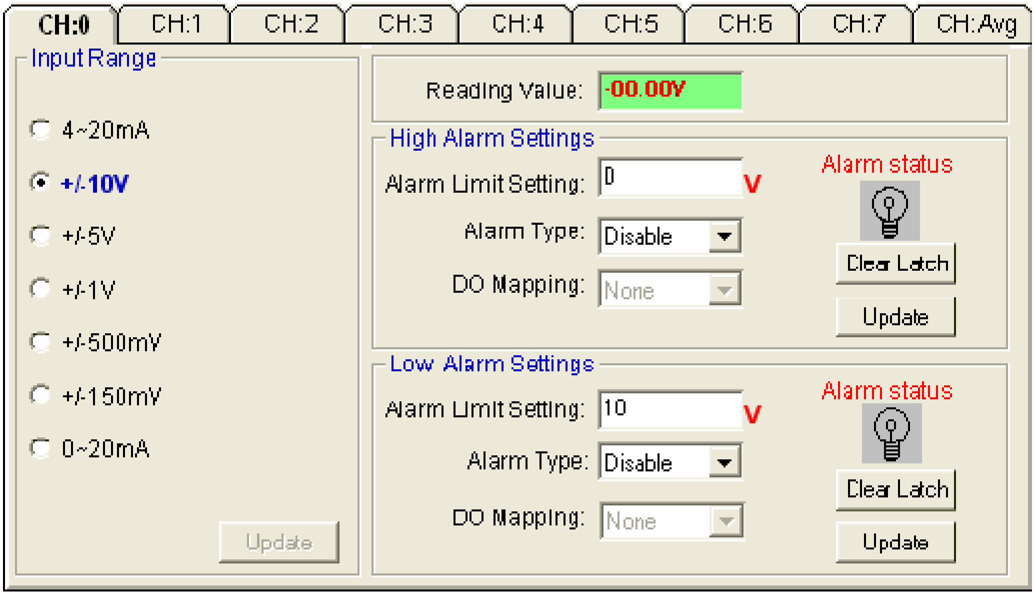
Act	Avg		Voltage	
☑	□	CH:0	-00.01	V
☑	□	CH:1	-00.01	V
☑	□	CH:2	-00.01	V
☑	□	CH:3	-00.01	V
☑	□	CH:4	-00.00	V
☑	□	CH:5	-00.02	V
☑	□	CH:6	-00.00	V
☑	□	CH:7	-00.00	V
		Avg	****	V

Avg Range: 4~20mA

5.11.2 Analog Input Setting Tab

♦ **Analog Input Type Setting**

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



Input Range

○ 4~20mA
● +/-10V
○ +/-5V
○ +/-1V
○ +/-500mV
○ +/-150mV
○ 0~20mA

Reading Value: -00.00V

High Alarm Settings

Alarm Limit Setting: 0 V
Alarm Type: Disable
DO Mapping: None

Low Alarm Settings

Alarm Limit Setting: 10 V
Alarm Type: Disable
DO Mapping: None

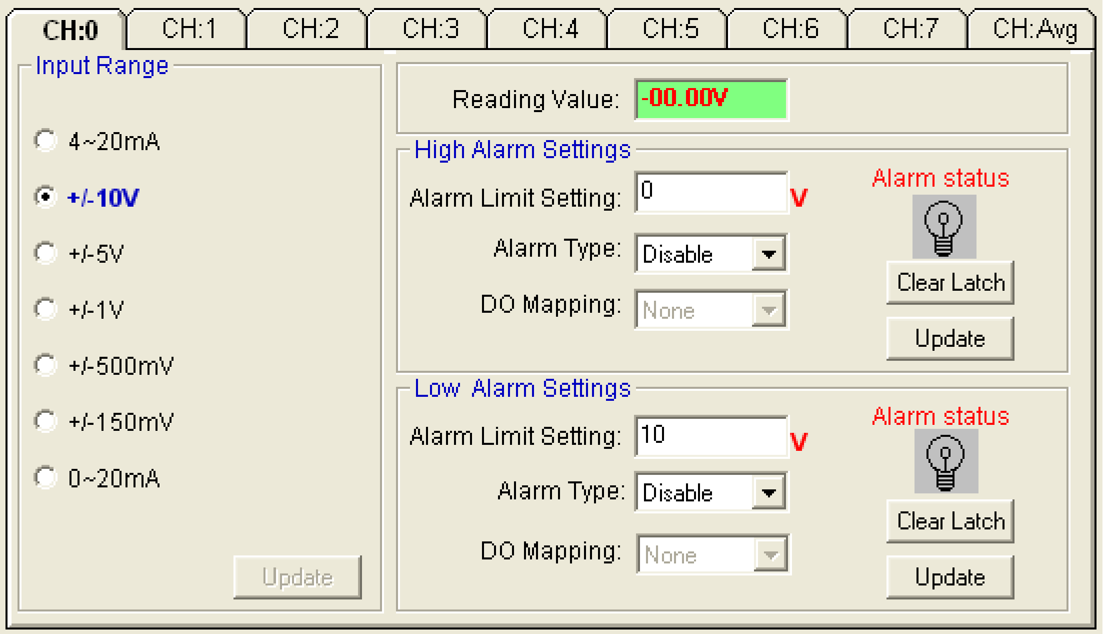
Figure 5-19 Input type setting

Note:

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

♦ **Analog Input 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 EDM-9017.



Input Range

○ 4~20mA
● +/-10V
○ +/-5V
○ +/-1V
○ +/-500mV
○ +/-150mV
○ 0~20mA

Reading Value: -00.00V

High Alarm Settings

Alarm Limit Setting: 0 V
Alarm Type: Disable
DO Mapping: None

Low Alarm Settings

Alarm Limit Setting: 10 V
Alarm Type: Disable
DO Mapping: None

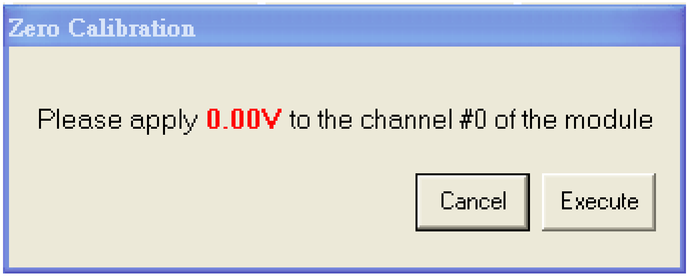
Figure 5-20 Alarm Setting

5.11.3 Analog Module Calibrations

Calibration is to adjust the accuracy of EDAM 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 EDAM-9017 is the first released analog module.

♦ **Zero Calibration**

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

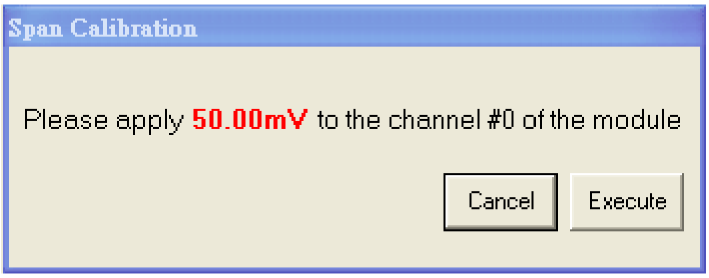


Zero Calibration

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

♦ **Span Calibration**

1. Follow the same procedure of zero calibration
2. Use a precision voltage source to apply a calibration voltage to the V+ and V- terminals of the EDAM-9017 module.
3. Click the Execute button.



Span Calibration

Please apply 50.00mV to the channel #0 of the module

Chapter 6 What is TCPDAQ ActiveX Control?

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

With TCPDAQ ActiveX Control, you can perform versatile I/O operations to control your Inlog EDAM-9000 module series.

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

Use the Inlog EDAM-9000 utility to configure the modules that is attached to your computer.

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

6.1 Installing the TCPDAQ ActiveX Controls

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

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

6.2 Building TCPDAQ ActiveX Control with Various Tools

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

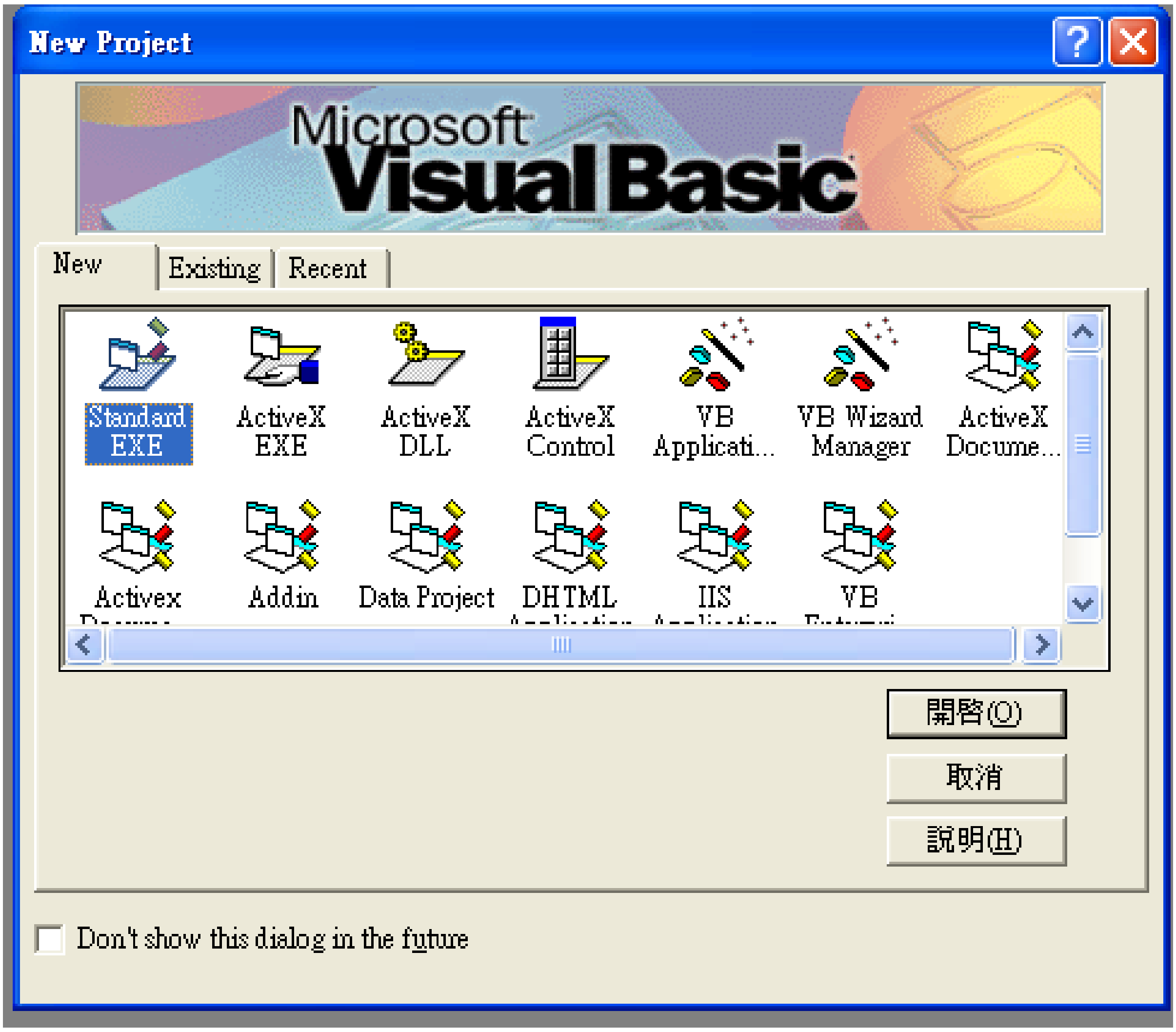
- ♦ Microsoft Visual C++ version 6.0 (SP5)
- ♦ Microsoft Visual Basic version 6.0 (SP5)
- ♦ Borland Delphi version 4.0 (with the Delphi 6 Update Pack fixes for ActiveX installed)
- ♦ Borland C++ Builder version 5.0

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

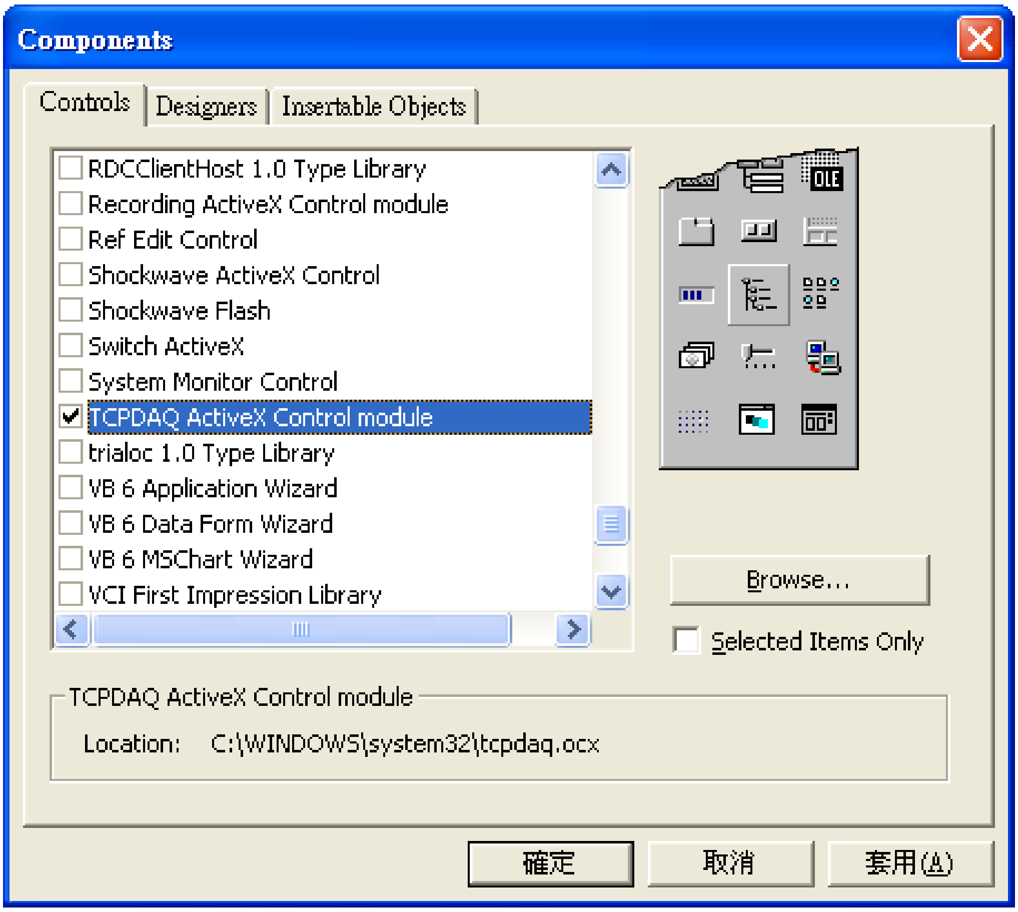
Note: For Borland Delphi 6, the Delphi 6 Update Pack fixes for ActiveX must be installed.

6.3 Building TCPDAQ Applications with Visual Basic

- Start Visual Basic.



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



Components

Controls | Designers | Insertable Objects

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

Browse...

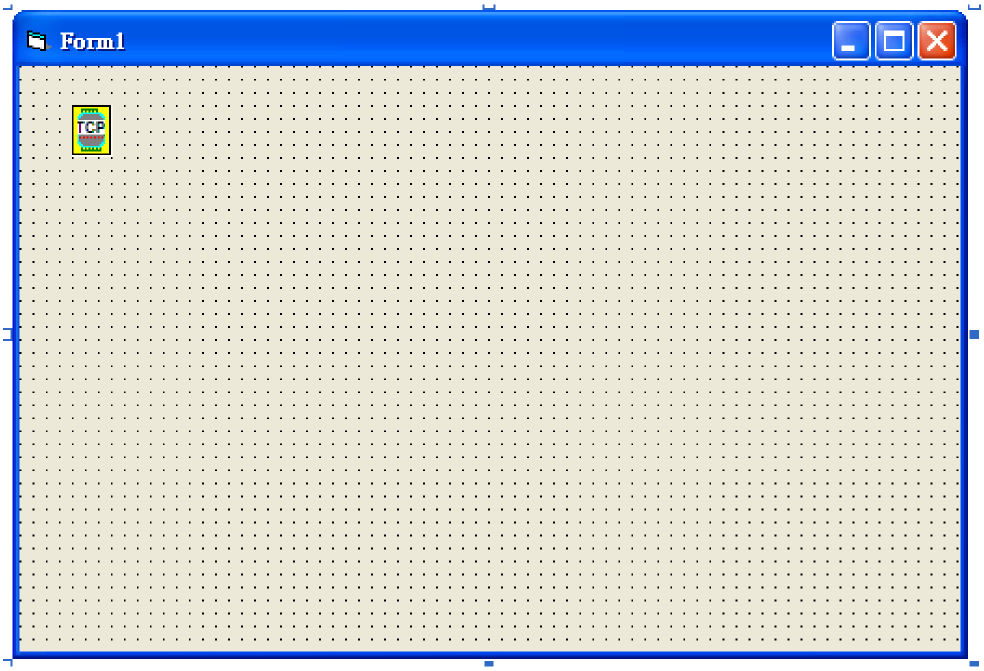
□ Selected Items Only

TCPDAQ ActiveX Control module

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

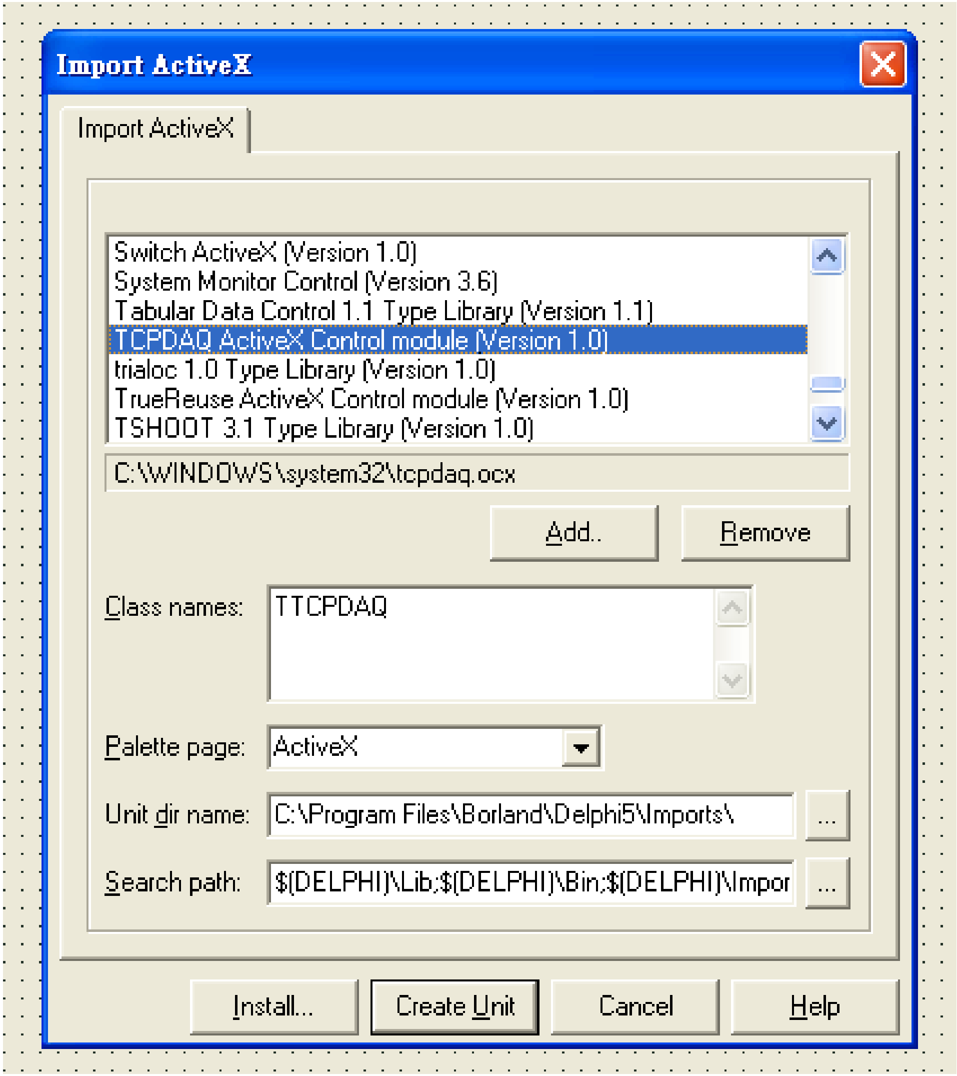
♦ Place a TCPDAQ control from the Toolbox on the form. Use the default name.

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



6.4 Building TCPDAQ Applications with Delphi

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


Import ActiveX

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

C:\WINDOWS\system32\tcpdaq.ocx

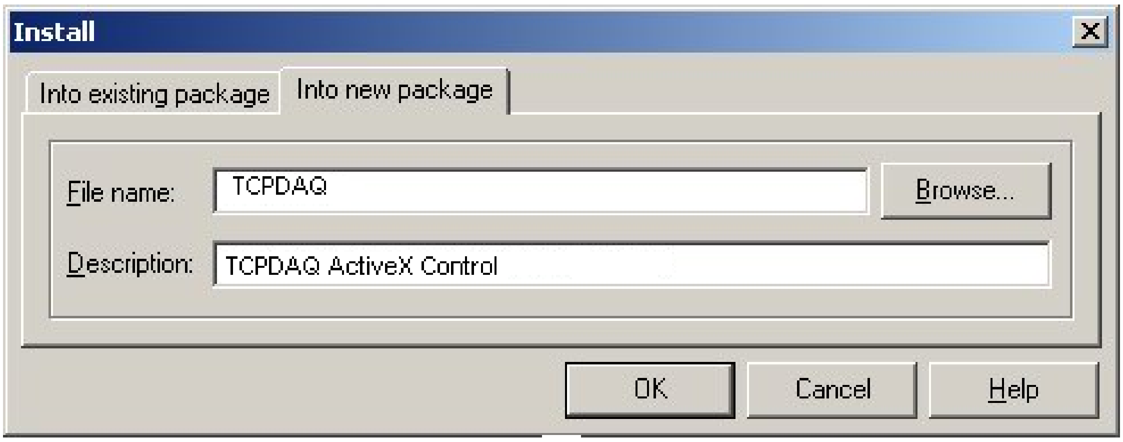
Class names: TTCPDAQ

Palette page: ActiveX

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

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

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

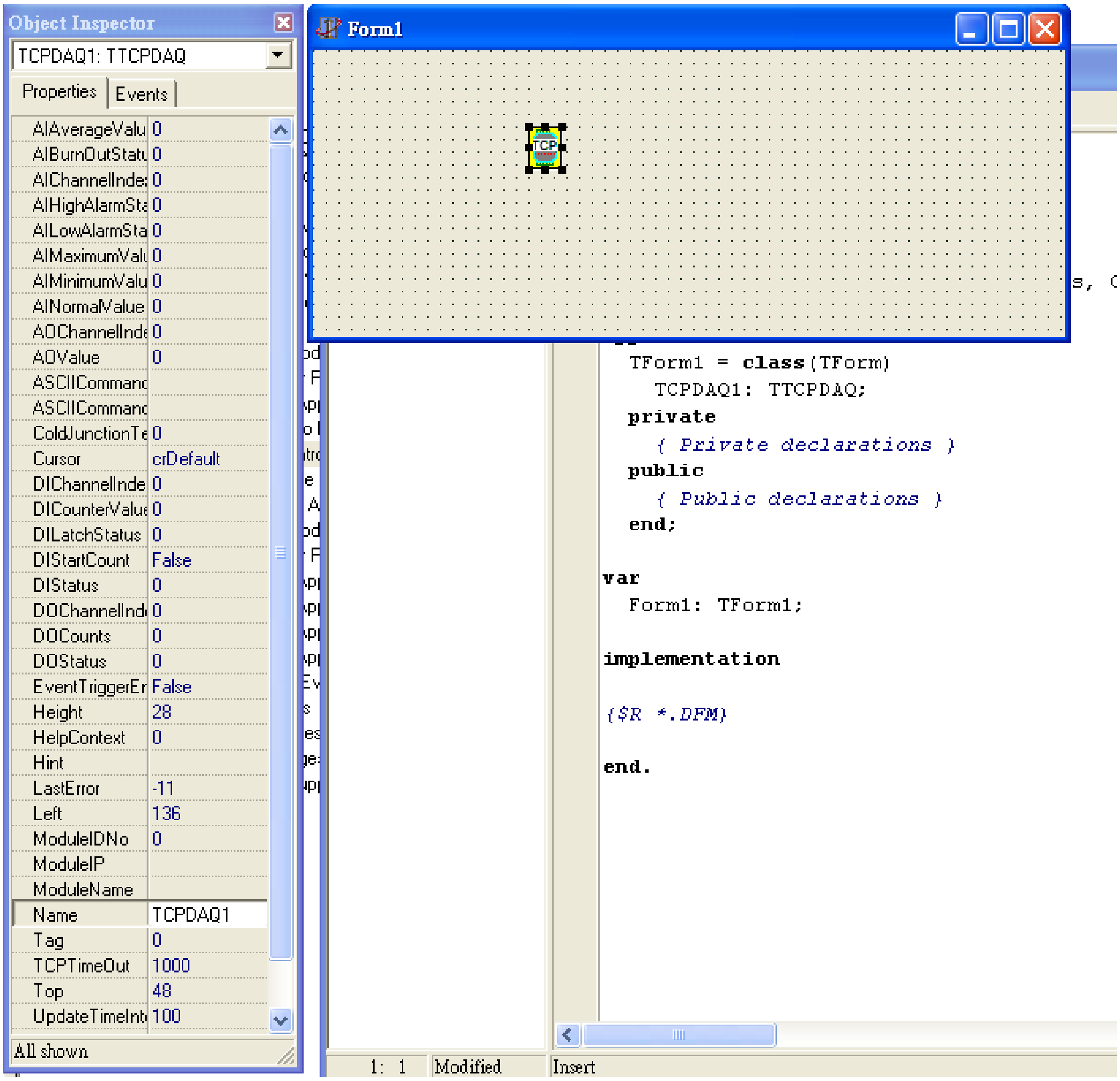

Install

Into existing package | Into new package

File name: TCPDAQ

Description: TCPDAQ ActiveX Control

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



TCPDAQ1: TTCPDAQ

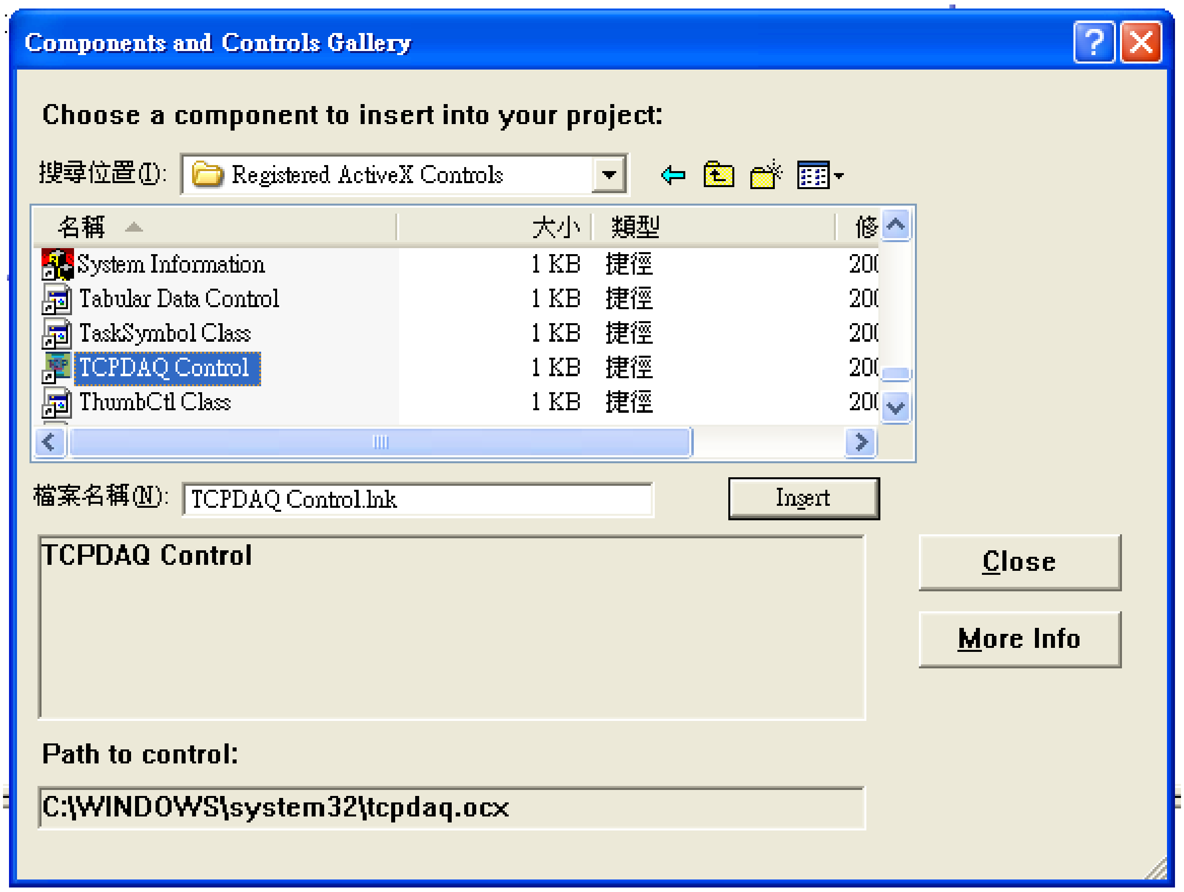
AIAverageValu	0
AIBurnOutStatu	0
AIChannelInde:	0
AIHighAlarmSta	0
AILowAlarmSta	0
AIMaximumValu	0
AIMinimumValu	0
AINormalValue	0
AOChannelInde	0
AOValue	0
ASCIICommanc	
ASCIICommanc	
ColdJunctionTe	0
Cursor	crDefault
DIChannelInde	0
DICounterValue	0
DILatchStatus	0
DIStartCount	False
DIStatus	0
DOChannelInd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

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

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

1: 1 Modified Insert

6.5 Building TCPDAQ Applications with Visual C++

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



Components and Controls Gallery

Choose a component to insert into your project:

搜尋位置(I): Registered ActiveX Controls

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

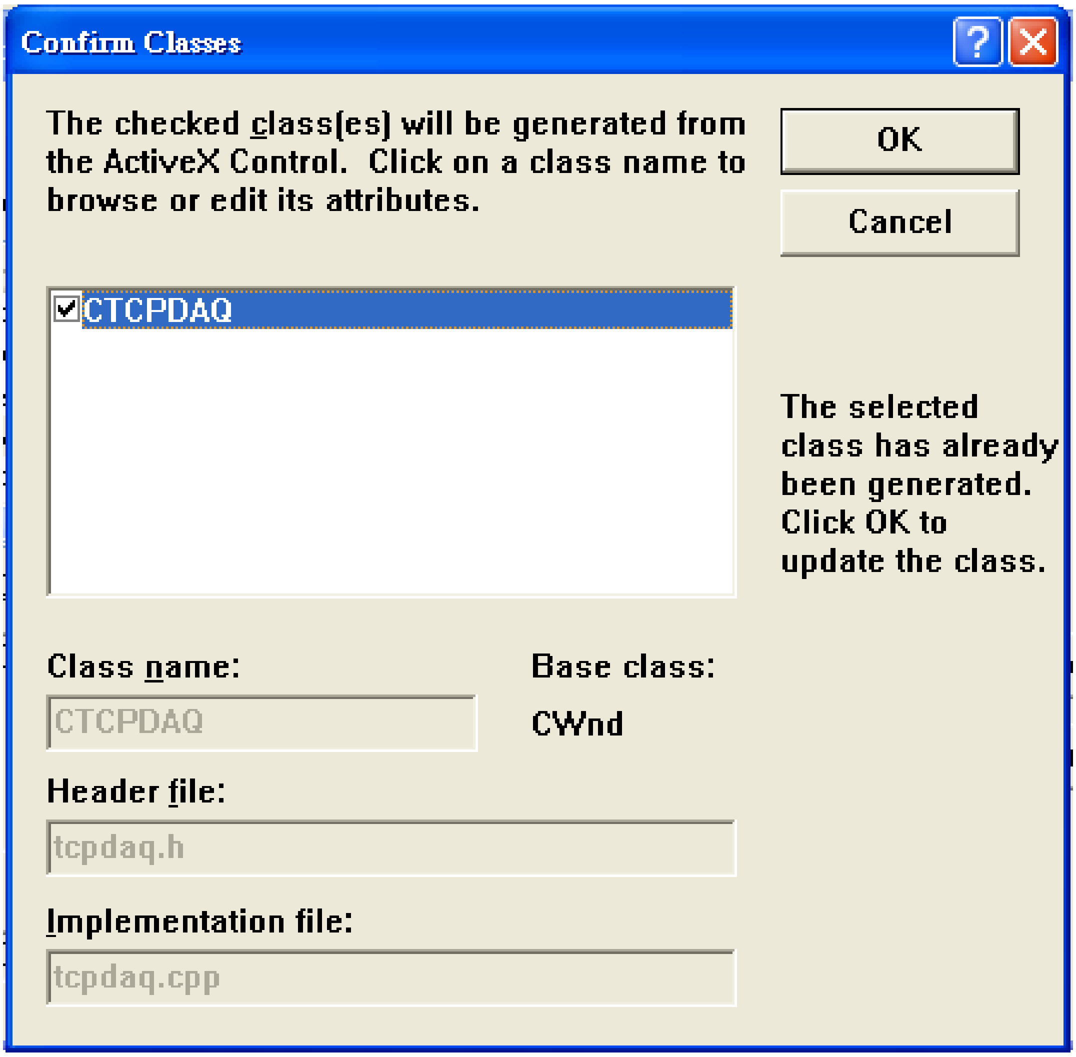
檔案名稱(N): TCPDAQ Control.lnk

TCPDAQ Control

Path to control:

C:\WINDOWS\system32\tcpdaq.ocx

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



Confirm Classes

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

☑ CTCPDAQ

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

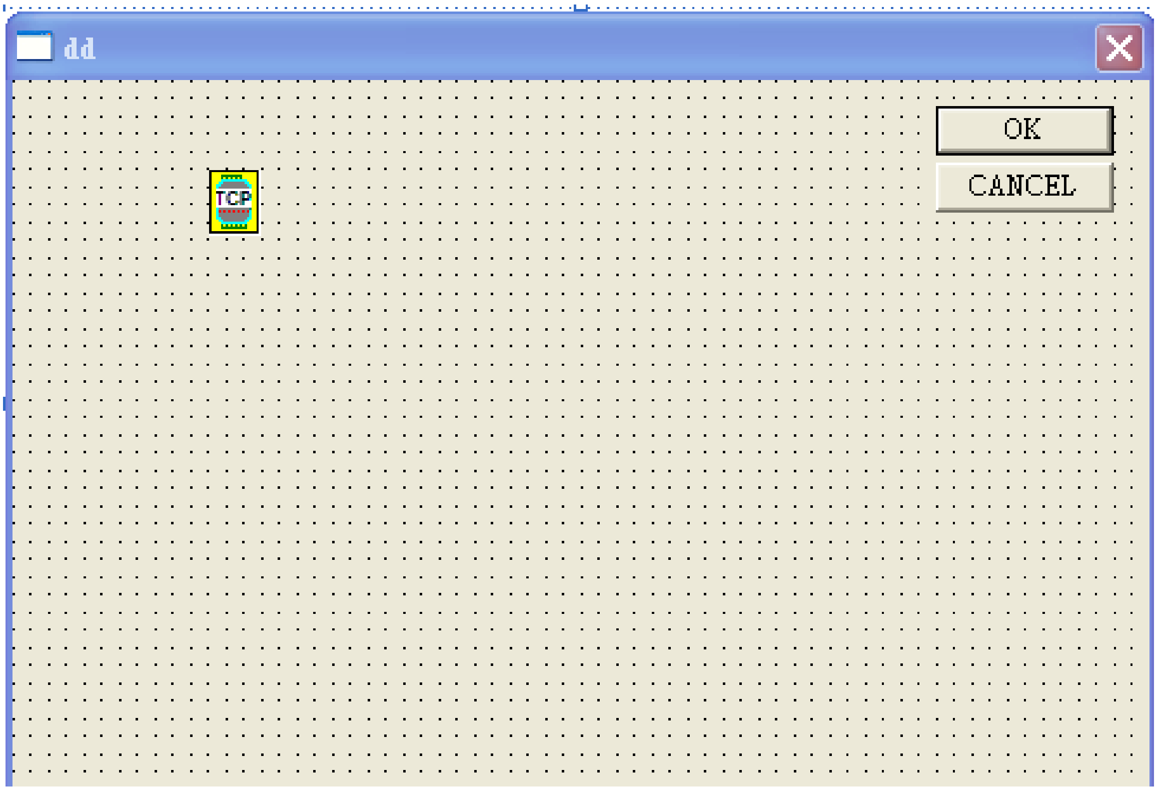
Class name: CTCPDAQ

Base class: CWnd

Header file: tcpdaq.h

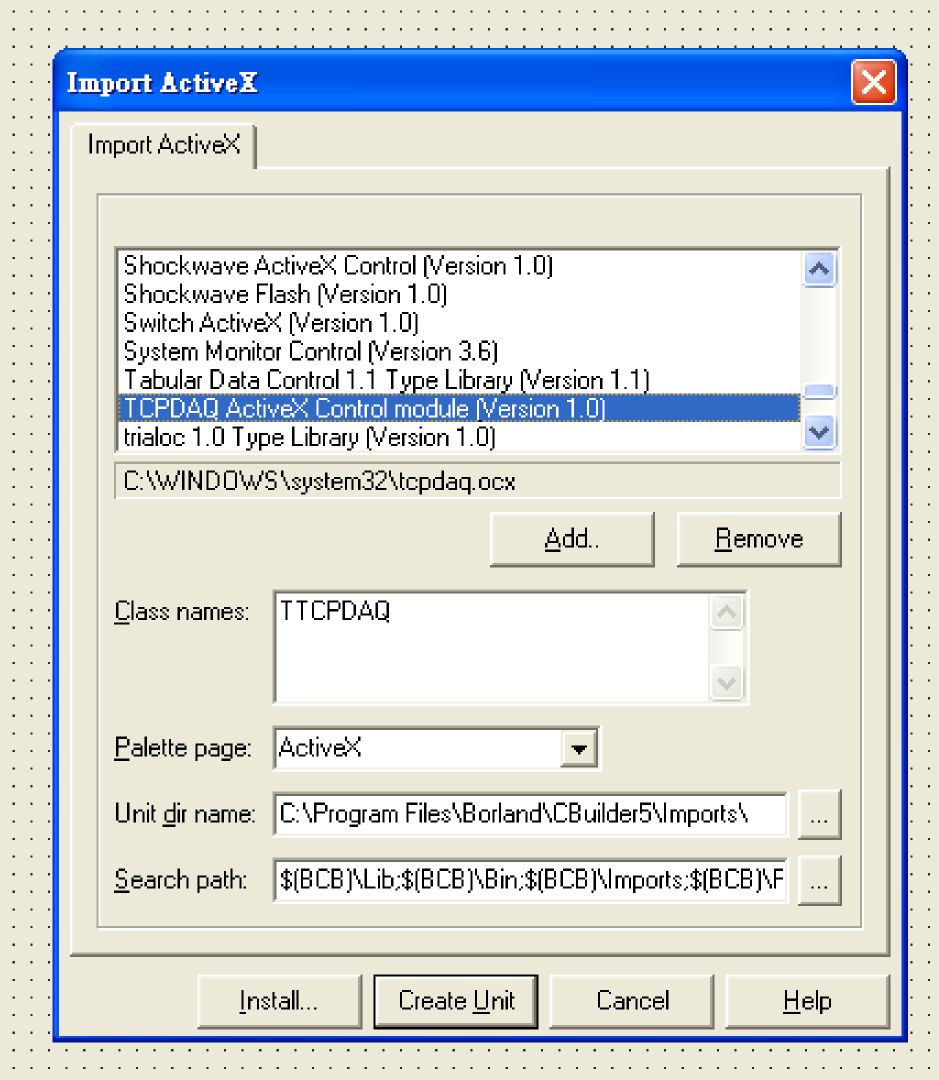
Implementation file: tcpdaq.cpp

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



6.6 Building TCPDAQ Applications with Borland C++ Builder

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



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

C:\WINDOWS\system32\tcpdaq.ocx

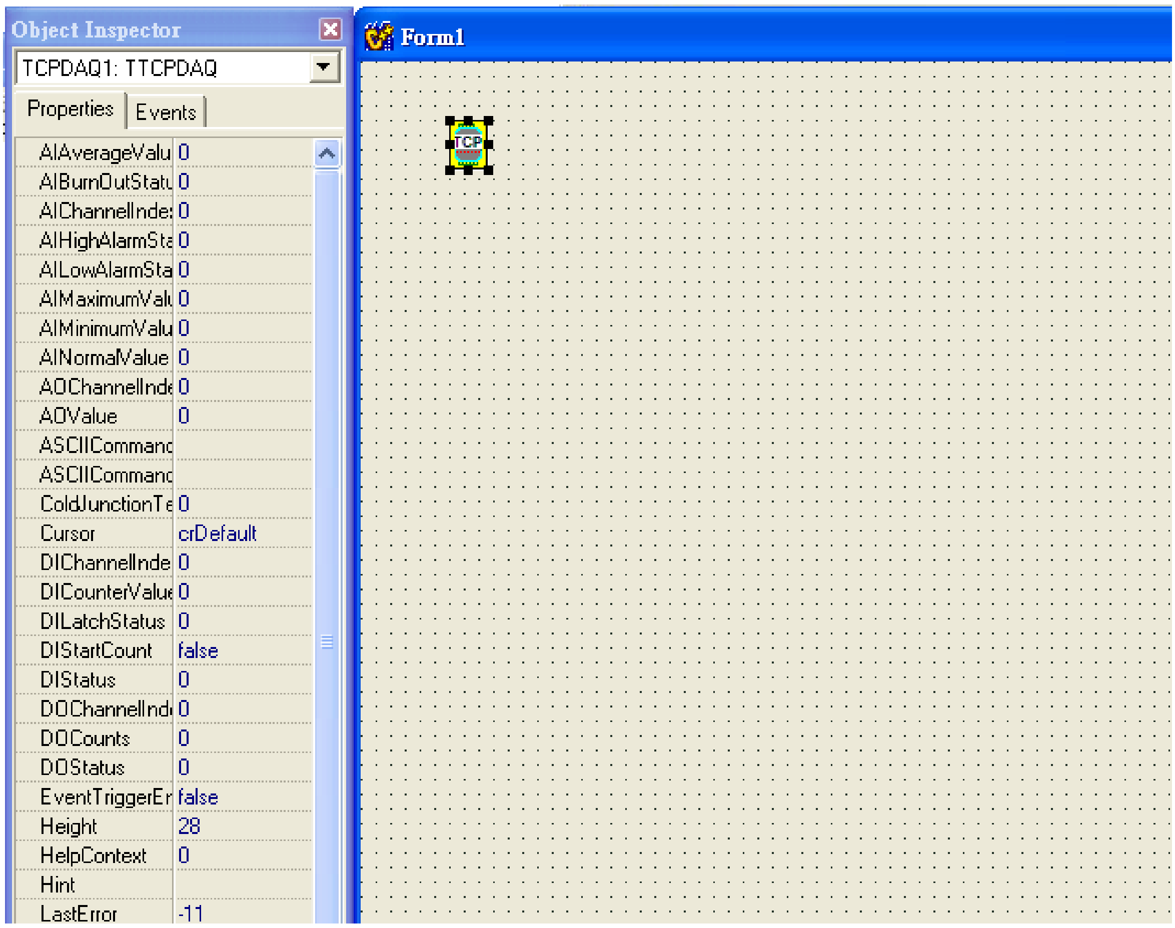
Class names: TTCPDAQ

Palette page: ActiveX

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

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

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



TCPDAQ1: TTCPDAQ

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

6.7 Properties of TCPDAQ ActiveX Control

Name	Type	Description	Avaliable Model(s)
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
ColdJunctionTemperature	double	Return the cold junction temperature	9019
DIChannelIndex	short	Specifies the digital input channel to perform other DI properties read/write operation.	All DIO models
DIounterValue	long	Return the counting value for the specific DI channel which functions in "Count/Frequency mode"	All DIO models
DILatchStatus	short	Return the latch status for the specific DI channel which functions in "Lo-Hi/Hi-Lo latch mode" (1=Latched, 0=No latched)	All DIO models
DIStartCount	boolean	Start/stop counting for the specific DI channel which functions in "Count/Frequency mode" (True=Start, 0=Stop)	All DIO models
DIStatus	short	Return the status for the specific DI channel which functions in "DI mode" (1=Active, 0=Inactive)	All DIO models
DOChannelIndex	short	Specifies the digital output channel to perform other DO properties read/write operation.	9017,9019 and All DIO models
DOCount	long	Set the output count value for the specific DO channel which functions in "Pulse output mode"	All DIO models

DOStatus	short	Return/set the status for the specific DO channel which functions in “D/O mode” (1=Active, 0=Inactive)	9017,9019 and All DIO models
EventTriggerEnable	boolean	Enable/disable event trigger mode (True=Enable, False=Disable)	All models
LastError	short	Return the Error code of operation	All models
MoudleIDNo	short	Return the module ID number	All models
ModuleIP	string	Set the remote module IP address	All models
ModuelName	string	Return the module name	All models
TCPTimeOut	long	Return/set the TCP/IP Timeout (ms)	All models
UpdateTimeInterval	long	Return/set data update time interval(ms)	All models

6.8 Methods of TCPDAQ ActiveX Control

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

6.9 Events of TCPDAQ ActiveX Control

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

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

6.10 Building TCPDAQ ActiveX Applications with Various Development Tools

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

Chapter 7 TCPDAQ DLL API

7.1 Common Functions

NO.	Function Name	Description	Sec.
1	TCP_Open	To initiate the TCPDAQ.dll to use.	7.6.1
2	TCP_Close	To terminates use of the TCPDAQ.dll.	7.6.2
3	TCP_Connect	To create a Window TCP socket then establishing a connection to a specific EDAM-9000	0
4	TCP_Disconnect	Disconnecting the Window TCP socket from all EDAM-9000 modules	7.6.4
5	TCP_ModuleDisconnect	Disconnecting the Window TCP socket from a specific EDAM-9000	0
6	TCP_SendData	Send data to a specific EDAM-9000 module	7.6.6
7	TCP_RecvData	Receive data to a specific EDAM-9000 module	7.6.7
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 EDAM-9000 and receiving the response from EDAM-9000	7.6.8
9	UDP_Connect	To create a Window UDP socket then establishing a connection to a specific EDAM-9000	7.6.9
10	UDP_Disconnect	Disconnecting the Window UDP socket from all EDAM-9000 modules	0
11	UDP_ModuleDisconnect	Disconnecting the Window UDP socket from a specific EDAM-9000	7.6.11
12	UDP_SendData	Send data to a specific EDAM-9000 module	7.6.12
13	UDP_RecvData	Receive data to a specific EDAM-9000 module	7.6.13
14	UDP_SendReceiveASCcmd	Direct send an ASCII format string as a command, and receive the response from EDAM-9000	7.6.14
15	TCP_GetModuleIPinfo	Return module IP information of a specific module	0
16	TCP_GetModuleID	Return module ID number of a specific module	7.6.16
17	TCP_GetIPFromID	Return IP address of a specific module ID number	7.6.17
18	TCP_ScanOnLineModules	Scan all on-line EDAM-9000 modules	7.6.18
19	TCP_GetDLLVersion	Return the DLL's version, that is the version of TCPDAQ.DLL	7.6.19
20	TCP_GetModuleNo	Return the module name of a specific IP address	7.6.20
21	TCP_GetLastError	Return the error code of the latest called function	7.6.21
22	TCP_PingIP	Ping to Remote IP address	7.6.22

7.2 Stream/Event Functions

TCP_StartStream	To instruct the PC to start to receive stream data that coming from EDAM-9000	7.6.23
TCP_StopStream	To instruct the PC to stop receiving stream data from all modules	7.6.24
TCP_ReadStreamData	To receive stream data that coming from the specific EDAM-9000	7.6.25
TCP_StartEvent	To instruct the PC to start to receive alarm event data that coming from EDAM-9000	7.6.26
TCP_StopEvent	To instruct the PC to stop receiving alarm event data from all modules	7.6.27
TCP_ReadEventData	To receive alarm event data that coming from the specific EDAM-9000	7.6.28

7.3 Digital I/O Functions

TCP_ReadDIOMode	To read the type for every D/I & D/O channels of an EDAM-9000 module	7.6.29
TCP_ReadDIO	To read DI/DO's status for an EDAM-9000 module	7.6.30
TCP_ReadDISignalWidth	To read the minimal high/low signal width of each D/I channel for an EDAM-9000 module	0
TCP_WriteDISignalWidth	To set the minimal high/low signal width of each D/I channel for an EDAM-9000 module	7.6.32
TCP_ReadDICounter	To read the counter value when a D/I channel function in 'Counter' mode	0
TCP_ClearDICounter	To clear the counter value when a D/I channel function in 'Counter' mode	7.6.34
TCP_StartDICounter	To start the counting when a D/I channel function in 'Counter' mode	0
TCP_StopDICounter	To stop the counting when a D/I channel function in 'Counter' mode	7.6.36
TCP_ClearDILatch	To clear the latch when a D/I channel function as 'Lo to Hi Latch' or 'Hi to Lo Latch'	7.6.37
TCP_ReadDILatch	To read the counter value when a D/I channel function in 'Counter' mode	7.6.38
TCP_WriteDO	To write some value to D/O channels for an EDAM-9000 module	7.6.39
TCP_WriteDOPulseCount	To write the pulse output count for EDAM-9000 DIO modules during runtime	7.6.40
TCP_WriteDODelayWidth	To set the pulse and delay signal widths to the specific EDAM-9000 DIO modules	0
TCP_ReadDODelayWidth	To read the pulse and delay signal width from the specific EDAM-9000 DIO modules	7.6.42

7.4 Analog I/O Functions

TCP_ReadAIAlarmTypes	To set all channel type	7.6.43
TCP_WriteAIAlarmType	To set all channel alarm type	7.6.44
TCP_ReadAITypes	To read type of all channels of a specific analog module	0
TCP_ReadAIValue	To read normal value of all channel	7.6.46
TCP_ReadAIMaxVal	To read maximum value of all channel	0
TCP_ReadAIMinVal	To read minimum value of all channel	7.6.48
TCP_ReadAIMultiplexChannel	To read active status of all channel	0
TCP_WriteAIMultiplexChannel	To set active status of all channel	7.6.50
TCP_ReadAIAverageChannel	To read in average status of all channel	0
TCP_WriteAIAverageChannel	To set/reset channels to be in average	7.6.52
TCP_ReadAIAlarmDOConnection	To read alarm DO connection status	0
TCP_WriteAIAlarmDOConnection	To set alarm DO connection	7.6.54
TCP_ReadAIAlarmStatus	To read alarm status	0
TCP_ClearAILatchAlarm	To clear alarm latch status when a A/I channel function in 'Alarm Latch mode' mode	7.6.56
TCP_ClearAIMaxVal	To clear maximum value to zero	0
TCP_ClearAIMinVal	To clear minimum value to zero	7.6.58
TCP_ReadAIBurnOutStatus	To read AI burn out status(EDAM9015/9019 only)	7.6.59
TCP_ReadAIAlarmLimit	To read channel high/low alarm limit value	7.6.60
TCP_WriteAIAlarmLimit	To set channel high/low alarm limit value	7.6.61
TCP_StartAIAlarm	To set channel alarm type of a specific analog module	7.6.62
TCP_StopAIAlarm	To disable channel alarm of a specific analog module	0
TCP_WriteCJCOffset	To set cold junction offset of a specific EDAM9019 module	7.6.64
TCP_ReadCJCOffset	To read cold junction offset from a specific EDAM9019 module	7.6.65
TCP_ReadCJCTemperature	To read cold junction temperature from a specific EDAM9019 module	7.6.66

7.5 MODBUS/TCP Functions

TCP_MODBUS_ReadCoil	To read the coil values at a specific range described in parameters	7.6.67
TCP_MODBUS_WriteCoil	To write the coil values at a specific range described in parameters.	7.6.68
TCP_MODBUS_ReadReg	To read the holding register value at a specific range described in parameters	7.6.69
TCP_MODBUS_WriteReg	To write values to the holding registers at a specific range described in parameters	7.6.70

7.6 Function Description

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

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

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

7.6.1 TCP_Open

Description: To initiate the TCPDAQ.dll to use.

Syntax:

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

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

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

int TCP_Open();

♦ **Delphi: (*see TCPDAQ.pas*)**

function TCP_Open(); StdCal;

♦ **VC++: *(see TCPDAQ.h*)**

int TCP_Open();

Parameters:

void

Return Code:

refer to the *Error code.*

7.6.2 TCP_Close

Description: To terminates use of the TCPDAQ.dll.

Syntax:

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

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

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

int TCP_ Close();

♦ **Delphi: *(see TCPDAQ.pas)***

function TCP_ Close(); StdCall;

♦ **VC++: *(see TCPDAQ.h)***

int TCP_ Close();

Parameters:

void

Return Code:

refer to the *Error code.*

7.6.3 TCP_Connect

Description: to create a Window TCP socket then establishing a connection to a specific EDAM-9000

Syntax:

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

Declare Function TCP_Connect Lib "TCPDAQ.dll" Alias "_TCP_Connect@20" (ByVal szIP As String, ByVal port As Integer, ByVal ConnectionTimeout As Long, ByVal SendTimeout As Long, ByVal ReceiveTimeout As Long) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
port[in]	: the TCP/IP port used by Modbus/TCP, it is 502
ConnectionTimeout[in]	: Connection timeout value (msec)
SendTimeout[in]	: Send timeout value (msec)
ReceiveTimeout[in]	: Receive timeout value (msec)

Return Code:

refer to the *Error code*.

7.6.4 TCP_Disconnect

Description: disconnecting the Window TCP socket from all EDAM-9000 modules

Syntax:

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

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

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

void TCP_Disconnect(void);

♦ **Delphi:** ***(see TCPDAQ.pas)***

procedure TCP_Disconnect ; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

void TCP_Disconnect(void);

Parameters:

void

Return Code:

none.

7.6.5 TCP_ModuleDisconnect

Description: disconnecting the Window TCP socket to a specific EDAM-9000

Syntax:

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

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

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

Int TCP_ModuleDisconnect(char szIP[]);

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

Int TCP_ModuleDisconnect(char szIP[]);

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

Return Code:

refer to the *Error code*.

7.6.6 TCP_SendData

Description: to send data to a specific EDAM-9000 module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

pData[in] : 8 bit data array

wDataLen[in] : length of data be sent

Return Code:

refer to the *Error code*.

7.6.7 TCP_RecvData

Description: receive data to a specific EDAM-9000 module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

pData[out] : 8 bit data array

wDataLen [in] : length of data array

Return Code:

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

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

7.6.8 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 EDAM-9000 and receiving the response from EDAM-9000

Syntax:

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

Declare Function TCP_SendReceiveASCcmd Lib "TCPDAQ.dll" Alias "_TCP_SendReceiveASCcmd@12" (ByVal szIP As String, ByVal Sendbuf As String, ByVal Recvbuf As String) As Long

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
Sendbuf [in]	: 8 bit data array to be sent
Recvbuf [out]	: 8 bit data array that stored the received data

Return Code:

refer to the *Error code.*

7.6.9 UDP_Connect

Description: to create a Window UDP socket then establishing a connection to a specific EDAM-9000

Syntax:

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

Declare Function UDP_Connect Lib "TCPDAQ.dll" Alias "_UDP_Connect@24" (ByVal szIP As String, ByVal s_port As Integer, ByVal d_port As Integer, ByVal ConnectionTimeout As Long, ByVal SendTimeout As Long, ByVal ReceiveTimeout As Long) As Long

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
s_port	: source port number
d_port	: destination port number
ConnectionTimeout	: timeout value for connection (msec)
SendTimeout	: timeout value for sending (msec)
ReceiveTimeout	: timeout value for receiving (msec)

Return Code:

refer to the *Error code.*

7.6.10 UDP_Disconnect

Description: disconnecting the Window UDP socket from all EDAM-9000 modules

Syntax:

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

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

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

void UDP_Disconnect(void);

♦ **Delphi: *(see TCPDAQ.pas)***

procedure UDP_Disconnect ; StdCall;

♦ **VC++: *(see TCPDAQ.h)***

void UDP_Disconnect(void);

Parameters:

void

Return Code:

none

7.6.11 UDP_ModuleDisconnect

Description: disconnecting the Window UDP socket from a specific EDAM-9000

Syntax:

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

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

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

int UDP_ModuleDisconnect(Char szIP[]);

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

int UDP_ModuleDisconnect(char szIP[]);

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be disconnected

Return Code:

refer to the *Error code.*

7.6.12 UDP_SendData

Description: send data to a specific EDAM-9000 module (Datagram)

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
pData[in]	: points to data buffer
wDataLen[in]	: length of data be sent

Return Code:

refer to the *Error code.*

7.6.13 UDP_RecvData

Description: receive data to a specific EDAM-9000 module (Datagram)

Syntax:

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

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

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

pData[out] : 8 bit array that stored the received data

wDataLen [in] : length of received data

Return Code:

refer to the *Error code*.

7.6.14 UDP_SendReceiveASCcmd

Description: send an ASCII format string as a command to EDAM-9000 and receiving the response from EDAM-9000

Syntax:

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

Declare Function UDP_SendReceiveASCcmd Lib "TCPDAQ.dll" Alias "_UDP_SendReceiveASCcmd@12" (ByVal szIP
As String, ByVal Txdata As _ String, ByVal Rxdata As String) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

Txdata [in] : 8 bit array that stored the data to be sent

Rxdata [out]: 8 bit array that stored the received data

Return Code:

refer to the *Error code*.

7.6.15 TCP_GetModuleIPinfo

Description: return module IP information of a specific module

Syntax:

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

Declare Function TCP_GetModuleIPinfo Lib "TCPDAQ.dll" Alias "_TCP_GetModuleIPinfo@8" (ByVal szIP As String, ByRef ModuleIP As ModuleInfo) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.16 TCP_GetModuleID

Description: return ID number of a specific module.

Syntax:

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

Declare Function TCP_GetModuleID Lib "TCPDAQ.dll" Alias "_TCP_GetModuleID@8" (ByVal szIP As String, ByRef ModuleID As Byte) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

ModuleID [in] : the ID number

Return Code:

refer to the *Error code.*

7.6.17 TCP_GetIPFromID

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

Syntax:

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

Declare Function TCP_GetIPFromID Lib "TCPDAQ.dll" Alias "_TCP_GetIPFromID@8" (ByVal szID As Byte, ByRef szIP As String) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

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

Return Code:

refer to the *Error code.*

7.6.18 TCP_ScanOnLineModules

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

Syntax:

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

Declare Function TCP_ScanOnLineModules Lib "TCPDAQ.dll" Alias "_TCP_ScanOnLineModules@8" (ModuleIP As ModuleInfo, ByVal Sortkey As Byte) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

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

Return Code:

refer to the *Error code.*

7.6.19 TCP_GetDLLVersion

Description: return the version number of TCPDAQ.dll

Syntax:

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

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

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

Int TCP_GetDLLVersion(void);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_GetDLLVersion: Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

Int TCP_GetDLLVersion(void);

Parameters:

void

Return Code:

the version number.

7.6.20 TCP_GetModuleNo

Description: return the module name of a specific IP address

Syntax:

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

Declare Function TCP_GetModuleNo Lib "TCPDAQ.dll" Alias "_TCP_GetModuleNo@8" _
(ByVal szIP As String, ByRef Mname As Byte) As Long

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

Mname[out] : 8 bit array that stored the module name string

Return Code:

refer to the *Error code*.

7.6.21 TCP_GetLastError

Description: return the error code of the latest called function

Syntax:

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

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

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

Int TCP_GetLastError(void);

- ♦ **Delphi: *(see TCPDAQ.pas)***

Function TCP_GetLastError: Longint ; StdCall;

- ♦ **VC++: *(see TCPDAQ.h)***

Int TCP_GetLastError(void);

Parameters:

void

Return Code:

refer to the *Error code*

7.6.22 TCP_PingIP

Description: ping to remote IP address

Syntax:

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

Declare Function TCP_PingIP Lib "TCPDAQ.dll" Alias "_TCP_PingIP@8" (ByVal IPadr As String, ByVal PingTimes As Integer) As Long

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

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

- ♦ **Delphi: *(see TCPDAQ.pas)***

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

- ♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
PingTimes [in]	:Timeout value

Return Code:

=-1, no response from remote IP

>0, response time from remote IP

7.6.23 TCP_StartStream

Description: to instruct the PC to start to receive stream data that coming from EDAM-9000

Syntax:

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

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

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code*.

7.6.24 TCP_StopStream

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

Syntax:

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

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

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

int TCP_StopStream(void);

♦ **Delphi:** ***(see TCPDAQ.pas)***

Function TCP_StopStream: Longint; StdCall;

♦ **VC++:** ***(see TCPDAQ.h)***

int TCP_StopStream(void);

Parameters:

void

Return Code:

refer to the *Error code*.

7.6.25 TCP_ReadStreamData

Description: to read stream data that coming from the specific EDAM-9000

Syntax:

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

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

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.26 TCP_StartEvent

Description: to start listening the alarm event trigger

Syntax:

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

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

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.27 TCP_StopEvent

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

Syntax:

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

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

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

Int TCP_StopEvent(void);

♦ **Delphi: *(see TCPDAQ.pas)***

Function TCP_StopEvent: Longint; StdCall;

♦ **VC++: *(see TCPDAQ.h)***

Int TCP_StopEvent(void);

Parameters:

void

Return Code:

refer to the *Error code*.

7.6.28 TCP_ReadEventData

Description: to read triggered alarm event message

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code*.

7.6.29 TCP_ReadDIOMode

Description: to read the mode of D/I & D/O channels of an EDAM-9000 module.

Syntax:

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

```
Declare Function TCP_ReadDIOMode Lib "TCPDAQ.dll" Alias "_TCP_ReadDIOMode@12" _
                    (ByVal szIP As String, ByRef DImode As Byte, ByRef DOmode As Byte) As Long
```

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

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

Return Code:

refer to the *Error code*.

7.6.30 TCP_ReadDIO

Description: to read DI/DO's status for an EDAM-9000 module

Syntax:

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

```
Declare Function TCP_ReadDIO Lib "TCPDAQ.dll" Alias "_TCP_ReadDIO@12" _
                    (ByVal szIP As String, ByRef ByDi As Byte, ByRef ByDo As Byte) As Long
```

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

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

Return Code:

refer to the *Error code*.

7.6.31 TCP_ReadDISignalWidth

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

Syntax:

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

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

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

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

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

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

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

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
ulLoWidth[out]	: an 32 bit array that stored channel low width value
ulHiWidth[out]	: an 32 bit array that stored channel high width value

Return Code:

refer to the *Error code.*

7.6.32 TCP_WriteDISignalWidth

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

Syntax:

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

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

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

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

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

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

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

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
ulLoWidth[in]	: an unsigned 32 bits array that stored the minimal low signal width for each D/I channel. The unit is 0.5 mSec
ulHiWidth[in]	: an unsigned 32 bits array that stored the minimal high signal width for each D/I channel. The unit is 0.5 mSec

Return Code:

refer to the *Error code.*

7.6.33 TCP_ReadDICounter

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
ulCounterValue[out]	:an unsigned 32 bits array that stored the counter value for each D/I channel

Return Code:

refer to the *Error code.*

7.6.34 TCP_ClearDICounter

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wChNo[in]	: the D/I channel to be cleared.

Return Code:

refer to the *Error code.*

7.6.35 TCP_StartDICounter

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.36 TCP_StopDICounter

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.37 TCP_ClearDILatch

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h*)**

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.38 TCP_ReadDILatch

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

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

Return Code:

refer to the *Error code.*

7.6.39 TCP_WriteDO

Description: to write some value to D/O channels for an EDAM-9000 module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

wStartDO[in] : the starting channel that to be written.

wCount[in] : how many channels to be written.

byDO[in] : an 8 bit array that stored the values that written to the connected EDAM-9000

Return Code:

refer to the *Error code*.

7.6.40 TCP_WriteDOPulseCount

Description: to write the pulse output count for EDAM-9000 DIO modules during runtime

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

wDoChannel[in] : the channel index for writing

ulPulseCount[in]: the pulse output count.

Return Code:

refer to the *Error code*.

7.6.41 TCP_WriteDODelayWidth

Description: to set the pulse and delay signal widths to specific EDAM-9000 DIO modules

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wChno[in]	: the channel index for writing
ulLoPulseWidth[in]	: the output pulse signal width at low level.
ulHiPulseWidth[in]	: the output pulse signal width at high level.
ulLoDelayWidth[in]	: the output signal delay width when set DO from high to low level.
ulHiDelayWidth[in]	: the output signal delay width when set DO from low to high level.

Return Code:

refer to the *Error code*.

7.6.42 TCP_ReadDODelayWidth

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

Syntax:

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

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

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

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

♦ **Delphi: (see TCPDAQ.pas)**

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

♦ **VC++: (see TCPDAQ.h)**

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wChno[in]	: the channel index for reading
ulLoPulseWidth[out]	: the pulse output signal width at low level
ulHiPulseWidth[out]	: the pulse output signal width at high level
ulLoDelayWidth[out]	: the delay output signal width at low level
ulHiDelayWidth) [out]	: the delay output signal width at high level

Return Code:

refer to the *Error code*.

7.6.43 TCP_ReadAIAlarmTypes

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

Syntax:

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

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

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

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

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

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

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

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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.*

7.6.44 TCP_WriteAIAlarmType

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

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

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

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

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

Delphi: *(see TCPDAQ.pas)*

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

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

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.45 TCP_ReadAITypes

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

Syntax:

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

Declare Fu 錯誤! 找不到參照來源。nction TCP_ReadAITypes Lib "TCPDAQ.dll" Alias "_TCP_ReadAITypes@8"
(ByVal szIP As String, ByRef szRange As Byte) As Long

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
szTypes[out]	: an array that stored the types of all A/I channels

Return Code:

refer to the *Error code.*

7.6.46 TCP_ReadAIValue

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 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.*

7.6.47 TCP_ReadAIMaxVal

Description: Read all channel maxmal value of a specific ananlog module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.48 TCP_ReadAIMinVal

Description: Read all channel minimal value of a specific ananlog module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.49 TCP_ReadAIMultiplexChannel

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

Syntax:

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

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

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.50 TCP_WriteAIMultiplexChannel

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

Syntax:

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

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

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

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

♦ **Delphi:** ***(see TCPDAQ.pas)***

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

♦ **VC++:** ***(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.51 TCP_ReadAIAverageChannel

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.52 TCP_WriteAIAverageChannel

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.53 TCP_ReadAIAlarmDOConnection

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

Syntax:

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

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

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

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

♦ **Delphi: (see TCPDAQ.pas)**

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

♦ **VC++: (see TCPDAQ.h)**

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

Parameters:

szIP[in]   : the IP address for an EDAM-9000 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.*

7.6.54 TCP_WriteAIAlarmDOConnection

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]   : the IP address for an EDAM-9000 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.*

7.6.55 TCP_ReadAIAlarmStatus

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 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*.

7.6.56 TCP_ClearAILatchAlarm

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.57 TCP_ClearAIMaxVal

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

Chno[in] : the channel index for clearing

Return Code:

refer to the *Error code*.

7.6.58 TCP_ClearAIMinVal

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

Chno[in] : the channel index for clearing

Return Code:

refer to the *Error code*.

7.6.59 TCP_ReadAIBurnOutStatus

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
dlBurnout[out]	: an 8 bit array that stored the burn-out status of EDAM-9019,9015 module (=0 normal, =1 burn-out)

Return Code:

refer to the *Error code.*

7.6.60 TCP_ReadAIAlarmLimit

Description: Read all channel high/low alarm limit value

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in	: the IP address for an EDAM-9000 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.*

7.6.61 TCP_WriteAIAlarmLimit

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.62 TCP_StartAIAlarm

Description: to start channel alarm of a specific analog module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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*.

7.6.63 TCP_StopAIAlarm

Description: to disable channel alarm of a specific analog module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 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

7.6.64 TCP_WriteCJCOffset

Description: to set cold junction offset of a specific EDAM9019 module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

CJoffset[in] : cold junction temperature offset

Return Code:

refer to the *Error code*.

7.6.65 TCP_ReadCJCOffset

Description: to read cold junction offset from a specific EDAM9019 module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

CJoffset[out]: cold junction offset

Return Code:

refer to the *Error code.*

7.6.66 TCP_ReadCJCTemperature

Description: to read cold junction temperature from a specific EDAM9019 module

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in] : the IP address for an EDAM-9000 that to be connected

CJTemp[out]: cold junction temperature

Return Code:

refer to the *Error code.*

7.6.67 TCP_MODBUS_ReadCoil

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wStartAddress[in]	: start address of coil registers (1 ~ 255)
wCount[in]	: the count that coil data be read
byData[in]	: the 8 bit array that stored the coil data (0=set, 1=reset)

Return Code:

refer to the *Error code.*

7.6.68 TCP_MODBUS_WriteCoil

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wStartAddress[in]	: start address of coil registers (1 ~ 255)
wCount[in]	: the count that coil data be written
byData[in]	: the 8 bit array that stored the coil data (0=set, 1=reset)

Return Code:

refer to the *Error code.*

7.6.69 TCP_MODBUS_ReadReg

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wStartAddress[in]	: start address of holding registers (1 ~ 255)
wCount[in]	: the count that holding data be read
byData[in]	: the 16 bit array that stored the holding data

Return Code:

refer to the *Error code*.

7.6.70 TCP_MODBUS_WriteReg

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

Syntax:

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

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

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

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

♦ **Delphi: *(see TCPDAQ.pas)***

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

♦ **VC++: *(see TCPDAQ.h)***

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

Parameters:

szIP[in]	: the IP address for an EDAM-9000 that to be connected
wStartAddress[in]	: start address of holding registers (1 ~ 255)
wCount[in]	: the count that holding data be read
byData[in]	: the 16 bit array that stored the holding data

Return Code:

refer to the *Error code.*

Chapter 8 ASCII Commands for EDAM-9000(A)/9400 Modules

8.1 About ASCII Commands

For users do not familiar to Modbus protocol, Inlog offers a function library as a protocol translator, integrating ASCII command into Modbus/TCP structure. Therefore, users familiar to ASCII command can access EDAM-9000 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.

EDAM9000(A) series also integrate ASCII command into ***UDP protocol with port 1025****. User can simply send the Command of ASCII format through UDP protocol (such as UPD_send (Dest_IP, "$01M")).*

8.2 Syntax of ASCII

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

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

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

The command set is divided into the following five categories:

♦ System Command Set

♦ Analog Input Command Set

♦ Analog Input Alarm Command Set

♦ Universal I/O Command Set

♦ Digital I/O Command Set

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

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

8.3 ASCII Command Set

8.3.1 Common commands

Command	Command Name	Description	modules	Sec.
$AAM	Read Module Name	Return the module name	All modules	8.4.1
$AAF	Read Firmware Version	Return the firmware version	All modules	8.4.2
~AADnnnnn	Set timout to search DHCP	Set timout to search DHCP	All modules	8.4.3
~AAD	Read timout to search DHCP	Read timout to search DHCP	All modules	8.4.4
$AAID	Read module ID	Return module ID number(For DLL)	All modules	0
$AAIDFF	Set Module ID number	Set module ID number(For DLL) (for firmware version 6.000 or later)	All 9000A series	8.4.6
$AAMD(data)	set module description	set module description (for firmware version 6.000 or later)	All 9000A series	0
$AAMD	read module description	read module description (for firmware version 6.000 or later)	All 9000A series	8.4.8
^AAMAC	read MAC address	read MAC address (for version 6.000 or later)	All 9000A series	0
$AARS	Reboot the module to the power-on state	Reboot the module to the power-on state (for firmware version 6.000 or later)	All 9000A series	8.4.10
$AAS1	Reloads the module factory default	Reloads the module factory default (for firmware version 6.000 or later)	All 9000A series	8.4.11
~AAI	Soft INIT command (Enable)	Soft INIT* command (Enable) (for firmware version 6.000 or later)	All 9000A series	8.4.12
~AATNN	Sets the soft INIT timeout value	Sets the soft INIT* timeout value (for firmware version 6.000 or later)	All 9000A series	8.4.13
$AA5	Reads the Reset Status of a module	Reads the Reset Status of a module (for firmware version 6.000 or later)	All 9000A series	8.4.14

8.3.2 Digital I/O commands

Commands	Command name	Descritpion	Modules	Sec.
$AA6	Read DI/O Channel Status	read the status of all DI(0~11) and DO (0~7) channels	9050 / 9051 / 9052	1.1.1
		read the status of all DI(0~15) and DO(0~15) channels. (ref. "@AA6" for read 16bits DIO) (for firmware version 6.000 or later)	All modules (Except 9050 /9051 / 9052)	
@AA	Read DI/O Status	read the status of all DI(0~11) and DO(0~7) channels. (ref. "@AA6" for read 16bits DIO)	9050 / 9051 / 9052	8.4.16
		read the status of all DI(0~15) and DO(0~15) channels. (ref. "@AA6" for read 16bits DIO) (for firmware version 6.000 or later)	All modules (Except 9050 / 9051 / 9052)	
$AA7	Read DI latch status	Read DI latch status	All modules	8.4.17
#AA00dd	Write DO channels(0~7)	Write a value to digital output channels(0~7)	All modules	8.4.18
#AA1ndd	Set single DO channel	Set the single digital output channel(0~15)	All modules	8.4.19

#AA2npppppppp	Write DO pulse counts	Write DO pulse counts to the specific DO channel	All modules	8.4.27
$AACLS	Clear all latch status	Clear latch status of all DI channels	All modules	8.4.28
~AADSMN	Set DI/O active state	Set digital input/output active state (for firmware version 6.000 or later)	All modules	8.4.29
~AADS	read DI/O active state	read digital input/output active state (for firmware version 6.000 or later)	All modules	8.4.30
@AA6	read the status of all DO and DI channels	read the status of all DO(4char) and DI(4char) channels. (for firmware version 6.000 or later)	All modules	8.4.35
@AA6DDDD	Write DO channels(0~15)	Write a value to digital output channels(0~15) (for firmware version 6.000 or later)	All modules	8.4.36
@AA6ONSS	Set single DO channel	Set the single digital output channel(0~15) (for firmware version 6.000 or later)	All modules	8.4.37
@AA6ON	Read a single digital output for channel N	Read a single digital output for channel N (0~15) (for firmware version 6.000 or later)	All modules	0
@AA6IN	Read a single digital input for channel N	Read a single digital input for channel N (0~15) (for firmware version 6.000 or later)	All modules	8.4.39
$AA9DNNLLLLUUUU	Set DO low/high delay width for channel N	Set DO Low/high delay output width for channel N (00~15), unit: 0.5ms. (for firmware version 6.070 or later)	9000A series only	8.4.40
$AA9PNNLLLLUUUU	Set DO pulse low/high width for channel N	Set DO pulse low/high width for channel N (00~15), unit: 0.5ms. (for firmware version 6.070 or later)	9000A series only	8.4.41
$AA9TN	Set a single DO channel for DO "Auto-Off Time Mode"	Set a single DO channel for Auto-Off Time Mode of the Digital Output. (for firmware version 6.070 or later)	9000A series only	8.4.42
$AA9NN	Read DO pulse low/high width for channel N	Read DO pulse low/high width and DO Low/high delay output width for ch. N (00~15), unit: 0.5ms. (for firmware version 6.000 or later)	9000A series only	8.4.49
$AACONNDD	set a single DO mode for channel N	set a single DO mode for channel N (for firmware version 6.070 or later)	All modules	0
$AACONN	read a single DO mode for channel N	read a single DO mode for channel N (for firmware version 6.070 or later)	All modules	8.4.32
$AACINNDD	set a single DI mode for channel N	set a single DI mode for channel N (for firmware version 6.070 or later)	All modules	8.4.33
$AACINN	read a single DI mode for channel N	read a single DOImode for channel N (for firmware version 6.070 or later)	All modules	0
$AAYM1CPSHHHHLLLL (data)	Set DI match/mismatch DO Toggler Mode	Set DI *match/mismatch DO Toggler Mode* for DIO Sync. (for Mirror Local DI to DO) (for firmware version 6.070 or later)	9000A series only	8.4.43
$AAYM2CPSTTTT (data)	Set DI match DO latch mode	Set DI data *DI match DO latch mode* for DIO Sync. (for firmware version 6.070 or later)	9000A series only	8.4.44
$AAYM3CPSTTTT (data)	Set DI mismatch DO latch mode	Set DI data *DI mismatch DO latch mode* for DIO Sync. (for firmware version 6.070 or later)	9000A series only	8.4.45

$AAYMRCS	Start(Run)/Stop DIO synchronize	Start(Run)/Stop DIO sync. operation (for firmware version 6.070 or later)	9000A series only	8.4.46
$AAYMC	Read DIO Sync. mode	Read DIO Sync. mode (for firmware version 6.070 or later)	9000A series only	0
$AAYMS	Read DIO Sync. DO active status	Read DIO Sync. DO active status (for firmware version 6.070 or later)	9000A series only	0

8.3.3 DI Counter Commands

Command	Command Name	Description	modules	Sec.
$AA0MCC	Read DI counter filter	Read DI counter filter signal width of channel N (uint = 0.5ms) (for firmware version 6.000 or later)	9000A series only (Except 9051A)	8.4.20
$AA0MCC(data)	Set DI counter filter	Set DI counter filter signal width of channel N (uint = 0.5ms) (for firmware version 6.000 or later)	9000A series only (Except 9051A)	8.4.21
$AAEcn	Enable/disable DI counter	Start/stop counter of the specific DI channel.	All modules	8.4.22
$AACn	Clear DI counter	Clear DI counter of the specific DI channel.	All modules	8.4.23
#AA	Read DI counter	Read all DI counter values.	All modules	8.4.24
#AAn	Read DI counter	Read counter value of the specific DI channel.	All modules	8.4.25
#AArn	Read DI counter with overflow	Read a single DI channel counter with overflow. (for firmware version 6.000 or later)	9000A series only	8.4.26

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

Command	Command Name	Description	modules	Sec.
~**	Host ok	host ok. Refresh WDT counter of all modules via broadcast,(No reply from modbus response	9000A series only	8.4.50
~AA**	Host ok	host ok. Refresh WDT counter of the specific module Response : !01	9000A series only	8.4.51
~AA0	Read host watchdog timeout status	Read host watchdog timeout status. bit(7) - watchdog Enable/Disable, bit(2) - watchdog timeout(1) status)	9000A series only	8.4.52
~AA1	Reset host watchdog timeout status	Reset host watchdog timeout status	9000A series only	8.4.53
~AA2	Read host communication Timeout value.	Read host communication Timeout value. (0.1sec)	9000A series only	8.4.54
~AA3EVVV	Set Host watchdog timeout interval.	Enable Host watchdog and set timeout interval (unit=0.1sec)	9000A series only	8.4.55
~AA4V	Read the power-on DO value or the safe DO value of a module	Read the power-on DO value or the safe DO value of a module	9000A series only	8.4.56
~AA5V	Sets the current value as the power-on DO value or the safe DO value	Sets the current value as the power-on DO value or the safe DO value	9000A series only	8.4.58
~AA3PPP	Set module Power-on delay time.	Set module Power-on delay time (unit=0.1sec) to start host communication timeout watchdog	9000A series only	0

~AA3P	read module Power-on delay time	Read module Power-on delay time (unit=0.1sec) to start host communication timeout watchdog	9000A series only	8.4.59

8.3.5 Analog commands

Command	Command name	Description	modules	Sec.
#AAn	Read single analog Input	Read the input value from the specific analog input channel	9015/9017/9019	8.4.60
#AA	Read all analog Input	Read the input values from all analog input channels	9015/9017/9019	8.4.61
#AAAcctt	Set analog input type	Set type of the specific analog input channel	9015/9017/9019	0
$AABhh	Read input type	read input type of the specific analog channel	9015/9017/9019	8.4.63
$AA0	Span Calibration	Calibrate the analog input module to correct the gain error	9015/9017/9019	8.4.64
$AA1	Offset Calibration	Calibrate the analog input module to correct the offset error	9015/9017/9019	8.4.65
$AA6	Read Channel Enable/Disable Status	Read the Enable/Disable status of all analog input channels	9015/9017/9019	8.4.66
$AA5mm	Enable/disable ananlog channel(s)	Enable/disable analog input channels	9015/9017/9019	8.4.67
#AAMH	Read all Max. Data	Read the maximum data from all analog input channels	9015/9017/9019	8.4.68
#AAMHn	Read single Max. Data	Read the maximum date from a specific analog input channel	9015/9017/9019	8.4.69
#AAML	Read all Min. Data	Read the minimum data from all analog input channels	9015/9017/9019	8.4.70
#AAMLn	Read single Min. Data	Read the minimum data from a specific analog input channel	9015/9017/9019	8.4.71
$AACjAhs	Set Alarm Mode	Set the High/Low alarm in either Momentary or Latching mode	9015/9017/9019	8.4.72
$AACjAh	Read Alarm Mode	Returns the alarm mode for the specific channels	9015/9017/9019	8.4.73
$AACjAhEs	Enable/Disable Alarm	Enables/Disables the high/low alarm of the specific channels	9015/9017/9019	8.4.74
$AACjCh	Clear Latch Alarm	Resets a latched alarm	9015/9017/9019	8.4.75
$AACjAhCCn	Set Alarm Connection	Connects the High/Low alarm of a specific input channel to interlock with a specific output channel	9017/9019	8.4.76
$AACjRhC	Read Alarm Connection	Returns the alarm configuration of a specific input channel	9017/9019	8.4.77
$AACjAhU	Set Alarm Limit	Sets the High/Low alarm limit value to a specific channel	9015/9017/9019	8.4.78
$AACjRhU	Read Alarm Limit	Returns the High/Low alarm limit value of the specific channel	9015/9017/9019	8.4.79
$AACjS	Read Alarm Status	Reads whether an alarm occurred in the specific Channel	9015/9017/9019	8.4.80
$AA3	Read cold junction	Return the Cold Junction temperature	9019	8.4.81

$AA9hhhhh	Set CJ offset	Set Cold Junction temperature offset	9019	8.4.82
$AA9	Read CJ offset	Return Cold Junction temperature offset	9019	8.4.83

8.4 ASCII Command Description

8.4.1 $AAM Read Module Name

Description: Returns the module name from a specific module.

Syntax: **$AAM(cr)**

$	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
M	is the Module Name command.
(cr)	is the terminating character, carriage return (0Dh).

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

?AA(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error, communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was in-valid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.
(data)	A string showing the name of the module (max. 6 chars.)
(cr)	is the terminating character, carriage return (0Dh).

Example:

command: $01M(cr)

response: !019050(cr)

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

8.4.2 $AAF Read Firmware Version

Description: Returns the firmware version from a specific module.

Syntax: $AAF(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

F is the Firmware Version command.

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000(A) module.

(data) firmware version of module(max. 6 chars.)

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

Example 1:

command: **$01F(cr)**

response: **!01M1.01(cr)**

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

Example 2:

command: **$01F(cr)**

response: **!0160.01(cr)**

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

8.4.3 ~AADnnnnn Set timout to search DHCP

Description: Set timout to search DHCP.

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

~ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

D is set DHCP search timeout command.

nnnnn is DHCP timeout value (10~1800 sec) (dec)

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.

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

Example:

command: **~01D01200(cr)**

response: **!01(cr)**

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

8.4.4 ~AAD Read timout to search DHCP

Description: Read timout to search DHCP.

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

~ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

D is read DHCP search timeout command.

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.

nnnnn Timeout value to search DHCP servo

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

Example:

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

response: **!0101200(cr)**

The command read timeout is 1200 seconds

8.4.5 $AAID Read module ID number

Description: Returns the ID number from a specific module. (for tcpdaq.dll driver)

Syntax: $AAID(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 EDAM-9000 module.

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).

8.4.6 $AAIDFF Set module ID number

Description: Set module ID number(for tcpdaq.dll driver). (command for version 6.000 or later)

Syntax: $AAIDFF(cr)

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

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

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.
(cr)	is the terminating character, carriage return (0Dh).

Example:

command: $01ID1A(cr)

response: !01(cr)

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

8.4.7 $AAMD(data) Set module description

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

Syntax: $AAMD(data)(cr)

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

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

?AA(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error, communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.
(cr)	is the terminating character, carriage return (0Dh).

Example:

command: $01MD12DI8DO(cr)

response: !01(cr)

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

8.4.8 $AAMD Read module description

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

Syntax: $AAMD(cr)

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

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

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.
(data)	module description (max 30 characters)
(cr)	is the terminating character, carriage return (0Dh).

Example:

command: $01MD(cr)

response: !0112DI8DO(cr)

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

8.4.9 ^AAMAC Read MAC address

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

Syntax: ^AAMAC(cr)

^ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

MAC command for read MAC address.

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.

MMMMMMMMMMMM MAC address(hex)

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

Example:

command: ^01MAC(cr)

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

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

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

Syntax: $AARS(cr)

$	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
RS	command to reboot the module to the power-on state
(cr)	is the terminating character, carriage return (0Dh).

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

?AA(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error, communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module
(cr)	is the terminating character, carriage return (0Dh).

Example:

command: $01RS(cr)

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

8.4.11 $AAS1 Reloads the module factory default

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

Syntax: $AAS1(cr)

$	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
S1	command to reload the factory default
(cr)	is the terminating character, carriage return (0Dh).

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

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module
(cr)	is the terminating character, carriage return (0Dh).

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

Example :

(1) Sets the soft INIT* timeout value of module 01 to 32 seconds and returns a valid response.

Command: ~01T32(cr)
Response: !01(cr)

(2) Sets the soft INIT* enable and returns a valid response.

Command: ~01I(cr)
Response: !01(cr)

(3) Reloads the module factory default

Command: $01S1(cr)
Response: !01(cr)

Related command: ~AATnn, ~AAI

8.4.12 ~AAI Set the Soft INIT*

Description: The Soft INIT* command is used to enable modification of the IP, Gateway and communication protocol settings using software only. (available for version 6.000 or later)

Syntax: ~AAI(cr)

~	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
I	command to set the Soft INIT* enable
(cr)	is the terminating character, carriage return (0Dh).

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

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000(A) module
(cr)	is the terminating character, carriage return (0Dh).

Example :

1. Sets the soft INIT* timeout value of module 01 to 16 seconds and returns a valid response.

   Command: ~01T10(cr)
   Response: !01(cr)

2. Sets the soft INIT* enable and returns a valid response.

   Command: ~01I(cr)
   Response: !01(cr)

3. Reloads the module factory default

   Command: $01S1(cr)
   Response: !01(cr)

Related command: ~AATnn, $AAS1

8.4.13 ~AATNN Sets the soft INIT* timeout value.

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

Syntax: ~AATNN(cr)

~	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
T	command to Sets the soft INIT* timeout value
NN	Two hexadecimal digits representing the timeout value in seconds. The maximum timeout value is 60 seconds. When changing the IP or Gateway settings without altering the INIT* slide switch, the ~AAI and (or $AAS1) commands should be sent consecutively and the time interval between the two commands should be less than the soft INIT* timeout. If the soft INIT* timeout is 0, then the IP and Gateway settings cannot be changed using software only. The power-on reset value of the soft INIT* timeout is 0.
(cr)	is the terminating character, carriage return (0Dh).

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

?AA(cr)	if an invalid operation was entered.
	There is no response if the module detects a syntax error, communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000(A) module
(cr)	is the terminating character, carriage return (0Dh).

Example :

1. Sets the soft INIT* timeout value of module 01 to 16 seconds and returns a valid response.

   Command: ~01T10(cr)
   Response: !01(cr)

2. Sets the soft INIT* enable and returns a valid response.

   Command: ~01I(cr)
   Response: !01(cr)

3. Reloads the module factory default

   Command: $01S1(cr)
   Response: !01(cr)

Related command: ~AAI, $AAS1

8.4.14 $AA5 Reads the Reset Status of a module

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

Syntax: $AA5(cr)

$	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
5	Command for read reset status.
(cr)	is the terminating character, carriage return (0Dh).

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

?AA(cr)	if an invalid operation was entered.
	There is no response if the module detects a syntax error, communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal address of an EDAM-9000 module.
S	the Reset Status of a module.
	= 0 - the module is not been reseted
	= 1 - the module is been reseted
(cr)	is the terminating character, carriage return (0Dh).

Example 1:

command:	$015(cr)	
response:	!011(cr)	; the module is been reseted

Example 2:

command:	$015(cr)	
response:	!010(cr)	; the module is not been reseted

8.4.15 1.1.9 $AA6 Read DI /DO Channel Status

Description: This command requests that the specific EDAM-9000(A) 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 address of the corresponding module.

6 is the Digital Data In command.

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

Response: !AA0(data1)(data2)(cr) if the command is valid.

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex address of the corresponding EDAM-9000(A) module.

0 The value is always 0.

For 9050/9051/9052:

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

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

For All EDAM-9000A DIO modules (Except 9050/9051/9052) :

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

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

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

Example 1:

command: **$016(cr)** ; read E-9050 DIO status

response : **!01003004(cr)**

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

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

Example 2:

command: **$016(cr)** ; read E-9050A DIO status

response: **!01000030004(cr)**

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

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

Related command: @AA, @AA6

8.4.16 @AA Read DIO status

Description: Read digital input and output status. All modules

Syntax: @AA(cr)

@ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

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

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

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

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

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

For 9050/9051/9052 :

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

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

For All EDAM-9000A modules (Except 9050/9051/9052) :

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

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

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

Example 1:

command: **@01(cr)** ; read E-**9050** DIO status

response: **>03004(cr)**

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

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

Example 2:

command: **@01(cr)** ; read E-**9050A** DIO status

response: **>00030004(cr)**

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

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

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

Related command: @AA6, $AA6

8.4.17 $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
7	represents read DI latch status command.
(cr)	is the terminating character, carriage return (0Dh)

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

?AA(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexadecimal Modbus address of an EDAM-9000(A) module.
DDDD	represent DI latch status
(cr)	is the terminating character, carriage return (0Dh)

Example:

command: $017(cr)
response: !010003(cr)

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

8.4.18 #AA00dd Write All Digital Output

Description: This command sets all digital output channels to the specific EDAM-9000 module.

Syntax: #AA00nn(data)(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
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 hex address of the corresponding module
(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

8.4.19 #AA1n0dd Set Single Digital Output Channel

Description: Set the digital output status of EDAM-9000(A) digital output module.

Syntax: #AA1n0d(cr)

# is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

n (range 0-F) represents the specific channel you want to set the output status.

dd represents the status you want to set to the specific channel.

= 00 – output inactive.

= 01 - output active.

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

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

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

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

! Delimiter indicating a valid command was received.

? Delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hexadecimal Modbus address of an EDAM-9000(A) module.

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

Example 1:

command: #011201(cr)

response: !01

The command set digital channel 2 "ON" status for the specific module at address 01h.

Example 2:

command: #011200(cr)

response: !01

The command set digital channel 2 "OFF" status for the specific module at address 01h.

8.4.20 $AA0Mcc Read DI counter debounce time

Description: Read DI counter pre-debounce and post-debounce of channel N (uint = 0.5ms)

Syntax: $AA0Mcc(cr)

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

0M command for read DI counter filter of channel N

cc represents DI channel number (00~0F)

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hex address of the corresponding EDAM-9000(A) module.

(data) DI counter min. LOW/HIGH witdh, 16-chars(L/H) each channel, (hex/unit=0.5ms)

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

Example:

command: $010M00(cr)

response: !010000000200000003(cr)

00000002 represents channel(0) Low signal width.

00000003 represents channel(0) High signal width.

8.4.21 $AA0MCC(data) Set DI counter debounce time

Description: Set DI counter pre-debounce and post-debounce of channel N (uint = 0.5ms)

Syntax: $AA0MCC(data)(cr)

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

0M command for read DI counter filter of channel N

cc represents DI channel number (00~0F)

(data) DI counter pre-debounce and post-debounce time, 16-chars(L/H) each channel, (unit=0.5ms)

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hex address of the corresponding EDAM-9000(A) module.

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

Example:

command: $010M0000000000200000003(cr)

response: !01(cr)

set channel(0) pre-debounce time to 00000002 x0.5msec=1 msec.

Set channel(0) post-debounce time to 00000003 x 0.5 msec=1.5 msec

8.4.22 $AAEcn Start/stop single DI counter

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

Syntax: $AAEcn(cr)

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

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

?AA(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	represents the 2-character hex address of the corresponding EDAM-9000(A) module.
(cr)	is the terminating character, carriage return (0Dh)

Example 1:

command: *$01E21(cr)*

response: *!01(cr)*

1 represents start DI(2) counter

Example 2:

command: $01E20(cr)

response: !01(cr)

0 represents stop DI(2) counter

8.4.23 $AACn Clear single DI counter value and overflow flag

Description: Clear single digital input counter value with overflow flag

Syntax: $AACn(cr)

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

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

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	represents the 2-character hex address of the corresponding EDAM-9000(A) module.
(cr)	is the terminating character, carriage return (0Dh)

Example:

command:	$01C2(cr)
response:	!01(cr)

2 represents DI counter channel 2

8.4.24 #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

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hex address of the corresponding EDAM-9000(A) module.

(data)… 10-characters(decimal) represents counter values

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

Example:

command: #01(cr)

response: !0100000001230230000001000000052300000050011010000432……(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 only valid for EDAM9050/9051/9052 digital I/O modules.

This command is supported for EDAM9050/9051/9052 with firmware V2.21 or later.

8.4.25 #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
n	represents DI channel number (0~F)
(cr)	is the terminating character, carriage return (0Dh)

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	represents the 2-character hex address of the corresponding EDAM-9000(A) module.
(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

8.4.26 #AArn Read single DI counter value and overflow flag

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

Syntax: #AArn(cr)

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

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	represents the 2-character hex address of the corresponding EDAM-9000(A) module.
r	DI Counter Overflow. = 0 - No counter overflow has occurred. = 1 - A counter overflow has occurred.
(data)	10-characters(decimal) represents counter value (0~4294967295)
(cr)	is the terminating character, carriage return (0Dh)

Example 1:

command: #01RC(cr)

response: !0110000000123(cr)

C represents DI counter channel 12,

10000000123 represents counter value is 123 and counter overflow has occurred.

Example 2:

command: #01RC(cr)

response: !0100000000123(cr)

C represents DI counter channel 12,

00000000123 represents counter value is 123 and No counter overflow has occurred.

8.4.27 #AA2npppppppp Write DO pulse counts

Description: Generates pulse output of the specific DO channel.

Syntax: #AA2npppppppp(cr)

# is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

2 represent generates DO pulse output command.

n represents DO channel n

pppppppp represents pulse counts (8 digits) (0000~FFFFFFFF)
if pppppppp = 00000000, continue DO pulse
if pppppppp = 00000001, stop DO pulse

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

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

?AA(cr) if an invalid operation was entered.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hex address of the corresponding EDAM-9000(A) module.

(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

Ref. command: $AA9NN, #AA2PNNST

8.4.28 $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

CLS represents clear DI latch status command.

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hex address of the corresponding EDAM-9000(A) module.

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

Example:

command: $01CLS(cr)

response: !01(cr)

This command clears all DI latch status

8.4.29 ~AADSMN Set DI/O active state

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

Syntax: ~AADSMN(cr)

# is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

DS Command for setting DIO active status

M Digital input channel active values,
= 0 - represent input value=0 is active (ON), input value=1 is inactive (OFF or OPEN).
= 1 - represent input value=1 is active (ON), input value=0 is inactive (OFF or OPEN).

N Digital output channel active values,
= 0 – represent output low is active (ON),output output high/open is inactive
= 1 - represent output high/open is active (ON),output output low is inactive

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

?AA(cr) if an invalid command has been issued.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex Modbus network address of a module

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

Example 1: read DI/DO active state

Command: ~01DS(cr)
Response: !0101(cr)
0-represent DI active state is 0

1- represent DO active state is 1
2-

Example 2: reads the value of the DI/DO channels

Command: @01(cr)
Response: >3DFFF(cr)

Example 2: set DI and DO active state

Command: ~01DS11(cr)
Response: !01(cr)
1- set input active state= 1, when input value=1

1- set output active state =1, when output value=high/open
2-

Example 2: reads the value of the DI/DO channels

Command: @01(cr)
Response: >3D000(cr)

8.4.30 ~AADS Read DI/O active state

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

Syntax: ~AADS(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
DS	Command for setting DIO active status

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

?AA(cr)	if an invalid command has been issued. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hex Modbus network address of a module
M	Digital input channel active state
N	Digital output channel active state
(cr)	is the terminating character, carriage return (0Dh)

Example:

Read EDAM-9050A active state

```
Command:   ~01DS(cr)
Response:   !0100(cr)
```

DO active state is 0 and DI active state is 0

Reads the value of the DI and DO channels

```
Command:   @01(cr)
Response:   >00FFF(cr)
```

Set input active state to 0 and output active state to 1(ON)

```
Command:   ~01DS01(cr)
Response:    !01(cr)
```

Reads the vaslue of the DI and DO channels

```
Command:   @01(cr)
Response:   >3FFFF(cr)
```

8.4.31 $AACONNDD Set a single DO channel mode

Description: Set a single DO channel mode. (available for version 6.070 or later)

Syntax: $AACONNDD(cr)

# is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal network address

CO command for set a single DO mode for channel N

NN channel number (00~1F)

DD DO mode setting (2 characters),

bit(2,1,0) - DO mode setting.

= 0x000 - Direct DO output

= 0x001 - Pluse output mode

= 0x010 - Low to high delay

= 0x011 - High to low delay

= 0x100 - DIO Sync. Mode

bit(6,5,4,3) - always 0,

bit(7) - Set digital output channel active state

= 0 – represent output low is active (ON), outputt high/open is inactive

= 1 - represent output high/open is active (ON), output low is inactive

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex Modbus network address of a module

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

Example:

Set DO channel(0) to *Low to high delay mode* and write 1 to digital output channel(0)

```
Command:  $01CO0082(cr)      // bit(2,1,0) = 0x010  &  bit(7)=1
Response:  !01(cr)
```

Ref. command: $AACONN, $AACIINNDD, $AACINN

8.4.32 $AACONN Read DO mode

Description: Read a single DO channel mode . (available for version 6.070 or later)

Syntax: $AACONN(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal network address
CO	command to read a single DO channel mode
NN	DO channel number(00~1F)

Response: !AADD(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 hex Modbus network address of a module

DD DO mode setting (2 characters),

bit(2,1,0) - DO mode setting.

= 0x000 - Direct DO output
= 0x001 - Pluse output mode
= 0x010 - Low to high delay
= 0x011 - High to low delay
= 0x100 - DIO Sync. Mode

bit(6,5,4,3) - always 0,

bit(7) – current active state of DO channel NN

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

Example:

Set DO channel(0) Low to high delay mode and digital output to 0

```
Command:  $01CO00(cr)
Response:  !0182(cr)          // bit(2,1,0) = 0x010   &   bit(7)=1
```

Ref. command: $AACONNDD, $AACIINNDD, $AACINN

8.4.33 $AACINNDD Set a single DI channel mode

Description: Set a single DI channel mode (available for version 6.070 or later)

Syntax: $AACINNDD(cr)

# is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal network address

CI command for set a single DI mode for channel N

NN channel number (00~1F)

DD DI mode setting (2 characters),
bit(2,1,0) - DI mode setting.
= 0x000 - Direct DI input
= 0x001 - Counter Mode
= 0x010 - Low to high latch
= 0x011 - High to low latch
= 0x100 - Input frequency mode
bit(6) - DI counter H/L width filter (Default H/L width =5ms)
= 0 - disable (off)
= 1 - enable (on)
bit(7) - Digital input channel active values
= 0 - represent input value=0 is active (ON), input value=1 is inactive (OFF or OPEN).
= 1 - represent input value=1 is active (ON), input value=0 is inactive (OFF or OPEN).

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

?AA(cr) if an invalid command has been issued.
There is no response if the module detects a syntax error or communication error or if the address does not exist.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex Modbus network address

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

Ref. command: $AACINN, $AACONNDD, $AACONN

8.4.34 $AACINN Read a single DI channel mode

Description: Read a single DI channel mode (available for version 6.070 or later)

Syntax: $AACINN(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal network address
CI	command for read a single DI mode for channel N
NN	channel number (00~1F)

Response: !AADD(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 hex Modbus network address

DD DI mode setting (2 characters),

bit(2,1,0) - DI mode setting.

= 0x000 - Direct DI input
= 0x001 - Counter Mode
= 0x010 - Low to high latch
= 0x011 - High to low latch
= 0x100 - Input frequency mode

bit(6) - DI counter H/L width filter. (Default H/L width =5ms)

= 0 - disable (off)
= 1 - enable (on)

bit(7) - Digital input channel active values

= 0 - represent input value=0 is active (ON), input value=1 is inactive (OFF or OPEN).

= 1 - represent input value=1 is active (ON), input value=0 is inactive (OFF or OPEN).

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

Ref. command: $AACINNDD, $AACONNDD, $AACONN

8.4.35 @AA6 Read DIO status

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

Syntax: @AA6(cr)

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

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
TTTT	represents the 4-character hexadecimal DO status (0000~FFFF).
NNNN	represents the 4-character hexadecimal DI status (0000~FFFF)
(cr)	is the terminating character, carriage return (0Dh)

Example: (for E-9050A)

command: @016(cr)

response: >00030004(cr)

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

0004 represents DI2 is ON and DI0, DI1, and DI3~DI15 are OFF

8.4.36 @AA6DDDD Write DO channels (0~15)

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

Syntax:	@AA6DDDD(cr)
@	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
6	command for read DIO status
DDDD	represents the data be written to digital output(channels 0~15)
(cr)	is the terminating character, carriage return (0Dh)

Response:	>(cr) if the command is valid.
?01(cr)	if an invalid operation was entered.
	There is no response if the module detects a syntax error or communication error or if the address does not exist.
>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

Example: (for E-9053A)

command: @016123A(cr)

response: >(cr)

8.4.37 @AA6Onss Set/reset a single digital output channel

Description: Set/reset a single digital output channel (available for version 6.000 or later)

Syntax: @AA6Onss(cr)

@ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

6O command to activate/deactivate a single digital output.

n channel number(0~F)

ss output state of DO channel n

= 00 – deactivate(reset) digital output channel n.

= 01 – activate(set) digital output channel n.

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

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

Example: Set DO(1) to active state

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

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

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

8.4.38 @AA6ON Read a single digital output channel

Description: Read status of a single digital output channel (available for version 6.000 or later)

Syntax: @AA6On(cr)

@	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
6O	command to read status of a single digital output channel.
n	channel number(0~F)
(cr)	is the terminating character, carriage return (0Dh)

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
>	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
DD	output status = 00 - OFF (inactive) = 01 - ON (active)
(cr)	is the terminating character, carriage return (0Dh)

Example: Read DO(1) status

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

8.4.39 @AA6IN Read a single digital input channel

Description: Read status of a single digital input channel (available for version 6.000 or later)

Syntax: @AA6IN(cr)

@ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

6I command to read status of a single digital input channel.

N DI channel number(0~F)

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

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

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

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

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

DD input value
= 00 - OFF (inactive)
= 01 - ON (active)

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

Example: Read the status of DI channel(1)

command: @016I1(cr)

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

8.4.40 $AA9DNNLLLLHHHH Set DO low/high delay time

Description: Set Low to High or High to Low delay time of DO channel N (unit: 0.5ms)
(available for version 6.070 or later)

Syntax: $AA9DNNLLLLHHHH(cr)

$	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
9D	Command to set DO low/high delay time of DO channel N
NN	DO channel number. = 00~0F (hex) - channel number. = FF (hex) - copy the setting to all DO channels
LLLL	4 char, DO low to high delay width (hex 0001~hex 3332,uint 0.5ms)
HHHH	4 char, DO high to low delay width (hex 0001~hex 3332,uint 0.5ms)
(cr)	is the terminating character, carriage return (0Dh)

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hex Modbus network address
(cr)	is the terminating character, carriage return (0Dh)

Example: Set DO Low/high delay output width of channel 2

```
command:    $019D0201230456(cr)
response:   !01(cr)          ; valid
```

Ref. command: $AA9NN, $AA9TN

8.4.41 $AA9PNNLLLLHHHH Set DO pluse high/low width of channel N

Description: Set DO Pluse High/ Low output width of channel N (unit: 0.5ms)
(available for version 6.070 or later)

Syntax: $AA9PNNLLLLHHHH(cr)

$	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
9P	command to set DO pluse high/low width of channel N
NN	DO channel number.
	= 00~0F (hex) - DO channel number.
	= FF (hex) - copy the setting to all DO channels
LLLL	4 char, DO pulse low signal width (hex 0001~hex 3332/uint 0.5ms)
HHHH	4 char, DO pulse high signal width (hex 0001~hex 3332/ uint 0.5ms)
(cr)	is the terminating character, carriage return (0Dh)

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hex Modbus network address
(cr)	is the terminating character, carriage return (0Dh)

Example: (for E-9063A) Set DO pulse Low/high output width of channel 2

```
command:    $019P0201230456(cr)
response:   !01(cr)           ; valid
```

Ref. command: $AA9NN, #AA2NPPPPPPPP, #AA2PNNST

8.4.42 $AA9TN Generate a output pulse of DO channel N

Description: This command works as a monostable multivibrator that generates an pule output.

When module receives this command, a pulse of pre-defined duration (High delay or Low delay width) is produced. The DO then returns to its quiescent state and produces no more output until next command received again. *(Only available when DO channel is set to "High to Low" or"Low to High" delay output mode)* (available for version 6.070 or later), (ref. to 0)

Syntax: $AA9TN(cr)

$	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
9T	command to start output pulse
N	channel number, 00~0F (hex).
(cr)	is the terminating character, carriage return (0Dh)

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

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

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

!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
AA	(range 00-FF) represents the 2-character hexl Modbus network address
(cr)	is the terminating character, carriage return (0Dh)

Example: Gerenate a pulse output with 3 sec width (DO active –> holds delay 3 secs –>inactive)

Set D/O channel(0) to "*<u>High to Low delay mode</u>*"

```
command:    $01CO0003(cr)
response:   !01(cr)         ; valid
```

Set D/O(0 channel(0) <u>low to high</u> delay width(0.5ms) and <u>high to low</u> delay width(3000ms).

```
command:    $019D0000011770(cr)
response:   !01(cr)         ; valid
```

Start DO channe(0) delay 0.5 ms and then generate a pulse output (pulse width=3000ms)

```
command:    $019T0(cr)
response:   !01(cr)         ; valid
```

Ref. command: $AACONNDD, $AA9DNNLLLLHHHH

8.4.43 $AAYM1CPSHHHHLLLL(data) Set DIO Sync. DI match/mismatch DO Toggle Mode

Description: Set *DI match/mismatch DO Toggle Mode* parameters of *DIO Synchronization* (Ref. to 14.11).

A single digital output channel is activated(1 or 0) dependent on the DI input value, When the DI input value match/mismatch DI mask pattern,the specific DO channel will be set to active(1 or 0).

(avalibale for version 6.070 or later).

Syntax: $AAYM1CPSHHHHLLLL(data)

$ is a delimiter character.

AA represents the 2-character hexadecimal ID address.

YM1 represents DO Toggle Mode of Automatic DIO Synchronization.

C DO channel number (hex 0~F) been mirrored.

P Enable/Disable Auto Run(Start) DIO Synchronization operation when power-on.

= 0 - Disiable

= 1 - Enable

S set digital output to active state(0/1) when DI value match DI mask pattern
set digital output to inactive state(0/1) when DI value mismatch DI mask pattern

HHHH DI channel pre-debounce time (hex 0000~FFFF ms)

LLLL DI channel post-debounce time (hex 0000~FFFF ms)

(data) represents monitored DI channels and DI mask pattern(16 char)
bits(15..0) - represents DI channel(15..0) state been monitored or ignored
= '1' -represent DI channel n is monitored and match state is '1'
= '0' - represent DI channel n is monitored and match state is '0'
= 'X' - don't care (DI channel isn't monitored)

(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.

AA (range 00-FF) represents the 2-character hex Modbus network address

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

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

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

command: $01YM1011012C0096XXXXXXXXXX1XX0X1(cr). (Ref. 14.11)

where P = 1 enable Auto Run(Start) DIO Synchronization operation when power-on.

S = 1 digital output active state(=1) when DI value meet data match pattern

HHHH = 0x012C (300ms)

LLLL = 0x0096 (150ms)

response: !01(cr) ; valid

Ref. command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM2CPSTTTT (data), $AAYM3CPSTTTT(data)

8.4.44 $AAYM2CPSTTTT (data) Set DIO Sync. DI match DO latch Mode

Description: Set DO to *DI match DO latch Mode* of *DIO Synchronization*. A single digital output channel is activated (1 or 0) and latched, when DI input value match DI mask pattern. (Ref. to 14.11). ***(available for version 6.070 or later),***

Syntax: $AAYM2CPSTTTT (data)

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

YM2 command to set DI match/ DO latch mode of *Auto DIO Sync*.

C mirrored DO channel number (hex 0~F).

P enable/Disable Auto Run(Start) DIO Synchronization operation when power-on.

= 0 - Disiable

= 1 - Enable

S digital output active state(0/1) when DI input value match DI mask pattern

TTTT DI channel pre-bounce time (hex 0000~FFFF ms)

(data) represent DI mask pattern is used to indicate the monitored input channels(0~15) and mask state (16 char)

bits(15..0) - indicate DI channel(15..0) to be monitored and mask state

= '1' - indicate DI channel n is monitored and mask state is '1'.

= '0' - indicate DI channel n is monitored and mask state is '0'.

= 'X' - don't care (not be monitored)

(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.

AA (range 00-FF) represents the 2-character hex Modbus network address

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

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

Example: Assume DI channel 0,2,5 are monitored and the ASCII form of DI mask pattern is XXXXXXXXXX1XX0X1.When DI input ch(0,5)=1 and ch(2)=0, the corresponding DO(0) will be set to ON(1) and latched. DI pre-debounce time is 300 msec. (Ref. to 14.11)

command: $01YM2011012CXXXXXXXXXX1XX0X1(cr)

P = 1 - Enable Auto Run(Start) DIO Synchronization operation when power-on.

S = 1 - digital output active state(=1) when DI value match DI mask pattern

TTTT = 0x012C (300ms)

response: !01(cr) ; valid

Ref. command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM1CPSHHHHLLLL (data), $AAYM3CPSTTTT(data)

8.4.45 $AAYM3CPSTTTT (data) Set DIO Sync. DI mismatch DO latch Mode

Description: Set DO to *DI mismatch DO latch Mode* for *Automatic DIO Synchronization*. A single digital output channel is activated (1 or 0) and latched, when DI input value mismatch DI mask pattern. (Ref. to 14.11)
(available for version 6.070 or later),

Syntax: $AAYM3CPSTTTT (data)

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

YM3 command to set DI mismatch/DO latch mode of Auto DIO Sync.

C mirrored DO channel number (hex 0~F).

P enable/Disable Auto Run(Start) DIO Synchronization operation when power-on.
= 0 - Disiable
= 1 - Enable

S digital output active state(0/1) when DI input value mismatch DI mask pattern

TTTT DI channel pre-debounce time (hex 0000~FFFF ms)

(data) represent DI mask pattern is used to indicate the monitored input channels(0~15) and mask state, (16 char)
bits(15..0) - indicate DI channel(15..0) to be monitored and mask state
= '1' - indicate DI channel n is monitored and mask state is '1'.
= '0' - indicate DI channel n is monitored and mask state is '0'.
= 'X' - don't care (not be monitored)

(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.

AA (range 00-FF) represents the 2-character hex Modbus network address

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

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

Example: Assume DI channel 0,2,5 are monitored and the ASCII form of DI mask pattern is XXXXXXXXXX1XX0X1. When DI input ch(0)≠1 or ch(5)≠1 or ch(2)≠0(mismatch DI mask pattern), the corresponding DO(0) will be set to ON(1) and latched. DI pre-debounce time is 300 msec.
(Ref. to 14.11)

command: $01YM3011012CXXXXXXXXXX1XX0X1(cr)
P = 1 - enable Auto Run(Start) DIO Synchronization operation when power-on.
S = 1 - digital output active state(=1) when DI value mismatch DI mask pattern
TTTT = 0x012C (300ms)
response: !01(cr) ; valid

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

8.4.46 $AAYMRCS Start(Run)/Stop *DIO Synchronization* operation

Description: This command is used to start (run)/stop the DIO Synchronization operation. (available for version 6.070 or later)

Syntax: $AAY6CS

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

YMR represent this command is used to start/stop DIO Synchronization operation.

C mirrored DO channel number (hex 0~F).

S start/stop DIO Synchronization operation.

= 0 - Stop DIO Synchronization operation (default)
= 1 - Start(Run) DIO Synchronization operation.

(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.

AA (range 00-FF) represents the 2-character hex Modbus network address

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

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

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

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

```
command:    $01CO0004(cr)
response:   !01(cr)     ; valid
```

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

```
command:    $01YM1011012C0096XXXXXXXXXX1XX0X1(cr)
response:   !01(cr)     ; valid
```

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

```
command:    @016O000(cr)
response:   !01(cr)     ; valid
```

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

```
command:    $01YMR01(cr)
response:   !01(cr)     ; valid
```

8.4.47 $AAYMC Read DIO Synchronization Mode parameters

Description: This command is used to read parameters of DIO Synchronization Mode (Ref. to 14.11)
(available for version 6.070 or later)

Syntax: $AAYMC

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

YM command to read parameters of DIO Synchronization Mode

C mirrored DO channel number (hex 0~F).

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

Response:

For " DI match DO toggle Mode ":

!AADPSHHHHLLLL(data)(cr) if the command is valid.

For " DI match/mismatch DO latch Mode ":

!AADPSTTTT(data)(cr) if the command is valid.

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex Modbus network address

D specific DO channel mode.
= 0 - specific DO channel isn't mirrored to DI hannel(s).
= 1 - DIO Synchronization- DI match DO toggle Mode
= 2 - DIO Synchronization- DI match DO latch Mode.
= 3 - DIO Synchronization- DI mismatch DO latch Mode.

P enable/Disable Auto Run(Start) DIO Synchronization operation when power-on.
= 0 - Disiable
= 1 - Enable

S digital output active state (0/1), when DI input value match/mismatch DI mask pattern.

For "DI match DO toggle Mode ":

HHHH - DI channel pre-debounce time (0000~FFFF/ms)
LLLL - DI channel post-debounce time(0000~FFFF/ms)

For " DI match/mismatch DO latch Mode ":

TTTT - DI channels pre-debounce time(0000~FFFF/ms).

(data) This parameter is used to indicate the monitored input channels(15~0) (16 char).
bits(15..0) - indicate DI channel(15..0) state to be monitored
bit n = '1' or '0' - indicate DI channel n(0~15) is monitored and mask state is 1 or 0
= 'X' - indicate the DI channel n(0~15) isn't monitored
(ex: DI channel(0,1,2,4,5,7) are monitored, (data) = "XXXXXXXX0X10X011"

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

Example: Read parametersof DO(0) DIO sync Mode *(Ref. 14.7)*

command: $01YM0(cr) // Read DIO sync Mode parameters of DO(0).
response: !01301012CXXXXXXXXXXXXXX01(cr) ; valid

Ref. command: $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MS, $AAYM1CPSHHHHLLLL (data),
$AAYM2CPSTTTT(data)

8.4.48 $AAYMS Read DO current status during DIO Synchronization operation

Description: Read DO channel current status (Ref. to 14.11) (available for version 6.070 or later)

Syntax: $AAYMS

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

YMS command to read DO current state during DIO Synchronization operation

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex Modbus network address

DDDD DO channels (15..0) status(1=active, 0=inactive).

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

Example: Read DO current status of DIO Synchronization (Ref.14.7)

```
command:   $01YMS(cr)
response:  !010005(cr)   ; valid and DO(0,2) have been activated
```

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

8.4.49 $AA9NN Read DO pulse and DO low/high delay width for channel N

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

Syntax: $AA9NN(cr)

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

9 command for Read a single digital input.

NN channel number.(00~0F (hex)

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

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

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

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

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-FF) represents the 2-character hex Modbus network address

(data) DO pulse time interval and low/high width (LLLLHHHHUUUUDDDD)

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

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

Example: Read DO pulse and Low/high delay output width of DO channel 2

command: $01902(cr)

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

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

Description: Host send this command via broadcast IP to tell all modules that host and network are alive (No reply from modules) (available for firmware version 6.000 or later)

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

Syntax: ~**(cr)

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

Response: No response.

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

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

Description: Host sends this command via specific module IP to tell the module that host and network are alive (available for firmware version 6.000 or later)

When host watchdog timer is enable, host computer must send this command to specific module before timeout otherwise "Host watchdog timer enabled" module will go to safety state.

Syntax: ~AA**(cr)

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

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

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

8.4.52 ~AA0 Read watchdog timeout status

Description: Read watchdog timeout status. (available firmware version 6.000 or later)

Syntax: ~AA0(cr)

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

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

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

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

! delimiter indicating a valid command was received.

SS two hexadecimal digits that represent the host watchdog status.

bit(7) - Host watchdog enable/disable ,

= 0 - Disable.

= 1 - Enable.

bit(2) - Host watchdog timeout status,

= 0 - indicates that no host watchdog timeout has occurred.

= 1 - indicates that a host watchdog timeout has occurred.

bit(6,5,4,3,1,0) - reserved(always 0)

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

? delimiter indicating the command was invalid.

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

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

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

Example 2: Reads the host watchdog status of module 01 and returns 04, meaning that a host watchdog timeout has occurred.

```
Command: ~010(cr)
Response:   !0104(cr)
```

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

8.4.53 ~AA1 Reset host watchdog timeout status

Description: Reset host watchdog timeout status (available for version 6.000 or later)

Syntax: ~AA1(cr)

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

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
!	delimiter indicating a valid command was received.
?	delimiter indicating the command was invalid.
(cr)	is the terminating character, carriage return (0Dh)

Example 1: Reads the host watchdog status of module 01 and shows that a host watchdog timeout has occurred.

```
command:  ~010(cr)
response:  !0104(cr)
```

Example 2: Resets the host watchdog timeout status of module 01 and returns a valid response.

```
command:  ~011(cr)
Response:  !01(cr)
```

Example 3: Reads the host watchdog status of module 01 and shows that no host watchdog timeout has occurred.

```
command:  ~010(cr)
response:  !0100(cr)
```

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

8.4.54 ~AA2 Read host communication Timeout value.

Description: Read host communication Timeout value. (available for firmware version 6.000 or later)

Syntax: ~AA2(cr)

~ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

2 command for reading watchdog timeout value

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

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

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

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

? delimiter indicating the command was invalid.

! delimiter indicating a valid command was received.

E Host watchdog enabled status

= 1 – Enable.

= 0 – Disable..

VVV Timeout value in hex format from hex 001 to 28F .(unit 0.1 sec),FF denotes 25.5 seconds

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

Example: Reads the host watchdog timeout value of module 01 and returns FF, which denotes that the host watchdog is enabled and the host watchdog timeout value is 25.5 seconds.

```
command:   ~012(cr)
response:  !011FF(cr)
```

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

8.4.55 ~AA3EVVV Set Host watchdog timeout interval

Description: Enable/disable Host watchdog and set timeout interval (unit = 0.1sec)
(available for firmware version 6.000 or later)

Syntax: ~AA3EVVV(cr)

~ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

3 command for setting host wdt Enable/ disable and host wdt timeout value.

E host watchdog enabled status
E = 1 – Enable.
E = 0 – Disable..

VVV Timeout value in hex format from hex 001 to 28F.(unit:0.1 sec) , FF denotes 25.5 seconds

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

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

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

? delimiter indicating the command was invalid.

! delimiter indicating a valid command was received.

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

Note:

*If host watchdog timer is enabled, the host should send Host OK (see "~**" or ~AA**) command periodically within Timeout value to refresh the timer, otherwise the module will be forced to safet state (see "~AA5V") and The Power-LED on the module will go to flash. After timeout the all of D/O commands are disabled.*

Example:

Set module (ID=01) to have watchdog timeout value 20.0 seconds and enable host watchdog.

command: ~01310C8(cr)
response: !01(cr)

Read watchdog timeout value form module (ID=01). The module returns 10C8, which denotes that the host watchdog is enabled and the host watchdog timeout value is 20.0 seconds.

command: ~012(cr)
response: !0110C8(cr)

Host send this command to all modules that host and network are alive

command: ~**(cr) ; Host OK (to all modules)
or
command: ~01**(cr) ; Host OK (to the Specific module)

Stop sending any command string to modules for at least 20.0 seconds. The Power- LED on the module will go to flash. The flash LED indicates the host watchdog is timeout and timeout status flag is set.

Read watchdog timeout status, the module returns 01, which denotes that a host watchdog timeout has occurred.

command: ~010(cr)
response: !0104(cr)

Reset watchdog timeout status. Watchdog timeout is cleared and LED stops flashing, and host watchdog is disabled

command: ~011(cr)
response: !01(cr)

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

command: ~010(cr)
response: !0100(cr) Timeout status is cleared

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

8.4.56 ~AA4V Read power-on or safe DO value of module

Description: Read power-on or safe DO value of module (available for firmware version 6.000 or later)

Syntax: ~AA4V(cr)

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
4	command for read the power-on DO value or the safe DO value of a module
V	read power-on value or safe value = P - read power-on value, = S - read safe value.
(cr)	is the terminating character, carriage return (0Dh)

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.
DDDD	powe-on or safe value.
(cr)	is the terminating character, carriage return (0Dh)

Example 1: Read Power on value and return power-on value 5A5A.

command: ~014P(cr)

response: !045A5A(cr)

Example 2: Read Power on value and return safe value AA00.

command: ~014S(cr)

response: !04AA00(cr)

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

8.4.57 ~AA3PPP Set module Power-on delay time.

Description: Set module Power-on delay time.(available for firmware version 6.000 or later)

Syntax: ~AA3PPP(cr)

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
3	command for set module Power-on delay time.
PPP	power-on delay time(unit:0.1sec) to start communication timeout. (range 001~28F) (default 5 sec).
(cr)	is the terminating character, carriage return (0Dh)

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.
(cr)	is the terminating character, carriage return (0Dh)

Example: **S**et power-on delay time to 096 (15 sec)

Command: **~013096(cr)**

Response: **!01(cr)**

Ref. command: ~AA3P

8.4.58 ~AA5V Sets the current DO value as power-on or safe value

Description: Set the current DO value as power-on or safe value (available for firmware version 6.000 or later)

Syntax: ~AA5V(cr)

~ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address

5 command for sets the current value as the power-on DO value or the safe DO value

V sets the current DO value as power-on value or safe value

= P - set to power-on value,

= S - set to safe value.

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

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

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

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

? delimiter indicating the command was invalid.

! delimiter indicating a valid command was received.

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

Example 1: Set module DO to have output value 2A.

Command: @012A(cr)

Response: >(cr)

Example2: Set current output value 2A as safe value.

Command: ~015S(cr)

Response: !01(cr)

Example 3: Set module to have output value 15.

Command: @0115(cr)

Response: >(cr)

Example 4: Set current output value 15 as power-on value.

Command: ~015P(cr)

Response: !01(cr)

Example 5: Read Power on value and return power-on value 0015.

Command: ~014P(cr)

Response: !010015(cr)

Example6: Read Power on value and return safe value 2A.

Command: ~014S(cr)

Response: !01002A(cr)

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

8.4.59 ~AA3P Read module Power-on delay time.

Description: Read module Power-on delay time. (available for firmware version 6.000 or later)

Syntax: ~AA3P(cr)

~	is a delimiter character.
AA	represents the 2-character hexadecimal Modbus address
3P	command for read module Power-on delay time.
(cr)	is the terminating character, carriage return (0Dh)

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

?01(cr)	if an invalid operation was entered. There is no response if the module detects a syntax error or communication error or if the address does not exist.
?	delimiter indicating the command was invalid.
!	delimiter indicating a valid command was received.
PPP	Power-on delay time (unit=0.1sec) to start communication timeout counting. (range 001~28F)
(cr)	is the terminating character, carriage return (0Dh)

Example 1: Set power-on delay time to 096 (15 sec)

```
command:    ~013096(cr)
response:   !01(cr)
```

Example 2: Read power-on delay time and return 096 (15 sec)

```
command:    ~013P(cr)
response:   !01096(cr)
```

Ref. command: ~AA3PPP

8.4.60 #AAn Read value of analog Input channel N

Description: Returns the input data from a specific analog input channel in a specific module.

Syntax: #AAn(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
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 EDAM-9000 analog module at address 01h responds with an input value +01.000.

8.4.61 #AA Read value of all Analog Input Channels

Description: Returns the input data from all analog input channels in a specific module.

Syntax: #AA(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
(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**

8.4.62 $AAAcctt Set analog input type (range)

Description: Set the analog input type (range) in EDAM-9000 analog input module.

Syntax: $AAAnntt(cr)

$ is a delimiter character.

01 represents the 2-character hexadecimal Modbus address

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

(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 hex address of the corresponding module. .

(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

8.4.63 $AABhh Read analog input type

Description: Returns the input type of the specific analog channel

Syntax: $AABhh(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 .

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 EDAM-9000 module. The second 2-character portion of the response is the type of channel (For each analog module, the type number is different)

Code(Hex)	Type
0x07	4-20mA
0x08	+/-10V
0x09	+/-5V
0x0a	+/-1V
0x0b	+/-500mV
0x0c	+/-150mV
0x0d	0-20mA
0x0e	J type -8824 uV ~ 69536 uV,
0x0f	K type -5891 uV ~ 54807 uV
0x10	T type -5603 uV ~ 20869 uV
0x11	E type -9835 uV ~ 76373 uV
0x12	R type -0000 uV ~ 21101 uV
0x13	S type -0000 uV ~ 18693 uV
0x14	B type -0000 uV ~ 13820 uV
0x20	IEC Pt100 -50C ~ 150C
0x21	IEC Pt100 0C ~ 100C
0x22	IEC Pt100 0C ~ 200C
0x23	IEC Pt100 0C ~ 400C
0x24	IEC Pt100 -200C ~ 200C
0x25	JIS Pt100 -50C ~ 150C

0x26	JIS Pt100 0C ~ 100C
0x27	JIS Pt100 0C ~ 200C
0x28	JIS Pt100 0C ~ 400C
0x29	JIS Pt100 -200C ~ 200C
0x2A	Pt1000 -40C ~ 160C
0x2B	BALCO500 -30C ~ 120C
0x2C	Ni -80C ~ 100C
0x2D	Ni 0C ~ 100C

Figure 8-1 Analog input types

8.4.64 $AA0 Span Calibration

Description: Calibrates a specific module to correct gain errors

Syntax: $AA0(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 EDAM-9000 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.

8.4.65 $AA1 Zero Calibration

Description: Calibrates a specific module to correct offset errors

Syntax: $AA1(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 hex Modbus address of EDAM-9000 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.

8.4.66 $AA6 Read AI channel enable/disable Status

Description: Asks a specific 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 address of the corresponding module.

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 hex Modbus address of an EDAM-9000 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).

8.4.67 $AA5mm Enable/disable all AI channels

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 address of the corresponding module.
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 address of the corresponding 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.

8.4.68 #AAMH Read Maximum Value of AI channels

Description: Read the maximum values from all analog input channels in a specific analog module

Syntax: #AAMH(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
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.
Data..	represents maximum analog data
(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.

8.4.69 #AAMHn Read Maximum Value of AI channel N

Description: Read the maximum value from a specific channel in a specific module

Syntax: #AAMHn(cr)

#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
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.
Data	represents maximum analog data
(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

8.4.70 #AAML Read Minimum Value of all AI channels

Description: Read the minimum values from all analog input channels in a specific module

Syntax: #AAML(cr)

# is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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.

Data.. represents minimum analog data

(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.

8.4.71 #AAMLn Read Minimum Value of AI channel N

Description: Read the minimum value from a specific analog input channel in a specific module

Syntax:	#AAMLn(cr)
#	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module.
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.
Data	represents minimum analog data
(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.

8.4.72 $AACjAhs Set Alarm Mode

Description: Sets the High/Low alarm of the specific input channel in the addressed EDAM-9000 module to either Latching or Momentary mode.

Syntax: $AACjAhs(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

Cj identifies the desired channel j (j =0 ~ 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 EDAM-9000 module.

(cr) represents terminating character, carriage return (0Dh)

Example:

command: $01C1AHL(cr)

response: !01(cr)

Channel 1 of the EDAM-9000 module at address 01h is instructed to set its High alarm in latching mode. The module confirms that the command has been received.

8.4.73 $AACjAh Read Alarm Mode

Description: Returns the alarm mode for the specific channel in the specific EDAM-9000 module.

Syntax: $AACjAh(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

Cj identifies the desired channel j (j = 0 ~ 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 Modbus network address of the corresponding 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 EDAM-9000 module at address 01h is instructed to return its Low alarm mode. The system responds that it is in Momentary mode.

8.4.74 $AACjAhEs Enable/disable alarm

Description: Enables/Disables the High/Low alarm of the specific input channel in the addressed module

Syntax: $AACjAhEs(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

Cj identifies the desired channel j (j = 0 ~ 7).

AhEs is the Set Alarm Mode command.
h indicates alarm type (H = High alarm, L = Low alarm)
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 EDAM-9000 module.

(cr) represents terminating character, carriage return (0Dh)

Example:

command: $01C1ALEE(cr)

response: !01(cr)

Channel 1 of the EDAM-9000 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.

8.4.75 $AACjCh Clear latch alarm

Description: Clear High/Low alarm flag of the specific input channel in the addressed module

Syntax: $AACjCh(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

Cj identifies the desired channel j (j= 0 ~ 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 module

(cr) represents terminating character, carriage return (0Dh)

Example:

command: $01C1CL(cr)

response: !01(cr)

Channel 1 of the EDAM-9000 module at address 01h is instructed to set its Low alarm flag to OFF.

8.4.76 $AACjAhCCn Set Alarm Connection

Description: Connects the High/Low alarm of the specific input channel to interlock the specific digital output in the addressed EDAM-9000 module

Syntax: $AACjAhCCn(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module

Cj identifies the desired analog input channel j (j= 0 ~ 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 ~ 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 (range 00-FF) represents the 2-character hexadecimal address of the corresponding module

(cr) represents terminating character, carriage return (0Dh)

Example 1:

command: $01C1ALCC0(cr)

response: !01(cr)

Channel 1 of the module at address 01h is instructed to connect its Low alarm to the digital output of channel 0 in the module.

Example 2:

command: $01C1ALCC*(cr)

response: !01(cr)

Channel 1 of the module at address 01h is instructed to connect its Low alarm does not interlock to the any digital output in the module.

8.4.77 $AACjRhC Read Alarm Connection

Description: Returns the High/Low alarm limit output connection of a specific input channel in the addressed module

Syntax: $AACjRhC(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module

Cj identifies the desired analog input channel j (j= 0 ~ 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 (range 00-FF) represents the 2-character hexadecimal address of the corresponding 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 is "*", the analog input alarm has no connection with any digital output point.

(cr) represents terminating character, carriage return (0Dh)

Example:

command: $01C1RLC(cr)

response: !01C0(cr)

Channel 1 of the 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.

8.4.78 $AACjAhU Set Alarm Limit

Description: Sets the High/Low alarm limit value of the specific input channel of a specific module.

Syntax: $AACjAhU(data)(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module

Cj identifies the desired analog input channel j (j= 0 ~ 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 (range 00-FF) represents the 2-character hexadecimal address of the corresponding 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 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.

8.4.79 $AACjRhU Read Alarm Limit

Description: Returns the High/Low alarm limit value for the specific input channel in the addressed module

Syntax: $AACjRhU(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module

Cj identifies the desired analog input channel j (j= 0 ~ 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 (range 00-FF) represents the 2-character hexadecimal address of the corresponding 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)

The command instructs the system to return the High alarm limit value for that channel 1. The system responds that the High alarm limit value in the desired channel is 2.0500 V.

8.4.80 $AACjS Read Alarm Status

Description: Reads whether an alarm occurred to the specific input channel in the specific module

Syntax:	$AACjS(cr)
$	is a delimiter character.
AA	(range 00-FF) represents the 2-character hexadecimal address of the corresponding module
Cj	identifies the desired analog input channel j (j= 0 ~ 7).
S	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	(range 00-FF) represents the 2-character hexadecimal address of the corresponding 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.

8.4.81 $AA3 Read cold junction temperature

Description: Return the Cold Junction temperature of EDAM-9019A

Syntax: $AA3(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 specific module. The response is +17.5C

8.4.82 $AA9hhhhh Set CJC offset

Description: Set Cold Junction temperature offset of EDAM-9019A

Syntax: $AA19hhhhh(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 (range 00-FF) represents the 2-character hexadecimal address of the corresponding module

? delimiter indicating the command was invalid.

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

Example:

command: $019000A0(cr)

response: !01(cr)

This example need to set cold junction offset to 2C , then the actual ASCII value should be 2 *80=160 (hex=000A0). Hence the complete ASCII command string is $019000A0(cr)

8.4.83 $AA9 Read CJC offset

Description: Return Cold Junction temperature offset of EDAM-9019A

Syntax: $AA9(cr)

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the corresponding module.

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 specific module. The response is +5.5C.

Chapter 9 MODBUS/TCP Command structure

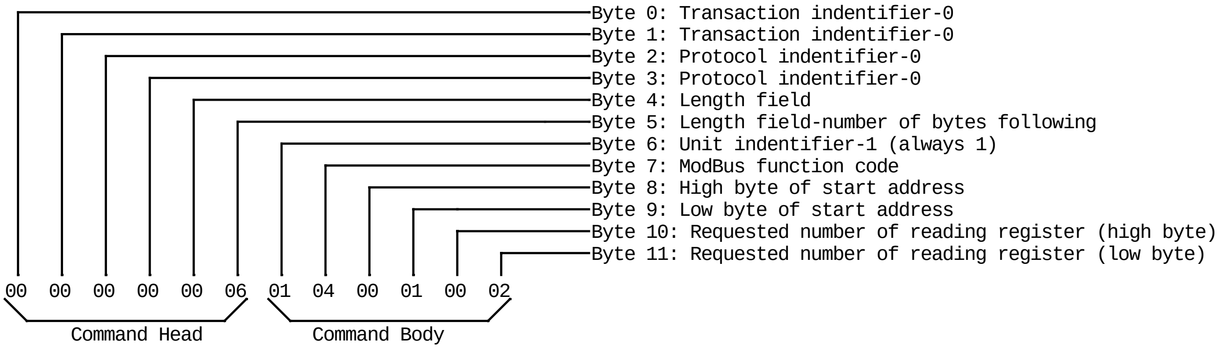
EDAM-9000(A) system accepts a command/response form with the host computer. When systems are not MODBUS/TCP Command structure. EDAM-9000(A)/9400 system accepts a command/response form with the host computer. When systems are not transmitting they are in listen mode. The host issues a command to a system with a specific address and waits a certain amount of time for the system to respond. If no response arrives, a time-out aborts the sequence and returns control to the host. This chapter explains the structure of the commands with Modbus/TCP protocol, and guides to use these command sets to implement user's programs.

9.1 Command Structure

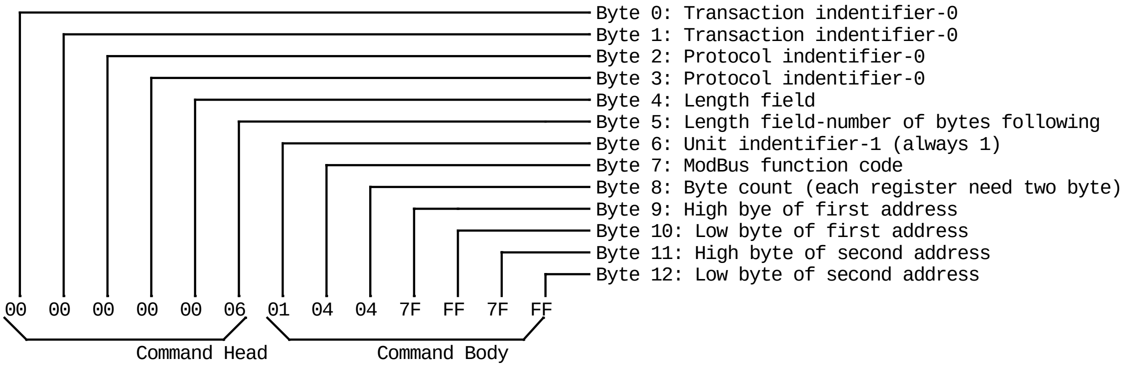
It is important to understand the encapsulation of a Modbus request or response carried on the Modbus/TCP network. A complete command is consisted of command head and command body. The command head is prefixed by six bytes and responded to pack Modbus format; the command body defines target device and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

Example:

If you want to read the first two values of EADM-9017 (address: 40001~40002), the request command should be:


Byte 0: Transaction indentifier-0
Byte 1: Transaction indentifier-0
Byte 2: Protocol indentifier-0
Byte 3: Protocol indentifier-0
Byte 4: Length field
Byte 5: Length field-number of bytes following
Byte 6: Unit indentifier-1 (always 1)
Byte 7: ModBus function code
Byte 8: High byte of start address
Byte 9: Low byte of start address
Byte 10: Requested number of reading register (high byte)
Byte 11: Requested number of reading register (low byte)
00 00 00 00 00 06 01 04 00 01 00 02
Command Head
Command Body

And the response should be:


Byte 0: Transaction indentifier-0
Byte 1: Transaction indentifier-0
Byte 2: Protocol indentifier-0
Byte 3: Protocol indentifier-0
Byte 4: Length field
Byte 5: Length field-number of bytes following
Byte 6: Unit indentifier-1 (always 1)
Byte 7: ModBus function code
Byte 8: Byte count (each register need two byte)
Byte 9: High bye of first address
Byte 10: Low byte of first address
Byte 11: High byte of second address
Byte 12: Low byte of second address
00 00 00 00 00 06 01 04 04 7F FF 7F FF
Command Head
Command Body

Note: (Byte 6) Unit Indentifier Always 1

9.2 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

9.3 For DIO Modules: Register Address (Unit: 16bits)

Where X = 40000 for function 03, function 06, function 16
X = 30000 for function 04

9.4 EDAM-9050: 12 Digital Input/6 Digital Output Module

9.4.1 Holding Register Address (Unit: 16bits)

Where X=00000 for function 01, function 05
X=10000 for function 02

Address	Channel	Item	type
X+0001~X+0024	Read DI Counter Count value, (For DI Counter Mode).	12 Channels, 32 Bits/channel.	R
X+0025~X+0036	For Pulse Output L level, time Unit:0.1ms	6 Channels, 32 Bits/channel.	R/W
X+0037~X+0048	For Pulse Output H level, time Unit:0.1ms	6 Channels, 32 Bits/channel.	R/W
X+0049~X+0060	Set DO pulse count value. (Set to 0= Continue mode & 1= stop)	6 Channels, 32 Bits/channel.	R/W

9.4.2 Bit Address (Unit: 1Bit)

Where X=00000 for function 01, function 05
X=10000 for function 02

Address	Channel	Item
X+0001~X+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)

9.5 EDAM-9051: 12 Digital Input/2 Counter/2 Output Module

9.5.1 Holding Register Address (Unit: 16bits)

Address	Channel	Item	Type
X+0001~X+0028	Read DI Counter Count value, (For DI Counter Mode).	14 Channels, 32 Bits per channel	R
X+0029~X+0032	For Pulse Output L level, time Unit:0.5ms	2 Channels, 32 Bits/channel	R/W
X+0033~X+0036	For Pulse Output H level, time Unit:0.5ms	2 Channels, 32 Bits/channel	R/W
X+0037~X+0040	Set DO pulse count value. (Set to 0= Continue mode & 1= stop)	2 Channels, 32 Bits/channel	R/W

9.5.2 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)

9.6 EDAM-9052: 8 channel digital Input /digital out Module

9.6.1 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	Read DI Counter Count value, (For DI Counter Mode).	8 Channels, 32 Bits/channel	R
X+0017~X+0032	For Pulse Output L level, time Unit:0.5ms	8 Channels, 32 Bits/channel	R/W
X+0033~X+0048	For Pulse Output H level, time Unit:0.5ms	8 Channels, 32 Bits/channel	R/W
X+0049~X+0064	Set DO pulse count value. (Set to 0= continue and 1= stop)	8 Channels, 16 Bits/channel (0,1 or 2~65535)	R/W
X+0065	Digital input status	8 channel,16 Bits	R
X+0066	Digital output status	8 channel,16 Bits	R/W

9.6.2 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

9.7 EDAM-9015 7-Channel RTD Input Module

9.7.1 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		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	

9.7.2 Bit Address (unit:1 bit)

Where X=00000 for function 01, function 05

X=10000 for function 02

Address	Channel	Item	Attribute
X+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

9.8 EDAM-9017 8-Channel Voltage/Current Input Module

9.8.1 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	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	

9.8.2 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	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

9.9 EDAM-9019 8-Channel T/C Input Module

9.9.1 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	

9.9.2 Bit Address (unit:1 bit)

Where X=00000 for function 01, function 05
X=10000 for function 02

Address	Channel	Item	Attribute
X+0017	0	DO value	R/W
X+0018	1	DO value	R/W
X+0101	0	Reset Max. value	R/W
X+0102	1	Reset Max. value	R/W
X+0103	2	Reset Max. value	R/W
X+0104	3	Reset Max. value	R/W
X+0105	4	Reset Max. value	R/W
X+0106	5	Reset Max. value	R/W
X+0107	6	Reset Max. value	R/W
X+0108	7	Reset Max. value	R/W
X+0109~X+0110		Reserved	
X+0111	0	Reset Min. value	R/W
X+0112	1	Reset Min. value	R/W
X+0113	2	Reset Min. value	R/W
X+0114	3	Reset Min. value	R/W
X+0115	4	Reset Min. value	R/W
X+0116	5	Reset Min. value	R/W
X+0117	6	Reset Min. value	R/W
X+0118	7	Reset Min. value	R/W
X+0119~X+0120	--	Reserved	
X+0121	0	Burnout flag	R
X+0122	1	Burnout flag	R
X+0123	2	Burnout flag	R
X+0124	3	Burnout flag	R
X+0125	4	Burnout flag	R
X+0126	5	Burnout flag	R
X+0127	6	Burnout flag	R
X+0128	7	Burnout flag	R
X+0129~X+0130	--	Reserved	
X+0131	0	High alarm flag	R
X+0132	1	High alarm flag	R
X+0133	2	High alarm flag	R
X+0134	3	High alarm flag	R
X+0135	4	High alarm flag	R
X+0136	5	High alarm flag	R
X+0137	6	High alarm flag	R
X+0138	7	High alarm flag	R
X+0139~X+0140	--	Reserved	
X+0141	0	Low alarm flag	R
X+0142	1	Low alarm flag	R
X+0143	2	Low alarm flag	R
X+0144	3	Low alarm flag	R
X+0145	4	Low alarm flag	R
X+0146	5	Low alarm flag	R
X+0147	6	Low alarm flag	R
X+0148	7	Low alarm flag	R

Chapter 10 Advanced EDAM-9000A/9400 MODBUS/TCP address Mapping

10.1 All EDAM-9000A/9400 Digital Input/Output Modules

All EDAM-9000A modules use the same MODBUS address mapping

10.1.1 Register Address (Unit: 16bits) (Firmware Ver: 6.000 or later)

10.1.2 Bit Address (Unit:1Bit)

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

X = 30000 for function 04

X+1453~X+1484	DO mode setting: = 0000 - Direct DO output, (default) = 0001 - Pulse output mode, = 0002 - low to high delay, = 0003 - high to low delay	16-Bits/channel.	R/W
X+1485~X+1516	DI mode setting: = 0000 - Direct DI input, (default) = 0001 - Counter Mode, = 0002 - low to high latch = 0003 - high to low latch = 0004 - Input frequency mode(0.3 ~1000 Hz max)	16-Bits/channel.	R/W
X+1321~X+1322	DI(0~31) input value,	32 Channels, 1-bit/channel. (X+1321) for DI(15~0) (X+1322) for DI(31~16)	R
X+1323~X+1324	DO(0~31) output value	32 Channels, 1-bit/channel. (X+1323) for DO(15~0) (X+1324) for DO(31~16)	R/W
X+1001~ X+1064	DI Counter Count value, (DI counter mode only).	32 Channels, 32-bit/channel (X+1001) for DI(0), bit 15~0 (X+1002) for DI(0), bit 31~16 ...... (X+1063) for DI(15), bit 15~0 (X+1064) for DI(15), bit 31~16	R
X+1065~X+1128	DO pulse output L level time Unit: 0.5ms,(1 ~13170) (DO pulse output mode only)	32 channels, 32-bit/channel (X+1065) for DO(0) , bit 15~0 X+1066) for DO(0) , bit 31~16 ……. (X+1127) for DO(31) , bit 15~0 (X+1128) for DO(31) , bit 31~16	R/W
X+1129~X+1192	For pulse output H level time Unit: 0.5ms,(1 ~13170) (DO pulse output mode only)	32 channels, 32-bit/channel (X+1129) for DO(0) , bit 15~0 (X+1130) for DO(0) , bit 31~16 ……. (X+1191) for DO(31) , bit 15~0 (X+1192) for DO(31) , bit 31~16	R/W
X+1193~X+1256	Start/stop DO Pulse output (0000= continue, 0001= stop, 0002= start)	32 channels, 32-bit/channel (X+1193) for DO(0) , bit 15~0 (X+1194) for DO(0) , bit 31~16 ..... (X+1255) for DO(31) , bit 15~0 (X+1256) for DO(31) , bit 31~16	R/W

X+1257~X+1320	DO pulse output count value (00000000~FFFFFFFF)	32 Channels, 32-bits/channel (X+1257) for DO(0) , bit 15~0 (X+1258) for DO(0) , bit 31~16 ……. (X+1319) for DO(31) , bit 15~0 (X+1320) for DO(31) , bit 31~16	R/W
X+1517~X+1548 (ver: 6.070 or later)	Start / Stop the_DIO Synchronization operation. =0000 - Stop DIO Sync. =0001 - Start DIO Sync.	32 channels, 16 bits/channel. (x+1517) for DO(0) , bit 15~0 ……. (x+1548) for DO(31) , bit 15~0	R/W
X+1549~X+1550 (ver: 6.070 or later)	Read DO status in DIO Synchronization operation. the bit is set when output is activated = 0 - output is inactivated = 1 - output is activated	32 channel,1-bit/channel. (x+1549).bit(0) for DO(0) (X+1549).bit(15) for DO(15) (X+1550).bit(0) for DO(16) (X+1550).bit(15) for DO(31)	R
X+1551~X+1582 (ver: 6.070 or later)	Set/read monitored DI channels (0~15) in *DIO Sync operation.* X+1551~X+1566- indicate monitored state of DI(n) =0001 when DI(n) is activated =0000 when DI(n) is inactivated X+1567~X+1582 - DI mask pattern =0001 - Enable DI(n) to be monitored =0000 -Don’t care.	16 DI channels, 16-bit/channel **(**X+1551) monitor state of DI(0) (X+1552) monitor state of DI(1) ….. (X+1566) monitor state of DI(15) (X+1567) Enable/disable DI(0) to be monitored (X+1569) Enable/disable DI(1) to be monitored … (X+1582) Enable/disable DI(15) to be monitored **(Ref. 14.7)**	R/W
X+1615~X+1646 (ver: 6.070 or later)	Set DIO SYNC mode, DO active output state and Enable/disable auto run when power-on for DO channel N. bit(1.0): DIO Sync. operation mode =01 - DI match DO toggler mode . =02 - DI match DO latch mode =03 - DI mismatch DO latch mode. bit(2) : enable/disable DIO Synchronization operation when power-on. = 0 - Disable. = 1 - Enable. bit(3) : digital output state when DI input value match DI mask pattern =0 - Inactive state =1 - Active state bit(15~4): Don’t care	32 DO channels,16 Bits/channel (X+1615) for Do(0) (X+1616) for Do(0) …… (X+1646) for Do(31)	R/W
X+1647~X+1710 (ver: 6.070 or later)	Set DIO SYNC DI debounce time for DO channel N **For DI match/mismatch DO toggle mode:** The first word contains the high order bits bit(31~16) :DI pre-debounce time when DI value match DI mask pattern. =0000~FFFF ms	32 Bits/DO channel for DO(0) X+1647=0000~FFFF X+1648=0000~FFFF or 0000 for DO(1) X+1649=0000~FFFF	R/W

	The second word contains the low order bits bit(15~0):DI post-debounce time when DI =0000~FFFF ms **For DI match/mismatch DO latch mode:** The first word contains the high order bits bit(31~16): = 0000 . The second word contains the low order bits bit(15~0) :DI pre-debounce time when DI value match or mismatch DI mask pattern. =0000~FFFF ms	X+1650=0000~FFFF or 0000 .... for DO(31) X+1709=0000~FFFF X+1710=0000~FFFF or 0000	
X+1711~X+1774 (ver: 6.070 or later)	Set DO channel High/Low delay output time (unit:**0.5ms**). The first word: bit(31~16) :DO(n) High to Low to (HHHH) output delay time The second word: bit(15~0) :DO(n) Low to High(LLLL) output delay time.	32 DO channel, 32-bits/channel X+1711~X+1714 for DO(0) X+1715~X+1718 for DO(1) X+1719~X+1722 for DO(2) ..... X+1771~X+1774 for DO(3)	R/W
X+1775~X+1806 (ver: 6.070 or later)	Set a single digital output channel for Auto-Off Time of DO = 1 - Set(active) a single digital output channel for Auto-Off	32 channels, 16-bit/channel. X+1775 for DO(0) X+1776 for DO(1) X+1777 for DO(2) ….. X+1806 for DO(31) **Example:** for DO(0) active Request 01 06 A3 2E 00 01 Resp: 01 06 A3 2E 00 01	R/W
x+5678	Informs all modules that the host is OK. (ref. ~AA**, or ~**)	(No reply to modbus response)	R
x+5601	host communication timeout value (unit: 0.1sec)	ref. (x+5604) Host watchdog timeout status	R/W

10.1.3 Bit Address (Unit:1Bit)

Where **X = 00000** for function 01, function 05, function 15
X = 10000 for function 02

Address	Channel	Item	Type
X+0001~X+0016	Read digital Input status	16 Channels, 1-bit/channel	R
X+0017~X+0032	Read/write digital output	16 Channels, 1-bit /channel	R/W
X+0033	Start/stop DI(0) counter	0xFF00=Start ,0x0000= Stop	R/W
X+0034	Clear DI(0) counter	0xFF00= Clear DI(0) counter	W
X+0035	Read/clear DI(0) counter overflow status	1= DI(0) overflow occurred or clear 0= DI(0) no overflow occurred	R/W
X+0036	Read/clear DI(0) latch status	1=DI(0) latched/clear latch 0=DI(0) no latched **Example:** # read ch0 Latch Status	R/W

		request: 01 02 27 33 00 01 response: 01 02 01 01 ;ch0 is latched # clear ch0 Latch Status request: 01 0F 00 23 00 01 FF FE response: 01 0F 00 23 00 01 # clear ch0 Latch Status request: 01 05 00 23 00 00 response: 01 05 00 23 00 00 # read ch0 Latch Status request: 01 02 27 33 00 01 response: 01 02 01 00 ;ch0 not latched	
X+0037	Start/stop DI(1) counter	0xFF00=Start ,0x0000= Stop	R/W
X+0038	Clear DI(1) counter	0xFF00= Clear DI(1) counter	W
X+0039	Read/clear DI(1) counter overflow status	1= DI(1) overflow occurred or clear 0= DI(1) no overflow occurred	R/W
X+0040	Read/clear DI(1) latch status	1=DI(1) latched/clear latch 0=DI(1) no latched	R/W
………..	X+0041~x+0152 For DI(2~29)	…………	R/W
X+0153	Start/stop DI(30) counter	0xFF00=Start ,0x0000= Stop	R/W
X+0154	Clear DI(30) counter	0xFF00= Clear DI(30) counter	W
X+0155	Read/clear DI(30) counter overflow status	1= DI(30) overflow occurred or clear 0= DI(30) no overflow occurred	R/W
X+0156	Read/clear DI(30) latch status	1=DI(30) latched/clear latch 0=DI(30) no latched	R/W
X+0157	Start/stop DI(31) counter	0xFF00=Start ,0x0000= Stop	R/W
X+0158	Clear DI(31) counter	0xFF00= Clear DI(2) counter	W
X+0159	Read/clear DI(31) counter overflow status	1= DI(31) overflow occurred or clear 0= DI(31) no overflow occurred	R/W
X+0160	Read/clear DI(31) latch status	1=DI(31) latched/clear latch 0=DI(31) no latched	R/W
X+0161~x+0192	Read DI(0~31) value	32 Channels, 1-bit/channel.	R
X+0193~x+0224	Read/write DO(0~31) value	32 Channels, 1-bit /channel.	R/W

Chapter 11 TCPDAQ Data Structure

11.1 Typedef struct _AlarmInfo

```
typedef struct _AlarmInfo                  //Alarm Event data structure
{
    u_cha         szIP[4];                 //The IP address which cause the alarm change
    u_short       szDateTime[6];           //E.x[ 2001]/[09/][23][10]:[12]:[34]

  //     (Year/Month/Day Hour:Minute:Second)
    u_short       byChannel;               //The Channel of which cause the alarm change
    u_short       byAlarmType;             //0x00:AIO Low Alarm
                                           //0x01:AIO High Alarm
                                           //0x20:DIO Alarm
                                           //0xF0:Connection Alarm
    u_short       byAlarmStatus;           //0:Alarm ON to OFF, 1:Alarm OFF to ON
    u_short       wValue;                  //Alarm value.For DIO, this value could be "0" or "1" means that "ON" or
                                               //"OFF"
                                           //For high or low alarm, this is the AIO value.
                                           //For connection lost, this value is '0'.
} _AlarmInfo;
```

11.2 Typedef struct _StreamData

```
typedef struct _StreamData                 //Stream Event data structure
{
    u_char        szIP[4];                 //The IP address which send the stream datae
    u_short       szDateTime[6];           //E.x [2001]/[09]/[23] [10]:[12]:[34]
                                           //     (Year/Month/Day Hour:Minute:Second)
    u_short       DIN;                     //Digital input data (DI#0~DI#15)
    u_short       DOUT;                    //Digital output data (DO#0~DO#15)
    u_short       wData[32];               //Digital input Counter (Each channel occupies 4 Byte)
} _StreamData;
```

11.3 Typedef struct ModuleInfo

```
typedef struct ModuleInfo                  // Used For Scan_Online_Modules(..)
{     u_char      szIP[4];                 //IP address
    u_char        szGate[4];               //Gateway
    u_char        szMask[4];               //Submask
    u_char        szDHCP;                  //DHCP status 01=enable, 00=disable
    u_char        szID;                    //Module ID number
    u_char        szMacAddr[6];            //MAC address of module
    u_short       szModuleNo;              //Module name
    u_char        szBuffer[12];            //Buffer reserved for TCPDAQ.DLL
} ModuleInfo;
```

11.4 Typedef struct ModuleData

```
typedef struct ModuleData                  //Used for function TCP_ReadAllDataFromModule (..)
{  u_char     Din[16];                     //Digital input data (DI#0~DI#15),avaliable for EDAM9050/51/52
   u_char         Dout[16];                //Digital output data (DO#0~DO#15),avaliable for
                                                           //EDAM9050/51/52/17/19
   u_char      DiLatch[16];          //Digital input latch status (DI#0~DI#15),avaliable for EDAM9050/51/52
   long        DiCounter[16];        //Digital input counter value (DI#0~DI#15),avaliable for EDAM9050/51/52
   double      AiNormalValue[16];    //Analog Input value(AI#0~AI#15),avaliable for EDAM9015/17/19
   double      AiMaxValue[16];       //Analog maximum value(AI#0~AI#15),avaliable for EDAM9015/17/19
   double      AiMinValue[16];       //Analog minimum value(AI#0~AI#15),avaliable for EDAM9015/17/19
   u_char      AiHighAlarm[16];      //Analog high alarm status(AI#0~AI#15),avaliable for EDAM9015/17/19
   u_char      AiLowAlarm[16];       //Analog low alarm status(AI#0~AI#15),avaliable for EDAM9015/17/19
   u_char      AiChannelType[16];    //Analog channel Type, avaliable for EDAM9015/17/19
   u_char      AiBurnOut[16] ;       //Analog channel burn out status,avaliable for EDAM9019/15 only
   double      CJCTemperature ;      //Cold junction temperature,avaliable for EDAM9019 only
} ModuleData;
```

Chapter 12 EDAM-9000 Web Server

12.1 What is TCPDAQ Web Server?

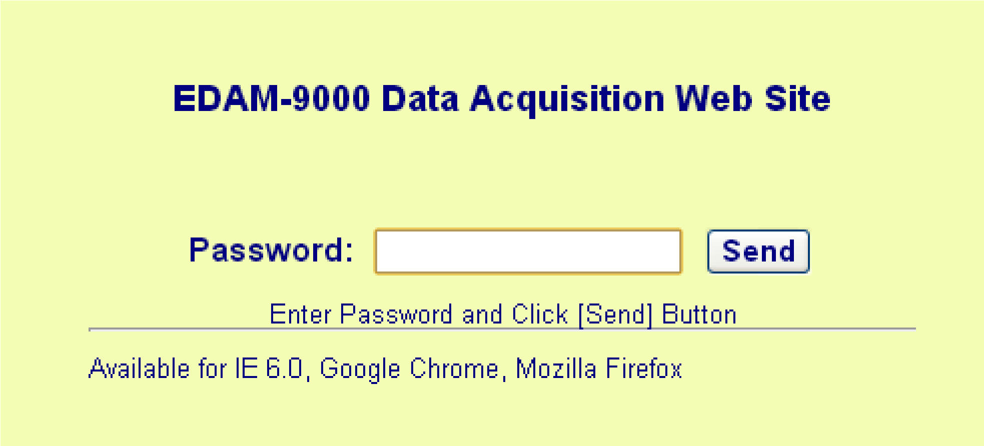
EDAM-9000 I/O modules all features built-in web server. Remote computer or devices can monitor and control I/O status on EDAM-9000 modules remotely through web browser. There is default built-in web page on EDAM-9000 modules.

To use your computer to browse the web page on EDAM-9000 module, you can simply type the IP address to connect to your EDAM-9000 module in web browser. There will be one dialog window asking you to enter the password. After you have typed the correct password, you can start to monitor or control I/O on EDAM-9000 modules.

Notice: Please use Windows Internet Explorer 5.5 (IE 5.5 or later version)

12.2 Home Page

- Type the **IP address** in the web browser (example: http:\\192.168.0.51)
- The home page will pop-up in the browser window to ask you to enter the password



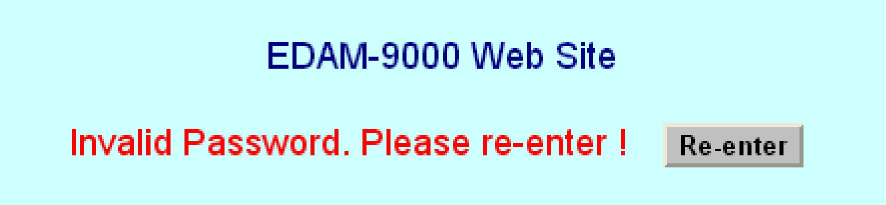
EDAM-9000 Data Acquisition Web Site

Password:

Enter Password and Click [Send] Button

Available for IE 6.0, Google Chrome, Mozilla Firefox

- Enter the correct password and click send button to verify the password. If the password is not correct, a warming message box will show up to remain you to reenter the password



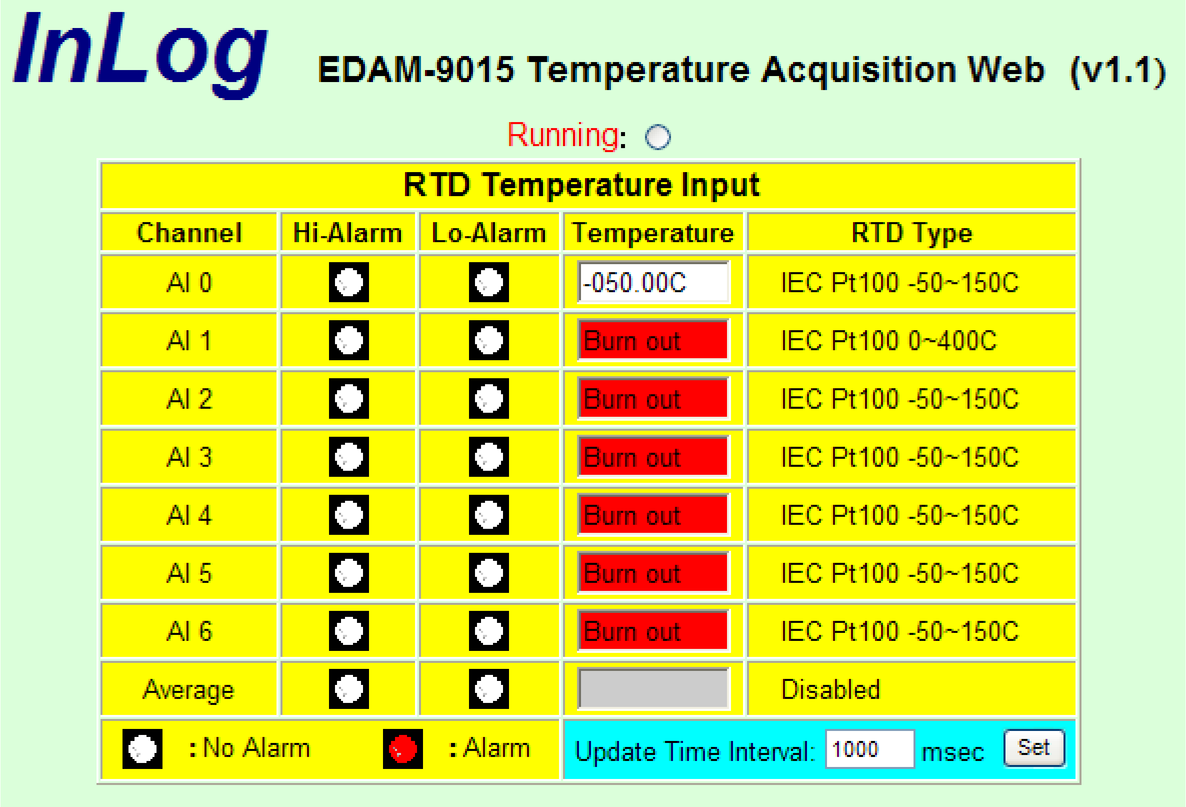
EDAM-9000 Web Site

Invalid Password. Please re-enter !

- If the password is correct, the module monitoring page will pop up in the web browser.

12.3 Module monitoring page

12.3.1 EDAM-9015 monitoring page



InLog EDAM-9015 Temperature Acquisition Web (v1.1)

Running: ○

RTD Temperature Input

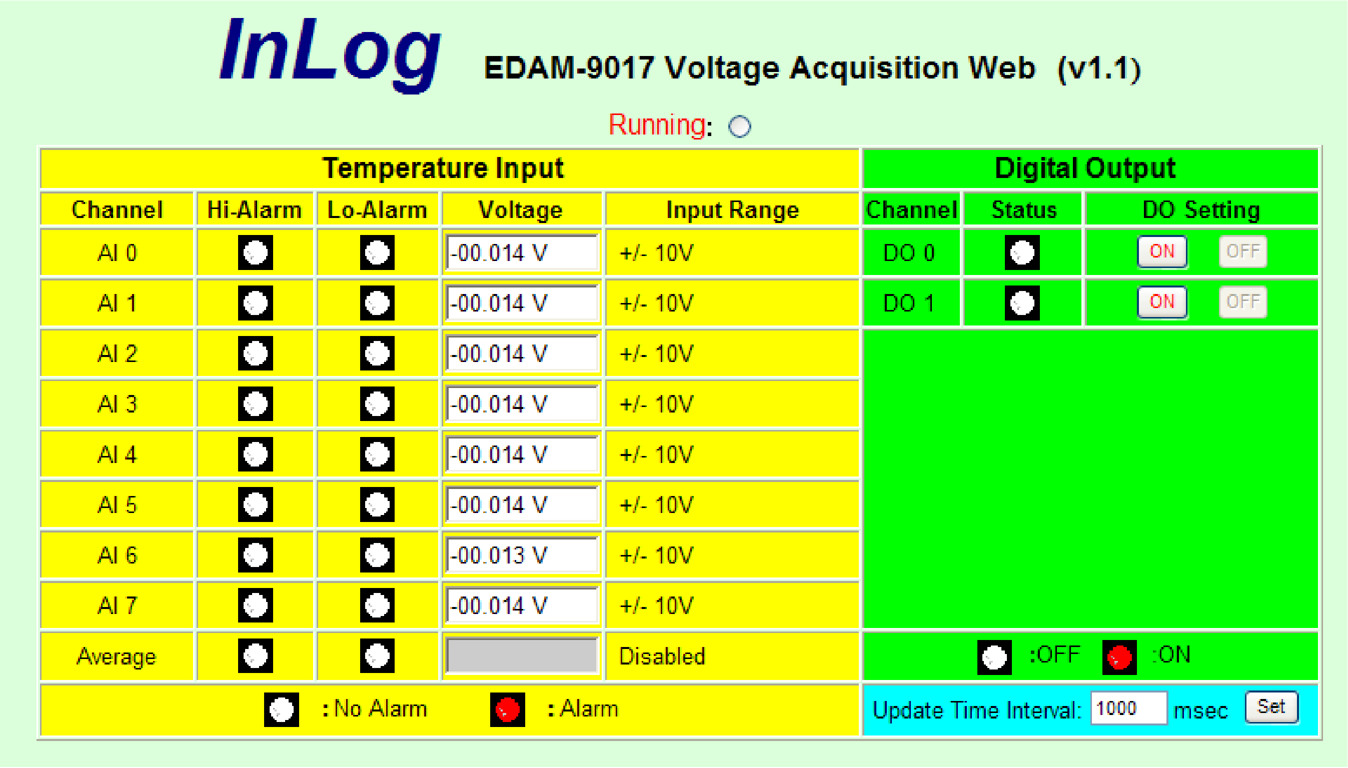
Channel	Hi-Alarm	Lo-Alarm	Temperature	RTD Type
AI 0	○	○	-050.00C	IEC Pt100 -50~150C
AI 1	○	○	Burn out	IEC Pt100 0~400C
AI 2	○	○	Burn out	IEC Pt100 -50~150C
AI 3	○	○	Burn out	IEC Pt100 -50~150C
AI 4	○	○	Burn out	IEC Pt100 -50~150C
AI 5	○	○	Burn out	IEC Pt100 -50~150C
AI 6	○	○	Burn out	IEC Pt100 -50~150C
Average	○	○		Disabled

○ : No Alarm ● : Alarm

Update Time Interval: 1000 msec

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

12.3.2 EDAM-9017 monitoring page



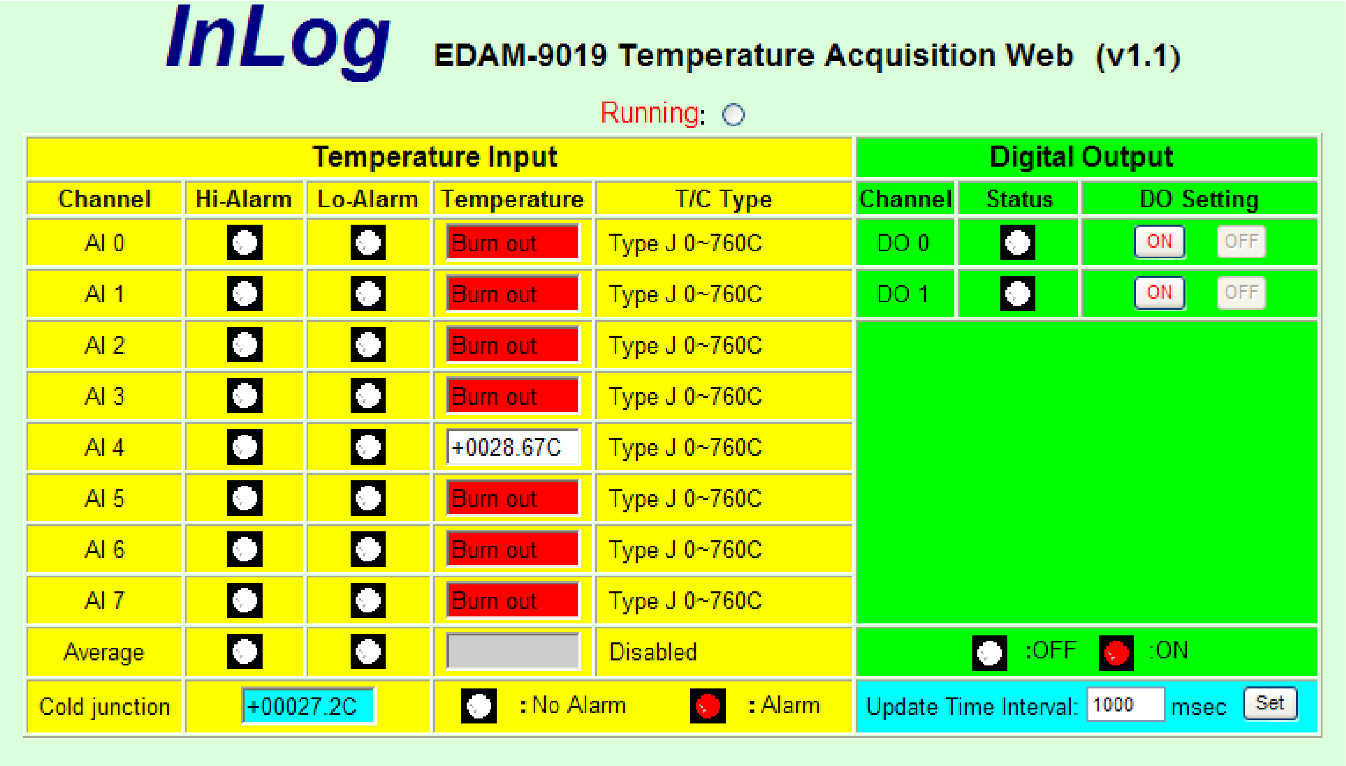
InLog EDAM-9017 Voltage Acquisition Web (v1.1)

Running: ○

Temperature Input					Digital Output		
Channel	Hi-Alarm	Lo-Alarm	Voltage	Input Range	Channel	Status	DO Setting
AI 0	○	○	-00.014 V	+/- 10V	DO 0	○	ON OFF
AI 1	○	○	-00.014 V	+/- 10V	DO 1	○	ON OFF
AI 2	○	○	-00.014 V	+/- 10V			
AI 3	○	○	-00.014 V	+/- 10V			
AI 4	○	○	-00.014 V	+/- 10V			
AI 5	○	○	-00.014 V	+/- 10V			
AI 6	○	○	-00.013 V	+/- 10V			
AI 7	○	○	-00.014 V	+/- 10V			
Average	○	○		Disabled	○ :OFF ● :ON		
○ : No Alarm ● : Alarm					Update Time Interval: 1000 msec Set		

Channel : Channel number of analog input or digital output
Hi-Alarm : Analog channel High alarm status
Lo-Alarm : Analog channel low alarm status
Voltage : Voltage value of analog input channel
Input Range : Range of analog input channel
Status : Digital output status
DO Setting : Set digital output on or off
Time interval : I/O status update time interval

12.3.3 EDAM-9019 monitoring page



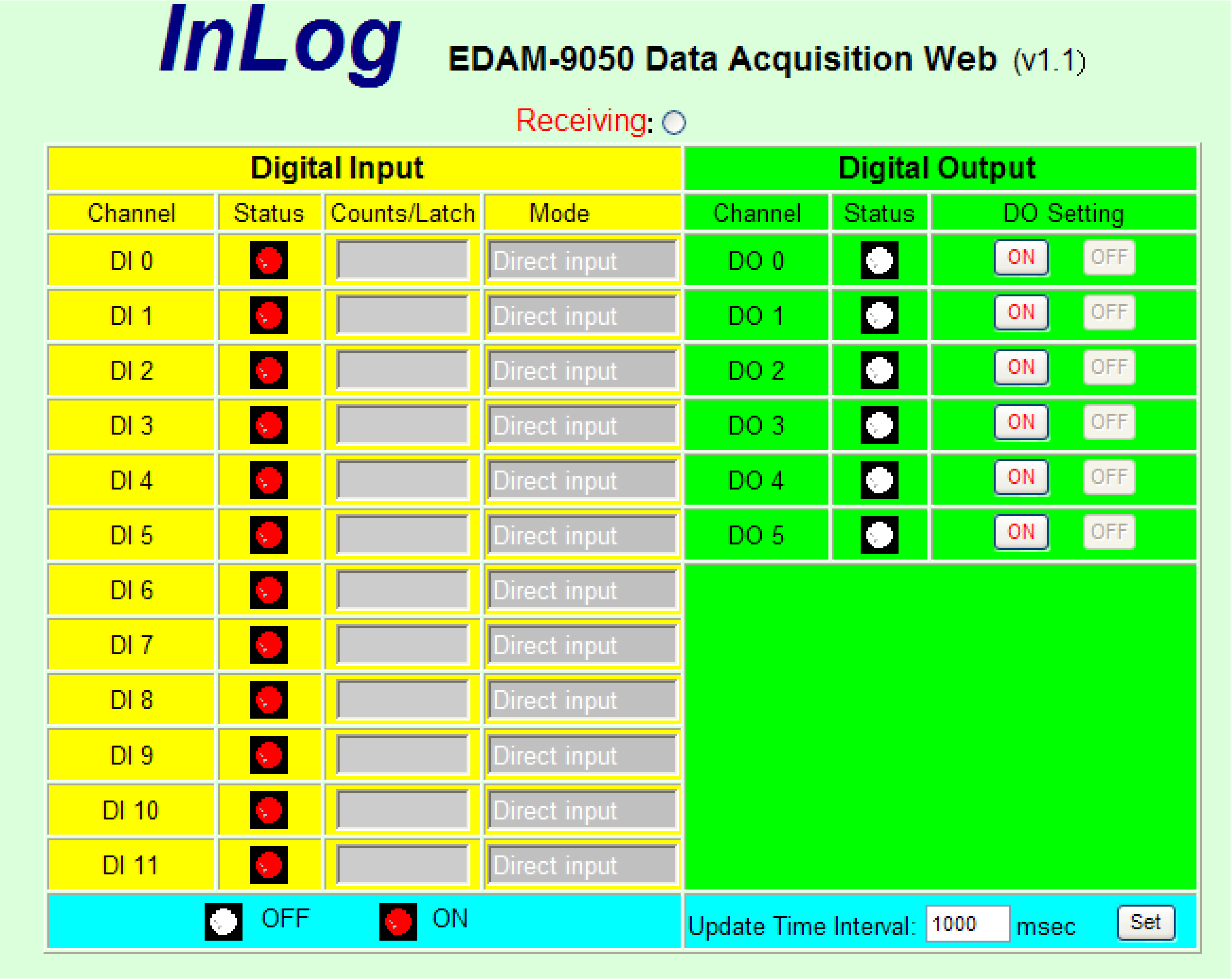
InLog EDAM-9019 Temperature Acquisition Web (v1.1)

Running: ○

Temperature Input					Digital Output		
Channel	Hi-Alarm	Lo-Alarm	Temperature	T/C Type	Channel	Status	DO Setting
AI 0	○	○	Burn out	Type J 0~760C	DO 0	○	ON OFF
AI 1	○	○	Burn out	Type J 0~760C	DO 1	○	ON OFF
AI 2	○	○	Burn out	Type J 0~760C			
AI 3	○	○	Burn out	Type J 0~760C			
AI 4	○	○	+0028.67C	Type J 0~760C			
AI 5	○	○	Burn out	Type J 0~760C			
AI 6	○	○	Burn out	Type J 0~760C			
AI 7	○	○	Burn out	Type J 0~760C			
Average	○	○		Disabled	○ :OFF ● :ON		
Cold junction	+00027.2C	○ : No Alarm ● : Alarm			Update Time Interval: 1000 msec Set		

Channel	: Channel number of analog input or digital output
Hi-Alarm	: Analog channel High alarm status
Lo-Alarm	: Analog channel low alarm status
Temperature	: Temperature value of T/C input channel
T/C type	: Thermal Couple type of input channel
Cold junction	: Temperature of T/C cold junction
Average	: Average value of channels which functions in average
Status	: Digital output status
DO Setting	: Set digital output on or off
Time interval	: I/O status update time interval

12.3.4 EDAM-9050 monitoring page



InLog EDAM-9050 Data Acquisition Web (v1.1)

Receiving: ○

Digital Input				Digital Output		
Channel	Status	Counts/Latch	Mode	Channel	Status	DO Setting
DI 0	🔴		Direct input	DO 0	○	ON OFF
DI 1	🔴		Direct input	DO 1	○	ON OFF
DI 2	🔴		Direct input	DO 2	○	ON OFF
DI 3	🔴		Direct input	DO 3	○	ON OFF
DI 4	🔴		Direct input	DO 4	○	ON OFF
DI 5	🔴		Direct input	DO 5	○	ON OFF
DI 6	🔴		Direct input			
DI 7	🔴		Direct input			
DI 8	🔴		Direct input			
DI 9	🔴		Direct input			
DI 10	🔴		Direct input			
DI 11	🔴		Direct input			
○ OFF 🔴 ON				Update Time Interval: 1000 msec Set		

Channel : Channel number of digital input or output

Status : Current input or output status

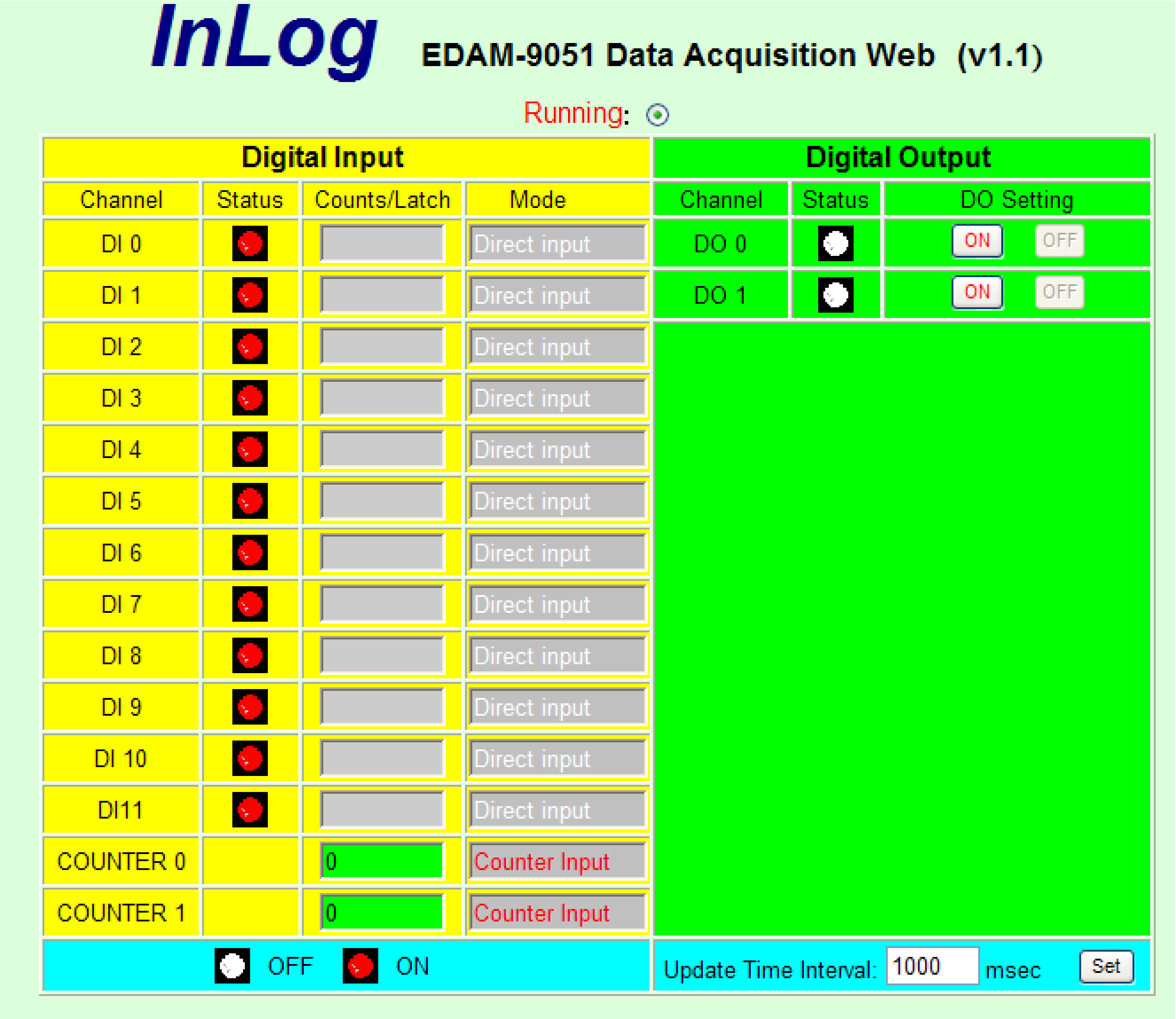
Count/Latch : Counter value or latch status of digital input which functions at “Counter” or “Latch” mode

Mode : Channel operating mode

DO Setting : Set digital output on or off

Time interval : I/O status update time interval

12.3.5 EDAM-9051 monitoring page



InLog EDAM-9051 Data Acquisition Web (v1.1)

Running: ◉

Digital Input				Digital Output		
Channel	Status	Counts/Latch	Mode	Channel	Status	DO Setting
DI 0	●		Direct input	DO 0	○	ON OFF
DI 1	●		Direct input	DO 1	○	ON OFF
DI 2	●		Direct input			
DI 3	●		Direct input			
DI 4	●		Direct input			
DI 5	●		Direct input			
DI 6	●		Direct input			
DI 7	●		Direct input			
DI 8	●		Direct input			
DI 9	●		Direct input			
DI 10	●		Direct input			
DI11	●		Direct input			
COUNTER 0		0	Counter Input			
COUNTER 1		0	Counter Input			
○ OFF ● ON				Update Time Interval: 1000 msec Set		

Channel : Channel number of digital input or output

Status : Current input or output status

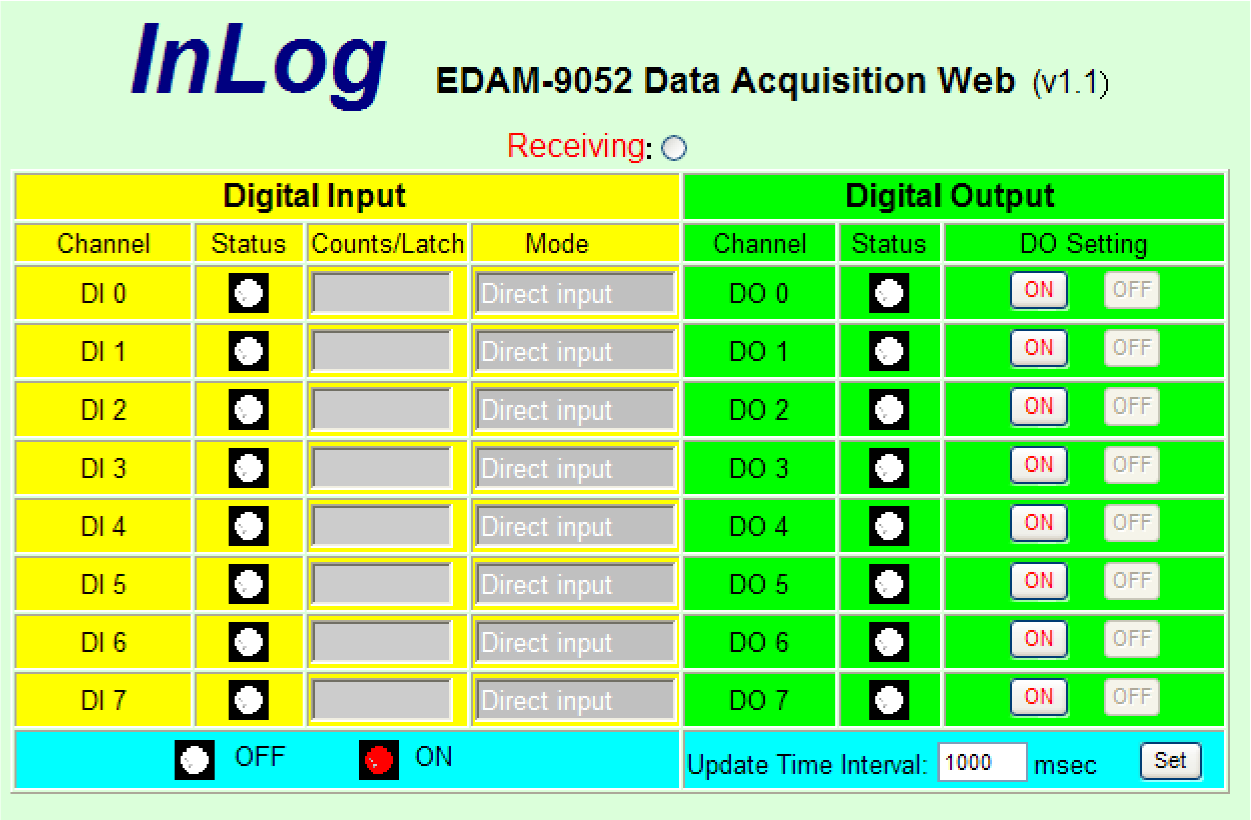
Count/Latch : Counter value or latch status of digital input which functions at "Counter" mode or "Latch" mode

Mode : Channel operating mode

DO Setting : Set digital output on or off

Time interval : I/O status update time interval

12.3.6 EDAM-9052 monitoring page



InLog EDAM-9052 Data Acquisition Web (v1.1)

Receiving: ○

Digital Input				Digital Output		
Channel	Status	Counts/Latch	Mode	Channel	Status	DO Setting
DI 0	○		Direct input	DO 0	○	ON OFF
DI 1	○		Direct input	DO 1	○	ON OFF
DI 2	○		Direct input	DO 2	○	ON OFF
DI 3	○		Direct input	DO 3	○	ON OFF
DI 4	○		Direct input	DO 4	○	ON OFF
DI 5	○		Direct input	DO 5	○	ON OFF
DI 6	○		Direct input	DO 6	○	ON OFF
DI 7	○		Direct input	DO 7	○	ON OFF
○ OFF 🔴 ON				Update Time Interval: 1000 msec Set		

Channel : Channel number of digital input or output

Status : Current input or output status

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

Mode : Channel operating mode

DO Setting : Set digital output on or off

Time interval : I/O status update time interval

Chapter 13 Appendix

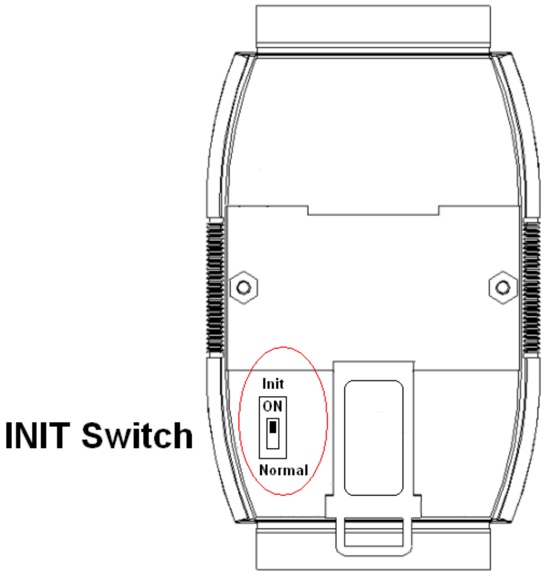
13.1 INIT* switch operation (EDAM-9000A/9400 only)

The EDAM-9000A “INIT*mode” has two purposes, one for reading module current configuration, and another for configuring the module **IP Address, Subnet Mask, and Gateway**.

- ♦ Reading module current configuration

  Each EDAM module has a built-in EEPROM which is used to store the configuration information such as address ID, type, DIO mode etc.. If the user unfurtunally forget the configuration of the module. User may use a special mode called “INIT* mode” to resolve the problem When the module is set to “INIT* mode”, the default settings are IP Address, Subnet Mask, and Default Gateway (10.0.0.1, 255.0.0.0 and 10.0.0.1)

- ♦ Originally, the INIT mode is accessed by connecting the INIT* terminal to the GND terminal. New EDAM-9000A modules have the INIT switch located on the rear side of the module to allow easier access to the INIT mode. For these modules, INIT mode is accessed by sliding the INIT switch to the Init position as shown below.



- ♦ The following steps show you how to enable INIT* mode and read the current configuration:

  1. Power off the module.
  2. Sliding the INIT switch to the “Init” position.
  3. Power on the module.
  4. Start up the Windows Utility, it will search all EDAM-9000(A) I/O modules on the host PC’ to read the current configuration stored in the EEPROM and set new **IP Address, Subnet Mask, and Default Gateway**,
  5. Power off the module again
  6. Sliding the INIT switch to the “Normal” position.

- ♦ Factory default settings:

1.	IP Address	: 10.0.0.1
2.	Subnet Mask	: 255.0.0.0
3.	Gateway	: 10.0.0.1
4.	DHCP	: Disabled
5.	Web Server	: Disabled
6.	Module ID	: 00
7.	Password	: 00000000

13.2 Module Status (version 6.000 or later)

Power-On Reset will let all output go to Power-On Value. The module may accept the host's command to change the output value. Host Watchdog Timeout will let all digital output go to Safe Value if the host watchdog timeout flag is set, and the output command will be ignored. The module's LED will go to flash and user must reset the module status via command to restore normal operation.

13.3 Dual Watchdog Operation (version 6.000 or later)

Dual Watchdog = Module Watchdog + Host Watchdog

The Module Watchdog is a hardware reset circuit to monitor the module's operating status. While working in harsh or noisy environment, the module may be down by the external signal. The circuit may let the module to work continues and never halt. The Host Watchdog is a software function to monitor the host's operating status. Its purpose is to prevent the network/communication from problem or host halt. While the timeout occurred, the module will turn the all output into safe state to prevent from unexpected problem of controlled target. The E-8000 module with Dual Watchdog may let the control system more reliable and stable.

13.4 Reset Status (version 6.000 or later)

The reset status of a module is set when the module is powered-on or when the module is reset by the module watchdog. It is cleared after the responding of the first $AA5 command. This can be used to check whether the module had been reset. When the $AA5 command responds that the reset status is cleared, that means the module has not been reset since the last $AA5 command was sent. When the $AA5 command responds that the reset status is set and it is not the first time $AA5 command is sent, it means the module has been reset and the digital output value had been changed to the power-on value.

13.5 Input counter and Input latch

Input counter:

Each input channel has internal counter used to software count the state change (*falling edge*) of input signal (max. 500Hz(E-9000A/9400) / 1KHz(E-9000)). The counting value can be read and cleared by sending "*Read digital input counter command*" or " *Clear digital input counter command*".

Input latch:

Each input channel has internal latch which is used to latch the pulse signal from the input. This latched state can be read by sending "*Read latched digital input* " command and cleared by sending "*Clear latched digital input*" command. For example, if the digital input is connected to a key switch. The key switch is a pulse signal. The user may lose the strike information by sending command $AA6. The digital input latch can latch the pulse and ready be read by sending "*Read latched digital input* " command. If the latched state=1 means that there is a key strike occurred.

13.6 Power-on & Safe value (version 6.000 or later)

Power-on value:

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

Safe value:

Safe value are used to set the module outputs into the specific values when Host watchdog timeout

If The host watchdog timer is enabled by sending *Set host watchdog timeout value*, the host should send *Host OK* command periodically within Timeout value to refresh the timer, otherwise the module will be forced to safety state.

13.7 **DIO Synchronization** (Mirror Local DI to DO)

EDAM-9000A/9400 series modules also provide a *DIO Synchronization* function. A single digital output channel can be activated (1 or 0) dependent on the digital input channels value. When the specific DI channels value changed from "match" to "mismatch" (or "mismatch" to "match")DI mask pattern, the corresponding DO will be set to active state(1 or 0) dependent on the DO setting. (only for version 6.070 or later)

13.7.1 The *DIO Synchronization* is divided into three modes:

1. DI match/mismatch DO Toggle Mode.
2. DI macth DO latch Mode.
3. DI mismatch DO latch Mode.

✓ Using ASCII command:

1. Configure DI match/mismatch DO Toggle Mode

   Step 1:

   Set a single DO channel to DIO SYNC. mode
   $AACONNDD

   Step 2:

   Set a single DO channel to "DI match/mismatch DO toggle mode".
   $AAYM1CPSHHHHLLLL(data)

   Step3:

   Set the digital output channel to ON or OFF..
   @AA6ONSS

   Step4:

   Start(run) DIO Synchronization operation.
   $AAYMRCS

2. Configure DI match/mismatch DO latch Mode

   Step 1:

   Set a single DO channel to DIO SYNC. mode
   $AACONNDD

   Step 2:

   Set DI mask pattern
   $AAYM2CPSTTTT(data) ; for DI *match DO latch* mode
   or
   $AAYM3CPSTTTT(data) **;** for DI *mismatch* DO latch mode

   Step 3:

   Set digital output channel to ON or OFF.
   @AA6ONSS

   Step 4:

   Start(run) DIO Sync.
   $AAYMRCS

Note: Before enable DIO Synchronization function, you must set DO to "***DIO SYNC. Mode***" first (Ref. $AACONNDD).

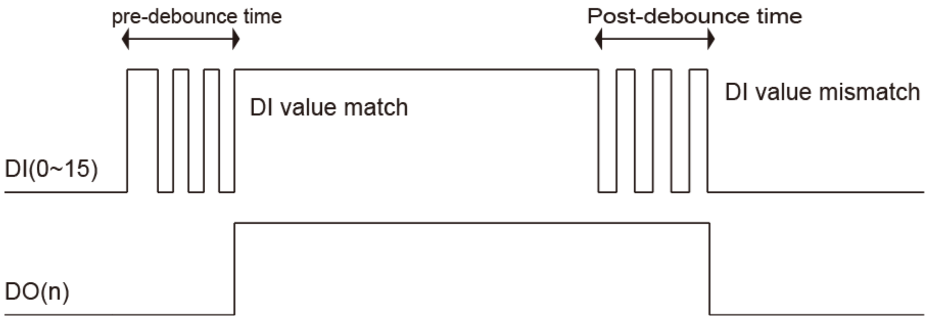
Ref. Command: $AACONNDD , **@AA6ONSS**, $AAYMC, $AAYMRCS, $AAYMS, $AAYM1CPSHHHHLLLL(data), $AAYM2CPSTTTT(data), $AAYM3CPSTTTT(data))

✓ Using Modbus-TCP commands:

Step 1:
(X+1453~X+1484) :Set a single DO channel to DIO SYNC. mode.
Step 2:
(X+1615~X+1646) :Set *DO active state* (0/1) when DI match/mismatch and power-on *auto run mode*
Step 3:
(X+1551~X+1582) :Set DI channels to be monitored and DI mask pattern.
Step 4:
(X+1647~X+1710) :Set DI pre-debounce and post-debounce time.
Step 5:
(X+1517~X+1548) :Run(Start)/Stop DIO Synchronization operation.

13.7.2 DIO Synchronization –DI match/mismatch DO Toggle Mode

A single digital output channel is activated or inactivated (1 or 0) dependent on the specific DI channels value, When the specific DI input value *match/mismatch* DI mask pattern, the corresponding DO will be set to *active/inactive*(1 or 0) state.



♦ **Example** (*using ASCII command*) : (DI match/mismatch DO Toggle Mode)

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

1. Set DO 0 to " DIO SYNC. Mode " (Ref. $AACONNDD)
   command: $01CO0000(cr)
   response: !01(cr) ; valid
2. Set DI mask pattern (DI(0,5)=1 and DI(2)=**0**) and DO(0) to "DI match/mismatch *DO toggle mode"*.

   DI pre-debounce time=300(0x 012C) msec and post-debounce time=150(0x0096)msec
   P = 1 - enable Auto Run(Start) DIO Synchronization operation when power-on.
   S = 1 - digital output active state(=1),when DI input value match DI mask pattern
   (Ref. $AAYM1CPSHHHHLLLL(data))
   command: $01YM1011012C0096XXXXXXXXXX1XX0X1(cr)
   response: !01(cr) ; valid
3. Set the digital output channel to OFF. *(ref. @AA6ONSS)*
   command: @016O000(cr)
   response: !01(cr) ; valid
4. Start/Run(S=1) DIO Sync. Operation *(Ref. $AAYMRCS)*
   command: $01YMR01(cr)
   response: !01(cr) ; valid

Ref. $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM1CPSTTTT (data), $AAYM2CPSTTTT(data)

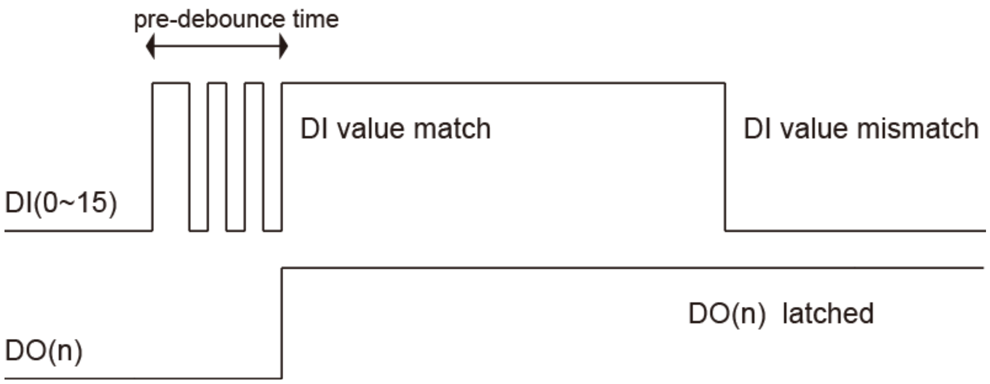
♦ **Example** (*using Modbus command*) : (DI match/mismatch DO Toggle Mode)

Assume DI channel 0,2,5 are monitored and the ASCII form of DI mask pattern is XXXXXXXXXX1XX0X1. When DI input ch(0,5)=1 and ch(2)=0, the corresponding DO(0) will be set to ON(1), otherwise (DI mismatch) DO(0) will be toggled to OFF(0). DI pre-debounce time is 400 msec and post-debounce time is 800 msec

1. (41453) Set DO(0) mode to " DIO SYNC. Mode "
   Request: 01 06 A1 EC 00 00 ; Modbus address=0xA1EC=41453-1=41452
   Response: 01 06 A1 EC 00 00 ; valid
2. (41615) Set DO(0) to "DI match/mismatch DO Toggle Mode" , DO active =1,and enable=1 auto run when power-on
   Request: 01 06 A2 8E 00 0D ; Modbus address=0xA28E=41615-1=41464
   Response: 01 06 A2 8E 00 0D ; valid
3. (41551) Set DI mask pattern (DI(2,3)=1 and DI(0)=0) for DIO Sync. operation.
   Request: 01 06 A2 4E 00 0C ; DI bits(15..0) = (0000 0000 0000 1100). Modbus
   ; address=0xA24E=41551-1=41550
   Response: 01 06 A2 4E 00 0C ; valid
4. (41552) Set DI channels (3,2,0) to be monitored for DIO Sync. operation.
   Request: 01 06 A2 4F 00 0D ; bits(15..0) = (0000 0000 0000 1101), Modbus
   ; address=0xA24F=41552-1=41551-
   Response: 01 06 A2 4F 00 0D ; valid
5. (41647) Set debounce time (pre-debounce time=0x0190(400)ms, post-debounce time=0x0320(800)ms
   Request: 01 10 A2 AE 00 02 04 01 90 03 20 ; Modbus address=0xA2AE=41647-1=41646
   Response: 01 10 A2 AE 00 02 ; valid
6. (41517) Run(start) DO(0) DIO Sync.
   Request: 01 06 A2 2C 00 01 ; Modbus address=0xA22C=41517-1=41516
   Response: 01 06 A2 2C 00 01 ; valid

13.7.3 DIO Synchronization –DI match DO latch mode

When DI input value "match" DI mask pattern, the specific single digital output channel will be activated (1 or 0) and latched.



♦ **Example** *(using ASCII command)*: (DI match DO latch mode)

When the specific DI channels (DI(0)=1, DI(2)=0) , the corresponding DO(0) will be set to ON(1). Assume DI pre-debounce time=0x0096(150)ms

1. Set DO(0) to " DIO SYNC. Mode " (Ref. $AACONNDD)
   command: $01CO0000(cr)
   response: !01(cr) ; valid
2. Assume DI(0)=1 and DI(2)=0 are monitored and DI(1,3,4,5,6..)=X (don't care).
   P = 1 - enable Auto Run(Start) DIO Synchronization operation when power-on.
   S = 1 - set digital output to active state (=1) when DI input data match DI mask pattern
   (Ref. $AAYM2CPSTTTT(data))
   command: $01YM2011012CXXXXXXXXXXXXX0X1(cr)
   response: !01(cr) ; valid
3. Set the DO(0) to inactive state (OFF). *(Ref. @AA6ONSS)*
   command: @016O000(cr)
   response: !01(cr) ; valid
4. Start/Run (S=1) DIO Sync. Operation on (Ref. $AAYMRCS)
   command: $01YMR01(cr)
   response: !01(cr) ; valid

Ref. $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM1CPSHHHHLLLL(data), $AAYM3CPSTTTT(data)

♦ **Example (using Modbus command) :** (DI match DO latch mode)

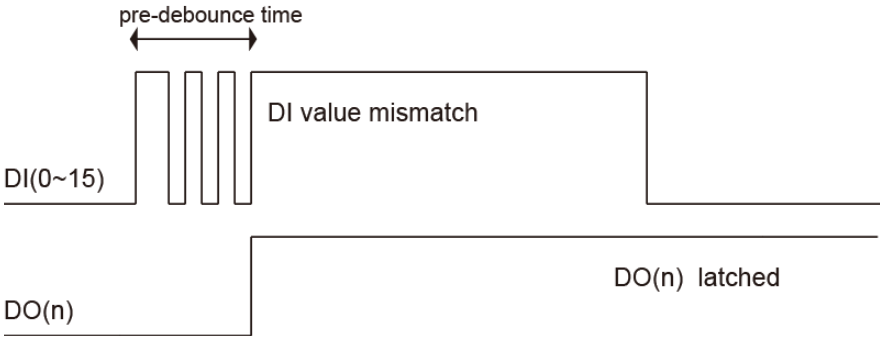
When the specific DI(2,4)=1 ,DI(0)=0, the corresponding DO(0) will be set to active state(ON). Assume DI pre-debounce time=400ms

1. (41453) Set DO(0) mode to " DIO SYNC. Mode "
   Request: 01 06 A1 EC 00 00 ;Modbus address=0xA1 EC =41453-1=41452
   Response: 01 06 A1 EC 00 00 ; valid
2. (41615) Set DO(0) to "DI Match DO latch mode" , DO latch output =1 and disable auto run when power-on
   Request: 01 06 A2 8E 00 0A ; Modbus address=0xA28E =41615-1=4164
   Response: 01 06 A2 8E 00 0A ; valid
3. (41551) Set DI channels DI(2,4,0) to be monitored and DI mask pattern
   high order word : = 0x0014 = (0000 0000 0001 0100) ; DI mask pattern
   low order word : = 0x0015 = (0000 0000 0001 0101) ; DI channels to be monitored

   ;(1=monitored DI chn,0=not monitored DI chn.)
   Request: 01 10 A2 4E 00 02 04 00 14 00 15 ;Modbus address=0xA24E=41551-1=41550
   Response: 01 10 A2 4E 00 02 ; valid
4. (41647) Set DI pre-debounce time to=400(0x0190)ms
   Request: 01 10 A2 AE 00 02 04 01 90 00 00 ; Modbus address=0xA2AE=41647-1=41646
   Response: 01 10 A2 AE 00 02 ; valid
5. (41517) Run(start) DO(0) DIO Sync.
   Request: 01 06 A2 2C 00 01 ; Modbus address=0xA22C=41517-1=41516
   Response: 01 06 A2 2C 00 01 ; valid

13.7.4 DIO Synchronization –DI mismatch DO latch mode

A single digital output channel is activated(1 or 0) dependent on the specific DI value, When the specific DI channels status mismatch DI mask pattern, the corresponding DO will be set to active state(1 or 0)



Example (using ASCII command) : (DI mismatch DO latch mode)

Assume DI mask pattern is 10000001 When DI(2)=1 ,DI(0)=1,, because DI(2)=0 mismatch the value of bit(2) of DI mask pattern, the corresponding DO(0) will be set to ON(1). DI pre-debounce time=300(0x012C) ms

1. Set DO(0) to "*DIO SYNC. Mode* " *(Ref. $AACONNDD)*
   command: $01CO0000(cr)
   response: !01(cr) ; valid
2. Set DI mask pattern and disable auto-run when power-on. DO active state=1
   P = 0 - disable Auto Run(Start) DIO Synchronization operation when power-on.
   S = 1 - digital output active state(=1) when DI value mismatch DI mask pattern
   (Ref. $AAYM3CPSTTTT(data))
   command: $01YM3001012CXXXXXXXXXXXXX0X1(cr)
   response: !01(cr) ; valid
3. Set the DO channel 0 to inactive(OFF). *(Ref. @AA6ONSS)*
   command: @016O000(cr)
   response: !01(cr) ; valid
4. *Start/Run* (S=1) DIO Sync. Operation *(Ref. $AAYMRCS)*
   command: $01YMR01(cr)
   response: !01(cr) ; valid

Ref. $AACONNDD, @AA6ONSS, $AAY6MRCS, $AAY6MC, $AAY6MS, $AAYM1CPSHHHHLLLL(data), $AAYM2CPSTTTT(data)

Example(using Modbus command) **:** (DI mismatch DO latch mode)

When the specific DI(2,4)=1 ,DI(0)=0, the corresponding DO(0) will be set to active state ON(1). Assume DI pre-debounce time=400ms

1. (41453) Set DO(0) to " DIO SYNC. Mode "
   Request: 01 06 A1 EC 00 00 ; Modbus address=0xA1EC =41453-1=41452
   Response: 01 06 A1 EC 00 00 ; valid
2. (41615) Set DIO SYNC. To "*DI misatch DO latch mode*" , DO(0) active output Low(=0) and disable(=0) auto run when power-on.
   Request: 01 06 A2 8E 00 08 ; Modbus address=0xA28E =41615-1=41614
   Response: 01 06 A2 8E 00 08 ; valid
3. (41551) Set DI monitored channels DI(4,2,1) (0000 0000 0001 0101=0x0015) ,DI mask pattern= 0000 0000 0001 0100=0x0014, and DO(0) is mirrored
   Request: 01 10 A2 4E 00 02 04 00 14 00 15 ; Modbus address=0xA24E =41551-1=41550
   Response: 01 10 A2 4E 00 02 ; valid

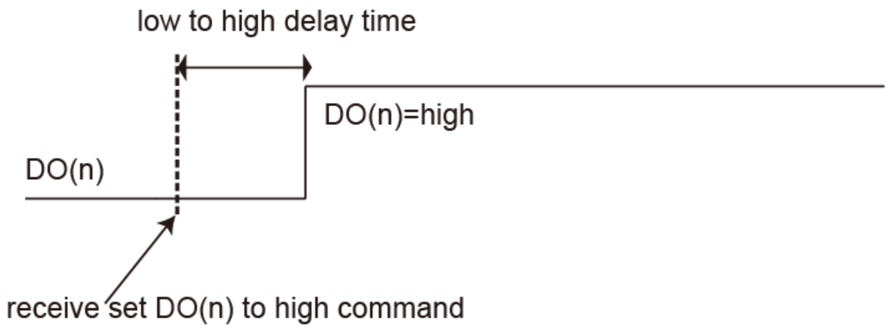
4. (41647) Set DI pre-debounce time=400(0x0190)ms.
   Request: 01 10 A2 AE 00 02 04 01 90 00 00 ; Modbus address=0xA2AE =41647-1=41646
   Response: 01 10 A2 AE 00 02 ; valid
5. (41517) Run(start) DO(0) DIO Sync. opreation
   Request: 01 06 A2 2C 00 01 ; Modbus address=0xA22C =41517-1=41516
   Response: 01 06 A2 2C 00 01 ; valid

13.8 High-to-Low and Low-to-High delay output

EDAM-9000A/9400 series modules supports **high-to-low and low-to-high delay** output function (available for version 6.070 or later)

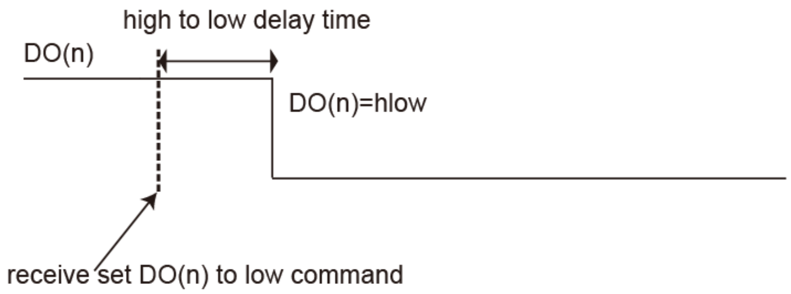
13.8.1 Low to High Delay output

When you choose *Low to High* delay mode, it is almost the same as choosing the DO direct output mode. The only difference is that there will be certain time delay when the output value changes from logic low to logic high. You can define the delay time by entering its value into the delay time text box in the setting area. After you complete the setting, click the" Apply" button. Then you can control the digital output value by the DO button and see its current value by the DO status LED display at the top of the module Display area.



13.8.2 High to Low Delay output

When you choose *High to Low* delay mode, it is almost the same as choosing the DO direct output mode. The only difference is that there will be certain time delay when the output value changes from logic high to logic low. You can define the delay time by entering its value into the Delay time text box in the Setting area. After you complete the setting, click the Apply button. Then you can control the digital output value by the DO button and see its current value by the DO status LED display at the top of the module Display area.



13.8.3 DO pulse output

This function is used to force the specific DO channel to work as a monstable operation. After a certain period of time, the DO returns to the stable state until another triggering command is applied.

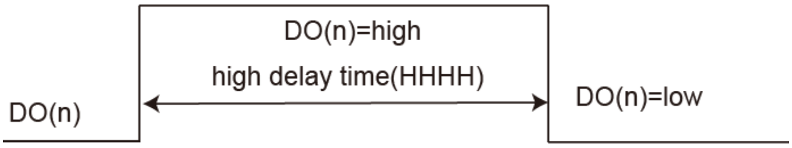
♦ **ASCII command:**

1.	$AACONNDD	;Set DO(n) to low to high or high to low DO pulse output mode
2.	$AA9DNNLLLLHHHH	;Set low/high delay width of DO(n)
3.	$AA9TN	;Start DO pulse output operation

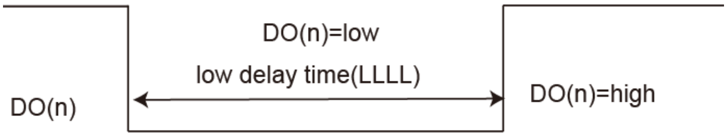
♦ **Modbus command:**

1.	X+1453 ~ X+1484	;Set DO(0~31) to low to high delay mode(=2) or high to low delay mode(=3)
2.	X+1711 ~ X+1774	;Set high/low delay time of the specific DO(n)
3.	X+1775 ~ X+1806	;Start DO pulse output operation

♦ **Low to High DO(n) pulse output**

♦ 

♦ **High to Low DO(n) pulse output**


DO(n)=low
low delay time(LLLL)
DO(n)
DO(n)=high