Modbus™

Application Specific Function Block Manual

GIDDINGS & LEWIS®

NOTE

Progress is an ongoing commitment at Giddings & Lewis. 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. Giddings & Lewis 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 Giddings & Lewis 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. Giddings & Lewis 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, Giddings & Lewis, 666 South Military Road, P.O. Box 1658, Fond du Lac, WI 54936-1658. Giddings & Lewis can be reached by telephone at (414) 921-7100.

401-55378-00

Version 2 - 1497

DB2-3489

BA22-4489

Table of Contents

Application Specific Function Block Guidelines

Installation 1
Revisions 2
ASFB Input/Output Descriptions 3
Using ASFBs 4

Modbus ASFB Software Package

1.1 Introduction 5
Background on Modbus protocol 6
1.2 Requirements 8
Hardware requirements 8
Cable connections 8
Software requirements 9
Compatibility 9
1.3 Installation 11
1.4 Modbus Function Block 12
C_MODMST 13
The ASCII Mode 17
The RTU Mode 3.8
Modbus Master 21
Modbus master example LDO 22
C_MODMST function block setup 25
C_MODSLV 26
The ASCII Mode 29
The RTU Mode 30
Modbus Slave 31
Modbus slave example LDO 32
C_MODSLV function block setup 34

. Index

NOTES

Application Specific Function Block Guidelines

Installation

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

1. Make a back up copy of the ASFB disk you receive and store the original in a safe place.
2. The disk you receive with the ASFB package will include the following:
   1. ASFBS directory containing:
      - .LIB file(s) containing the ASFB(s)
      - source .LDO(s) from which the ASFB(s) was made
   2. EXAMPLES directory containing:
      - 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

   It is recommended that you copy the .LIB and the source LDO files to your hard drive on the PC in the following way. Remember that ASFB libraries (.LIB) files and source (.LDO) files must be kept in the same directory.

   - Create a directory that will hold all ASFB LIBs and source LDOs. For example, you may have the Motion ASFB package and the Communication ASFB package. Copy the appropriate files on the disks to a directory on your PC called ASFB.

     When you installed PiCPro, the PiCLib statement was automatically entered in your autoexec.bat file as shown below:

     SET PICLIB=C:\PICLIB

     NOTE: If you chose to alter your PICLIB statement during installation, it will look different than what appears above.

     Now add the ASFB directory to your PICLIB = statement as shown below:

     SET PICLIB=C:\PICLIB;C:\ASFB

   - Put the example file(s) in your working directory.
     For example, if you always run PiCPro from the directory which holds all your LDO files, then copy all the ASFB example LDOs to the LDO directory.

Revisions

3. The first three networks of each ASFB source ladder provide the following information.

Network 1

The first 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 1
|...1.....
X-Name ASFB Source Revision History

Located in Library X-LIB

Requirements:
PiCPro Ver 4.0 or higher

Date	Version	Using PiCPro	Comments
MM-DD-YY	EN00	4.1	Original

Network 2

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

If you revise the ASFB, do the following:

1. Do a 'M'odule, save 'A's in order to save the original ASFB before you begin modifying.
2. Change the number on the first input to the ASFB in the software declarations table to a 50 or greater (for example, EN00 would be changed to EN50).
3. Update the revision history in network 1.

ASFB Input/Output Descriptions

Network 3

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

ASFB Description

INPUTS:

Name	Data Type	Definition
EN00	BOOL	enables execution

OUTPUTS:

Name	Data Type	Definition
OK	BOOL	execution complete

4. 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.

     NOTE: To keep the original example LDO, use the 'save As' command. This copies the example LDO to an LDO with the application name you give it.

   - If you already have an application LDO, merge the example LDO with the application LDO using the optional LDOMERGE software package. The software declaration tables for both LDOs will also merge.

   - Enter the ASFB into your application LDO.

     NOTE: This method is not recommended if the software declarations table is lengthy. It requires that you manually enter all the inputs and outputs to the ASFB in the table. With some packages, this is time-consuming. Any structure, array, array of structures, or strings must be entered exactly as it appears in the original table. This is critical to the correct functioning of the ASFB.

Modbus ASFB Software Package

1 .1 Introduction

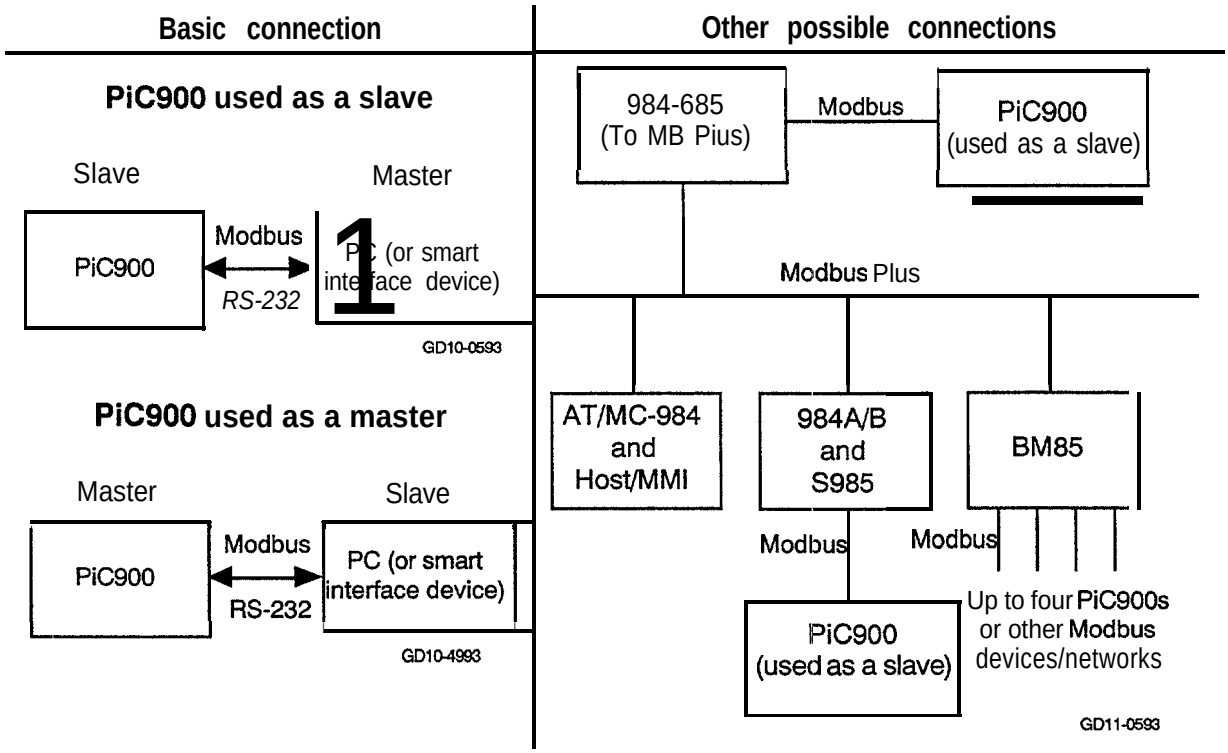
The Modbus ASFB software package from Giddings & Lewis allows the PiC900 to communicate as a Modbus master or slave using the Modbus protocol. The Modbus protocol determines how each controller on a network will know its device address, recognize a message addressed to it, determine the kind of action to be taken, and extract any data or other information contained in the message. If a reply is required, the PiC900 will construct the reply message and send it using Modbus protocol.

Communications is done using a master/slave mode. Only one device, the master, can initiate a transaction called a query. The other devices, the slaves, respond with the data requested by the master or by taking the action specified in the query.

The PiC900 is used as a master or slave device. Communication takes place through the PiC900 serial ports. The serial ports include the USER PORT 2 on the CPU or the ports on a PiC900 serial communications module (2 or 4 port models available).

The master is typically a host processor, an operator interface, or a PiC900. The master can address individual slaves or can initiate a broadcast message to all slaves. Slaves respond to queries that are addressed to them individually. Responses are not returned to broadcast queries from the master. Some possible Modbus configurations are shown in the block diagrams below.

Figure 1. Configurations for using Modbus communications



The interface devices shown in the diagram are described below.

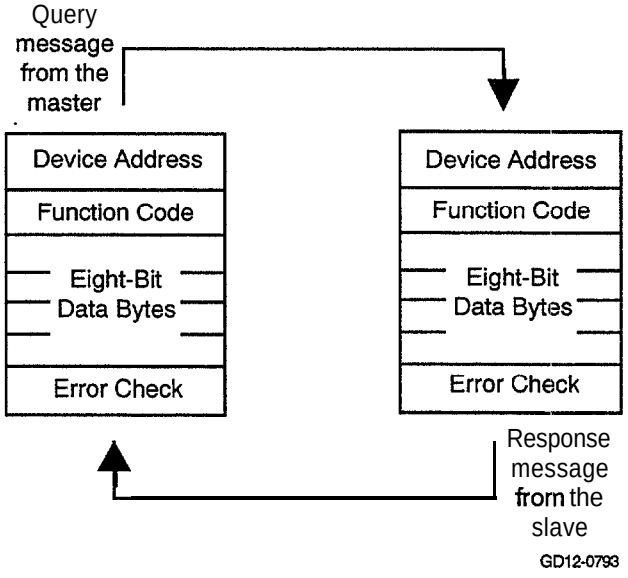
984-685	Modicon controller
AT/MC-984	IBM AT™/Modicon controller
MMI	Man Machine Interface
984 A/B	Modicon controller
S985	Remote I/O bus
BM85	Bridge Multiplexer - Modbus Plus to Modbus bridge

Background on Modbus protocol

As shown below, the Modbus protocol establishes the format for the master query by placing into the query message a device or broadcast address, a function code defining the requested action, any data bytes to be sent, and an error-checking field. The slave response message contains fields confirming the action taken, any data to be returned, and an error-checking field.

If an error occurred in receipt of the message, or if the slave is unable to perform the requested action, the slave will construct an error message and send it as its response.

The Master/Slave Query/Response Cycle



Query

The code in the query tells the addressed slave device what kind of action to perform. The data bytes contain any additional information that the slave will need to execute the command.

For example, code 03 will query the slave to read holding registers and respond with their contents. The data field must contain the information telling the slave which register to start at and how many registers to read. The error check field provides a method for the slave to validate the query message.

Response

If the slave makes a normal response, the code in the response is an echo of the code in the query. The data bytes contain the data collected by the slave, such as register values or status.

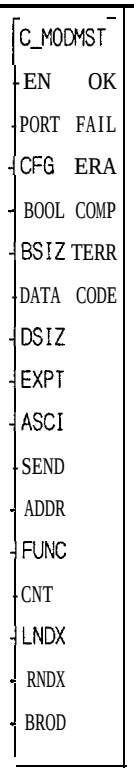
If an error occurs, the code is modified to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the response message contents are valid.

The Modbus software that allows the PiC900 to communicate using the Modbus protocol includes the following ASFBs that you install in PiCPro and use in your application ladder.

C_MODMST

Communications-Modbus master Function block

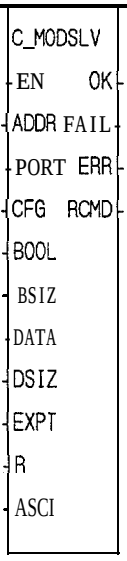
Allows the PiC900 to be used as a master with the Modbus protocol.



C_MODSLV

Communications-Modbus slave function block

Allows the PiC900 to be used as a slave with the Modbus protocol.



1.2 Requirements

The hardware and software requirements when using the Modbus interface to communicate with a compatible device are covered in this section.

Hardware requirements

- A PiC900 programmable industrial computer with approximately 1.3K of data bytes free and approximately 11.5K of ladder code bytes free.

  NOTE: The number of free bytes can be checked in the Download complete box in PiCPro. The box appears after you download a ladder. An example is shown on the right.

```
Download complete:

 * Memory Usage *
  69 of 8k data bits
2465 of 32k data bytes
 220 ladder code bytes
27392 total code bytes

Press any key to continue
```

- A serial port (either USER PORT 2 on the PiC900 CPU or one of four serial ports on a serial communications module.)
- A serial cable to connect the PiC900 to the remote device.

Cable connections

The pinouts for the various Modbus communications connections are shown below. Choose the one for your system.

PiC900 to an operator interface

PiC900 CPU (1 O-pin screw terminal)			to an operator interface (25-pin female)	
GND	8	———	**7**	GND
RECV	9	———	2	TRANS
TRANS	10	———	**3**	RECV

PiC900 to a PC

PiC900 CPU (1 O-pin screw terminal)			to a PC (9-pin female)	
GND	8	———	**5**	GND
RECV	9	———	**3**	TRANS
TRANS	10	———	**2**	RECV

Software requirements

- Modbus ASFB software
- PiCPro Version 4.1 or higher
- LDOMERGE software (Optional software that allows you to merge ladders.)

Compatibility

When the PiC900 is used as a Modbus slave, 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 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 PiC900 is communicating with must also support these commands. If the PiC900 receives a command it does not support or recognize, it will return an error response to the sender.

When the PiC900 is used as a Modbus master, 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 PiC900 is communicating with must support the commands the PiC900 is going to generate. If the PiC900 sends a command the other device does not recognize, it will respond with an error response.

Message Addressing

The addressing between the PiC900 and Modbus 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)

1.3 Installation

The Modbus software disk contains the files listed below. The Main group includes the ASFB library (LIB), source ladders for the ASFBs (LDOs), and remark files containing the comments in the source ladders (.REMs). The Example group includes the example LDO and REM files. The Auxiliary group contains the LIB, LDOs, and REMs for the UDFBs used in the source ladders for the ASFBs. NOTE: It should never be necessary for you to access any of the files in the Auxiliary group. The LIB is required in order for the ASFB to work and the LDOs allow you to view the source ladders when troubleshooting if necessary.

Follow the guidelines found at the beginning of the manual. Always make a back up copy of the disk and store the original in a safe place. The recommended destination' directory for each file is listed in the last column.

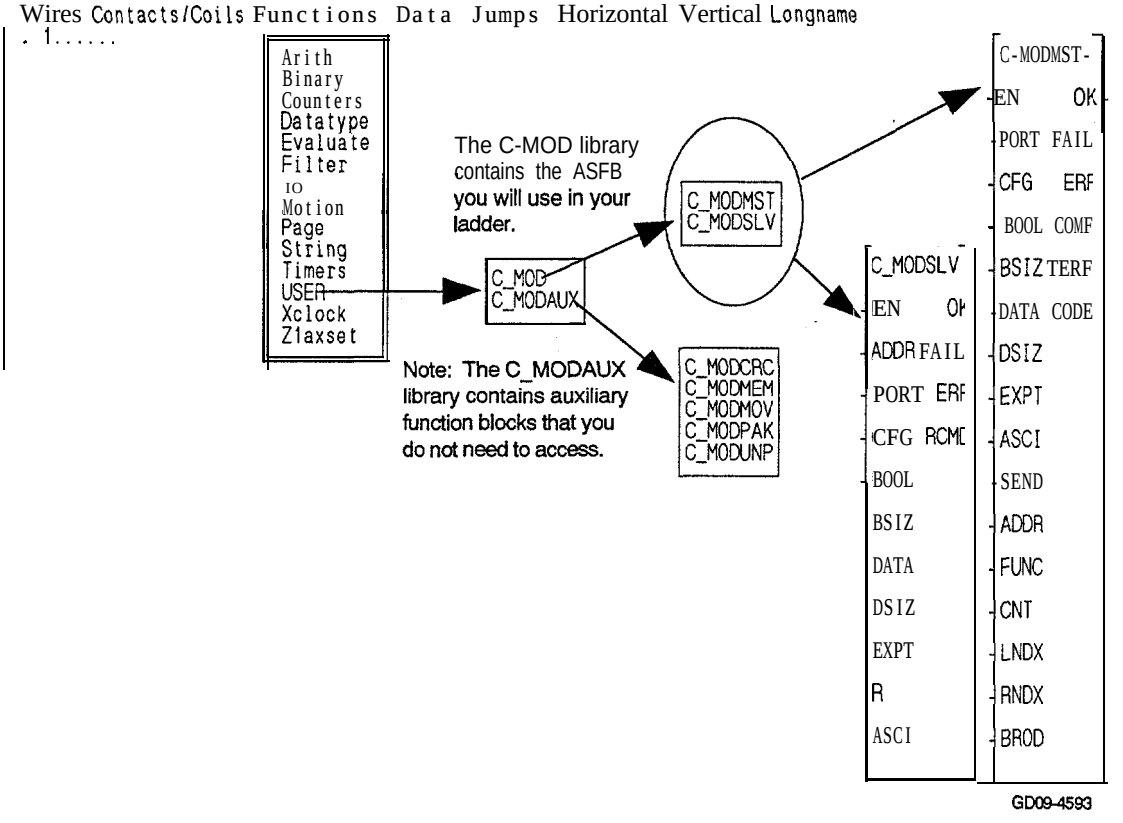
Group	File	Description	Directory
Main	C_MOD.LIB	The library containing the application specific function block used to perform Modbus communications.	ASFB
	C_MODSLV.LDO	The source ladder for the transceiver function block.	ASFB
	C_MODSLV.REM	The remark file for the source ladder	ASFB
	C_MODMST.LDO	The source ladder for the C_MODMST function block.	ASFB
	C_MODMST.REM	The remark file for the source ladder	ASFB
Example	C_MASTEX.LDO C_MASTEX.REM	The example for Modbus master LDO from which you can build a new application LDO or to which you can merge an existing one.	Working
	C_MODEX.LDO C_MODEX.REM	The example for Modbus slave LDO from which you can build a new application LDO or to which you can merge an existing one.	Working
Auxiliary	C_MODAUX.LIB	The library that holds all the function blocks used in the source ladder for the ASFB.	ASFB
	C_MODCRC.LDO C_MODCRC.REM	Source ladder Remark file	ASFB ASFB
	C_MODMEM.LDO C_MODMEM.REM	Source ladder Remark file	ASFB ASFB
	C_MODMOV.LDO C_MODMOV.REM	Source ladder Remark file	ASFB ASFB
	C_MODPAK.LDO C_MODPAK.REM	Source ladder Remark file	ASFB ASFB
	C_MODUNP.LDO C_MODUNP.REM	Source ladder Remark file	ASFB ASFB

NOTE: The libraries containing the ASFBs and their source ladders must always be in the same directory. If they are not in the same directory, PiCPro will not be able to find the source ladder module when the View User function option is selected.

1.4 Modbus Function Blocks

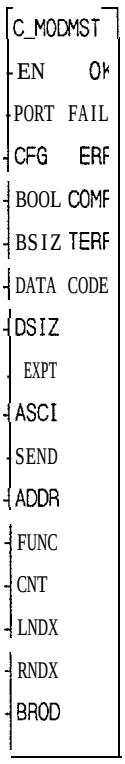
The function block for the Modbus interface is described in this section. When PiCPro is running, you can find the Modbus function blocks by choosing the Function menu, then USER, then C-MOD as shown in Figure 2.

Figure 2. Location of ASFBs in PiCPro



C_MODMST
USER/C-MOD

Communi-cations_ Modbus master



Inputs:

EN (BOOL) - enables execution

PORT (STRING) - identifies the communication serial port

CFG (STRING) - configuration string for the port

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)

ASCI (BOOL) - selects ASCII mode if set; selects RTU mode if not set

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

ADDR (USINT) - address of the slave device to which messages will be sent (range from l-255)

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

BROD (BOOL) - set to send a broadcast frame

IMPORTANT

The C_MODMST function block cannot be used with the PiC911/912 or PiC90 CPUs. The serial port on the EC186 processor in these CPUs is not compatible with the modbus protocol for a master. These CPUs will work as slaves with the C_MODSLV function block.

Outputs: **OK** (BOOL) - execution completed without error

FAIL (BOOL) - initialization failed

ERR (INT) - 0 if initialization is successful; ≠ 0 if initialization is unsuccessful

RCMD (BOOL) - energized if a message is received

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

NOTE: C_MODMST function block cannot be used with the C_MODSLV function block on the same serial port.

The C_MODMST function block provides PiC900 communication capabilities on a Modbus network with the PiC900 acting as a Modbus master. The link to the network must be made through one of the PiC900 serial ports.

When this function is enabled, it will open the PiC900 serial port specified at the PORT input. This port will be configured based on the information specified at the CFG input. If the port configures properly, the OK output will energize and the system will be ready to generate requests as the Modbus master. If a problem occurs in the open or configuration process, the FAIL output will be energized and the OK will not be set. See Appendix B in the PiC900 software manual for the error codes at the ERR output.

To establish communications on the Modbus network, this function block is needed only once and should be enabled every scan.

Inputs

EN

The EN input is energized every scan to respond to a query over the Modbus 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.

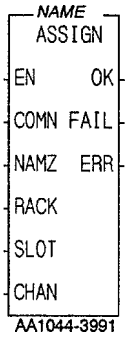
PORT

The PORT input specifies which serial port this function block will use to communicate over. Place a string type variable at this input that has been initialized with the name of the port that is to be used.

For example, if the PiC900 User port 2 is being used, initialize a string as:

USER:$OO

If one of the channels on the serial communications module is being used, the variable you enter at the NAMZ input of the ASSIGN function block is the variable you enter at the PORT input of the C_MODMST function block.



CFG

The string variable at the CFG input holds the initialized configuration string you will use.

If the RTU mode is chosen (see ASCI input), then the configuration string would typically be:

9600, E, 8, 1, N, $00

The port will be set up at 9600 baud, even parity, 8 data bits, no handshaking. See the OPEN function block description in the PiC900 Software Manual for more information about this configuration string.

If the ASCII mode is chosen (see ASCI input), then the configuration string would typically be :

9600, E, 7, 2, N, $00

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.

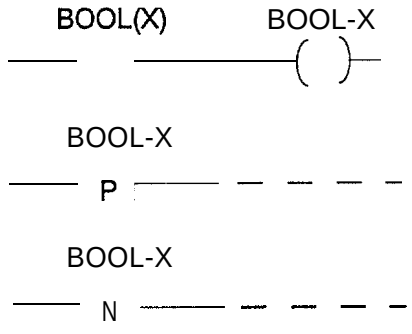
For example, if the array of boolean variables is called BOOL and a write request is made from register 00222, the PiC900 would send the data in BOOL(221).

The array size can range from 2 (O..l)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.

ASCI Controllers can be setup to communicate on Modbus networks using either of two transmission modes: ASCII or RTU. You select the desired mode at this ASCI input. If ASCI input is set, then the ASCII mode is in effect. If it is not set, then the RTU mode is in effect. More information on these two modes can be found in your Modbus manual.

NOTE: The mode and serial parameters must be the same for all devices on the network. The modes define the bit contents of message fields transmitted serially on the network. They determine how information will be packed into the message fields and decoded.

The ASCII Mode

When controllers are setup to communicate on a network using ASCII mode, each 8-bit byte in a message is sent as two ASCII characters.

The format for each byte in the ASCII mode is:

Coding System	Hexadecimal, ASCII characters O-9, A-F One hexadecimal character contained in each ASCII character of the message
Bits per Byte	1 start bit 7 data bits, least significant bit sent first 1 bit for even/odd parity; no bit for no parity 1 stop bit if parity is used; 2 bits if no parity
Error Check Field	Longitudinal Redundancy Check (LRC)

In the ASCII mode, the message frame starts with a 'colon' (:) character (ASCII 3A hex) and ends with a 'carriage return - line feed' (CRLF) pair (ASCII OD and OA).

The allowable characters transmitted for all other fields are hexadecimal O-9, A-F. Networked devices monitor the network bus continuously for the 'colon' character. When one is received, each device decodes the next field (the address field) to find out if it is the addressed device.

Intervals of up to one second can elapse between characters within the message. If a greater interval occurs, the receiving device assumes an error has occurred. A typical message frame is shown below.

Start	Address	Function	Data	LRC check	End
1 character	2 characters	2 characters	*n* characters	2 characters	2 characters CRLF

The RTU Mode

When controllers are setup to communicate on a network using RTU mode, each 8-bit byte in a message contains two 4-bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII for the same baud rate. Each message must be transmitted in a continuous stream.

The format for each byte in RTU mode is:

Coding System: Hexadecimal, ASCII characters O-9, A-F
One hexadecimal character contained in each ASCII character of the message

Bits per Byte: 1 start bit
8 data bits, least significant bit sent first
1 bit for even/odd parity; no bit for no parity
1 stop bit if parity is used; 2 bits if no parity

Error Check Field: Cyclical Redundancy Check (CR.C-16)

In the RTU mode, the message frame starts with a silent interval of at least 3.5 character times. The first field then transmitted is the device address.

The allowable characters transmitted for all fields are hexadecimal O-9, A-F. Networked devices monitor the network bus continuously including during the silent intervals. When the first field is received, each device decodes it to find out if it is the addressed device.

Following the last transmitted character, a similar interval of at least 3.5 character times marks the end of the message. A new message can begin after this interval.

The entire message frame must be transmitted as a continuous stream. If a silent interval of more than 1.5 character times occurs before completion of the frame, the receiving device flushes the incomplete message and assumes that the next byte will be the address field of a new message.

Similarly, if a new message begins earlier that 3.5 character times following a previous message, the receiving device will consider it a continuation of the previous message. This will set an error, as the value in the final CRC field will not be valid for the combined messages. A typical message frame is shown below.

Start	Address	Function	Data	LRC check	End
T1-T2-T3-T4	8 bits	8 bits	*n* * 8 bits	16 bits	T1-T2-T3-T4

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

ADDR The ADDR input specifies the address of the slave device to which messages will be sent.

The range of numbers that this input will accept is 1 to 255 (decimal).

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.

BROD The BROD input is set by you when a broadcast type frame is sent.

outputs

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

FAIL The FAIL 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.

ERR The ERR 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.

RCMD The RCMD output is a one-shot output that energizes when the PiC900 has received a query from the master. The information describing the nature of the response will be placed in the data structure placed at the R input of this function block.

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 slave and reported at the CODE output in the form of exception responses. For a complete explanation of these errors, see your Modicon Modbus Protocol Reference Guide, Appendix A: Exception Responses.

TERW 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 query data field is not an allowable value for the slave.
4	Slave device failure	An unrecoverable error occurred while the slave was attempting to perform the requested action.
5	Acknowledge	The slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the master.
6	Slave device busy	The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free.
7	NA	
8	Memory parity error	The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device.

The table below contains the error codes that are detected locally the PiC900 and reported at the CODE output.

TERR Code	Name	Description
100	CRC error	The message from the slave device has failed CRC.
101	Address error	The message from the slave device has an unexpected value in the address field.
102	Function error	The message from the slave device has an unexpected value in the function field.
103	Time-out error	No response message has been received for the slave device with 350 ms.
104	Invalid function	The value at the FUNC input is invalid. No functions above function 16 are currently supported.
105	Invalid function	The value at the FUNC input is invalid. The function number is not supported.
106	Broadcast error	The function requested will not support broadcast mode as defined by Modbus.
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.

Modbus Master

As a Modbus master, the PiC900 will query a remote device for integer and boolean data. The C_MODMST function block described above is entered in your application program one time. It is responsible for all communications to and from the PiC900 for Modbus support. Enable it every scan. Each input must have the appropriate variable attached to it.

All data being sent to or retrieved from the PiC900 will have a function code number associated with it. Although this number has no direct equivalent in the PiC900, 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 master example LDO

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

If you want to add the C_MASTEX.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 C_MODMST function block already entered. You can modify this to fit your application.

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

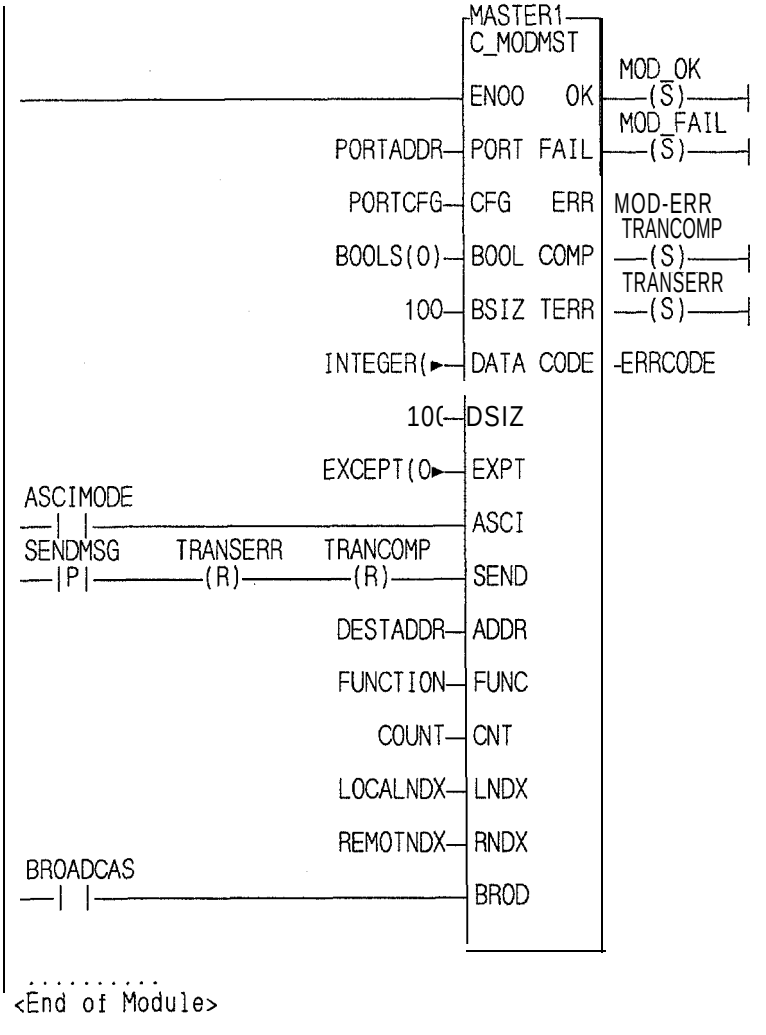
```
Workstation  Processor  Module Declarations Network  Element  View
...1.......
this function enables the PiC900 to be a Modbus Master over an RS232
connect ion.
It is configured as: Station 1

                     RTU mode at 9600 baud, Even Parity, 8 data bits, 1 stop bit
                     100 booleans have been defined (USER ADJUSTABLE)
                     100 integers have been defined (USER ADJUSTABLE)
```

This function will OPEN and CONFIGURE the serial port specified at the PORT input on the rising edge of the ENOO input. This input should be enabled all the time, or as long as communication is required (typically all the time).

HOW TO START A TRANSACTION WITH A SLAVE STATION:

Each time the SENDMSG contact transition from off to on the C_MODMST function will use the data at the function's inputs to build a frame to be sent to a slave station. Once sent it will wait up to 350 milliseconds for a response. During this time period, no additional requests will be processed. Once the response data has been received and decoded the COMP output will be energized to indicate that the transaction is complete. If a problem occurs during the transaction, and the transfer fails, the FAIL output will be energized and an error code will appear at the ERR output to indicate what the problem is.

NOTE: If this Modbus Master ASFB is going to be used over the USER port on the CPU module and the port needs to be configured for any setup other than 8 data bits, No parity, 1 Stop bit, it will be necessary to have version 11 or higher system eproms in the CPU. If the Serial Communications Module is being used this is not necessary.



The software declarations table for the example is shown below. You can modify it to fit your application.

Name	Type	I/O Pt. = Init. Val.	Long Name
MASTER1	<fb>C_MODMST		Modbus\Master\Driver
BOOLS	BOOL(0..99)		Booleans: \On Modbus\00001 to 00999
INTEGER	INT(0..99)		Integers: \On Modbus\40001 to 40999
PORTADDR	STRING[15]	USER:$00	Serial\Port\Name\
PORTCFG	STRING[15]	9600,E,8,1►	Serial\Port\Configure\String
MOD-OK	BOOL		Function\Initialize\OK
MOD_FAIL	BOOL		Function\Init\Failure
MOD ERR	INT		Function\Init\Error\Code
EXCEPT	BOOL(0..7)		User\Definable\Exception\Status
SENDMSG	BOOL		Start\a Modus\Transaction
DESTADDR	USINT		Destination\Address\of Slave\Device
FUNCTION	USINT		Modbus\Function\Number
COUNT	UINT		Number of\Words or\Bools to\transfer
LOCALNDX	UINT		Local\Index into\data area
REMOTNDX	UINT		Remote\(slave)\index into data area
BROADCAS	BOOL		Set to\enable\BROADCAST type mess
ASCIMODE	BOOL		Set to\Enable\ASCII mode
TRANCOMP	BOOL		Energizes\When a\transfer completes
TRANSERR	BOOL		Energizes\When a\transfer fails
ERRCODE	INT		Failed\Transfer\Error\Code
end-table	void		

Alt-M modifies attribute==Press F10 to exit==Alt-E enters field=Bottom

C_MODMST function block setup

The steps for setting up the C_MODMST function block allowing the PiC900 to function as a Modbus master follows.

1. Determine the address for the slave device and enter it at the ADDR input.
2. Determine which serial port is going to be used for the Modbus communications. If the USER port is going to be used, initialize a string type variable as follows:

   PORTADDR STRING(lO) "USER:$00"

3. Determine the proper communications configuration for the serial port. Then assign a string type variable an Initial Value that when placed at the PORTCFG input will configure the communications channel. See the CFGZ input on the CONFIG function in the PiC900 Software Manual for more information on the configuration string.

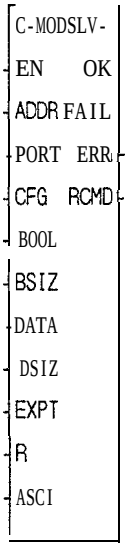
   ASCII Example: For 9600 baud, Even parity, 7 data bits, 1 stop bit,
   CONFIG STRING(15) "9600,E,7,2,N,$00"

   RTU Example: For 19200 baud, Even parity, 8 data bits, 2 stop bit
   CONFIG STRING(15) "19200,E,8,1,N,$00"

4. Determine how many booleans (bit type) will be needed for your application.
5. 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.
6. The size of the boolean array (BOOLS) must be entered in BSIZ(boolean size).
7. Determine how many integers will be needed for your application. The acceptable range is from 1 to 999.
8. 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.
9. The size of the integer array (INTEGER) must be entered in DSIZ (data size).
10. Determine whether ASCII or RTU mode will be used and set the ASCI input accordingly.

C_MODSLV
USER/C_MOD

Communi-cations Modbui slave



Inputs:

EN (BOOL) - enables execution

ADDR (USINT) - address for the PiC900 (range from l-255)

PORT (STRING) - identifies the communication serial port

CFG (STRING) - configuration string for the port

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)

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

ASCI (BOOL) - selects ASCII mode if set; selects RTU mode if not set

outputs:

OK (BOOL) - execution completed without error

FAIL (BOOL) - initialization failed

ERR (INT) - 0 if initialization is successful; ≠ 0

if initialization is unsuccessful

RCMD (BOOL) - energized if a message is received

The C_MODSLV function block provides PiC900 communication capabilities on a Modbus network. The link to the network must be made through one of the PiC900 serial ports.

When this function is enabled, it will open the PiC900 serial port specified at the PORT input. This port will be configured based on the information specified at the CFG input. If the port configures properly, the OK output will energize and the system will be ready to respond to queries from the Modbus master. If a problem occurs in the open or configuration process, the FAIL output will be energized and the OK will not be set. See Appendix B in the PiC900 software manual for the error codes at the ERR output.

To establish communications on the Modbus network, this function block is needed only once and should be enabled every scan.

Inputs

EN

The EN input is energized every scan to respond to a query over the Modbus 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.

ADDR

The ADDR input specifies the address this PiC900 will be on the network. This input must have a unique number which represents the PiC900 address.

The range of numbers that this input will accept is 1 to 255 (decimal).

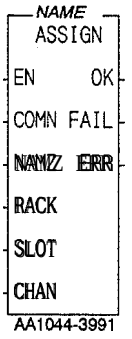
PORT

-The PORT input specifies which serial port this function block will use to communicate over. Place a string type variable at this input that has been initialized with the name of the port that is to be used.

For example, if the PiC900 User port 2 is being used, initialize a string as:

USER: $00

If one of the channels on the serial communications module is being used, the variable you enter at the NAMZ input of the ASSIGN function block is the variable you enter at the PORT input of the C_MODSLV function block.



CFG

The string variable at the CFG input holds the initialized configuration string you will use.

If the ASCII mode is chosen (see ASCI input), then the configuration string would typically be :

9600, E, 7, 2, N, $00

If the RTU mode is chosen (see ASCI input), then the configuration string would typically be:

9600, E, 8, 1, N, $00

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.

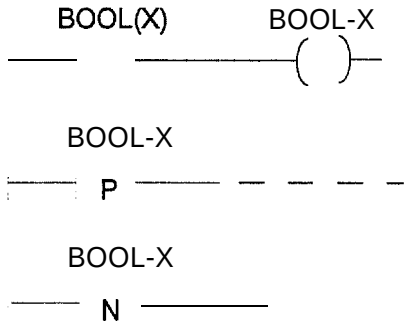
For example, if the array of booleans variable is called BOOL and a query is made for register 00222, the PiC900 would respond with the data in BOOL(221).

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 PiC900 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 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 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.

ASCI Controllers can be setup to communicate on Modbus networks using either of two transmission modes: ASCII or RTU. You select the desired mode at this ASCI input. If ASCI input is set, then the ASCII mode is in effect. If it is not set, then the RTU mode is in effect. More information on these two modes can be found in your Modbus manual.

NOTE: The mode and serial parameters must be the same for all devices on the network. The modes define the bit contents of message fields transmitted serially on the network. They determine how information will be packed into the message fields and decoded.

The ASCII Mode

When controllers are setup to communicate on a network using ASCII mode, each 8-bit byte in a message is sent as two ASCII characters.

The format for each byte in the ASCII mode is:

Coding System	Hexadecimal, ASCII characters O-9, A-F One hexadecimal character contained in each ASCII character of the message
Bits per Byte	1 start bit 7 data bits, least significant bit sent first 1 bit for even/odd parity; no bit for no parity 1 stop bit if parity is used; 2 bits if no parity
Error Check Field	Longitudinal Redundancy Check (LRC)

In the ASCII mode, the message frame starts with a 'colon' (:) character (ASCII 3A hex) and ends with a 'carriage return - line feed' (CRLF) pair (ASCII OD and OA).

The allowable characters transmitted for all other fields are hexadecimal O-9, A-F. Networked devices monitor the network bus continuously for the 'colon' character. When one is received, each device decodes the next field (the address field) to find out if it is the addressed device.

Intervals of up to one second can elapse between characters within the message. If a greater interval occurs, the receiving device assumes an error has occurred. A typical message frame is shown below.

Start	Address	Function	Data	LRC check	End
1 character	2 characters	2 characters	*n* characters	2 characters	2 characters CRLF

NOTE: The data field cannot exceed 128 bytes in length.

The RTU Mode

When controllers are setup to communicate on a network using RTU mode, each 8-bit byte in a message contains two 4-bit hexadecimal characters. The main advantage of this mode is that its greater character density allows better data throughput than ASCII for the same baud rate. Each message must be transmitted in a continuous stream.

The format for each byte in RTU mode is:

Coding System:	Hexadecimal, ASCII characters O-9, A-F One hexadecimal character contained in each ASCII character of the message
Bits per Byte:	1 start bit 8 data bits, least significant bit sent first 1 bit for even/odd parity; no bit for no parity 1 stop bit if parity is used; 2 bits if no parity
Error Check Field:	Cyclical Redundancy Check (CRC-16)

In the RTU mode, the message frame starts with a silent interval of at least 3.5 character times. The first field then transmitted is the device address.

The allowable characters transmitted for all fields are hexadecimal O-9, A-F. Networked devices monitor the network bus continuously including during the silent intervals. When the first field is received, each device decodes it to find out if it is the addressed device.

Following the last transmitted character, a similar interval of at least 3.5 character times marks the end of the message. a new message can begin after this interval.

The entire message frame must be transmitted as a continuous stream. If a silent interval of more than 1.5 character times occurs before completion of the frame, the receiving device flushes the incomplete message and assumes that the next byte will be the address field of a new message.

Similarly, if a new message begins earlier that 3.5 character times following a previous message, the receiving device will consider it a continuation of the previous message. This will set an error, as the value in the final CRC field will not be valid for the combined messages. Atypical message frame is shown below.

Start	Address	Function	Data	LRC check	End
T1-T2-T3-T4	8 bits	8 bits	*n* * 8 bits	16 bits	T1-T2-T3-T4

NOTE: The data field cannot exceed 128 bytes in length.

outputs

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

FAIL The FAIL 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.

ERR The ERR 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.

RCMD The RCMD output is a one-shot output that energizes when the PiC900 has received a query from the master. The information describing the nature of the query will be placed in the data structure placed at the R input of this function block.

Modbus Slave

As a Modbus slave, the PiC900 will receive and respond to Modbus functions from other devices but will not initiate any transfers. The C_MODSLV function block described above is entered in your application program one time. It is responsible for all communications to and from the PiC900 for Modbus support. Enable it every scan. Each input must have the appropriate variable attached to it.

All data being sent to or retrieved from the PiC900 will have a code number associated with it. Although this number has no direct equivalent in the PiC900, it is used to determine where to place or retrieve data.

The PiC900 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.

Mqdbus slave example LDO

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

If you want to add the C_MODEX.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 C_MODSLV function block already entered. You can modify this to fit your application.

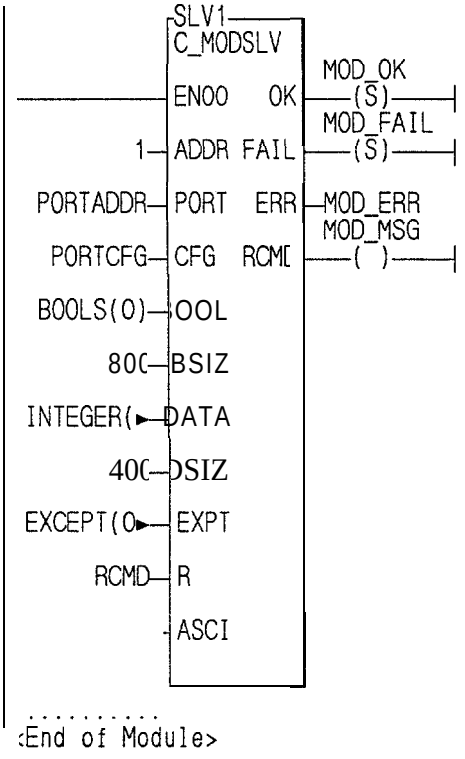
You can also insert the C_MODSLV function block into an existing ladder and enter the software declarations yourself.

C_MODEX.LDO

Workstation Processor Module Declarations Network Element View
Comment Editor: Insert Mode Line: 1 of 19 Col: 1
|...1..
this function enables the PiC900 to be a Modbus Slave over an RS232 connection.
It is configured as: Station 1
RTU mode at 9600 baud, Even Parity, 8 data bits, 1 stop bit,
No hardware handshaking
800 booleans have been defined (USER ADJUSTABLE)
400 integers have been defined (USER ADJUSTABLE)

Over Modbus the Integer array maps into register: 40001 to 40400 and the Boolean array is accessed over Modbus with bit numbers: 00001 to 00800. Any requests for data to/from Ixxxx, 2xxxx, or 3xxxx will cause an error response to be returned from the PiC900 Modbus Slave.

NOTE: If this Modbus Slave ASFB is going to be used over the USER port on the CPU module and the port needs to be configured for any setup other than 8 data bits, No parity, 1 Stop bit, it will be necessary to have version 11 or later system EPROMs in the CPU. If the Serial Communications Module, is being used this is not necessary.



The software declarations table for the example is shown below. You can modify it to fit your application.

Name	Type	I/O Pt.	Init. Val.	Long Name==Top
SLV1	<fb>C_MODSLV			Modbus\Slave\Driver
BOOLS	BOOL(0..799)			Booleans:\On Modbus\00001 to 00999
INTEGER	INT(0..399)		...ARRAY...	Integers:\On Modbus\40001 to 40999
PORTADDR	STRING[15]		USER:$00	Serial\Port\Name
PORTCFG	STRING[15]		9600,E,8,1,N,$00	Serial\Port\Configure\String
RCMD	STRUCT			Received\Commands\Information\Structure
.ADDRESS	USINT			Address of\Incomming\Command\Frame
.FUNCTION	USINT			Function\Number of Incoming\Command
	END_STRUCT			
MOD_OK	BOOL			
MOD_FAIL	BOOL			
MOD_ERR	INT			
MOD_MSG	BOOL			
EXCEPT	BOOL(0..7)			User\Definable\Exception\Status
end_table	void			

Alt-M modifies attribute=Press F10 to exit=Alt-E enters field=Bottom

C_MODSLV function block setup The steps for setting up the C_MODSLV function block allowing the PiC900 to function as a Modbus slave follows.

1. Determine the address for the PiC900 and enter it at the ADDR input.
2. Determine which serial port is going to be used for the Modbus communications. If the USER port is going to be used, initialize a string type variable as follows:

   PORTADDR STRING(lO) "USER: $00"
3. Determine the proper communications configuration for the serial port. Then assign a string type variable an Initial Value that when placed at the PORTCFG input will configure the communications channel. See the CFGZ input on the CONFIG function in the PiC900 Software Manual for more information on the configuration string.

   ASCII Example: For 9600 baud, Even parity, 7 data bits, 1 stop bit,
   CONFIG STRING(15) "9600,E,7,2,N,$00"

   RTU Example: For 19200 baud, Even parity, 8 data bits, 2 stop bit
   CONFIG STRING(15) "19200,E,8,1,N,$00"
4. Determine how many booleans (bit type) will be needed for your application.
5. 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.
6. The size of the boolean array (BOOLS) must be entered in BSIZ(boolean size).
7. Determine how many integers will be needed for your application. The acceptable range is from 1 to 999.
8. 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.
9. The size of the integer array (INTEGER) must be entered in DSIZ (data size).
10. Determine whether ASCII or RTU mode will be used and set the ASCI input accordingly.

Index

A

ADDR 18, 27
addressing 9
ASCI 15, 29
ASCII mode 16, 29
ASFBs 1
  guidelines 1
  revising 2, 3
  using 4

B

BOOL 15, 27
BROD 18
BSIZ* 15, 27

C

cables 8
CFG 15, 27
CNT 18
CODE 19
codes
  Modbus function 9
  Modbus master function 18, 20, 21
  Modbus slave function 28
COMP 19
compatibility 9
configurations 5
CPUs 13
C_MODMST function block 7, 13
  inputs 14
  outputs 19
C_MODSLV function block 7, 26
  inputs 27
  outputs 31

D

DATA 15, 28
directories 11
directory 1
DSIZ* 15, 28

E

EN 2, 14, 27
EPROM
  version 32
ERR 19, 31
example LDO
  master 22
  slave 32
EXAMPLES
  directory 1
EXPT 15, 28

F

FAIL 19, 31
files 11
FUNC 18
function blocks
  location 12
function codes
  Modbus 9
  Modbus master 18, 20, 21
  Modbus slave 28

G

guidelines
  ASFBs 1

H

hardware requirements 8

I

installation 1, 11
interface devices 6

L

ladder
  source 2
LDO
  master example 22
  slave example 32
LDO files 1
LIB files 1
LNDX 18

M

Modbus 5
  function codes 9
  master example 22
  master function codes 18, 20, 21
  protocol 5, 6
  slave example 32
  slave function codes 28
mode
  ASCII 16, 29
  master 21
    setup 25
  RTU 16, 30
  slave 31
    setup 34

0

OK 19, 31

P

pinouts 8
PORT 14, 27
ports 5, 8

Q

Query 7
query response cycle 6

R

R 28
    structure 28
RCMD 19, 31
Response 7
revising ASFBs 2, 3
RNDX 18
RQ 2
RTU mode 16, 30

S

SEND 18
software declarations 24, 33
software requirements 9
source ladder 2

T

TERR 19
transitionals 16, 28

V

version numbers 2

W

wiring 8