Open Modbus/TCP

Application Specific Function Block Manual

Part Number M.1302.1690

Version 1.0

Sheffield Automation, LLC

NOTE

Progress is an on-going commitment at Sheffield Automation. We continually strive to offer the most advanced products in the industry; therefore, information in this document is subject to change without notice. The illustrations and specifications are not binding in detail. Sheffield Automation shall not be liable for any technical or editorial omissions occurring in this document, nor for any consequential or incidental damages resulting from the use of this document.

DO NOT ATTEMPT to use any Sheffield Automation product until the use of such product is completely understood. It is the responsibility of the user to make certain proper operation practices are understood. Sheffield Automation products should be used only by qualified personnel and for the express purpose for which said products were designed.

Should information not covered in this document be required, contact the Customer Service Department, Sheffield Automation, 660 South Military Road, P.O. Box 1658, Fond du Lac, WI 54936-1658. Sheffield Automation can be reached by telephone at (920) 921–7100.

DISCLAIMER: All programs in this release (application demos, application specific function blocks (ASFB's), etc.), are provided "AS IS, WHERE IS", WITHOUT ANY WARRANTIES, EXPRESS OR IMPLIED. There may be technical or editorial omissions in the programs and their specifications. These programs are provided solely for user application development and user assumes all responsibility for their use. Programs and their content are subject to change without notice.

M.1302.1690

Release 2302

Table of Contents:
Open Modbus/TCP AFSB Manual

NOTES

CHAPTER 1 Application Specific Function Block Guidelines

Installation

The following guidelines are recommended ways of working with Application Specific Function Blocks (i.e. ASFBs) from Giddings & Lewis.

The Applications CD includes the ASFB package as follows:

- .LIB file(s) containing the ASFB(s)
- source .LDO(s) from which the ASFB(s) was made
- example LDO(s) with the ASFB(s) incorporated into the ladder which you can then use to begin programming from or merge with an existing application ladder

When you install the Applications CD, the ASFB paths default to:

C:\Program Files\Giddings & Lewis\Open Modbus TCP ASFB **V*x.x***\ASFB

and

C:\Program Files\Giddings & Lewis\Open Modbus TCP ASFB **V*x.x***\Examples

The .LIB files and source .LDO files are put in the ASFB subdirectory. The example .LDO files are put in the Examples subdirectory.

Revisions

The first four networks of each ASFB source ladder provide the following information:

Network 1

The first network just informs you that the ASFB is provided to assist your application development.

Network 2

The second network is used to keep a revision history of the ASFB. Revisions can be made by Giddings & Lewis personnel or by you.

The network identifies the ASFB, lists the requirements for using this ASFB, the name of the library the ASFB is stored in, and the revision history.

The revision history includes the date, ASFB version (see below), the version of PiCPro used while making the ASFB, and comments about what the revision involved.

When an ASFB is revised, the number of the first input (EN_ _ or RQ_ _) to the function block is changed in the software declarations table. The range of numbers available for Giddings & Lewis personnel is 00 to 49. The range of numbers available for you is 50 to 99. See chart below.

Revision	Giddings & Lewis revisions	User revisions
1st	EN00	EN50
2nd	EN01	EN51
.	.	.
.	.	.
.	.	.
50th	EN49	EN99

Network 3

The third network describes what you should do if you want to make a revision to the ASFB.

ASFB Input/Output Descriptions

Network 4

The fourth network describes the ASFB and defines all the inputs and outputs to the function block.

Using ASFBs

When you are ready to use the ASFB in your application, there are several approaches you can take as shown below.

- Create a new application LDO starting with the example LDO for the ASFB package. The advantage is that the software declarations table for the ASFB has been entered for you.
- If you already have an application LDO, copy and paste the example LDO into yours. The software declaration tables for both LDOs will also merge.

CHAPTER 2 Configuration and Software Installation

Introduction

The Open Modbus/TCP ASFB software package from Giddings & Lewis allows the MMC, MMC for PC, or PiC900/90 to communicate with other Open Modbus devices over Ethernet using the Open Modbus protocol. The Giddings & Lewis control can be programmed as a Client, a Server or both.

When programmed as a Client, the G&L Control issues read or write commands to another Open Modbus/TCP device configured as a Server. Conversely, when programmed as a Server, the G&L control responds to read or write commands from another Open Modbus/TCP Client device.

When programmed as both a Client and a Server, the G&L control can issue read and write commands to Open Modbus/TCP Server devices as well as respond to read and write commands from another Open Modbus/TCP device.

Throughout this document G&L is used as the generic description for the Giddings & Lewis MMC, MMC for PC, or PiC900/90 control platforms. Also, the term Modbus/TCP will be used interchangeably with Open Modbus/TCP.

Modbus/TCP Description

Open Modbus/TCP or Modbus/TCP is a variant of the Modbus family of serial communication protocols intended for supervision and control of automation equipment. Specifically, it covers the use of Modbus messaging in an Intranet or Internet environment using the Ethernet TCP/IP protocol for connection to PLCs, I/O modules, and other simple fieldbuses or I/O networks.

Modbus/TCP uses the Ethernet connection-oriented TCP protocol that maintains an individual Modbus/TCP transaction by enclosing it in a connection that can be identified, supervised and closed without requiring specific action on the part of the client and server applications. This gives the mechanism a wide tolerance to network performance changes, and allows security features such as firewalls and proxies to be easily added.

A Modbus/TCP request and response body, from the function code to the end of the data portion, have exactly the same layout and meaning as in other Modbus variants, such as;

- Modbus serial port - ASCII encoding
- Modbus serial port - RTU (binary) encoding
- Modbus PLUS network - data path

Data Input/Output Descriptions

Modbus/TCP, like serial Modbus, bases its data model on a series of tables:

input descretes - single bit, provided by an I/O system, read only

output discretes - single bit, alterable by an application program, read-write

input registers - 16-bit quantitiy, provided by an I/O system, read only

output registers - 16 bit quantity, alterable by an application program, read-write

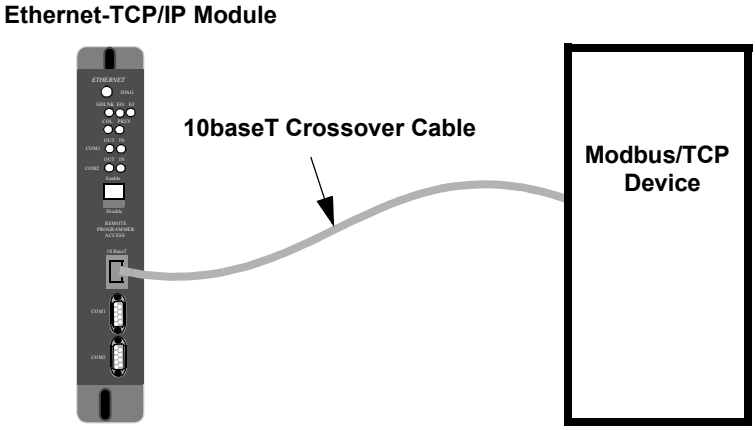
For each of the primary tables, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span multiple consecutive data items up to a date size limit that is dependent on the transaction function code.

For example, a Modbus message requesting the read of a register at offset 0 would return the value know to the application programmer as found in register 4:00001 (memory type 4 = output register, reference 00001).

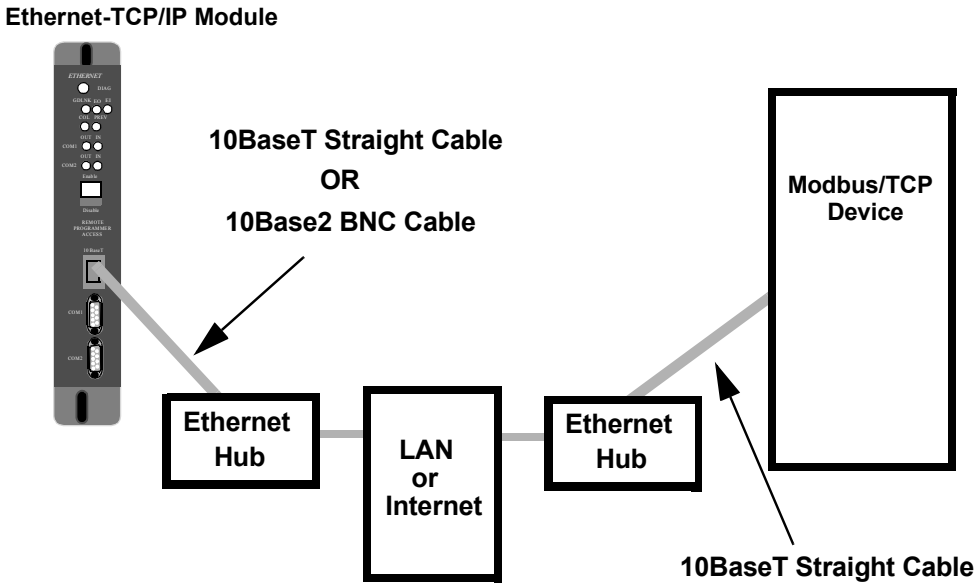
Hardware Configuration

Refer to the MMC and PiC900 Hardware Manuals for detailed information on configuring the hardware.

The G&L and Modbus/TCP Device modules can be connected directly together using a 10baseT crossover cable.



If you are communicating over a plant network you can connect the G&L to an Ethernet HUB using a 10baseT straight cable or a 10Base2 BNC cable. The Modbus/TCP Device is then connected to an Ethernet HUB using a 10baseT straight cable. Both of the Devices would then be connected to the plant network.



G&L Client - Open Modbus/TCP Client Design

The following is a list of requirements for the G&L Client – Open Modbus Client design.

1. Establish a TCP connection to port 502.
2. Submit the Modbus request, including its 6 -byte Modbus prefix, to the server.
3. Wait for a response from the server to appear on the same TCP connection.
4. Read the first 6 bytes of the response, indicating the actual length of the response message.
5. Read the remaining bytes of the response.
6. If no further communications is expected to the server in the immediate future, close down the TCP connection so that the resources at the server can be used to serve other clients.

In the event of a timeout waiting for a response, issue a unilateral close of the connection. If desired, it is up to the ladder application to re-establish a connection.

G&L Server - Multithreaded Open Modbus/TCP Server Design

The following is a list of requirements for the G&L Server – Multithreaded Open Modbus Server design.

1. Wait for incoming connections on TCP port 502.
2. When a new connection request is received, accept it and spawn a new thread to handle the connection.

Within the new thread, do the following in an infinite loop:

3. Read the first 6 byte Modbus/TCP header from the client request.
4. Analyze the header. If if appears corrupt (e.g. the protocol field is non-zero or the length of the message is larger than 256) then unilaterally close the connection.
5. Read the remaining bytes of the message, whose length is now known.
6. Process the incoming Modbus message based on the type of function code.
7. Generate the Modbus/TCP prefix for the response.
8. Send the response, including the Modbus/TCP prefix, as a single buffer for transmission on the connection, using send().
9. Go back and wait for the next 6 byte prefix (go back to step 3 above).

In the event of a timeout waiting for a response, issue a thread close of the connection.

Software Requirements

- Giddings & Lewis Open Modbus/TCP ASFB software
- Giddings & Lewis PiCPro for Windows V13.0 or higher Programming Software
- LDOMERGE software (Optional software that allows you to merge ladders)

When the G&L is used as a Modbus/TCP server, it responds to the following Modbus commands;

Function Code	Command	Description
01	Read Coil Status	Obtains current status (ON/OFF) of a group of logic coils.
03	Read Holding Registers	Obtain current binary value in one or more holding registers.
05	Force Single Coil	Force logic state to a state of ON or OFF.
06	Preset Single Register	Place a specific binary value into a holding register.
07	Read Exception Status	Obtain the status of the eight internal coils whose addresses are controller dependent. You can program these coils to indicate slave status. Short message lenght allows rapid reading of status.
15	Force Multiple Coils	Forces a series of consecutive logic coils to defined ON or OFF status.
16	Preset Multiple Registers	Places specific binary values into a series of consecutive holding registers.

The device the G&L is communicating with must support these commands. If the G&L receives a command it does not support or recognize, it will return an error response to the sender.

When the G&L is used as a Modbus/TCP CLIENT, it responds to the following commands.

Function Code	Command	Description
01	Read Coil Status	Obtains current status (ON/OFF) of a group of logic coils.
02	Read Input Status	Obtain current status of the physical inputs (Inputs 10000 to 19999).
03	Read Holding Registers	Obtain current binary value in one or more holding registers.
04	Read Input Registers	Obtain current value in one or more physical input registers (Inputs 20000 to 29999).
05	Force Single Coil	Force logic coil to a state of ON or OFF.
06	Preset Single Register	Place a specific binary value into a holding register.
07	Read Exception Status	Obtain the status (ON/OFF) of the eight internal coils whose addresses are controller dependent. You can program these coils to indicate slave status. Short message length allows rapid reading of status.
15	Force Multiple Coils	Forces a series of consecutive logic coils to defined ON or OFF states.
16	Preset Multiple Registers	Places specific binary values into a series of consecutive holding registers.

The device the G&L is communicating with must support the commands. If the G&L sends a command the other device does not recognize, it will respond with an error response.

Message Addressing

The addressing between the G&L and Modbus/TCP is as follows:

BOOLEANS		INTEGERS	
Modbus	**PiC900**	**Modbus**	**PiC900**
00001	BOOL(0)	40001	DAT(0)
00002	BOOL(1)	40002	DAT(1)
.	.	.	.
.	.	.	.
00999	BOOL(998)	40999	DAT(998)

Software Installation

Insert the Giddings & Lewis Open Modbus/TCP ASFB software CD. If the CD doesn't auto run go to Start-Run-Setup.exe to install the software. The files will be copied to the default folder c:\Program Files\Giddings & Lewis\Open Modbus TCP ASFB ***Vx.x***\Examples. During installation you can change the destination folder.

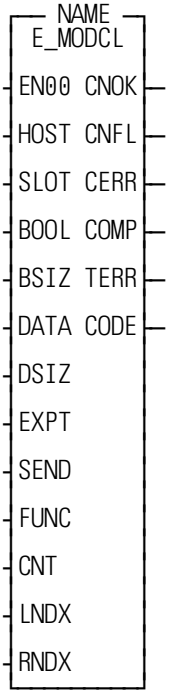
NOTES

CHAPTER 3 Open Modbus/TCP ASFBs

The following table lists the files for the Open Modbus/TCP application specific function blocks.

NOTE: Every .LDO file on the CD has a corresponding .REM file. The REM files contain all the comments found in the LDO files. If you move an .LDO file to a different location, be sure to move its REM file to the same directory.

G&L Server ASFBs		
	E_MODSEX.LDO	Example MODBUS/TCP ladder with the G&L as the Server.
	E_MODSVR.LDO	MODBUS/TCP Server source ladder.
G&L Client ASFBs		
	E_MODCEX.LDO	Example MODBUS/TCP ladder with the G&L as the Client.
	E_MODCL.LDO	MODBUS/TCP Client source ladder.
PiCPro LIB ASFBs		
	E_MODTCP.LIB	Library containing the compiled MODBUS/TCP ASFBs.
	E_MODTCP.CHM	Online function block help file.
MODBUS/TCP Support ASFBS		
	E_MODPRC.LDO	This function block handles all incoming requests from a MODBUS/TCP Client.
	E_MODRD.LDO	This function block reads an incoming TCP message.
	E_MODMOV.LDO	This function block moves integer data from the TCP message buffer to local Integer data or local Integer data to the TCP message buffer.
	E_MODPAK.LDO	This function block packs local Boolean data into the TCP message buffer.
	E_MODUNP.LDO	This function block unpacks the message buffer into local Boolean data.



Inputs: EN00 (BOOL) - enables execution

HOST (STRING) - IP address of MODBUS/TCP Server

SLOT (USINT) - slot number of Ethernet module

BOOL (ARRAY OF BOOL) - boolean data area

BSIZ (UINT) - size of the BOOL data area

DATA (ARRAY OF INT) - variable data area

DSIZ (UINT) - size of the DATA area

EXPT (ARRAY OF BOOL) - booleans read by the read exception status code (07)

SEND (BOOL) - energize to send a message to a Modbus slave

FUNC (USINT) - function code number to send to the Modbus slave device

CNT (UINT) - number of items to transfer over Modbus

LNDX (UINT) - local index where data received from a slave is stored

RNDX (UINT) - remote index where data will be sent/retrieved in the slave device

Outputs: CNOK (BOOL) - execution completed without error

CNFL (BOOL) - initialization failed

CERR (INT) - 0 if initialization failed; not equal to 0 if initialization is unsuccessful

COMP (BOOL) - energized when a transfer is complete

TERR (BOOL) - an error occurred in the transaction

CODE (INT) - number of error code,
code number < 100 = error returned from a slave device via an exception response,
code number > 99 = local error

```
<<INSTANCE NAME>>:E_MODCL(EN99 := <<BOOL>>, HOST :=
  <<STRING>>, SLOT := <<USINT>>, BOOL := <<ARRAY OF BOOL>>,
  BSIZ := <<USINT>>, DATA := <<ARRAY OF INT>>, DSIZ := <<UINT>>,
  EXPT := <<ARRAY OF BOOL>>, SEND := <<BOOL>>, FUNC :=
  <<USINT>>, CNT := <<UINT>>, LNDX := <<UINT>>, RNDX :=
  <<UINT>>, CNOK => <<BOOL>>, CNFL => <<BOOL>>, CERR =>
  <<INT>>, COMP => <<BOOL>>, TERR => <<BOOL>>, CODE =>
  <<INT>>);
```

The E_MODCL ASFB is a Modbus/TCP Client and provides data exchange over Ethernet between a PiC, MMC, MMC for PC or another device acting as a Modbus/TCP Server.

When the EN00 enable input is energized, the function block will open a TCP socket and connect to the device specified at the HOST input. If a connection is established, the CNOK output will be energized and the system will be ready to generate Client requests. If the attempt to establish a connection fails, the CNFL output will be energized and an associated error number will be displayed at the CERR output. To maintain a connection, the function block must remain enabled. In the event of a connection failure, disable and re-enable the function block.

A Client data request is issued with a positive transition at the SEND input. Upon a successful completion of the request, the COMP output will be energized. If the request was unsuccessful due to invalid input data, the TERR transaction error output will be energized and the associated error number will be dispalyed at the CODE output. If the request failed due to a connection problem, the CNFL output will be energized and an associated error number will be displayed at the CERR output.

The Client has a watchdog timeout condition that will end a Client/Server communication session. If a Client data request was initiated and the request was not sent to the Server within 5 seconds, the session will be terminated. The CNFL and TERR outputs will be energized and the associated error number will be displayed at the CODE output.

INPUTS:

EN00 The EN input is energized every scan to initiate a query over the Ethernet network. In a typical system, this input will be wired to the vertical or power bus rail.

NOTE: De-energizing this input will cause communication to stop.

HOST The HOST input specifies the IP address of MODBUS/TCP Server.

SLOT Slot number of Ethernet module.

BOOL The BOOL input is an array that specifies the boolean (bit) data area that is used for any boolean (bit) transfers. Queries to a slave device for data items 00001 to 09999 are placed here.

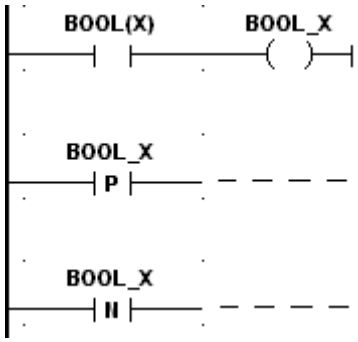
For example, if the array of boolean variables is called BOOL and a write request is made from register 00222, the PiC900 would send the data in BOOL(221).

The array size can range from 2 (0..1)to 999 (0..998) booleans.

IMPORTANT

Do not use a positive or negative transistional contact in your LDO with the BOOL array.

If it is necessary to set up a transistional contact with a BOOL array, use the BOOL array to energize another boolean coil. Then use this boolean for the transistional contact as shown in the example below.



BSIZ* Enter the number of booleans (up to 999).

*It is very important that the value in BSIZ and the size of the array in BOOL are the same. The size is user adjustable from 2 to 999 elements.

DATA The DATA input is used to specify the name of the main data area. Queries to a slave device for data items 40001 to 49999 are placed here.

For example, if the array of integer variables is called DAT and a read request is made for register 40005, the PiC900 would place the data it received in response in DAT(4).

This data area is an array of integers.

DSIZ* Enter the number of integers (up to 999).

*It is very important that the value in DSIZ and the size of the array in DATA are the same. The size is user adjustable from 2 to 999 elements.

EXPT The EXPT input is an array of eight booleans. If the PiC900 asks a remote station for the exception status (function 7), its response will be placed in this array. The bits in this array have no special meaning in the PiC900. But they have special meaning in a Modicon Control and are provided here to allow the PiC900 to read them.

SEND The SEND input must be energized each time a message is sent to one of the Modbus slaves.

FUNC The FUNC input holds the Modbus function code. The function code number shown in the table below is sent to the Modbus slave device.

Function Code	Name of function	Description
01	Read Coil Status	Reads the ON/OFF status of discrete outputs in the slave.
02	Read Input Status	Reads the status of the physical inputs (Inputs 10000 to 19999).
03	Read Holding Registers	Reads the binary contents of holding registers in the slave.
04	Read Input Registers	Reads one or more physical input registers (Inputs 20000 to 29999).
05	Force Single Coil	Forces a single coil to either ON or OFF. When broadcast, the command forces the same coil reference in all attached slaves.
06	Preset Single Register	Presets a value into a single holding register. When broadcast, the command presets the same register reference in all attached slaves.
07	Read Exception Status	Reads the contents of eight Exception Status coils within the slave. These eight coils are user-defined.
15	Force Multiple Coils	Forces each coil in a sequence of coils to either ON or OFF. When broadcast, the function forces the same coil references in all attached slaves.
16	Preset Multiple Registers	Presets values into a sequence of holding registers. When broadcast, the function presets the same register references in all attached slaves.

CNT The CNT input is the number of items to transfer over the Modbus.

LNDX The LNDX input is the local index. It is the location in this control where data received/sent from a slave device will be stored/retrieved.

RNDX The RNDX input is the remote index. It is the location in the slave device where data will be sent/retrieved.

OUTPUTS:

CNOK The OK output when energized indicates that a connection has been established and is ready for communication. If this output does not energize, check the FAIL output and the ERR output to identify the problem.

CNFL The CNFL output when energized indicates that the transceiver initialization failed. When this output is energized, the OK will not be energized and an error code will appear at the ERR output to identify the problem.

CERR The CERR output is 0 if initialization is successful and is _ 0 if initialization is unsuccessful. The error codes that appear at this output are system errors. See Appendix B in the PiC900 Software Manual for a description of each error.

COMP The COMP output energizes when a transfer is complete.

TERR The TERR output energizes when an error in the transaction has occurred.

CODE The CODE output gives the number of the transaction error that has occurred. If the number is less than 100, the error code has been returned from a slave station via an exception response. If the number is greater than 99, the error code is local. See the tables that follow.

The table below contains error codes returned from a Modbus/TCP Client and reported at the CODE output in the form of exception responses. For a complete explanation of these errors, see the Open Modbus/TCP specification, Section 6: Exception Codes.

TERR

Code	Name	Description
1	Illegal function	The function code received in the query is not an allowable action for the slave.
2	Illegal data address	The data address received in the query is not an allowable value for the slave.
3	Illegal data value	A value contained in the Boolean Data query data field is not an allowable value for the slave.

The table below contains the error codes that are detected locally by the G&L and reported at the CODE output.

TERR Code	Name	Description
103	Time-out error	A data request was initiated and the request was not sent within 5 seconds.
104	Invalid function	The value at the FUNC input is invalid. No functions above function 21 are currently supported.
105	Invalid function	The value at the FUNC input is invalid. The function number is not supported.
107	Boolean array size error	The amount of data requested would overflow the boolean data array defined by the user.
108	Integer array size error	The amount of data requested would overflow the integer data array defined by the user.
109	LNDX value error	The offset of data requested would overflow the boolean data array defined by the user. In other words, the location of the data is too close to the end of the array, given the amount of data being transferred.
110	LNDX value error	The amount of data requested would overflow the integer data array defined by the user.
111	Function not supported	The amount of data requested would overflow the integer data array defined by the user.

Modbus/TCP Client

As a Modbus/TCP Client, the G&L will query a remote device for integer and boolean data. The E_MODCL function block described above is entered in your application program one time. It is responsible for all communications to and from the G&L for Modbus/TCP support. Enable it every scan. Each input must have the appropriate variable attached to it.

All data being sent to or retrieved from the G&L will have a function code number associated with it. Although this number has no direct equivalent in the G&L, it is used to determine where to place or retrieve data.

All functions that read registors from a remote device will have their response data placed in the integer array specified at the DATA input. All functions that read booleans from a remote device will have their response data placed in the boolean array specified at the BOOL input.

Modbus/TCP Client example LDO

The example LDO called E_MODCEX.LDO is included with the software files you received. If you are creating a new application ladder, open the E_MODCEX.LDO and use the *save AS* command to name it whatever your application will be called.

If you want to add the E_MODCEX.LDO to an existing application ladder, copy and paste or drag and drop the application ladder contents into your ladder.

Both of these methods produce an application ladder with the software declarations for the E_MODCL function block already entered. You can modify this to fit your application.

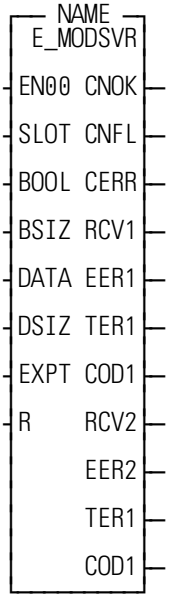
You can also insert the E_MODCL function block into an existing ladder and enter the software declarations yourself.

Please review the example ladders provided with this package.

E_MODCL function block setup

The steps for setting up the E_MODCL function block allowing the G&L to function as a Modbus client follows.

1. Determine the IP address for the Modbus/TCP Server and enter it at the HOST input.
2. Determine how many booleans (bit type) will be needed for your application.
3. Modify the size of the BOOLS array in the software declaration table by setting it to the size determined in step 4. In the software declarations table, place the cursor on the data item named BOOLS and press <Alt> A to enter the array length. The acceptable range is from 2 to 999.
4. The size of the boolean array (BOOLS) must be entered in BSIZ(boolean size).
5. Determine how many integers will be needed for your application. The acceptable range is from 1 to 999.
6. Modify the size of the INTEGER array in the software declaration table by setting it to the size determined in step 7. In the software declarations table, place the cursor on the data item named INTEGER and press <Alt> A to enter the array length.
7. The size of the integer array (INTEGER) must be entered in DSIZ (data size).



Inputs: EN00 (BOOL) - enables execution

SLOT (USINT) - slot where the Ethernet module resides

BOOL (Array of BOOL) - boolean data area

BSIZ (UINT) - size of the BOOL data area

DATA (Array of INTEGERS) - integer data area

DSIZ (UINT) - size of the DATA area

EXPT (Array of BOOL) - booleans read by exception status function code 07

R (STRUCT) - message information including address and function code

Outputs: CNOK (BOOL) - Ethernet connection without error

CNFL (BOOL) - Ethernet connection failed

CERR (INT) - 0 if connection is successful; non 0 if connection failed

RCV1 (BOOL) - energized if a client request was successfully processed on communication session 1

EER1 (INT) - 0 if a client request was processed without error on communication session 1, non 0 if request failed

TER1 (BOOL) - energized if an error occurred in the client message transaction on communication session 1

COD1 (INT) - 0 if client message message transaction was successful on communication session 1, non 0 if transaction was aborted

RCV2 (BOOL) - energized if a client request was successfully processed on communication session 2

EER2 (INT) - 0 if a client request was processed without error on communication session 2, non 0 if request failed

TER2 (BOOL) - energized if an error occurred in the client message transaction on communication session 2

COD2 (INT) - 0 if client message message transaction was successful on communication session 2, non 0 if transaction was aborted

<<INSTANCE NAME>>:E_MODSVR(EN00:= <<BOOL>>, SLOT := <<USINT>>, BOOL := <<ARRAY OF BOOL>>, BSIZ := <<USINT>>, DATA := <<ARRAY OF INT>>, DSIZ := <<UINT>>, EXPT := <<ARRAY OF BOOL>>, R := <<STRUCT>>, CNOK => <<BOOL>>, CNFL => <<BOOL>>, CERR => <<INT>>, RCV1 => <<BOOL>>, EER1 => <<INT>>, TER1 => <<BOOL>>, COD1 => <<INT>>, RCV2 => <<BOOL>>, EER2 => <<INT>>, TER2 => <<BOOL>>, COD2 => <<INT>>;

The E_MODSVR ASFB is a Modbus/TCP multi-communication Server and provides data exchange over Ethernet between a PiC, MMC, MMC for PC or other device acting as Modbus/TCP Client. The Server supports simultaneous connections allowing two Client/Server communication sessions.

When the EN00 enable input is energized the ASFB will open a TCP socket. If a socket is established, the CNOK output will be energized and the system will be ready respond to Client requests. When a request to connect is received by the Server, the socket will bind the Client to one of two possible communication sessions. If the attempt to establish a socket fails, the CNFL output will be energized and an associtated error number will be displayed at the CERR output. To maintain a socket or communication session, the function block must be enabled. In the event of a CNFL connection failure, disable and re-enable the ASFB.

When a Client request is received the Server will process it and respond. If the response is successful, the RCVx* output will be energized. If the response was unsuccessful, various outputs will display the error. If the failure was due to invalid request data, the TERx* transaction error output will be energized and the associated error number will be dispalyed at the ERRx* output only. In this case, it is important to realize that the failure occurred on one of two sessions. Because the other connection is still valid, the CNFL output is not energized.

The Server has two watchdog timeout conditions that will end a Client/Server communication session. The first is a session timeout where if the session exceed 30 seconds without any data transfer activity, the session will be terminated. The second is a response timeout. The response timeout results if a request from the Clent was received and a Server response was not returned within 5 seconds. If this occurs the session will be terminated. In either case, the TERx* transaction error output will be energized and the associated error number will be displayed at the CODx* output.

* "x" denotes Client/Server communication session 1 or 2.

INPUTS:

EN00 The EN input is energized every scan to respond to a query over the Ethernet network. In a typical system, this input will be wired to the vertical or power bus rail.

NOTE: De-energizing this input will cause communication to stop.

SLOT Slot number of Ethernet module.

BOOL The BOOL input is an array that specifies the boolean (bit) data area that is used for any boolean (bit) transfers. Queries from the master device for data items 00001 to 00999 are found here.

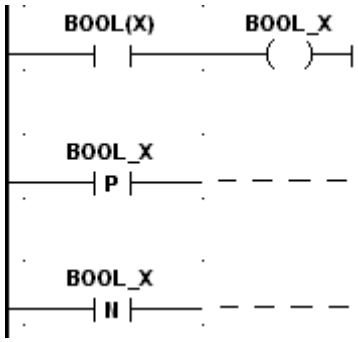
For example, if the array of booleans variable is called BOOL and a query is made for register 00222, the PiC900 would respond with the data in BOOL(221).

The array size can range from 2 (0..1)to 999 (0..998) booleans.

IMPORTANT

Do not use a positive or negative transistional contact in your LDO with the BOOL array.

If it is necessary to set up a transistional contact with a BOOL array, use the BOOL array to energize another boolean coil. Then use this boolean for the transistional contact as shown in the example below.



BSIZ* Enter the number of booleans (up to 999).

*It is very important that the value in size in BSIZ and the size of the array in BOOL are the same. The size is user adjustable from 2 to 999 elements.

DATA The DATA input is used to specify the name of the main data area. Queries from the master device for data items 40001 to 40999 are found here.

For example, if the array of integers variable is called DAT and a query is made for register 40005, the G&L would respond with the data in DAT(4).

This data area is an array of integers.

DSIZ* Enter the number of integers (up to 999).

*It is very important that the value in size in DSIZ and the size of the array in DATA are the same. The size is user adjustable from 2 to 999 elements.

EXPT When the read exception status command is issued by the master device, the values in this boolean array are returned. The booleans are user-defined.

R The R structure specifies a data area that information about the last query is placed. When a query is received, Modbus/TCP function data is placed in the data area specified by this input. The structure placed at this input must have the format shown below.

Declared array of structures for R input

R	STRUCT
.ADDRESS	USINT
.FUNCTION	USINT
	END_STRUCT

The function codes for the Modbus/TCP functions are as follows:

Function Code	Name of function	Description
01	Read Coil Status	Reads the ON/OFF status of discrete outputs in the slave.
03	Read Holding Registers	Reads the binary contents of holding registers in the slave.
05	Force Single Coil	Forces a single coil to either ON or OFF. When broadcast, the command forces the same coil reference in all attached slaves.
06	Preset Single Register	Presets a value into a single holding register. When broadcast, the command presets the same register reference in all attached slaves.
07	Read Exception Status	Reads the contents of eight Exception Status coils within the slave. These eight coils are user-defined.
15	Force Multiple Coils	Forces each coil in a sequence of coils to either ON or OFF. When broadcast, the function forces the same coil references in all attached slaves.
16	Preset Multiple Registers	Presets values into a sequence of holding registers. When broadcast, the function presets the same register references in all attached slaves.

OUTPUTS:

CNOK When energized, the CNOK output indicates that a socket has been established and is ready for communication. If this output does not energize, check the CNFL output and the CERR output to identify the problem.

CNFL When energized, the CNFL output indicates that the transceiver initialization failed. When this output is energized, the OK will not be energized and an error code will appear at the CERR output to identify the problem.

CERR The CERR output is 0 if initialization is successful and is _ 0 if initialization is unsuccessful. The error codes that appear at this output are system errors.

RCV1 The RCV1 output energizes when a transfer on communication session 1 is complete.

EER1 The EER1 output energizes when an error in the transaction has occurred on communication session 1.

TERR1 The TERR1 output energizes when an error in the transaction has occurred on communication session 1.

COD1 The COD1 output gives the number of the transaction error that has occurred on communication session 1. If the number is less than 100, the error code has been returned from a Client via an exception response. If the number is greater than 99, the error code is local. See the tables that follow.

RCV2 The RCV2 output energizes when a transfer on communication session 2 is complete.

EER2 The EER2 output energizes when an error in the transaction has occurred on communication session 2.

TERR2 The TERR2 output energizes when an error in the transaction has occurred on communication session 2.

COD2 The COD2 output gives the number of the transaction error that has occurred on communication session 2. If the number is less than 100, the error code has been returned from a Client via an exception response. If the number is greater than 99, the error code is local. See the tables that follow.

The table below contains error codes returned from a Modbus/TCP Server and reported at the CODE output in the form of exception responses. For a complete explanation of these errors, see the Open Modbus/TCP specification, Section 6: Exception Codes.

TERR Code	Name	Description
1	Illegal function	The function code received in the query is not an allowable action for the slave.
2	Illegal data address	The data address received in the query is not an allowable value for the slave.
3	Illegal data value	A value contained in the Boolean Data query data field is not an allowable value for the slave.

The table below contains the error codes that are detected locally by the G&L and reported at the CODE output.

TERR Code	Name	Description
103	Time-out error	The session exceeded 30 seconds without any data transfer activity or a response was not returned within 5 seconds.
104	Invalid function	No functions above function 21 are currently supported.
105	Invalid function	The function number is not supported.
107	Boolean array size error	The amount of data requested would overflow the boolean data array defined by the user.
108	Integer array size error	The amount of data requested would overflow the integer data array defined by the user.
109	LNDX value error	The offset of data requested would overflow the boolean data array defined by the user. In other words, the location of the data is too close to the end of the array, given the amount of data being transferred.
110	LNDX value error	The amount of data requested would overflow the integer data array defined by the user.
111	Function not supported	The amount of data requested would overflow the integer data array defined by the user.

Modbus/TCP Server

As a Modbus/TCP Server, the G&L will receive and respond to Modbus functions from other devices but will not initiate any transfers. The E_MODSVR function block described above is entered in your application program one time. It is responsible for all communications to and from the G&L for Modbus/TCP support. Enable it every scan. Each input must have the appropriate variable attached to it.

All data being sent to or retrieved from the G&L will have a code number associated with it. Although this number has no direct equivalent in the G&L, it is used to determine where to place or retrieve data.

The G&L will only respond to requests directed at one of the function codes it supports. Requests made to any other function codes will generate an error response to the device that made the request.

Modbus/TCP Server example LDO

The example LDO called E_MODSEX.LDO is included with the software files you received. If you are creating a new application ladder, open the E_MODSEX.LDO and use the *save AS* command to name it whatever your application will be called.

If you want to add the E_MODSEX.LDO to an existing application ladder, you can use the optional LDOMERGE software to combine them.

Both of these methods produce an application ladder with the software declarations for the E_MODSVR function block already entered. You can modify this to fit your application.

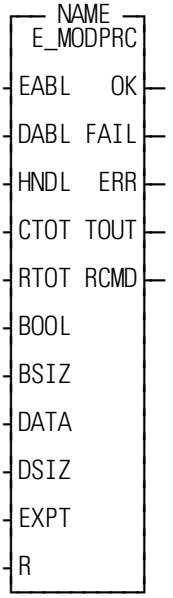
You can also insert the E_MODSVR function block into an existing ladder and enter the software declarations yourself.

Please refer to the example ladders included with this package.

E_MODSVR function block setup

The steps for setting up the E_MODSVR function block allowing the G&L to function as a Modbus slave follows.

1. Determine the slot where the Ethernet module resides and enter it at SLOT.
2. Determine how many booleans (bit type) will be needed for your application.
3. Modify the size of the BOOLS array in the software declaration table by setting it to the size determined in step 4. In the software declarations table, place the cursor on the data item named BOOLS and press <Alt> A to enter the array length. The acceptable range is from 2 to 999.
4. The size of the boolean array (BOOLS) must be entered in BSIZ(boolean size).
5. Determine how many integers will be needed for your application. The acceptable range is from 1 to 999.
6. Modify the size of the INTEGER array in the software declaration table by setting it to the size determined in step 7. In the software declarations table, place the cursor on the data item named INTEGER and press <Alt> A to enter the array length.
7. The size of the integer array (INTEGER) must be entered in DSIZ (data size).



Inputs: EABL (BOOL) - enables execution

DABL (BOOL) - disables execution

HNDL (UINT) - socket handle

CTOT (TIME) - connection timeout

RTOT (TIME) - response timeout

BOOL (ARRAY OF BOOL) - boolean data area

BSIZ (UINT) - size of the BOOL data area

DATA (ARRAY OF INTGERS) - integer data area

DSIZ (UINT) - size of the data area

EXPT (ARRAY OF BOOL) - booleans read by exception status function code 07

R (STRUCT) - message information including address and function code

Outputs: OK (BOOL) - execution completed without error

FAIL (BOOL) - Ethernet connection failed

ERR (INT) - 0 if connection is successful; non zero if connection failed

TOUT (BOOL) - energized if a connection or response timeout occurred

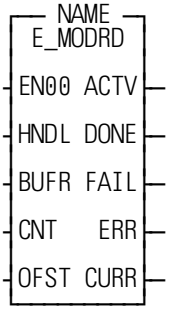
RCMD (BOOL) - energized if Client request was processed without error

<<INSTANCE NAME>>:E_MODPRC(EABL := <<BOOL>>, DABL := <<BOOL>>, HNDL := <<UINT>>, CTOT := <<TIME>>, RTOT := <<TIME>>, BOOL := <<ARRAY OF BOOL>>, BSIZ := <<UINT>>, DATA := <<ARRAY OF INTEGERS>>, DSIZ := <<UINT>>, EXPT := <<ARRAY OF BOOL>>, R := <<STRUCT>>, OK => <<BOOL>>, FAIL => <<BOOL>>, ERR => <<INT>>, TOUT => <<BOOL>>, RCMD => <<BOOL>>);

The E_MODPRC ASFB is nested in the E_MODSVR ASFB and is used to process the Client request.

E_MODRD

Reads data on the TCP connection **USER/E_MODTCP**



Inputs: EN00 (BOOL) - enables execution

HNDL (UINT) - socket handle from IPSOCK

BUFR (ARRAY OF BYTE) - byte data area

CNT (UINT) - number of bytes to read

OFST (UINT) - byte offset into data message to read

Outputs: ACTV (BOOL) - read active

DONE (BOOL) - read failed

FAIL (BOOL) - read failed

ERR (INT) - 0 if read is successful; non zero if read failed

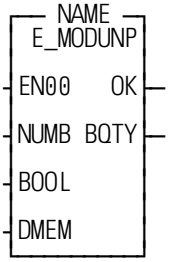
CURR (UINT) - number of bytes actually read

<<INSTANCE NAME>>:E_MODRD(EN00 := <<BOOL>>, HNDL := <<UINT>>, HNDL := <<UINT>>, BUFR := <<ARRAY OF BYTE>>, CNT := <<UINT>>, OFST := <<UINT>>, ACTV => <<BOOL>>, DONE => <<BOOL>>, FAIL => <<BOOL>>, ERR => <<INT>>, CURR => <<UINT>>);

The E_MODRD ASFB is nested in the both the E_MODCL and E_MODPRC ASFBs and is used to read data on the TCP connection.

E_MODUNP

Unpacks data into discrete boolean data **USER/E_MODTCP**



Inputs: EN00 (BOOL) - enables execution

NUMB (UINT) - number of bytes to unpack

BOOL (ARRAY OF BOOL) - booean data memory area

DMEM (ARRAY OF BYTE) - data memory area

Outputs: OK (BOOL) - execution completed without error

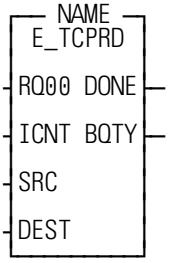
BQTY (UINT) - actual number of bytes unpacked

<<INSTANCE NAME>>:E_MODUNP(EN00 := <<BOOL>>, NUMB := <<UINT>>, BOOL := <<ARRAY OF BOOL>>, DMEM := <<ARRAY OF BYTE>>, OK => <<BOOL>>, BQTY => <<UINT>>);

The E_TCPRD ASFB is nested in the both the E_MODCL and E_MODPRC ASFBs and is used to unpack data into discrete boolean data.

E_MODMOV

Moves data from one memory area to another **USER/E_MODTCP**



Inputs: RQ00 (BOOL) - enables execution
ICNT (UINT) - number of data to move
SRC (ARRAY OF USINT) - source data memory area
DEST (ARRAY OF USINT) - destination data memory area

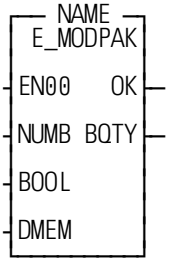
Outputs: DONE (BOOL) - execution completed without error
BQTY (UINT) - actual number of data moved

<<INSTANCE NAME>>:E_MODMOV(RQ00 := <<BOOL>>, ICNT := <<UINT>>, SRC := <<ARRAY OF USINT>>, DEST := <<ARRAY OF USINT>>, DONE => <<BOOL>>, BQTY => <<UINT>>);

The E_MODMOV ASFB is nested in the both the E_MODCL and E_MODPRC ASFBs and is used to move data from one memory area to another.

E_MODPAK

Packs discrete boolean data into byte data **USER/E_MODTCP**



Inputs: RQ00 (BOOL) - enables execution
NUMB (UINT) - number of bytes to pack
BOOL (ARRAY OF BOOL) - boolean data memory area
DMEM (ARRAY OF USINT) - data memory area

Outputs: OK (BOOL) - Ethernet connection made without error
BQTY (UINT) - actual number of data packed

<<INSTANCE NAME>>:E_MODPAK(EN00 := <<BOOL>>, NUMB := <<UINT>>, BOOL := <<ARRAY OF BOOL>>, DMEM := <<ARRAY OF USINT>>, OK => <<BOOL>>, BQTY => <<UINT>>);

The E_MODPAK ASFB is nested in the both the E_MODCL and E_MODPRC ASFBs and is used to pack discrete boolean data into byte data.

Index