MODBUS INTEGRATION KIT FOR PAC CONTROL USER'S GUIDE

Form 2009-150522—May 2015

OPTO 22
Automation made simple.

43044 Business Park Drive • Temecula • CA 92590-3614
Phone: 800-321-OPTO (6786) or 951-695-3000
Fax: 800-832-OPTO (6786) or 951-695-2712
www.opto22.com

Product Support Services
800-TEK-OPTO (835-6786) or 951-695-3080
Fax: 951-695-3017
Email: support@opto22.com
Web: support.opto22.com

Modbus Integration Kit for PAC Control User's Guide
Form 2009-150522—May 2015

The information in this manual has been checked carefully and is believed to be accurate; however, Opto 22 assumes no responsibility for possible inaccuracies or omissions. Specifications are subject to change without notice.

Opto 22 warrants all of its products to be free from defects in material or workmanship for 30 months from the manufacturing date code. This warranty is limited to the original cost of the unit only and does not cover installation, labor, or any other contingent costs. Opto 22 I/O modules and solid-state relays with date codes of 1/96 or newer are guaranteed for life. This lifetime warranty excludes reed relay, SNAP serial communication modules, SNAP PID modules, and modules that contain mechanical contacts or switches. Opto 22 does not warrant any product, components, or parts not manufactured by Opto 22; for these items, the warranty from the original manufacturer applies. These products include, but are not limited to, OptoTerminal-G70, OptoTerminal-G75, and Sony Ericsson GT-48; see the product data sheet for specific warranty information. Refer to Opto 22 form number 1042 for complete warranty information.

Opto 22
Automation Made Simple.

Table of Contents

1: Getting Started

The Modbus Integration Kit for PAC Control (part # PAC-INT-MB) allows Opto 22 hardware controllers using PAC Control to communicate using the Modbus Serial RTU, Modbus Serial ASCII or Modbus/TCP protocol. It also allows a SoftPAC software-based controller to communicate using Modbus/TCP.

This kit replaces the Modbus/TCP Integration Kit for PAC Control (part # PAC-INT-MBTCP) and the Modbus Serial Integration Kit for PAC Control (part # PAC-INT-MBSER).

The Integration Kit contains:

- A set of PAC Control master subroutines that are added to a strategy to enable an Opto 22 controller to communicate as a Modbus master
- An example Modbus Master chart using the subroutines
- A PAC Control slave subroutine that is added to a strategy to enable an Opto 22 controller to communicate as a Modbus slave
- An example Modbus Slave chart using the subroutine
- A demo data chart that can be used to load data and data types for testing
- Example initialization files to load data types in the data type tables

Both the master and slave subroutines transmit message strings as specified in the *Modbus Application Protocol Specification v1.1a*, the *Modbus Messaging on TCP/IP Implementation Guide v1.0a*, and the *Modbus Over Serial Line Specification & Implementation Guide v1.0*. The guides are available on the web at www.modbus.org.

The master subroutines and slave subroutine transmit and receive messages using Modbus standard holding and input registers, and input and coil numbers. You store or retrieve the desired information using PAC Control numeric tables.

This manual assumes that you fully understand how to use PAC Control, Modbus protocol, and the Modbus device to be used.

This chapter includes the following topics:

Understanding Modbus Protocol

This toolkit assumes that you are knowledgeable about Modbus protocol addressing, and register, coil, and input numbering. Even for those who are experienced, we highly recommend reading the *Modicon Modbus Protocol Reference Guide*, which is available at this link:

http://www.modbus.org/docs/PI_MBUS_300.pdf

We especially recommend the following two sections of the Modbus guide:

- *Chapter 2: Data and Control Functions* of the Modicon guide explains the key subtleties of register, coil, and input numbering/naming as opposed to register, coil, and input addressing. This helps eliminate a common point of confusion, even for those who are experienced with Modbus protocol.
- *Appendix A: Exception Responses* of the Modicon guide discusses the possible exception codes a Modbus device can reply with and what the codes mean. This helps when diagnosing communication problems.

What is Required

Before including the subroutines in your strategy, you will need:

- A PC running PAC Control software and the Modbus Integration Kit
- PAC Project (Basic or Pro) 9.1 or newer
- SNAP PAC controller with firmware version 9.1 or newer, or SoftPAC software-based controller, version 9.1 or newer

Modbus Functions Supported

The following Modbus function codes are supported by PAC Control subroutines.

Modbus Function Code	Name	PAC Control Subroutine
01	Read Coils	PACModbusMaster01_Read_Coil_Status
02	Read Discrete Inputs	PACModbusMaster02_Read_Input_Status
03	Read Holding Registers	PACModbusMaster03_Read_Holding_Registers

Modbus Function Code	Name	PAC Control Subroutine
04	Read Input Registers	PACModbusMaster04_Read_Input_Registers
05	Force Single Coil	PACModbusMaster05_Force_Single_Coil
06	Preset Single Register	PACModbusMaster06_Preset_Single_Register
08	Diagnostics	PACModbusMaster08_Diagnostics (slave subroutine supports subfunction 00)
15	Force Multiple Coils	PACModbusMaster15_Force_Multple_Coils
16	Preset Multiple Registers	PACModbusMaster16_Preset_Multiple_Holding_Registers
17	Report Slave ID	PACModbusMaster17_Report_Slave_ID (Master only)
22	Mask Write 4X Register	PACModbusMaster22_Mask_Write_4X_Register
23	Read/Write 4X Registers	PACModbusMaster23_Read_Write_4X_Registers

Data Types Supported in the Input and Holding Registers

The following data types are supported for input and holding registers:

Value	Data Type
0	16-bit unsigned (Modbus standard and default)
1	16-bit signed
2	Floating point (Uses two registers)
3	Floating point (Swapped. Uses two registers.)
4	32-bit signed (Uses 2 registers)
5	32-bit signed (Swapped. Uses 2 registers.)

NOTE: Most Modbus devices store 32-bit data values in two consecutive 16-bit registers. However, Opto 22 SNAP PAC controllers store 32-bit data values in individual table elements because tables support full 32-bit data values.

When accessing 32-bit data in most Modbus protocol devices, the data is stored in two consecutive 16-bit registers. Data that is 16 bits is sometimes referred to as a *word*, just as 8-bit data is referred to as a *byte*.

Modbus protocol messages treat each 16-bit register, or word, as 2 bytes with the high order byte coming before the low order byte. However, when device manufactures started supporting 32-bit data in standard Modbus messages, there was not a standard regarding the order of 16-bit registers (words) in the message when it comes to 32-bit data. Because of this, some Modbus protocol devices put the bytes of the high order register (word) first, and the low order register (word) second. Other devices do just the opposite.

In order to provide flexibility when communicating with both types of devices, the PAC Modbus toolkit supports *word swapping* when using 32-bit data types. We normally send the high-order

word before the low order word in the message. If the data is not correct, it may be because the word order is backwards compared to your Modbus device. If this is the case, you can simply change the word order by selecting the appropriate swapped data type.

For example, if you are dealing with floating point data using Data Type 2, you could try Data Type 3 in order to swap the order of the 16-bit words in the message.

Using the Data Type Integer 32 Tables

The example uses the following tables to designate the data type for each of the input and holding registers:

- ntMB_Holding_Register_4X_Data_TypeM1 for master holding registers
- ntMB_Input_Register_3X_Data_TypeM1 for master input registers
- ntMB_Holding_Register_4X_Data_TypeS2 for slave holding registers
- ntMB_Input_Register_3X_Data_TypeS2 for slave input registers

Index 0 of these tables sets the way the data type tables are used by the subroutines.

Master Subroutines

Index 0 = 0
Use one data type. The data type used is in index 1. The length of the table must be 2 or more. The register offset is not used for the data type table.

Index 0 = 1
The data and the data type are in the same index for each table. If the register offset is set to -7000 and the register to read is 7001 then the data will be stored in index 1 of the Modbus data table based on the data type at index 1 of the data type table. The length of the tables should be (start register + quantity + register offset + 1). The register offset is used for the data type table.

Index 0 = 2
The data type index and register number are the same. If you read register 8001 it will use the data type at index 8001 of the data type table. The length of the table should be (start register + quantity + 1). The register offset is not used for the data type table.

Slave Subroutines

Index 0 = 0
Use one data type. The data type used is in index 1. The length of the table should be (highest register number + register offset + 1). The register offset is used for the data type table.

Index 0 = 1
The data and the data type are in the same index for each table. If the register offset is set to -7000 and the register to read is 7001 then the data will be stored in index 1 of the Modbus data table based on the data type at index 1 of the data type table. The length of the tables should be (start register + quantity + register offset + 1). The register offset is used for the data type table.

Index 0 = 2
Not supported

Protocols Supported

The following protocols are supported:

Comm Mode	Name
0	Serial RTU protocol
1	Serial ASCII protocol
2	Modbus/TCP

Important Note for Users Upgrading

This Modbus integration kit replaces the Modbus/TCP Integration Kit for PAC Control (part # PAC-INT-MBTCP) and the Modbus Serial Integration Kit for PAC Control (part # PAC-INT-MBSER).

This kit has the following characteristics:

- The master subroutines support Modbus Serial RTU, Modbus Serial ASCII, or Modbus/TCP.
- The slave subroutine can support from 1 – 4 ports. The ports can be configured as a mix of Modbus RTU, Modbus ASCII, or Modbus/TCP.
- The subroutines support six data types for input and holding register data:
  - 16-bit unsigned
  - 16-bit signed
  - floating pt
  - floating pt swapped
  - 32-bit signed integer
  - 32-bit signed integer swapped

  A data type can be set for each register. This allows the reading and writing of multiple registers with different data types.
- The subroutines support an offset variable when an offset is needed between the registers and index of the data tables.
- The subroutines support a register mode variable used to make the system compatible with Modbus devices that store floats and 32-bit integers in one register and those that use two registers.
- The subroutines have a string table that returns the status of communications, the transmit string and the receive string.
- The subroutines have an integer 32 table used to return any error codes. The master subroutines also have the "OK" and "failed count" status for each packet sent.

Installing the Integration Kit

To install the integration kit on your computer, unzip the PACModbus.zip file to your C: drive. The expanded files will be placed automatically in the PACModbus directory.

Changing the Modbus Slave TCP Port

An Opto 22 SNAP PAC controller has a built-in Modbus/TCP slave capability, which only provides access to the I/O portions of the memory map (see Note below). The slave toolkit, on the other hand, provides access to both the strategy, portions of the memory map, and the I/O. Therefore, in order for the slave toolkit to work, you must disable the built-in Modbus slave functionality by changing the Modbus slave TCP port to a value of 0 (zero) as follows.

NOTE: I/O access only applies to rack-mounted controllers, such as the SNAP-PAC-R1 or R2.

1. Open PAC Manager.
2. In the PAC Manager main window, click the Inspect button.
3. In the Device Name field, type the IP address for the SNAP PAC controller (or choose it from the drop-down list).
4. Click Communications and choose Network Security.
5. Under PORTS, click the Value field for Modbus.
6. Change the value to 0 (zero). Click Apply.
7. Click Status Write.
8. Under Operation Commands, choose Send configuration to flash. Click Send Command.
9. Under Operation Commands again, choose Restart Device from powerup. Click Send Command.

For more details on using PAC Manager, see the *PAC Manager User's Guide*, form 1704.

Running the Example Strategies

The kit includes an example strategy to demonstrate how to use the subroutines in a PAC Control strategy. Both the slave and master charts have user setup blocks where user data is entered and initialization blocks that set the needed parameters before a block that calls the subroutine. Before importing the subroutines into your own strategy, it is recommended that you first run the example strategy to see how to use the subroutines in a PAC Control strategy, especially the configuration of variables.

To run the example strategy, start PAC Control, and then open the strategy file, PACModbus.idb.

Using Communication Handles

Be sure to use a separate TCP communication handle for each chart that uses the Modbus subroutines.

In PAC Control, if two charts were to run simultaneously while sharing an open communication handle, each chart would be able to read and write data from the communication handle as if the other running chart didn't exist. Because these reads and writes are not synchronized between the charts, it is possible for one chart to read the other chart's data.

2: Using the Master Subroutines

This chapter includes the following topics:

Adding the Master Subroutines

The Modbus master subroutines allow an Opto 22 controller to function as a Modbus master device. Each master subroutine in the integration kit supports one Modbus function code and can function independently of the other subroutines. Therefore, you need only use the subroutines for the Modbus functions that you require. For more information about subroutines, see form 1700, the *PAC Control User's Guide*.

When you decide which subroutines you need, include them in your strategy as follows:

1. Start PAC Control in Configure Mode and open the strategy that you intend to use with the integration kit.
2. Select Configure > Subroutines Included to open the Subroutine Files dialog.
3. Click the Add button and use the browser to select each subroutine file (.ISB extension) you wish to include in your strategy from the folder PACModbus\Subs.
4. Click OK.

The subroutines appear in the Subroutines Included folder and are ready to be used in your strategy.

To copy the Modbus Master chart to your strategy, you must export the chart Modbus_Master_M1 as a PAC Control chart export file (.cxf file) and then import it into your strategy. For more information, see form 1700, the *PAC Control User's Guide*.

This chart has the user setup parameters and subroutine parameters needed to make the subroutines function.

Start the Modbus Master chart in the Powerup chart of your strategy.

Data Tables Used By Master Subroutines

These are the names used for Modbus data tables by the master subroutines:

- ntMB_Coils_0XM1 (integer 32 table)
- ntMB_Inputs_1XM1 (integer 32 table)
- ntMB_Input_Register_3X_IntM1 (integer 32 table)
- ntMB_Holding_Register_4X_IntM1 (integer 32 table)
- ftMB_Input_Registers_3X_FloatM1 (float table)
- ftMB_Holding_Registers_4X_FloatM1 (float table)

You can name the tables however you want because the names of the tables are passed to the subroutines.

You may need to adjust the lengths of these tables to accommodate the amount of Modbus data and the register, coil, and input numbers expected to be accessed by the Modbus master device.

Use strategy logic to populate data in or retrieve data from these tables from the most recent Modbus message received.

NOTE: An integer table is used for data types 0, 1, 4, and 5. A float table is used for data types 2 and 3.

See "Important Note for Users Upgrading" on page 5.

Operation Mode Details for Master Subroutines

The Operation Mode feature uses the following two variables for added flexibility:

- Master Register Mode and Master Register Offset for the master
- Slave_Register_Mode1- 4 and RegisterOffset for the slave

Using Master Register Mode

The Master Register Mode uses an Integer 32 Variable. It is only used for data type 2, 3, 4 or 5.

For read functions:

Master Register Mode = False: It will read the quantity of registers and write every other index of the Modbus table.

Master Register Mode = True: It will read the quantity of registers and write consecutive indexes of the Modbus table.

For write functions:

Master Register Mode = False: It will read every other index and write using two registers per value.

Master Register Mode = True: It will read consecutive indexes and write using two registers per value.

Using Master Register Offset

The Master Register Off set is an Integer 32 Variable used to offset the data from the read register or write register.

Example 1: The first read register is 7001 and the offset is -7000

The data will be stored starting at index 1 of the Modbus table in the strategy.

Example 2: The first write register is 7001 and the offset is -7000

The data to write will start at index 1 of the Modbus table in the strategy

Using Register Offset

The Register Offset is an Integer 32 Table. It is used to simulate a device with high number register without needing to have Modbus tables with high lengths.

The index assignments are as follows:

- Index 0 = Not used
- Index 1 = Offset for function 1
- Index 2 = Offset for Function 2
- Index 3 = Offset for Function 3
- Index 4 = Offset for Function 4
- Index 15 = Offset for Function 15
- Index 16 = Offset for Function 16
- Index 23 = Offset for Function 23 Read
- Index 24 = Offset for Function 23 Write

Examples Using the Master Register Parameter

Case 1

Subroutines Related to Coils and Inputs

For Coils and Inputs, there is always a one-to-one correlation between the number of coils or inputs in the slave and the number of table elements used for the data in the master.

The Master Register parameter only affects the starting coil (or input) number used in the master data tables. This corresponds to the data table index number.

Example 1A: When the Master Register Offset parameter is 0, the subroutine uses the value of the Start Coil (or Start Input) parameter as the starting Coil (or Input) number in the slave and also as the starting table index in the master.

Parameter	Value
Master Register Offset	0

Parameter	Value
Slave Register	19
Qty of Coils	3

Master Table Index		Slave Coil Number
19	←	19
20	←	20
21	←	21

Example 1B: When the Master Register Offset parameter is non zero, the subroutine uses the value of the Start Coil (or Start Input) parameter as the starting Coil (or Input) number in the Slave and it uses the value of the starting Coil (or Input) number + the Master Register Offset as the starting table index in the master.

Parameter	Value
Master Register Offset	-100
Slave Register	101
Qty of Inputs	3

Master Table Index		Slave input Number
1	←	101
2	←	102
3	←	103

Case 2

Subroutines related to Input Registers and Holding Registers when the Data Type Parameter is 0 or 1 (both 16-bit data types)

For Input Registers and Holding Registers, when using 16-bit data types, there is always a one-to-one correlation between the number or registers in the slave and the number of table elements used for the data in the master.

The Master Register parameter only affects the starting register number used in the master data tables. This corresponds to the data table index number.

Example 2A: When the Master Register Offset parameter is 0, the subroutine uses the value of the Start Register parameter as the starting register number in the slave and also as the starting table index in the master.

Parameter	Value	Comment
Master Register Offset	0	
Slave Register	7001	
Qty of H Registers	3	
Data Type	0 or 1	data type is 16-bit

Master Table Index		Slave Register Number
7001	→	7001
7002	→	7002
7003	→	7003

Example 2B: When the Master Register Offset parameter is non zero, the subroutine uses the value of the Start Register parameter as the starting register number in the Slave and it uses the value of the starting register number + Master Register Offset as the starting table index in the master.

Parameter	Value	Comment
Master Register Offset	-6902	
Slave Register	7001	
Qty of I Registers	3	
Data Type	0 or 1	data type is 16-bit

Master Table Index		Slave Register Number
99	←	7001
100	←	7002
101	←	7003

Case 3

Subroutines related to Input Registers and Holding Registers when the Data Type parameter is 2, 3, 4, or 5 (all 32-bit data types)

NOTE: Opto 22 SNAP- PAC tables support 32-bit data; tables start with index 0. A table with length of 10 has indexes 0 through 9.

For Input Registers and Holding Registers, when using 32-bit data types, there is always a *two*-to-one correlation between the number of (16-bit) registers in the slave and the number of (32-bit) table elements used for the data in the master. This is because most slave devices store data in 16-bit registers. Consequently, 32-bit data in these devices are stored in two consecutive 16-bit registers.

For 32-bit data, the Master Register Offset and Master Register Mode parameter affects the starting register number used in the master data tables, but also the quantity of registers accessed in the slave.

Example 3A: When the Master Register Offset parameter is 0, the subroutine uses the value of the Start Register parameter as the starting register number in the slave and also as the starting table index in the master.

In addition, the Qty of H Registers (or Qty of I Registers) parameter determines the quantity of 16-bit registers read from or written to the slave. The number of 32-bit registers in the master will be half the quantity of this parameter.

The Quantity of H Registers (or Quantity of I Registers) parameter must be an even number because two registers are used for these data types.

The Start Register parameter *in most slave devices will be an odd number.*

The Start Register value will determine the first table index used for the 32-bit data in the master. Additional 32-bit values will be put into subsequent *odd indexes* of the table so that the table indexes in the master will match the first register number for each set of two consecutive 16-bit registers in the slave. All even table indexes are unused.

Parameter	Value	Comment
Master Register Offset	0	
Master Register Mode	False	It will read the Quantity of registers and write every other index of the Modbus table.
Slave Register	7001	must be an odd number
Qty of I Registers	4	must be an even number
Data Type	2,3,4 or 5	data type is 32-bit

Master Table Index		Slave Register Number
7001	←	7001
		7002
7003	←	7003
		7004

Example 3B: When the Master Register Offset parameter is non zero, the subroutine uses the value of the Start Register parameter as the starting register number in the Slave and it uses the value of the Start Register parameter + the Master Register Offset as the starting table index in the master.

The Quantity of H Registers (or Quantity of I Registers) parameter *must be an even number because two registers are used for these data types*. In addition, the Quantity of H Registers (or Qty of I Registers) parameter determines the quantity of 32-bit data values you want to read from or write to the slave and the number of 32-bit table elements used in the master. However, the Modbus protocol has no mechanism for identifying 32-bit data in the messages, so the subroutine requests twice the quantity of registers because the data in the slave is assumed to be 16-bit.

The Master Register value will determine the first table index used for the 32-bit data in the master. Additional 32-bit values will be put into consecutive indexes of the table so that there will not be any gaps in the table.

Parameter	Value	Comment
Master Register Offset	-6993	
Master Register Mode	True	It will read the Quantity of registers and write consecutive indexes of the Modbus table.
Slave Register	7001	must be an odd number
Qty of H Registers	3	Qty of 32-bit values
Data Type	2,3,4 or 5	data type is 32-bit

Master Table Index		Slave Register Number
8	→	7001
		7002
9	→	7003
		7004
10	→	7005
		7006

3: Using the Slave Subroutine

Adding the Slave Subroutine

The Modbus slave subroutine allows an Opto 22 controller to function as a Modbus slave device. Unlike the subroutines used in master strategies, which are called as needed, the Modbus slave subroutine never exits after being called. It continuously monitors the configured ports for Modbus traffic.

If the slave subroutine is needed, include it in your strategy as follows:

1. Start PAC Control in Configure Mode and open the strategy that you intend to use with the integration kit.
2. Select Configure > Subroutines Included to open the Subroutine Files dialog.
3. Click the Add button and use the browser to select the PACModbusSlave.isb subroutine file from the folder PACModbus\Subs.
4. Click OK.

The subroutine appears in the Subroutines Included folder and is ready to be used in your strategy.

To copy the Modbus Slave chart to your strategy, you must export the chart Modbus_Slave_S2 as a PAC Control chart export file (.cxf file) and then import it into your strategy. For more information, see form 1700, the *PAC Control User's Guide*.

This chart has the user setup parameters and subroutine parameters needed to make the subroutine function.

Start the Modbus Slave chart in the Powerup chart of your strategy.

Configure the user parameters in block 1, User Setup, of the Modbus_Slave_S2 chart.

Using Slave Register Mode

The Slave Register Mode is an Integer 32 Variable assigned per port. It is only used for data type 2, 3, 4 or 5.

For read functions:

Slave Register Mode = False: It will read every other index of the Modbus table.

Slave Register Mode = True: It will read consecutive indexes of the Modbus table.

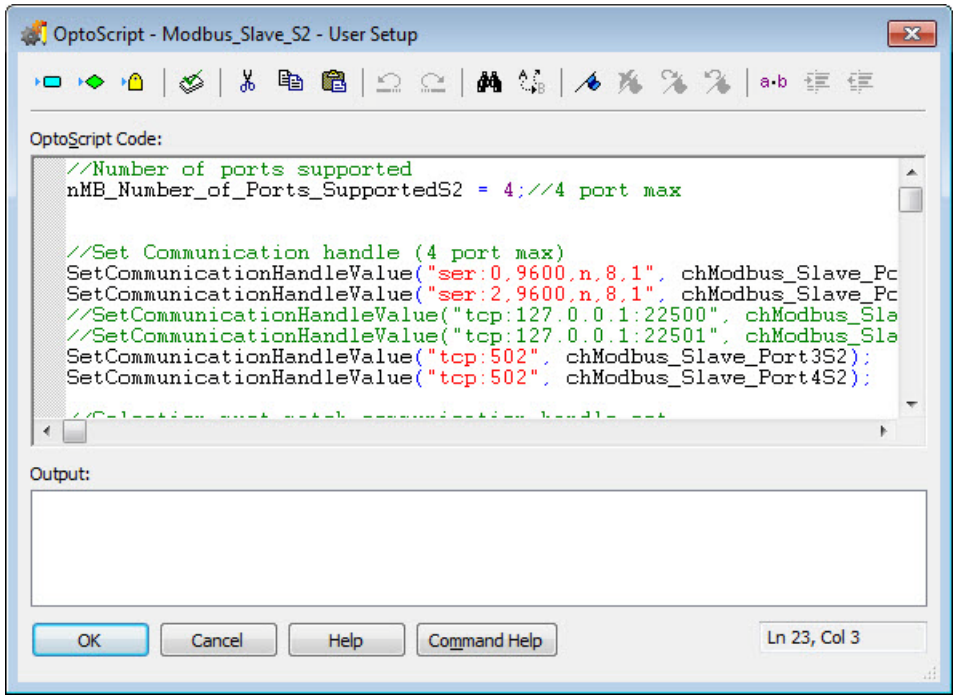
For write functions:

Slave Register Mode = False: It will write every other index of the Modbus table.

Slave Register Mode = True: It will write consecutive indexes of the Modbus table.

Configuring the Slave Subroutine

The following OptoScript window provides the configuration parameters for the slave subroutine. To see the window, double-click the User Setup block in the example slave chart, Modbus_Slave_S2.



Each port (1, 2, 3, and 4) has four related parameters that end in *PortXS2* or *ModeXS2*, where *X* represents the number of ports supported. For example, the four parameters related to port 1 end in either Port1S2 or Mode1S2 as follows:

- SetCommunicationHandleValue("ser:1,9600,n,8,1", chModbus_Slave_Port1S2)
- SetCommunicationHandleValue("ser:1,9600,n,8,1", chModbus_Slave_Port1S2)
- nMB_Comm_Mode1S2 = 0
- SetVariableFalse(nMBSlave_Register_Mode1S2)

In the example strategy, variable and table names that end in S2 are used with the slave subroutine and those that end in M1 are used with the master subroutines.

Set Number of Slave Ports Used (1 - 4)

The Slave subroutine can support from 1 to 4 Modbus ports. The ports can be any combination of serial ASCII, serial RTU, and Modbus/TCP. The example is set to use 4 slave ports. If you only need to use 1 port, then change the value to 1.

```
//Number of ports supported
nMB_Number_of_Ports_SupportedS2 = 4;//4 port max
```

The example is set to support 4 slave ports.

The number you set affects the configuration parameters that end in *PortXS2* or *ModeXS2*, where *X* represents the number of ports used. For example, if the number of ports used is set to 1, then the parameters ending in 2S2, 3S2, and 4S2 do not apply and are ignored by the subroutine.

Set Communication Handles for the Ports Used

Set Communication handle for each port used (4 ports max)

Example:

SetCommunicationHandleValue("ser:0,9600,n,8,1", chModbus_Slave_Port1S2);

SetCommunicationHandleValue("ser:2,9600,n,8,1", chModbus_Slave_Port2S2);

SetCommunicationHandleValue("tcp:127.0.0.1:22500", chModbus_Slave_Port3S2);

SetCommunicationHandleValue("tcp:502", chModbus_Slave_Port4S2);

Set Comm Mode

The Comm Mode must match the setting for SetCommunicationHandleValue for each port.

Comm Mode (Modbus Serial RTU = 0 ASCII = 1) (Modbus/TCP = 2)

Example:

nMB_Comm_Mode1S2 = 0;

nMB_Comm_Mode2S2 = 1;

nMB_Comm_Mode3S2 = 1;

nMB_Comm_Mode4S2 = 2;

Set Serial Slave Address

Example:

nMB_Slave_AddressS2 = 1;

Set Slave Register Mode for Each Port

The Slave Register Mode is an Integer 32 Variable assigned per port. It is only used for data type 2, 3, 4 or 5.

For read functions:

Slave Register Mode = False: It will read every other index of the Modbus table.

Slave Register Mode = True: It will read consecutive indexes of the Modbus table.

For write functions:

Slave Register Mode = False: It will write every other index of the Modbus table.

Slave Register Mode = True: It will write consecutive indexes of the Modbus table.

Example:

SetVariableFalse(nMBSlave_Register_Mode1S2);

SetVariableFalse(nMBSlave_Register_Mode2S2);

SetVariableFalse(nMBSlave_Register_Mode3S2);

SetVariableFalse(nMBSlave_Register_Mode4S2);

Set Data Type by Holding Register

Index 0 is a flag variable indicating whether all holding registers will have the same data type or different data types.

If all holding registers will have data of the same type, set the value of index 0 to 0. Then set the value of index 1 to the data type to be used for all holding registers.

For example:

ntMB_Holding_Register_4X_Data_TypeS2[0] = 1;

ntMB_Holding_Register_4X_Data_TypeS2[1] = 0;

If the holding registers will have data of different types, set the value of index 0 to 1. Then set the data type for each register index by index, in the same indexes as the register table.

For example:

ntMB_Holding_Register_4X_Data_Type[0] = 1;

ntMB_Holding_Register_4X_Data_Type[1] = 3;

ntMB_Holding_Register_4X_Data_Type[2] = 3;

ntMB_Holding_Register_4X_Data_Type[201] = 2;

ntMB_Holding_Register_4X_Data_Type[202] = 2;

ntMB_Holding_Register_4X_Data_Type[525] = 4;

ntMB_Holding_Register_4X_Data_Type[526] = 4;

The table can be loaded from an Initialization file (example files are included).

Set Data Type by Input Register

Index 0 is a flag variable indicating whether all holding registers will have the same data type or different data types.

If all holding registers will have data of the same type, set the value of index 0 to 0. Then set the value of index 1 to the data type to be used for all holding registers.

For example:

ntMB_Input_Register_3X_Data_TypeS2[0] = 0;

ntMB_Input_Register_3X_Data_TypeS2[1] = 0;

If the holding registers will have data of different types, set the value of index 0 to 1. Then set the data type for each register index by index, in the same indexes as the register table.

For example:

ntMB_Input_Register_3X_Data_Type[0] = 1;

ntMB_Input_Register_3X_Data_Type[1] = 3;

ntMB_Input_Register_3X_Data_Type[2] = 3;

ntMB_Input_Register_3X_Data_Type[201] = 2;

ntMB_Input_Register_3X_Data_Type[202] = 2;

ntMB_Input_Register_3X_Data_Type[525] = 4;

ntMB_Input_Register_3X_Data_Type[526] = 4;

The table can be loaded from an Initialization file (example files are included).

Set Modbus Tables Used by Subroutine

The example shows the names of the data tables used in the example strategy. If your strategy uses different data table names, change this portion to reflect the names used in your strategy.

potMBDataTablesS2[0] = &ntMB_Coils_0XS2;//Modbus Coil Table

potMBDataTablesS2[1] = &ntMB_Inputs_1XS2;//Modbus Input Table

potMBDataTablesS2[2] = &ntMB_Input_Register_3X_IntS2;//Modbus Input Register Integer Table

potMBDataTablesS2[3] = &ftMB_Input_Registers_3X_FloatS2;//Modbus Input Register Float Table

potMBDataTablesS2[4] = &ntMB_Holding_Register_4X_IntS2;//Modbus Holding Register Integer Table

potMBDataTablesS2[5] = &ftMB_Holding_Registers_4X_FloatS2;//Modbus Holding Register Float Table

Register Offset

Register offsets can be set independently for each function code using the table named ntMB_RegisterOffsetS2. The index of the table identifies the function code the offset value is to be used for. Note that index 24 is used to identify the function code 23 write offset.

ntMB_RegisterOffsetS2

Index 0 = N/A

Index 1 = Function 1 Offset

Index 2 = Function 2 Offset

Index 3 = Function 3 Offset

Index 4 = Function 4 Offset

Index 15 = Function 15 Offset

Index 16 = Function 16 Offset

Index 23 = Function 23 Read Offset

Index 24 = Function 23 Write Offset

Example:

ntMB_RegisterOffsetS2[1] = 0;

ntMB_RegisterOffsetS2[2] = 0;

```
ntMB_RegisterOffsetS2[3] = 0;
ntMB_RegisterOffsetS2[4] = -7000;
ntMB_RegisterOffsetS2[5] = 0;
ntMB_RegisterOffsetS2[6] = 0;
ntMB_RegisterOffsetS2[15] = 0;
ntMB_RegisterOffsetS2[16] = 0;
ntMB_RegisterOffsetS2[22] = 0;
ntMB_RegisterOffsetS2[23] = 0;
ntMB_RegisterOffsetS2[24] = 0;"
```

4: Subroutine Parameters

This chapter includes the following topics:

Modbus Master Subroutines

The following tables list the parameters for each function code and describe the type of data for each master subroutine parameter:

01: Read Coils

PACModbusMaster01_Read_Coil_Status

Name	Description
Data Table	Integer 32 Table (MB Coil 0X table)
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Coil_Start_Register Index 2 = Quantity_Of_Coils Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35 Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

02: Read Discrete Inputs

PACModbusMaster02_Read_Input_Status

Name	Description
Data Table	Integer 32 Table (MB Input 1X table)
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Input_Start_Register Index 2 = Quantity_Of_Inputs Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

03: Read Holding Registers

PACModbusMaster03_Read_Holding_Registers

Name	Description
Data Table	Pointer Table Index 0 = Holding_Register_4X_Integer table Index 1 = Holding_Register_4X_Float Table
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Start_Register Index 2 = Quantity_Of_Registers (see note 14 on page 39) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Master_Register_Mode (see note 4 on page 35) Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Data Type	Integer 32 Table (see note 5 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

04: Read Input Registers

PACModbusMaster04_Read_Input_Registers

Name	Description
Data Table	Pointer Table Index 0 = Holding_Register_3X_Integer table Index 1 = Holding_Register_3X_Float Table
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Start_Register Index 2 = Quantity_Of_Registers (see note 14 on page 39) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Master_Register_Mode (see note 4 on page 35) Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Data Type	Integer 32 Table (see note 5 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

05: Force Single Coil

PACModbusMaster05_Force_Single_Coil

Name	Description
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Coil_Register Index 2 = Coil State (0 = off, 1 = on) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Not used Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

06: Preset Single Register

PACModbusMaster06_Preset_Single_Register

Name	Description
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Holding Register Index 2 = Register Value Integer (data type 0, 1, 4, 5) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Reg Value Float	Float (data type 2,3)
Data Type	Integer 32 Table (see note 5 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

08: Diagnostics

PACModbusMaster08_Diagnostics

Name	Description
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = SubFunction (see note 11 on page 38) Index 2 = Send_Data (see note 11 on page 38) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Not used Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Receive_Data	Integer 32 Variable (see note 11 on page 38)
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

15: Force Multiple Coils

PACModbusMaster15_Force_Multiple_Coils

Name	Description
Data Table	Integer 32 Table (MB Coil 0X table)
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Coil_Start_Register Index 2 = Quantity_Of_Coils Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

16: Preset Multiple Registers

PACModbusMaster16_Preset_Multiple_Registers

Name	Description
Data Table	Pointer Table Index 0 = Holding_Register_4X_Integer table Index 1 = Holding_Register_4X_Float Table
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Start_Holding Register Index 2 = Quantity_Of_Registers (see note 14 on page 39) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Master_Register_Mode (see note 4 on page 35) Index 6 = Master_RegisterOffset (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35)
Data Type	Integer 32 Table (see note 5 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

17: Report Slave ID

PACModbusMaster17_Report_Slave_ID

Name	Description
Response Table	Integer 32 Table Data is Device Specific
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Not Used Index 2 = Not Used Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Not Used Index 6 = Not used Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

22: Mask Write 4X Registers

PACModbusMaster22_Mask_Write_4X_Register

Name	Description
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Reference_Register Index 2 = Not used Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = AND_Mask Index 6 = OR_Mask Index 7 = Transaction_Identifier (see note 2 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

23: Read/Write 4X Registers

PACModbusMaster23_Read_Write_4X_Registers

Name	Description
Data Table	Pointer Table Index 0 = Holding_Register_4X_Integer table Index 1 = Holding_Register_4X_Float Table
Setup Parameter	Integer 32 Table Index 0 = Slave Address Index 1 = Read_Holding_Start_Register Index 2 = Read_Quantity_Of_Holding_Registers (see note 14 on page 39) Index 3 = Wait_Time_ms Index 4 = Comm_Mode (0=RTU, 1=ASCII, 2=TCP) Index 5 = Master_Register_Mode (see note 4 on page 35) Index 6 = Master_RegisterOffsetRead (see note 1 on page 35) Index 7 = Transaction_Identifier (see note 2 on page 35) Index 8 = Write_Holding_Start_Register Index 9 = Write_Quantity_Of_Holding_Registers (see note 14 on page 39) Index 10 = Master_RegisterOffsetWrite (see note 1 on page 35)
Data Type	Integer 32 Table (see note 5 on page 35)
Modbus Port	Communication Handle
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status (see note 3 on page 35) Index 2 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = OK Count Index 1 = Port Status (0=OK or Error Code) Index 2 = Fail Count
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

Modbus Slave Subroutine

PACModbusSlave

The Modbus slave subroutine can support from 1 to 4 ports. The ports can be configured as a mix of Modbus Serial RTU, Modbus Serial ASCII or Modbus/TCP.

Name	Description
Data Tables	Pointer Table Index 0 = Coil 0X table Index 1 = Inputs_1X table Index 2 = Input_Register_3X_Integer table Index 3 = Input_Registers_3X_Float table Index 4 = Holding_Register_4X_Integer table Index 5 = Holding_Register_4X_Float Table
Setup Tables	Pointer Table Index 0 = Input_Register_3X_Data_Type (see note 5 on page 35) Integer 32 table Index 1 = Holding_Register_4X_Data_Type (see note 5 on page 35) Integer 32 table Index 2 = RegisterOffset (see note 6 on page 37) Integer 32 table
Setup Parameters	Pointer Table Index 0 = Number_of_Masters_Supported (1 – 4) Index 1 = Modbus_Slave_Port1 comm handle (see note 10 on page 37) Index 2 = Modbus_Slave_Port2 comm handle (see note 10 on page 37) Index 3 = Modbus_Slave_Port3 comm handle (see note 10 on page 37) Index 4 = Modbus_Slave_Port4 comm handle (see note 10 on page 37) Index 5 = Comm_Mode1 (0=RTU, 1=ASCII, 2=TCP) Index 6 = Comm_Mode2 (0=RTU, 1=ASCII, 2=TCP) Index 7 = Comm_Mode3 (0=RTU, 1=ASCII, 2=TCP) Index 8 = Comm_Mode4 (0=RTU, 1=ASCII, 2=TCP) Index 9 = Slave_Register_Mode1 (see note 7 on page 37) Index 10 = Slave_Register_Mode2 (see note 7 on page 37) Index 11 = Slave_Register_Mode3 (see note 7 on page 37) Index 12 = Slave_Register_Mode4 (see note 7 on page 37) Index 13 = Slave_Address Index 14 = OpenDelay1 down timer (see note 8 on page 37) Index 15 = OpenDelay2 down timer (see note 8 on page 37) Index 16 = OpenDelay3 down timer (see note 8 on page 37) Index 17 = OpenDelay4 down timer (see note 8 on page 37) Index 18 = Modbus_Port_1 up timer (see note 9 on page 37) Index 19 = Modbus_Port_2 up timer (see note 9 on page 37) Index 20 = Modbus_Port_3 up timer (see note 9 on page 37) Index 21 = Modbus_Port_4 up timer (see note 9 on page 37)
Status Table	String Table Index 0 = TX String (RTU and TCP use Hexadecimal to view) Index 1 = Status 1 (see note 12 on page 38) Index 2 = Status 2 (see note 12 on page 38) Index 3 = Status 3 (see note 12 on page 38) Index 4 = Status 4 (see note 12 on page 38) Index 5 = RX String (RTU and TCP use Hexadecimal to view)
Port Status	Integer 32 Table Index 0 = Not used Index 1 = Port 1 Status (see note 13 on page 39) Index 2 = Port 2 Status (see note 13 on page 39) Index 3 = Port 3 Status (see note 13 on page 39) Index 4 = Port 4 Status (see note 13 on page 39)
Put Status In	Integer 32 Variable 0 = Success -67 = Out of memory -69 = Null object error

Subroutine Notes

1. **Master Register Offset**

   Integer 32 Variable. Used to offset the data from the read register or write register.

   – Example 1: If the first read register is 7001 and the offset is -7000.The data will be stored starting at index 1 of the Modbus table in the strategy.

   – Example 2: If the first write register is 7001 and the offset is -7000. The data to write will start at index 1 of the Modbus table in the strategy

2. **Transaction Identifier**

   Integer 32 Variable. Transaction Identifier is used by the subroutine to identify the packet. It is not used for serial RTU or ASCII mode.

3. **Status**

   String Table. This is the status of the last packet.

   – No Session

   – Timeout

   – Too Many Characters

   – CRC Mismatch

   – Wrong Slave Address

   – LRC Mismatch

   – Identifier Mismatch

   – Exception code

   – Invalid Table Index

   – OK

4. **Master Register Mode**

   Integer 32 Variable. Only used for data type 2, 3, 4 or 5.

   For read functions:

   Master Register Mode = False: It will read the quantity of registers and write every other index of the Modbus table.

   Master Register Mode = 1: It will read the quantity of registers and write consecutive indexes of the Modbus table.

   For write functions:

   Master Register Mode = False: It will read every other index and write using two registers per value.

   Master Register Mode = True: It will read consecutive indexes and write using two registers per value.

5. **Data Type**

   Integer 32 Table. This allows you to read or write multiple register of different data types. This is an integer 32 table that will have a value of 0 - 5 for each register (index).

   In the User Setup block of the example master chart:

Index 0 = 0
Use one data type. The data type used is in index 1. The length of the table must be 2 or more. The register offset is not used for the data type table.

Index 0 = 1
The data and the data type are in the same index for each table. If the register offset is set to -7000 and the register to read is 7001 then the data will be stored in index 1 of the Modbus data table based on the data type at index 1 of the data type table. The length of the tables should be (start register + quantity + register offset + 1). The register offset is used for the data type table.

Index 0 = 2
The data type index and register number are the same. If you read register 8001 it will use the data type at index 8001 of the data type table. The length of the table should be (start register + quantity + 1). The register offset is not used for the data type table.

In the User Setup block of the example slave chart

Index 0 = 0
Use one data type. The data type used is in index 1. The length of the table should be (highest register number + register offset + 1). The register offset is used for the data type table.

Index 0 = 1
The data and the data type are in the same index for each table. If the register offset is set to -7000 and the register to read is 7001 then the data will be stored in index 1 of the Modbus data table based on the data type at index 1 of the data type table. The length of the tables should be (start register + quantity + register offset + 1). The register offset is used for the data type table.

Index 0 = 2
Not supported

Example: If holding register 1, 2, 3, 4 are data type 2, holding register 5 and 6 are data type 4 and holding register 7 - 200 are data type 1

4X data type table:

- Index 0 = 1
- Index 1 = 2
- Index 2 = 2
- Index 3 = 2
- Index 4 = 2
- Index 5 = 4
- Index 6 = 4
- Index 7 - 200 = 1

If the Master Register Mode = False and Master Register Offset = 0 then there will be a float value at register 1 and 3, a 32 bit signed integer at index 5 and 16 bit unsigned integers at index 7 - 200.

The data type table values can be downloaded from an Initialization file after the strategy. For more information, see form 1700, the *PAC Control User's Guide*.

6. **Register Offset**

   Integer 32 Table. Used to simulate a device with high number register without needing to have Modbus tables with high lengths.

   - Index 0 = Not used
   - Index 1 = Offset for function 1
   - Index 2 = Offset for Function 2
   - Index 3 = Offset for Function 3
   - Index 4 = Offset for Function 4
   - Index 15 = Offset for Function 15
   - Index 16 = Offset for Function 16
   - Index 23 = Offset for Function 23 Read
   - Index 24 = Offset for Function 23 Write

   Example: If the master must read holding register 7001 and the offset at index 3 of the Offset table = -7000. The data will be read from index 1 of the Modbus 4X table

7. **Slave Register Mode**

   Integer 32 Variable, per port. Only used for data type 2, 3, 4 or 5.

   For read functions:

   Slave Register Mode = False: It will read every other index of the Modbus table.

   Slave Register Mode = True: It will read consecutive indexes of the Modbus table.

   For write functions:

   Slave Register Mode = False: It will write every other index of the Modbus table.

   Slave Register Mode = True: It will write consecutive indexes of the Modbus table.

8. **Open Delay**

   Down timer, per port. This timer is used by subroutine to delay between opening attempts on port errors.

9. **Modbus Port Timer**

   Up timer, per port. This timer is used by subroutine to check status of the ports for activity every 30 seconds.

10. **Modbus Slave Port**

    Communication Handle, per port. Communication Handle for each port used. The communication handle and Comm Mode set in the User Setup block must match. The subroutine supports from 1 to 4 ports. The ports can be configured for Serial RTU, Serial ASCII, or Modbus/TCP in any order.

    Example: Set in User Setup block

    /////Communication Handle for serial module on rack

    SetCommunicationHandleValue("tcp:127.0.0.1:22500", chModbus_Slave_Port1S2);

    /////Communication Handle for Modbus/TCP

    SetCommunicationHandleValue("tcp:502", chModbus_Slave_Port2S2);

    /////Communication Handle for serial port on the controller

    SetCommunicationHandleValue("ser:1,9600,n,8,1", chModbus_Slave_Port3S2);

/////Communication Handle for Modbus/TCP
SetCommunicationHandleValue("tcp:502", chModbus_Slave_Port4S2);

nMB_Comm_Mode1S2 = 0; = Serial RTU
nMB_Comm_Mode2S2 = 2; = Modbus/TCP
nMB_Comm_Mode3S2 = 1; = Serial ASCII
nMB_Comm_Mode4S2 = 2; = Modbus/TCP

11. Function 08 Diagnostics Sub-function Codes

Sub-Function	Name	Data (Send)	Data (Receive)
0	Return Query Data	Any	0
1	Restart Communication Option	0 or 1 = Clear Log	0
2	Return Diagnostic Register	0	Register data
3	Change ASCII Input Delimiter	Decimal Value of Character	0
4	Force Listen Only Mode	0	0
5	Reserved		
6	Reserved		
7	Reserved		
8	Reserved		
9	Reserved		
10	Clear Counters and Diagnostic Register	0	0
11	Return Bus Message Count	0	Message Ct.
12	Return Bus Communication Error Count	0	Error Ct.
13	Return Bus Exception Error Count	0	Error Ct.
14	Return Slave Message Count	0	Message Ct.
15	Return Slave No Response Count	0	No Response Ct.
16	Return Slave NAK Count	0	NAK Ct.
17	Return Slave Busy Count	0	Busy Ct.
18	Return Bus Character Overrun Count	0	Overrun Ct.
19	Reserved		
20	Clear Overrun Counter And Flag	0	0
21	Reserved	0	

12. Status

String Table

- Listen TCP
- Open RTU
- Open ASCII
- Too Many Characters
- CRC Mismatch
- Wrong Slave Address
- LRC Mismatch
- Exception code
- OK RTU
- OK ASCII
- OK TCP
- Not Configured
- TX Error + Error code

13. Port Status

Integer 32 Table

- 0 = OK
- Any negative number = Error
- On ports configured for Modbus/TCP, -52 and -442 are normal until the Modbus master opens the session or after the Modbus master closes the session.

14. Quantity_Of_Registers

For data types 2, 3, 4 and 5 use a quantity of 2 per value. These data types require two Registers in the Modbus packet.

5: Troubleshooting

Troubleshooting the Master Subroutines

Each master subroutine has a status table and a port status table to help verify and troubleshoot the operation of the subroutine.

Status Table

String Table. For example, stMBStatus01M1.

- Index 0 = TX String (RTU and TCP use Hexadecimal to view)
- Index 1 = Status

  This is the status of the last packet.

  - No Session
  - Timeout
  - Too Many Characters
  - CRC Mismatch
  - Wrong Slave Address
  - LRC Mismatch
  - Identifier Mismatch
  - Exception code
  - Invalid Table Index
  - OK
- Index 2 = RX String (RTU and TCP use Hexadecimal to view)

Port Status Table

Integer 32 Table. For example, ntMBPortStatus01M1.

- Index 0 = OK Count
- Index 1 = Port Status (0=OK or Error Code)
- Index 2 = Fail Count

Troubleshooting the Slave Subroutine

The slave subroutine has a status table and a port status table to help verify and troubleshoot the operation of the subroutine.

Status Table

String Table, stMBStatusS2.

- Index 0 = TX String (RTU and TCP use Hexadecimal to view)
- Index 1 = Status port 1

  This is the status of the last packet.

  - Listen TCP
  - Open RTU
  - Open ASCII
  - Too Many Characters
  - CRC Mismatch
  - Wrong Slave Address
  - LRC Mismatch
  - Exception code
  - OK RTU
  - OK ASCII
  - OK TCP
  - Not Configured
  - TX Error + Error code
- Index 2 = Status port 2
- Index 3 = Status port 3
- Index 4 = Status port 4
- Index 5 = RX String (RTU and TCP use Hexadecimal to view)

Port Status Table

Integer 32 Table, ntMBPortStatusS2

- Index 0 = Not Used
- Index 1 = Port 1 Status
  - 0 = OK
  - Any negative number = Error

    On ports configured for Modbus/TCP, -52 and -442 are normal until the Modbus master opens the session or after the Modbus master closes the session.
- Index 2 = Port 2 Status
- Index 3 = Port 3 Status
- Index 4 = Port 4 Status