Table of Contents

1. Function Summary

1.1 Common Functions

TCP_Open	: To initiate the TCPDAQ.dll to use.
TCP_Close	: To terminates use of the TCPDAQ.dll.
TCP_Connect	: To create a Window TCP socket then establishing a connection to a specific EDAM-9000
TCP_Disconnect	: Disconnecting the Window TCP socket from all EDAM-9000 modules
TCP_ModuleDisconnect	: Disconnecting the Window TCP socket from a specific EDAM-9000
TCP_SendData	: Send data to a specific EDAM-9000 module
TCP_RecvData	: Receive data to a specific EDAM-9000 module
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
UDP_Connect	: To create a Window UDP socket then establishing a connection to a specific EDAM-9000
UDP_Disconnect	: Disconnecting the Window UDP socket from all EDAM-9000 modules
UDP_ModuleDisconnect	: Disconnecting the Window UDP socket from a specific EDAM-9000
UDP_SendData	: Send data to a specific EDAM-9000 module
UDP_RecvData	: Receive data to a specific EDAM-9000 module
UDP_SendReceiveASCcmd	: Direct send an ASCII format string as a command, and receive the response from EDAM-9000
TCP_GetModuleIPinfo	: Return module IP information of a specific module
TCP_GetModuleID	: Return module ID number of a specific module
TCP_GetIPFromID	: Return IP address of a specific module ID number
TCP_ScanOnLineModules	: Scan all on-line EDAM-9000 modules
TCP_GetDLLVersion	: Return the DLL's version, that is the version of TCPDAQ.DLL
TCP_GetModuleNo	: Return the module name of a specific IP address
TCP_GetLastError	: Return the error code of the latest called function
TCP_PingIP	: Ping to Remote IP address

1.2 Stream/Event Functions

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

1.3 Digital I/O Functions

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

1.4 Analog I/O Functions

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

1.5 MODBUS/TCP Functions

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

2. 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

2.1 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*.

2.2 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*.

2.3 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*.

2.4 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.

2.5 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*.

2.6 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*.

2.7 TCP_RecvData

Description: receive data from 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*.

2.8 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*.

2.9 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*.

2.10 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

2.11 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*.

2.12 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*.

2.13 UDP_RecvData

Description: receive data from 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*.

2.14 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(SOCKET UDPsock,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*.

2.15 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*.

2.16 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*.

2.17 TCP_GetIPFromID

Description: get IP address for a specific module's 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*.

2.18 TCP_ScanOnLineModules

Description: search on-line EDAM9000 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*.

2.19 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.

2.20 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*.

2.21 TCP_GetLastError

Description: return the error code of the latest called function

Syntax:

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

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

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

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

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

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

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

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

Parameters:

void

Return Code:

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

2.22 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

2.23 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*.

2.24 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*.

2.25 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*.

2.26 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*.

2.27 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*.

2.28 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*.

2.29 TCP_ReadAllDataFromModule

Description: to read all data from a specific module (see data_struture.pdf)

Syntax:

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

```
Declare Function TCP_ReadAllDataFromModule Lib "TCPDAQ.dll" Alias
                "_TCP_ReadAllDataFromModule@8" (ByVal szIP As String, _
                ByRef ModuleData As ModuleData) As Long
```

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

```
int     TCP_ReadAllDataFromModule(char szIP[],struct ModuleData *ModuleData);
```

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

```
Function   TCP_ReadAllDataFromModule(SzIP: PChar; Var ModuleData: TModuleData);
```

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

```
int     TCP_ReadAllDataFromModule(char szIP[],struct ModuleData *ModuleData);
```

Parameters:

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

ModuleData [out]: points to ModuleData structure that stored module data (ref. to TCPDAQ.H)

Return Code:

refer to the *Error code*.

2.30 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*.

2.31 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

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

Return Code:

refer to the *Error code*.

2.32 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*.

2.33 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*.

2.34 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*.

2.35 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*.

2.36 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*.

2.37 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*.

2.38 TCP_ClearDILatch

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

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*.

2.39 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' mode

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*.

2.40 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*.

2.41 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*.

2.42 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*.

2.43 TCP_ReadDODelayWidth

Description: to read the pulse and delay signal widths from specific EDAM-9000 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*.

2.44 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*.

2.45 TCP_ReadAITypes

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

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

```
Declare Function 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*.

2.46 TCP_ReadAIMaxVal

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

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

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

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

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

Delphi: *(see TCPDAQ.pas)*

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

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

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

Parameters:

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

2.47 TCP_ReadAIMinVal

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

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

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

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

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

Delphi: *(see TCPDAQ.pas)*

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

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

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

Parameters:

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

2.48 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*.

2.49 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*.

2.50 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*.

2.51 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*.

2.52 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*.

2.53 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*.

2.54 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*.

2.55 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*.

2.56 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*.

2.57 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*.

2.58 TCP_ClearAILatchAlarm

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

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

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

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

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

Delphi: *(see TCPDAQ.pas)*

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

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

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

Parameters:

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

2.59 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*.

2.60 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*.

2.61 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*.

2.62 TCP_ReadAIAlarmLimit

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

Syntax:

Visual Basic: (*see TCPDAQ.bas*)

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

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

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

Delphi: *(see TCPDAQ.pas)*

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

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

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

Parameters:

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

2.63 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*.

2.64 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

2.65 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*.

2.66 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*.

2.67 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*.

2.68 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*.

2.69 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*.

2.70 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*.

2.71 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*.