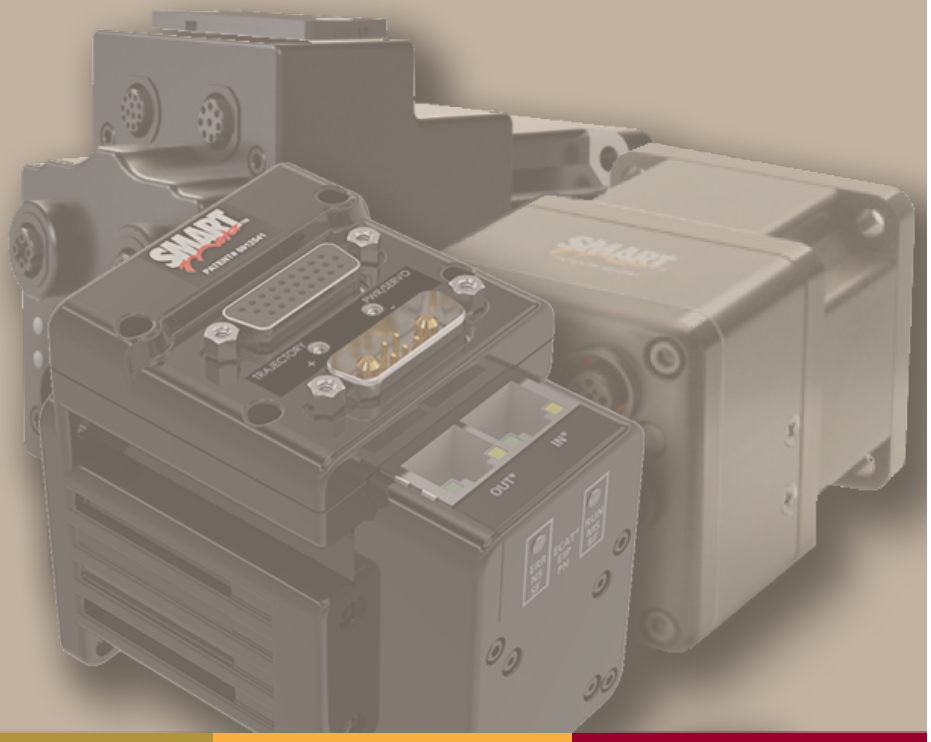
MODBUS® RTU IMPLEMENTATION FOR

FULLY INTEGRATED SERVO MOTORS

CLASS 5 AND 6 SMARTMOTOR™ TECHNOLOGY



Rev. E, September 2022

DESCRIBES THE CLASS 5 AND 6 SMARTMOTOR™ SUPPORT FOR THE MODBUS® RTU PROTOCOL

www.animatics.com

MOOG ANIMATICS

Copyright Notice

Moog Animatics Class 5 and 6 SmartMotor™ Modbus RTU Guide, Rev. E, PN: SC80100014-001.

Please let us know if you find any errors or omissions in this manual so that we can improve it for future readers. Such notifications should contain the words "Modbus RTU Guide" in the subject line and be sent by e-mail to: animatics_marcom@moog.com. Thank you in advance for your contribution.

Contact Us:

Americas - West
Moog Animatics
2581 Leghorn Street
Mountain View, CA 94043
USA

Tel: 1 650-960-4215

Americas - East
Moog Animatics
1995 NC Hwy 141
Murphy, NC 28906
USA

Support: 1 (888) 356-0357

Website: www.animatics.com

Email: animatics_sales@moog.com

Table Of Contents

Introduction

This chapter provides information on the purpose and scope of this manual. It also provides information on safety notation, related documents and additional resources.

Purpose

This Modbus® guide describes the Modbus RTU protocol support provided by the Moog Animatics Class 5 and 6 SmartMotor™. It describes the major concepts that must be understood to integrate a SmartMotor follower with a PLC or other Modbus RTU (Remote Terminal Unit) controller. However, it does not cover all the low-level details of the Modbus RTU protocol.

NOTE: The feature set described in this version of the manual refers to motor firmware 5.x.4.31 (Class 5 D/M) or higher, and 6.0.2.41 (Class 6 M) / 6.4.2.50 (Class 6 D) or higher.

This manual is intended for programmers or system developers who have read and understand the *Modbus Serial Line Protocol and Implementation Guide V1.02*, which is published and maintained by Modbus.org. Therefore, this manual is not a tutorial on that specification or the Modbus RTU protocol. Instead, it should be used to understand the specific implementation details for the Moog Animatics SmartMotor. For a general overview of Modbus RTU, see the FAQ page and other resources at www.modbus.org.

New Feature Highlights

This section highlights several new features have been added for Class 5 firmware versions 5.0.4.53 and later for D-style motors, and 5.98.4.53 and later for M-style motors. For more details on these and other new features, see New Feature Details on page 27.

NOTE: These features are not currently implemented in the Class 6 motors.

- **Command Code Mappings**
    - Read multiple values, packed data (see Read Packet Data from Modbus on page 29).
        - Gives priority to reading many values with minimal overhead.
        - Friendly to PLC environments that have a constant range of registers read during operation.
    - Write single value (see Modbus Single Command Write on page 34).
        - Option to call G after value is written to support the common use-case of "VT=1234 G".
        - Optional cyclic edge-triggered protection/feedback to support PLC programming environments, which may not have explicit control of message transmission cycles. Or, if desired, every Modbus write to the motor will execute the commanded action.
        - Friendly to PLC environments that have a constant range of registers read during operation.
- **GOSUB R2** (see GOSUB R2 through Modbus on page 37).
    - Same implementation used in CANopen object 0x2309.
    - For behavior similar to a function call, a quantity of user registers can optionally be written in the same “write multiple registers” Modbus operation as the GOSUB R2. When the subroutine runs, it can access these newly-written values as input arguments.
    - Special option to allow cyclic calling of GOSUB with or without edge-sensitive behavior (cycling between -1 and the intended subroutine number as the written data value).
    - This particular GOSUB interface always prevents re-entry until the subroutine completes with a RETURN. This protects from stack overflow due to repeated calling of the GOSUB. However, it doesn’t prevent self-induced stack overflow within the program if program is poorly written to GOSUB to itself.
    - A status flag is available to indicate when the subroutine has completed.
    - Writing to this location when still busy will return an error through Modbus.
    - Friendly to PLC environments that have a constant range of registers read during operation.

- **Parity Checking**
    - Parity and framing errors reported by the UART are used to reject a Modbus packet if any received character shows such an error. If parity mode is not selected when the channel is opened (OCHN), then only the framing error information will be used. CRC checking is still used (implemented previously).
    - Except for Modbus, parity and framing checking do not apply to other modes of serial port operation, and will only indicate a user flag when these occur.

Safety Information

This section describes the safety symbols and other safety information.

Safety Symbols

The manual may use one or more of these safety symbols:

WARNING: This symbol indicates a potentially nonlethal mechanical hazard, where failure to comply with the instructions could result in serious injury to the operator or major damage to the equipment.

CAUTION: This symbol indicates a potentially minor hazard, where failure to comply with the instructions could result in slight injury to the operator or minor damage to the equipment.

NOTE: Notes are used to emphasize non-safety concepts or related information.

Other Safety Considerations

The Moog Animatics SmartMotors are supplied as components that are intended for use in an automated machine or system. As such, it is beyond the scope of this manual to attempt to cover all the safety standards and considerations that are part of the overall machine/system design and manufacturing safety. Therefore, this information is intended to be used only as a general guideline for the machine/system designer.

It is the responsibility of the machine/system designer to perform a thorough "Risk Assessment" and to ensure that the machine/system and its safeguards comply with the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the site where the machine is being installed and operated. For more details, see Machine Safety on page 11.

Motor Sizing

It is the responsibility of the machine/system designer to select SmartMotors that are properly sized for the specific application. Undersized motors may: perform poorly, cause excessive downtime or cause unsafe operating conditions by not being able to handle the loads placed on them. The *System Best Practices* document, which is available on the Moog Animatics website, contains information and equations that can be used for selecting the appropriate motor for the application.

Replacement motors must have the same specifications and firmware version used in the approved and validated system. Specification changes or firmware upgrades require the approval of the system designer and may require another Risk Assessment.

Environmental Considerations

It is the responsibility of the machine/system designer to evaluate the intended operating environment for dust, high-humidity or presence of water (for example, a food-processing environment that requires water or steam wash down of equipment), corrosives or chemicals that may come in contact with the machine, etc. Moog Animatics manufactures specialized IP-rated motors for operating in extreme conditions. For details, see the *Moog Animatics Product Catalog*.

Machine Safety

In order to protect personnel from any safety hazards in the machine or system, the machine/system builder must perform a "Risk Assessment", which is often based on the ISO 13849 standard. The design/implementation of barriers, emergency stop (E-stop) mechanisms and other safeguards will be driven by the Risk Assessment and the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the site where the machine is being installed and operated. The methodology and details of such an assessment are beyond the scope of this manual. However, there are various sources of Risk Assessment information available in print and on the internet.

NOTE: The next list is an example of items that would be evaluated when performing the Risk Assessment. Additional items may be required. The safeguards must ensure the safety of all personnel who may come in contact with or be in the vicinity of the machine.

In general, the machine/system safeguards must:

- Provide a barrier to prevent unauthorized entry or access to the machine or system. The barrier must be designed so that personnel cannot reach into any identified danger zones.
- Position the control panel so that it is outside the barrier area but located for an unrestricted view of the moving mechanism. The control panel must include an E-stop mechanism. Buttons that start the machine must be protected from accidental activation.
- Provide E-stop mechanisms located at the control panel and at other points around the perimeter of the barrier that will stop all machine movement when tripped.
- Provide appropriate sensors and interlocks on gates or other points of entry into the protected zone that will stop all machine movement when tripped.
- Ensure that if a portable control/programming device is supplied (for example, a hand-held operator/programmer pendant), the device is equipped with an E-stop mechanism.

  NOTE: A portable operation/programming device requires *many* additional system design considerations and safeguards beyond those listed in this section. For details, see the safety standards specified by the governing authority (for example, ISO, OSHA, UL, etc.) for the site where the machine is being installed and operated.
- Prevent contact with moving mechanisms (for example, arms, gears, belts, pulleys, tooling, etc.).
- Prevent contact with a part that is thrown from the machine tooling or other part-handling equipment.
- Prevent contact with any electrical, hydraulic, pneumatic, thermal, chemical or other hazards that may be present at the machine.
- Prevent unauthorized access to wiring and power-supply cabinets, electrical boxes, etc.
- Provide a proper control system, program logic and error checking to ensure the safety of all personnel and equipment (for example, to prevent a run-away condition). The control system must be designed so that it does not automatically restart the machine/system after a power failure.
- Prevent unauthorized access or changes to the control system or software.

Documentation and Training

It is the responsibility of the machine/system designer to provide documentation on safety, operation, maintenance and programming, along with training for all machine operators, maintenance technicians, programmers, and other personnel who may have access to the machine. This documentation must include proper lockout/tagout procedures for maintenance and programming operations.

It is the responsibility of the operating company to ensure that:

- All operators, maintenance technicians, programmers and other personnel are tested and qualified before acquiring access to the machine or system.
- The above personnel perform their assigned functions in a responsible and safe manner to comply with the procedures in the supplied documentation and the company safety practices.
- The equipment is maintained as described in the documentation and training supplied by the machine/system designer.

Additional Equipment and Considerations

The Risk Assessment and the operating company's standard safety policies will dictate the need for additional equipment. In general, it is the responsibility of the operating company to ensure that:

- Unauthorized access to the machine is prevented at all times.
- The personnel are supplied with the proper equipment for the environment and their job functions, which may include: safety glasses, hearing protection, safety footwear, smocks or aprons, gloves, hard hats and other protective gear.
- The work area is equipped with proper safety equipment such as first aid equipment, fire suppression equipment, emergency eye wash and full-body wash stations, etc.
- There are no modifications made to the machine or system without proper engineering evaluation for design, safety, reliability, etc., and a Risk Assessment.

Safety Information Resources

Additional SmartMotor safety information can be found on the Moog Animatics website; open the topic "Controls - Notes and Cautions" located at:

https://www.animatics.com/support/downloads/knowledgebase/controls---notes-and-cautions.html

OSHA standards information can be found at:

https://www.osha.gov/law-regs.html

ANSI-RIA robotic safety information can be found at:

http://www.robotics.org/robotic-content.cfm/Robotics/Safety-Compliance/id/23

UL standards information can be found at:

http://ulstandards.ul.com/standards-catalog/

ISO standards information can be found at:

http://www.iso.org/iso/home/standards.htm

EU standards information can be found at:

http://ec.europa.eu/growth/single-market/european-standards/harmonised-standards/index_en.htm

Additional Documents

The Moog Animatics website contains additional documents that are related to the information in this manual. Please refer to these lists.

Related Guides

- Moog Animatics SmartMotor™ Installation and Startup Guides

  http://www.animatics.com/install-guides

- *SmartMotor™ Developer's Guide*

  http://www.animatics.com/smartmotor-developers-guide

- *SmartMotor™ Homing Procedures and Methods Application Note*

  http://www.animatics.com/homing-application-note

- *SmartMotor™ System Best Practices Application Note*

  http://www.animatics.com/system-best-practices-application-note

In addition to the documents listed above, guides for fieldbus protocols and more can be found on the website: https://www.animatics.com/support/downloads.manuals.html

Other Documents

- SmartMotor™ Certifications

  https://www.animatics.com/certifications.html

- *SmartMotor Developer's Worksheet*
  (interactive tools to assist developer: Scale Factor Calculator, Status Words, CAN Port Status, Serial Port Status, RMODE Decoder and Syntax Error Codes)

  https://www.animatics.com/support/downloads.knowledgebase.html

- *Moog Animatics Product Catalog*

  http://www.animatics.com/support/moog-animatics-catalog.html

Additional Resources

The Moog Animatics website contains useful resources such as product information, documentation, product support and more. Please refer to these addresses:

- General company information:

  http://www.animatics.com

- Product information:

  http://www.animatics.com/products.html

- Product support (Downloads, How-to Videos, Forums and more):

  http://www.animatics.com/support.html

- Contact information, distributor locator tool, inquiries:

  https://www.animatics.com/contact-us.html

- Applications (Application Notes and Case Studies):

  http://www.animatics.com/applications.html

Modbus Resources

Modbus is a common standard maintained by Modbus.org:

- Modbus.org website:

  http://www.modbus.org

System Connections

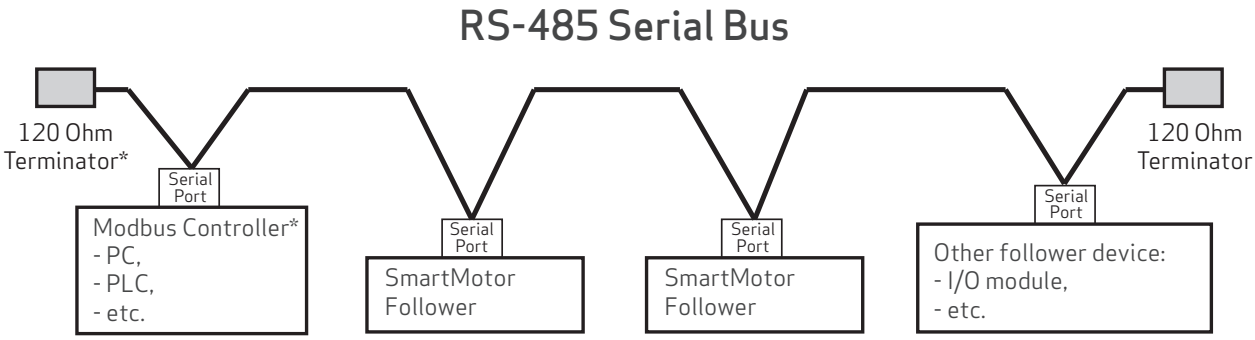
These sections describe the system connections.

NOTE: For information on your motor's connector pinouts and status LEDs, refer to your motor's SmartMotor Installation and Startup Guide. Also, note that if your Class 5 motor is equipped with the CANopen option, LEDs 2 and 3 do not apply to Modbus RTU.

Cable Diagram

The next figure shows a Modbus RTU controller connected to a serial daisy chain of follower devices. Although different network topologies are possible, the daisy chain provides the most reliable performance. If drops are made from the main trunk line, they should be kept as short as possible.

NOTE: When calculating the overall (total) cable length, you must account for all cable segments in the network.



**Controller may have termination option; see controller's documentation for details.*

NOTE: RS-485 serial communications uses a voltage differential signal that requires proper termination with a 120 ohm resistor at both ends of the network cable. This conforms with RS-485 standards for biasing to ensure reliable performance.

Maximum Cable Length

NOTE: When calculating the overall (total) cable length, you must account for all cable segments in the network.

Moog Animatics recommends a maximum cable length of 1000 meters or a maximum baud rate of 115200. As baud rate increases, the maximum cable length decreases. The maximum cable length allowed depends on the baud rate, gauge and other physical properties of the cable, operating environment and other factors.

For more details, see the *Modbus Serial Line Protocol and Implementation Guide V1.02*.

Using Modbus

These sections describe how to enable Modbus communications with your SmartMotor, along with information on supported function codes, input registers and holding registers.

Modbus RTU Description

Modbus RTU is a standard that allows industrial devices to communicate over serial connections. The Moog Animatics Class 5 and 6 SmartMotors support communication to a PLC, HMI, or other host device over serial RS-485 only.

NOTE: The Moog Animatics Class 6 SmartMotor also supports the Modbus TCP/IP protocol over Ethernet TCP/IP connections. Refer to that guide for details.

OCHN Command

The OCHN command is used to open the serial port with Modbus RTU support. Refer to the next examples. By default, SmartMotor serial ports are not open in this mode.

The Class 5 and 6 SmartMotors will act as a follower devices in such a network. User integer variables have read/write access as word (16-bit), or long (32-bit values). Status words can be read as 16-bit words. Also, subroutines may be called via GOSUB(value).

NOTE: In this guide, hexadecimal numbers are prefixed by 0x. Therefore 0x0001 is a hexadecimal one in 16-bit representation, and 0x00000001 is a hexadecimal one in 32-bit representation.

M-style Motor Example

```
OCHN(MB4,0,N,115200,1,8,D) ' Modbus COM0 M-series circular connector.
```

NOTE: Because the M-style SmartMotor has only one serial port, it is mutually exclusive with applications that need to interact with the SMI software over serial communications. If the above code is used in an M-style motor, it may be necessary to use the SMI software Communication Lockup Wizard to disable to program when SMI is needed to configure the motor.

D-style Motor Example

```
OCHN(MB4,1,N,115200,1,8,D) ' Modbus COM1 D-series 15-pin or
                           ' HD26-pin D-sub connector.
SADDR1 ' Modbus uses this address to identify itself.
```

For more details on the OCHN command, see the *SmartMotor™ Developer's Guide*.

Legacy Modbus RTU Discussion

Memory/registers designated as being in the 4X space are referred to as read/write space. This is an association consistent with legacy Modbus RTU.

Memory/registers designated as being in the 3X space are referred to as read-only space. This is an association consistent with legacy Modbus RTU.

For Class 5 and 6 SmartMotor access to resources described in this document, note that 0-based addressing is used. At the Modbus network level, this is the address that is transmitted. Some controllers may ask the user to specify addresses that are not 0-based. The address in the 4X space of 40001 is actually 0000(hex) on the network. Further, because legacy space is small, a network address of 8000(hex) is converted to a 4X reference as shown:

32768 + 40001 = 72769 (some Modbus RTU tools may do this)

Likewise, in the 3X space, the network address of 8000(hex) is converted to legacy format as shown:

32768 + 30001 = 62769

Note that address offsets are actually separated by function codes in the 4X and 3X legacy reference spaces. Therefore, what looks like an overlap in controller memory is not. The controller memory has been separated to respect the convention of read/write memory and read-only memory, 4X and 3X, respectively.

Some controllers may handle this differently. Therefore, it is the responsibility of the system programmer to be aware of the method used in the host controller. The SmartMotor simply expects 0-based addresses and is not aware of any translation that the host may conduct.

Supported Function Codes

A small set of Modbus function codes are supported for simple access to variables and status words. The GOSUB feature of the AniBasic language can be accessed through register write as well.

16-Bit Access

This table shows the codes, descriptions and functions for 16-bit access.

Code	Description	Function
03	Read Holding Registers (4X space)	Read 16-bit value or values.
04	Read Input Registers (3X space)	Read 16-bit (read-only) value or values.
06	Write Single Register (4X space)	Write 16-bit value or values.
16	Write Multiple Registers (4X space)	Write 16-bit value or values.

32-Bit Access

This table shows the codes, descriptions and functions for 32-bit access.

Code	Description	Function
03	Read Holding Registers (4X space)	Read 32-bit value or values.
16	Write Multiple Registers (4X space)	Write 32-bit value or values.
NOTE: Low word of 32-bit values is stored at lower Modbus address.		

Text Access (encapsulated command)

This table shows the codes, descriptions and functions for text / encapsulated command access.

This feature is provided primarily for use with SMI software when port settings configured as Modbus, and will not be described in detail here.

Code	Description	Function
65	Encapsulated command	Animatics command and report strings (including response strings.)
NOTE: Requires for Class 5 firmware 5.x.4.57 or higher. Requires for Class 6 firmware 6.x.2.35 or higher. NOTE: RTU mode support only.		

Input Registers - 3X

The Modbus 3X input registers are 16-bit registers used to read data to the PLC (i.e., they are read only). Regarding the SmartMotor, the set of data that can be read includes the Moog Animatics AniBasic "RW(x)" status words — the physical I/O state inputs RW(16) and, optionally, RW(17), and other RW(x) status words. Refer to the next table.

3X Mapping

This table describes the 3X mapping.

Address (hex)	Byte #	Description	Comments
0x0000	2	Status Register 0	Drive state and hardware limits
0x0001	2	Status Register 1	Index capture and software limits
0x0002	2	Status Register 2	Programs and communications
0x0003	2	Status Register 3	PID and motion
0x0004	2	Status Register 4	Timers
0x0005	2	Status Register 5	Interrupts
0x0006	2	Status Register 6	Commutation and bus
0x0007	2	Status Register 7	Trajectory details
0x0008	2	Status Register 8	Cam and interpolation user bits
0x0009	2	Status Register 9	N/A
0x000a	2	Status Register 10	N/A
0x000b	2	Status Register 11	N/A
0x000c	2	Status Register 12	User bits word 0
0x000d	2	Status Register 13	User bits word 1
0x000e	2	Status Register 14	N/A
0x000f	2	Status Register 15	N/A
0x0010	2	Status Register 16	I/O state, word 0
0x0011	2	Status Register 17	I/O state, word 1 (Class 5 D-style with AD1 option only)

NOTES:
1. Addresses shown are 0-based. Legacy Modbus addresses may be translated differently by the host controller.
2. Refer to the *SmartMotor Developer's Guide* for a full description of status word functionality.

LIMITATIONS: Up to 29 words can be read at a time (for the purposes of the input registers, reading is only meaningful up to the index shown in the previous table).

Holding Registers - 4X

The Modbus 4X holding registers are 16-bit registers used to read data to and write data from the PLC. Regarding the SmartMotor, the set of data that can be read/written includes the Moog Animatics AniBasic variables a-zzz, ab, aw and al, and the GOSUB command. Refer to the next table.

4X Mapping

This table describes the 4X mapping.

Address (hex)	Byte #	AniBasic Command Description	Comments
0x2000-2033	-	a to z	User memory
0x2034-2067	-	aa to zz	User memory
0x2068-209B	-	aaa to zzz	User memory, includes zzz
0x209C-0x2101		ab[0]-ab[203] al[0]-al[50] aw[0]-aw[101]	User memory array
0x8004		GOSUB(label)	Execute subroutine specified by label
NOTES: 1. Addresses shown are 0-based. Legacy Modbus addresses may be translated differently by the host controller. 2. User memory is word-addressable only. The low-addressed word is the lower half of a 32-bit number in the controller.			

LIMITATIONS: Up to 29 words can be read at a time. However, if accessing SmartMotor variables a, b, c, etc., which are 2 words each as 32-bit variables, then 14 variables can be accessed in a read operation. Writing multiple registers has a restriction of up to 27 words (13 variables that are 32-bits each).

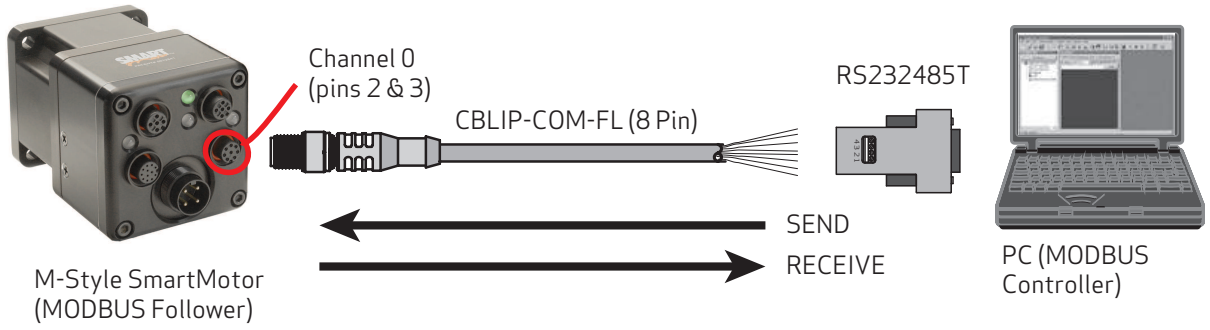
Modbus RTU Communications Example

This topic contains Modbus communications examples.

Modbus RTU Communication Setup

This section describes a typical setup for Modbus RTU (serial) communications.

- Connect the SmartMotor's RS-485 pins to the PLC, HMI or other device that is serving as the Modbus controller. For details, refer to Cable Diagram on page 16.
    - For Class 5 D-style motors, COM channel 1 (pins 5 and 6 on the 15-pin D-sub I/O connector) provides the RS-485 connections for Modbus RTU.
    - For Class 6 D-style motors, COM channel 1 (pins 19 and 20 on the HD26 high density, 26-pin D-sub connector) provides the RS-485 connections for Modbus RTU.
    - For Class 5 and 6 M-style motors, COM channel 0 (pins 2 and 3 on the Communication connector) provides the RS-485 connection for Modbus RTU.
- Verify that the OCHN (Open Channel) command is in a user program in the connected SmartMotor. For details, see OCHN Command on page 18.
- Verify that the SmartMotor's serial address is also used for the Modbus follower ID (i.e., both the motor address and Modbus follower ID must match). The SADDR= command is used in the program to set the SmartMotor serial address. Refer to the SADDR example in OCHN Command on page 18.
- For testing, you can use a PC as the Modbus controller along with the free utility program QModBus (http://qmodbus.sourceforge.net/). Refer to the next diagram.



Modbus RTU Communication Test Example

Although this wouldn't be used for a real application, it allows you to communicate with a SmartMotor as the Modbus RTU follower. For examples, see Modbus RTU Sample Command Sequences on page 23.

Modbus RTU Sample Command Sequences

This topic contains some sample Modbus RTU (serial) command sequences. These examples show the data sent from and received by the Modbus controller communicating with a SmartMotor. For these examples, a utility software, QModBus, is used to simulate the controller, and the SmartMotor uses Follower ID 5.

For each of these sections:

- Section title = action being performed
- SEND to motor = formatted byte stream sent from controller to the SmartMotor
- RECV from motor = formatted byte stream received by the controller from the SmartMotor

For each of these tables:

NOTE: A table is provided to illustrate the parts of the byte sequence only. The byte sequence must be transmitted as a stream of bytes shown in the SEND/RECEIVE strings above the table (i.e., no pause or null for the blank cells).

- Follower ID = device node address
- Funct = function code (see Supported Function Codes on page 19)
- Start Addr = start address in memory or single register address (see Input Registers - 3X on page 20 and Holding Registers - 4X on page 21)
- No. of Reg. = number of coils or number of registers
- Byte Cnt =byte count
- Data low word = data (low word)
- Data high word = data (high word)
- CRC = cyclic redundancy check

Read input registers (status word 3)

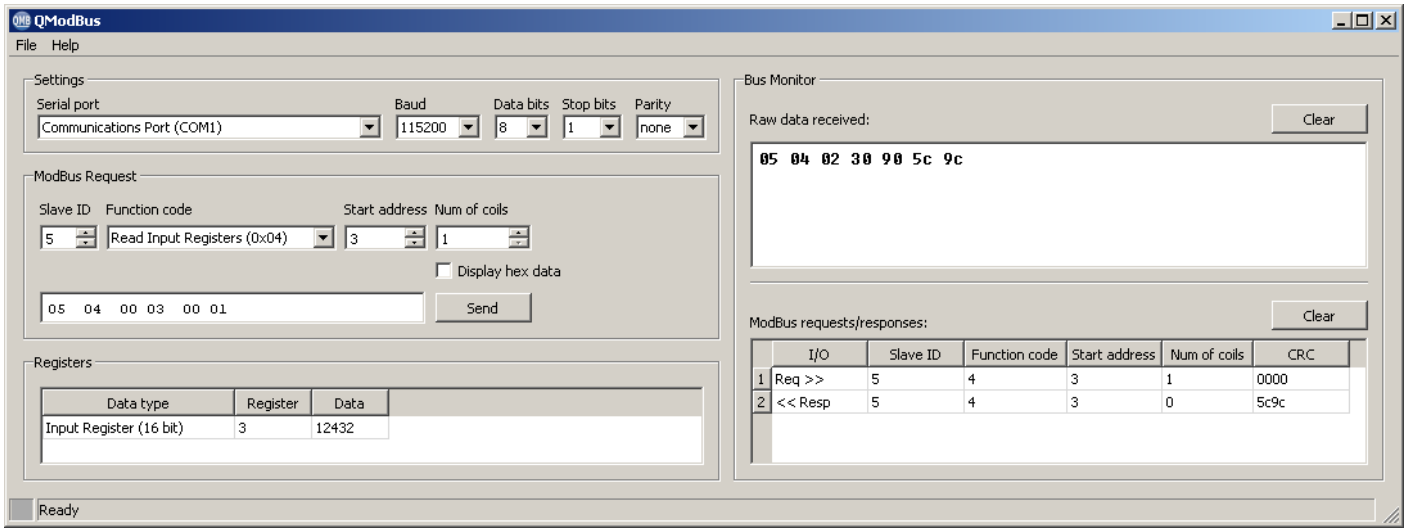
NOTE: For information on input registers, see Input Registers - 3X on page 20.

SEND to motor: 05 04 00 03 00 01 C0 4E

RECV from motor: 05 04 02 30 90 5C 9C

	Follower ID	Funct	Start Addr	No. of Reg.	Byte Cnt	Data low word	Data high word	CRC
SEND	05	04	00 03	00 01				C0 4E
RECV	05	04			02	30 90		5C 9C

A table is provided to illustrate the parts of the byte sequence only. The byte sequence must be transmitted as a stream of bytes shown in the SEND/RECEIVE strings above the table (i.e., no pause or null for the blank cells).



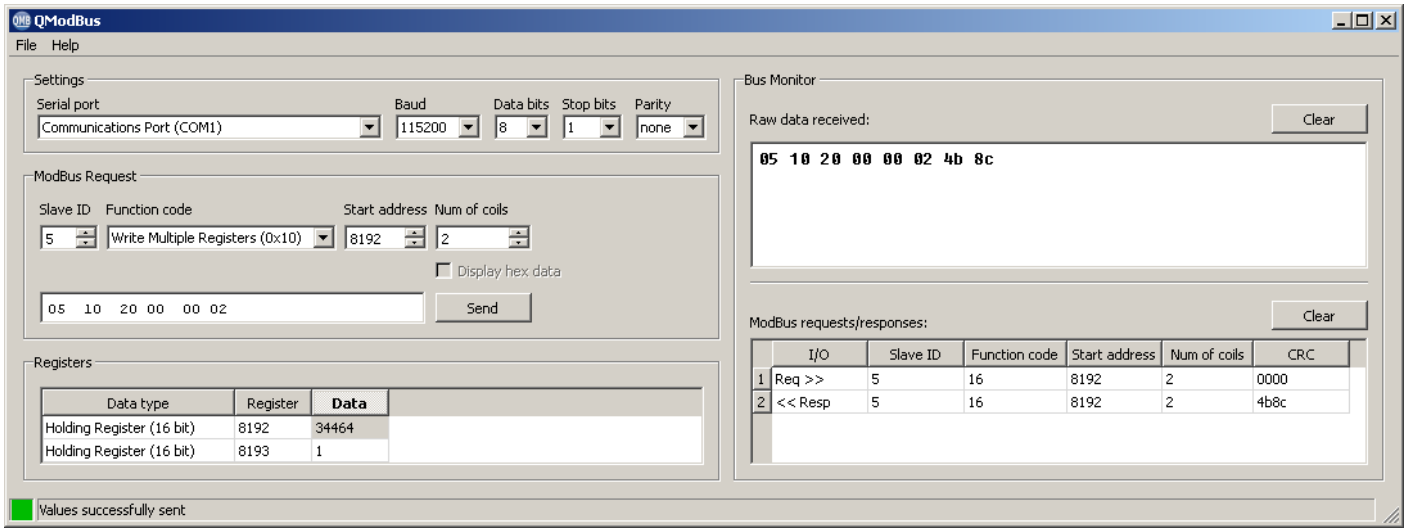
QModBus Utility Showing SEND / RECEIVE Data

Write variable "a" (a=100000)

SEND to motor: 05 10 20 00 00 02 04 86 A0 00 01 97 F4

RECV from motor: 05 10 20 00 00 02 4B 8C

	Follower ID	Funct	Start Addr	No. of Reg.	Byte Cnt	Data low word	Data high word	CRC
SEND	05	10	20 00	00 02	04	86 A0	00 01	97 F4
RECV	05	10	20 00	00 02				4B 8C
A table is provided to illustrate the parts of the byte sequence only. The byte sequence must be transmitted as a stream of bytes shown in the SEND/RECEIVE strings above the table (i.e., no pause or null for the blank cells).								



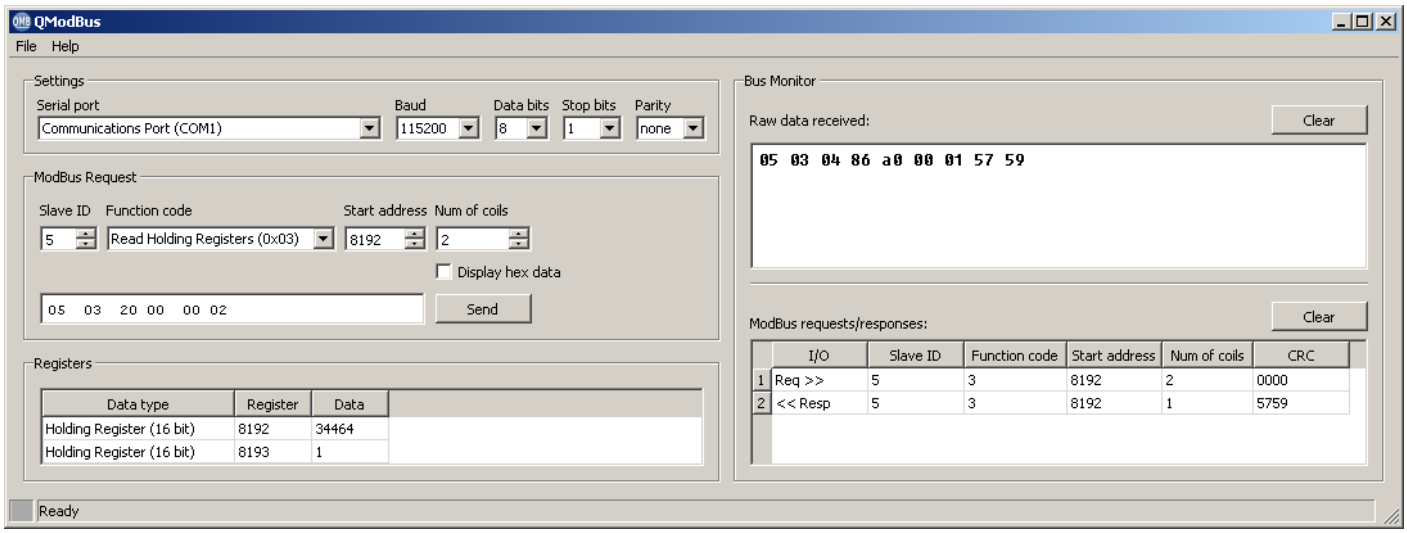
QModBus Utility Showing SEND / RECEIVE Data

Read variable "a" (value returned is 100000)

SEND to motor: 05 03 20 00 00 02 CE 4F

RECV from motor: 05 03 04 86 A0 00 01 57 59

	Follower ID	Funct	Start Addr	No. of Reg.	Byte Cnt	Data low word	Data high word	CRC
SEND	05	03	20 00	00 02				CE 4F
RECV	05	03			04	86 A0	00 01	57 59
A table is provided to illustrate the parts of the byte sequence only. The byte sequence must be transmitted as a stream of bytes shown in the SEND/RECEIVE strings above the table (i.e., no pause or null for the blank cells).								



QModBus Utility Showing SEND / RECEIVE Data

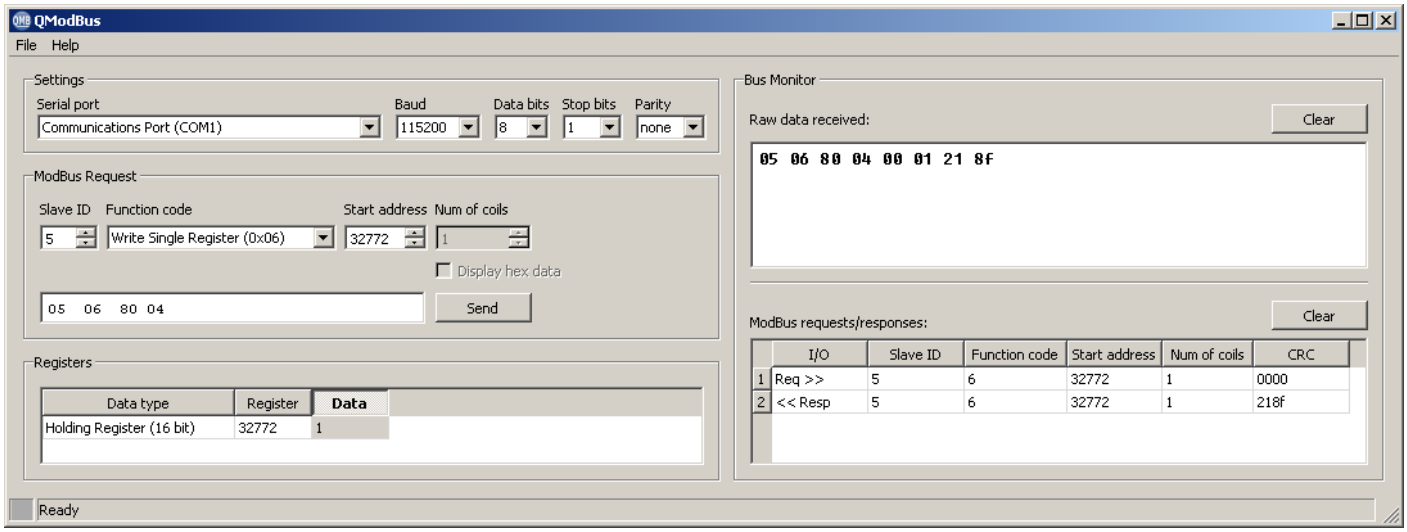
Call GOSUB(1) (Success)

NOTE: If the program label doesn't exist (it must be loaded as a user program in the motor), then the SmartMotor will return exception code 0x86 instead of the function code 0x06.

SEND to motor: 05 06 80 04 00 01 21 8F

RECV from motor: 05 06 80 04 00 01 21 8F

	Follower ID	Funct	Start Addr	No. of Reg.	Byte Cnt	Data low word	Data high word	CRC
SEND	05	06	80 04			00 01		21 8F
RECV	05	06	80 04			00 01		21 8F
A table is provided to illustrate the parts of the byte sequence only. The byte sequence must be transmitted as a stream of bytes shown in the SEND/RECEIVE strings above the table (i.e., no pause or null for the blank cells).								



QModBus Utility Showing SEND / RECEIVE Data

New Feature Details

These sections provide details on the recently added features in the Modbus RTU implementation for the Moog Animatics SmartMotor.

Read Packet Data from Modbus

Class 5 SmartMotors, with firmware versions 5.0.4.53 and later for D-style motors, and 5.98.4.53 and later for M-style motors, allow your application to read packet data from Modbus. This section provides details on that capability.

- Uses Modbus function code 0x04 “Read Input Registers” (traditional “3X” space).

  Modbus register address 256 (0-based, i.e., 3x notation is 300257).

  NOTE: For the feature of reading the packet data, this is always the starting register address regardless of the number of words desired. Starting at address 258, etc., will result in an error.

- Select number of words / registers.

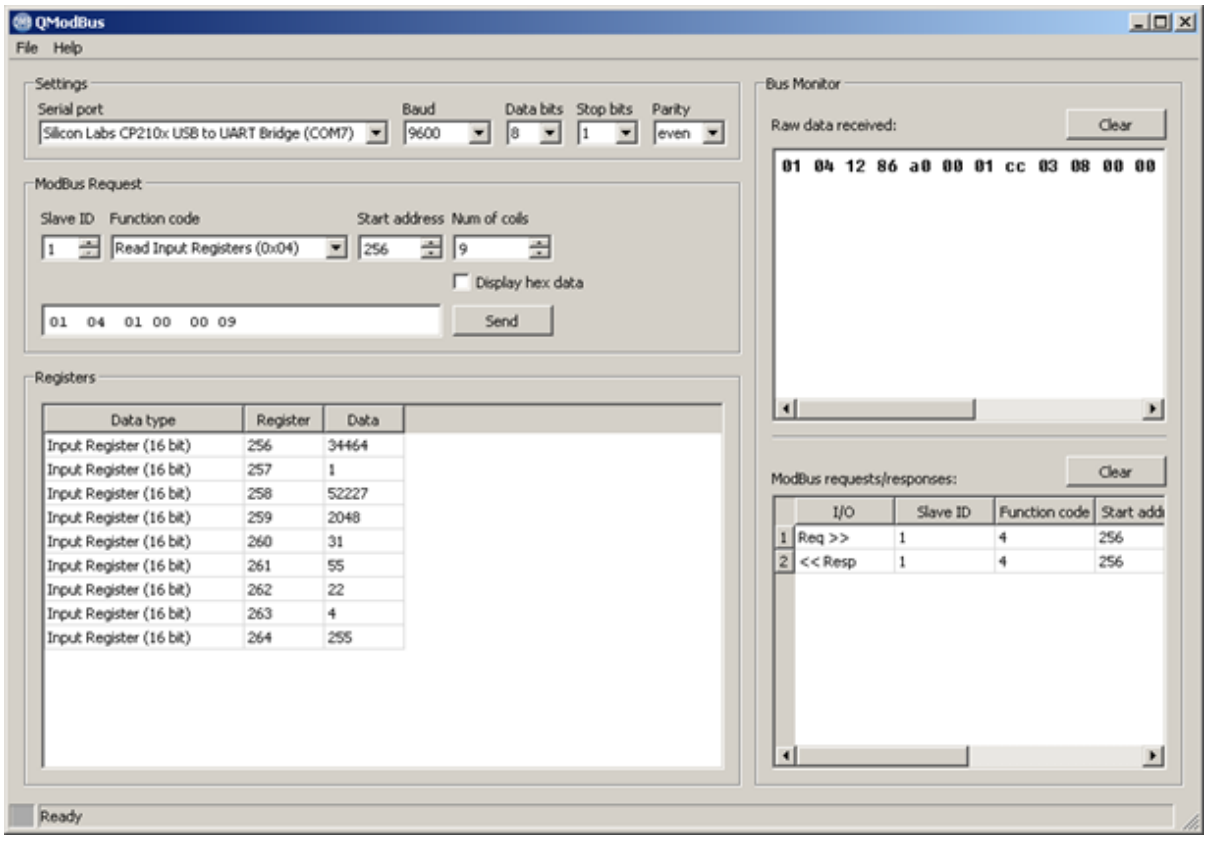
  Choose a quantity that minimizes the amount of data you need to transfer.

- Mapping of specific data elements is flexible.

  It is programmed by setting up SmartMotor user variables aw[0]-aw[31].

Example Read Using QModBus Program

QModBus is a free, open-source program that allows testing of the read feature and its configuration. However, it only allows a single event of read or write at a time and does not continuously poll. The program can be downloaded from http://qmodbus.sourceforge.net/.



QModBus Example Screen

Limitations of Read Packet Mapping

Read packet mapping has these limitations.

- There is a maximum of 16 read mappings.
- Data returned may be 16 or 32 bits. 8-bit values are not permitted. Use the word-sized equivalent commands (read RIN(W,0), or Raw[ ] respectively).
- Floating-point, double-precision registers af[ ] are not supported. See the DFS and LFS commands for loading and unloading values in a program between 32-bit integer storage and af[ ] registers.
- There is a maximum of 26 words of data due to buffer limitations.
- Mapping information can reduce the size of the returned data (e.g., truncate a 32-bit to 16-bit), but it is not allowed to increase the size.

Configuration Details

Each mapping requires 2 words in the aw[ ] array starting with aw[0] (even) and aw[1] (odd) for Mapping 0. For the list of available objects that can be mapped into the read packet, see Report Command Codes on page 43.

Even Word

This is the "mapping info" command code, desired response size, and special features.

- Desired response size allows values to be packed to a smaller size. Typically this would be because system values read as 32-bits, but often that would be a waste due to a limited expected range. For instance, temperature can be reduced to 16-bits.
- Special features:
    - Enable: every intended mapping must have bit 12 (value 4096) set to enable the mapping. When enable bit is cleared (but the rest of that mapping info word remains the same), the output data will maintain its data size (number of Modbus registers output) so that mapped items after that point still align the same way.
    - Mapping info == 8192 results in the "validate data" command bits reporting in this mapping entry. Each mapping entry has a corresponding "valid data" bit. This indicates success of the command code, and the returned data in the read packet is valid for that entry. The validate data command returns that array of bits 0 to 15 in a single Modbus register. This should be set as the final mapping entry, i.e., if 10 mappings are created, then the 11th mapping should be the valid bits, if desired. If they are not wanted, the valid bits can be omitted.

Odd Word

This is the index field typically placed in the data portion of the request.

- Some requested values require it, for example, user variable access (letter variables and al[ ], aw [ ] arrays).
- If not specially mentioned, the value for the index should be set to 0 for future compatibility.

Read Mapping Info

This table details read mapping info for even-numbered aw[ ] registers.

Bits 15 to 13	Bit 12	Bits 11 to 10	Bits 9 to 0
Control field, set to 0 for mappings	1: Enable this mapping 0: Disable this mapping Reports as the value 0, maintains the same data size compared to enabled state so that response data doesn't shift locations.	Desired response size. (Will be smaller if an object smaller than this is requested.) 00: reserved (not valid) 01: 2 bytes (1 Modbus word) 10: 4 bytes (2 Modbus words) 11: reserved (not valid)	Command code

Example: Mapping Info Bits Detail

This table provides an example of the mapping info bits detail.

Desired command	Mapping info field (aw[0], aw[2], ...aw[30])			
	Enabled (bit 12 set / +4096) Decimal value	Binary value of mapping info	Binary value of the highlighted data	Meaning
RVA	6212	0001 1000 0100 0100		
Enable		###1 #### #### ####	1	Enabled
Desired response size field (see ??)		#### 10## #### ####	10	4 bytes max requested
Command code field		#### ##00 0100 0100	00 0100 0100	Command code

Read Packet Configuration Registers

This table details the configuration registers for read packet with example results.

NOTE: aw[0] up to aw[31] are occupied by this logic — if fewer than 16 mappings are needed, then fewer aw[ ]registers are needed. Due to memory overlay, this also occupies al[0]-al[15] and ab[0]-ab [63]. Therefore, care must be taken to avoid altering these locations unintentionally, such as in a user program.

aw[ ] register	Mapping slot	Purpose	Example				
			Value (decimal)	Command	Resulting data size in read packet	Corresponds to Modbus read input register	Validity bit
aw[0]	Mapping 0	Mapping info	6164	RPA	32-bit (2 words)	Register: 256 (0x0100) Low data word of RPA	1
aw[1]		Index (16-bit)	0			Register: 257 (0x0101) High data word of RPA	
aw[2]	Mapping 1	Mapping info	2068	RPA (not enabled)	32-bit (2 words)	Register: 258 (0x0102) (value 0, not enabled)	0
aw[3]		Index (16-bit)	0			Register: 259 (0x0103) (value 0, not enabled)	
aw[4]	Mapping 2	Mapping info	5161	Raw[ ] Index to Raw[45]: 156+45 = 201	16-bit (1 word)	Register: 260 (0x0104)	1
aw[5]		Index (16-bit)	201				
aw[6]	Mapping 3	Mapping info	5620	RTEMP(0) reduced to 16-bit	16-bit (1 word)	Register: 261 (0x0105)	1
aw[7]		Index (16-bit)	0				
aw[8]	Mapping 4	Mapping info	0	Fill unused with 0	16-bit (1 word)	Register: 262 (0x0106) Value: 0 (nothing mapped or enabled)	0
aw[9]		Index (16-bit)	0				
aw[10]	Mapping 5	Mapping info	8192	Report validity data in this slot as a result of above mappings	16-bit (1 word)	Register: 263 (0x0107) Value: 29 binary: 0000 0000 0010 1101	1
aw[11]		Index (16-bit)	0				
aw[12]	Mapping 6	Mapping info	0	Fill unused with 0	16-bit (1 word)	Register: 264+ Value: 0 (nothing mapped or enabled)*	N/A**
aw[13]		Index (16-bit)	0				
. . .	Mapping...	Mapping info	0	Fill unused with 0	16-bit (1 word)	Register: 264+ Value: 0 (nothing mapped or enabled)*	N/A**
. . .		Index (16-bit)	0				
aw[30]	Mapping 15 (max pos-sible)	Mapping info	0	Fill unused with 0	16-bit (1 word)	Register: 264+ Value: 0 (nothing mapped or enabled)*	N/A**

aw[ ] register	Mapping slot	Purpose	Example				
			Value (decimal)	Command	Resulting data size in read packet	Corresponds to Modbus read input register	Validity bit
aw[31]		Index (16-bit)	0				
* There is no requirement to read these null values, they are shown to demonstrate that it is acceptable to read them anyway, such as if a fixed quantity transfer is already established and repeating like in a PLC. ** Because these items appear after the mapping assigned to the validity report, the state of these bits won't appear in the validity report.							

Example: Read Mapping Setup

This is an example Moog Animatics SmartMotor user program that will set up the read mapping. It is loaded into the motor using the SmartMotor Interface (SMI) software, which is available as a free download from the Moog Animatics website:www.animatics.com/smi.

```
x=0
aw[x]=2068+4096 aw[x+1]=0   x=x+2  ' RPA                32-bit
aw[x]=1428+4096 aw[x+1]=0   x=x+2  ' RW(0)              16-bit
aw[x]=1428+4096 aw[x+1]=1   x=x+2  ' RW(1)              16-bit
aw[x]=1524+4096 aw[x+1]=0   x=x+2  ' RTEMP as           16-bit
aw[x]=1065+4096 aw[x+1]=188 x=x+2  ' aw[32]             16-bit
aw[x]=1359+4096 aw[x+1]=1   x=x+2  ' GOSUB R2 echo as   16-bit
aw[x]=1359+4096 aw[x+1]=0   x=x+2  ' special as         16-bit
aw[x]=8192      aw[x+1]=0   x=x+2  ' validity report    16-bit
```

NOTE: +4096 sets the enable bit in each case.

Example: Data Request and Response (Byte-by-Byte)

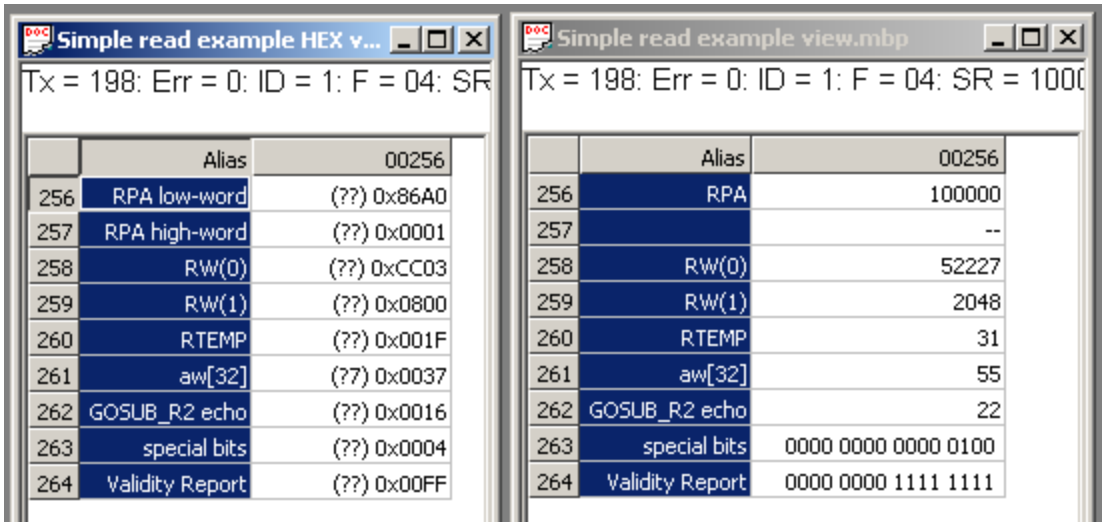
This section provides a data request and response byte-by-byte example. Use this key to understand the data parts:

- Device address 1
- Modbus function code 4
- Starting register 256 (0x0100) quantity 9 registers
- CRC is last 2 bytes of request (output) and response (input)

```
Output: 01 04 01 00 00 09 31 f0
Input: 01 04 12 86 a0 00 01 cc 03 08 00 00 1f 00 37 00 16 00 04 00 ff 9d 17
```

Resulting Data Packet

The next figures are Modbus polling software screen shots of the resulting data packet in Hex display (left) and in appropriate data formats (right).



Resulting Data Packet in Hex Display (left) and in Appropriate Data Formats (right)

Modbus Single Command Write

Class 5 SmartMotors, with firmware versions 5.0.4.53 and later for D-style motors, and 5.98.4.53 and later for M-style motors, allow your application to write packet data via Modbus. This section provides details on that capability.

- Holding register write multiple to address 512. (4x style: 400513) See Output Registers on page 41.
- Optionally, 512 and 513 can be written to set the map info/index (in aw[64], aw[65]) prior to sending data field in 514-517.
- If the write starts at 514, then the previously written values of aw[64], aw[65] are used for the map-info and index, respectively.
- Address 514 is the 'active' location that enacts the write.
- It is important to be aware of how the host/PLC processes multi-word values. For example setting VT= is 2 words. If the host / PLC environment is decoupled in the way it updates its memory for those 2 locations vs. when the Modbus write is made, then consider the one-time event control bit so that the value can be updated without a race condition. Otherwise, if the host knows it can send multi-word values with consistency, then the continuous update bit 12 can be used instead.

A Note on Status/Error Bits

Status/error bits are reflected through the "special status word", not as exceptions. Industry experts advise that exceptions are only for communication-stopping problems because many PLCs will stop, retry or do something undesired for continuous operation. Therefore, any errors generated by the attempt to write by the application are handled as status bits on which the PLC program can close the loop and not as communications exceptions.

Mapped Write Operation

For Modbus mapped write operations, configuration is handled in registers 512 and 513, and data in registers 514 and higher.

For write operations, this arrangement allows for flexibility to write the configuration information and data every Modbus write cycle or just the data. I.e., your application could start at register 514 to continuously write just the data using the existing configuration.

NOTE: aw[64] / aw[65] are used to direct how the data written in registers 514-517 is processed. Care must be taken to avoid altering these locations aw[64] and aw[65] unintentionally, such as in a user program.

Modbus register	Smart Motor Variable	Hex: 0x0### – 0xF###				Hex: 0x#0## – 0x#C##	Hex: 0x#0## – 0x#C##
		Bit 15	Bit 14	Bit 13	Bit 12	Bits 11 to 10	Bits 9 to 0
512 (0x0200) Mapping info for write single command operation.	aw[64]	Reserved	1: G command after update. This supports the common use-case of "VT=100000 G" 0: no G command added. NOTE: G not issued if the command generates an error.	One-time write on rising edge of this bit (unless bit 12 is a 1, thus overriding) Sets ack bit* See next section for specific values.	1: continuous write (overrides one-time write bit 13). Sets ack bit* 0: not continuous write, not overriding bit 13. See section 3.2 for specific values.	Data size. Exact match; otherwise, an error is thrown.	Command code

Modbus register	Smart Motor Variable	Hex: 0x0### – 0xF###				Hex: 0x#0## – 0x#C##	Hex: 0x#0## – 0x#C##
		Bit 15	Bit 14	Bit 13	Bit 12	Bits 11 to 10	Bits 9 to 0
513 (0x0201) Index info for write single command operation.	aw[65]	Index value (16-bit) Typically for user variables or command codes that involve an array of values.					
514-517 (0x0202-0x0205)	N/A	Data to be written. (Single value written according to aw[64] and aw[65])					
* See special status word for ack bit in section 0.							

See Example: Host to Call "VT=100000 G" on page 47.

Handling of Enable Bits 12 and 13 for Write Process

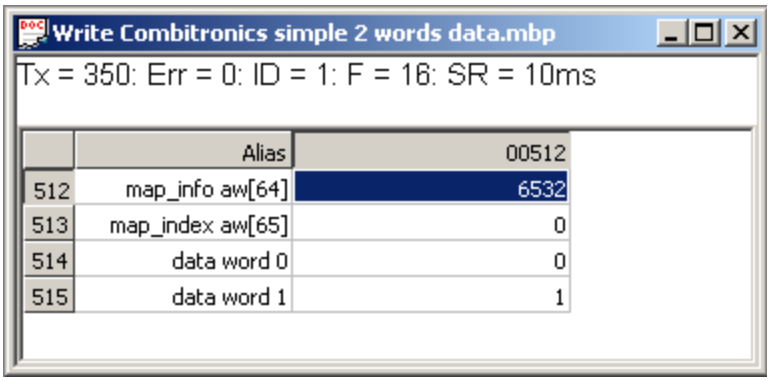
This table provides details on how enable bits 12 and 13 are handled for the write process.

Bit 13	Bit 12	Resulting condition
x (don't care)	1	Data written to Modbus registers 514-517 is accepted every cycle that it is written (ack bit in special status 1 at this time)
0	0	No action (ack bit in special status is 0 at this time)
0 -> 1 (rising edge)	0	Data written to Modbus registers 514-517 is accepted on this cycle that it is written (ack bit in special status shows rising edge 0 -> 1 in response at this time)
1	0	No action (ack bit in special status remains 1 at this time)
1 -> 0 (falling edge)	0	No action (ack bit in special status shows falling edge 1 -> 0 in response at this time).

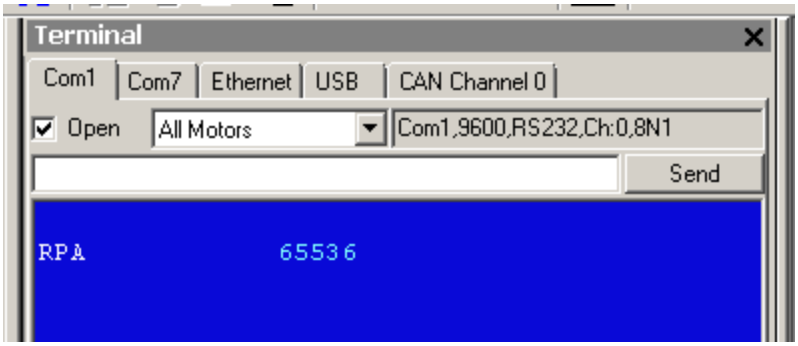
Example

These figures provide a simple example of a two-word (high word and low word) write and the resulting change.

- Write command code for 0=value (32-bit) Continuous write enabled (bit12).



- High word = 1, low word= 0. I.e., 0x00010000 is 65536 decimal. Note that RPA has changed as a result of writing 0=65536.



GOSUB R2 through Modbus

Class 5 SmartMotors, with firmware versions 5.0.4.53 and later for D-style motors, and 5.98.4.53 and later for M-style motors, allow your application to use subroutines (GOSUB R2) through Modbus. This section provides details on that capability.

- To perform more complex operations, it may be necessary to use a subroutine in a user program instead of directly writing the single command code.
- Writing holding registers 576-592 allows a number of aw[ ] registers to first be written, then a GOSUB call to a specific program label to be called. This effectively acts like a function call where the aw[ ] registers are input arguments. The subroutine can apply these values any way it chooses.
- The write may be a single (or multiple write of length 1) to the GOSUB R2 register 592, or may initiate a multiple register write starting at any point in the range 576-592. The purpose is to allow a flexible number of input values to the routine while minimizing transmission time overhead. Writing location 593 by any means will result in an error.
- GOSUB R2 offers specific protections against stack overflow. Once GOSUB R2 is called, it must complete and exit before it can be called again. Therefore, the routine must not contain endless loops unless there is no intent to call it a second time.
- By default, GOSUB R2 requires that the value -1 (65535) be written between calls to the desired subroutine. This defends against PLC systems that cyclically write to the register, but their application logic does not intend the routine to be called that often (and they don't have control over the transmission cycle time). That allows the PLC application program to control the number of times the routine is called.
- There is a special command to remove the requirement of this value transition. CANCTL(19,1) will set this mode, so that every write to address 592 will attempt to call that subroutine number. However, it will still be ignored if a subroutine is currently running. CANCTL(19,0) will return to the default mode, which requires writing value -1 between GOSUB calls.
- Status/error bits are reflected through the "special status word" instead of exceptions (industry advice from experts is that exceptions are only for communication-stopping problems because many PLCs will stop, retry or do something undesired for continuous operation. Application-level conditions should be handled with read registers/status bits that the PLC program can close the loop on.

  NOTE: Label C0 should be avoided if there is any chance that the PLC will initially and unintentionally write the value 0 to that location. Doing so would cause C0 to be executed!
- In a cyclic PLC environment, one way to control the pace of the logic and determine if the GOSUB number has been written to the motor is to include the GOSUB R2 readback echo command code as one of the mapped objects.
  - aw[x]=1359+4096 aw[x+1]=1 x=x+2 ' GOSUB R2 echo as 16-bit

GOSUB R2 Procedure

This section describes the steps for using GOSUB R2 in your Modbus application.

1. Set 592 (GOSUB R2) to -1.
2. Check the value of the GOSUB R2 echo that is mapped from that location.

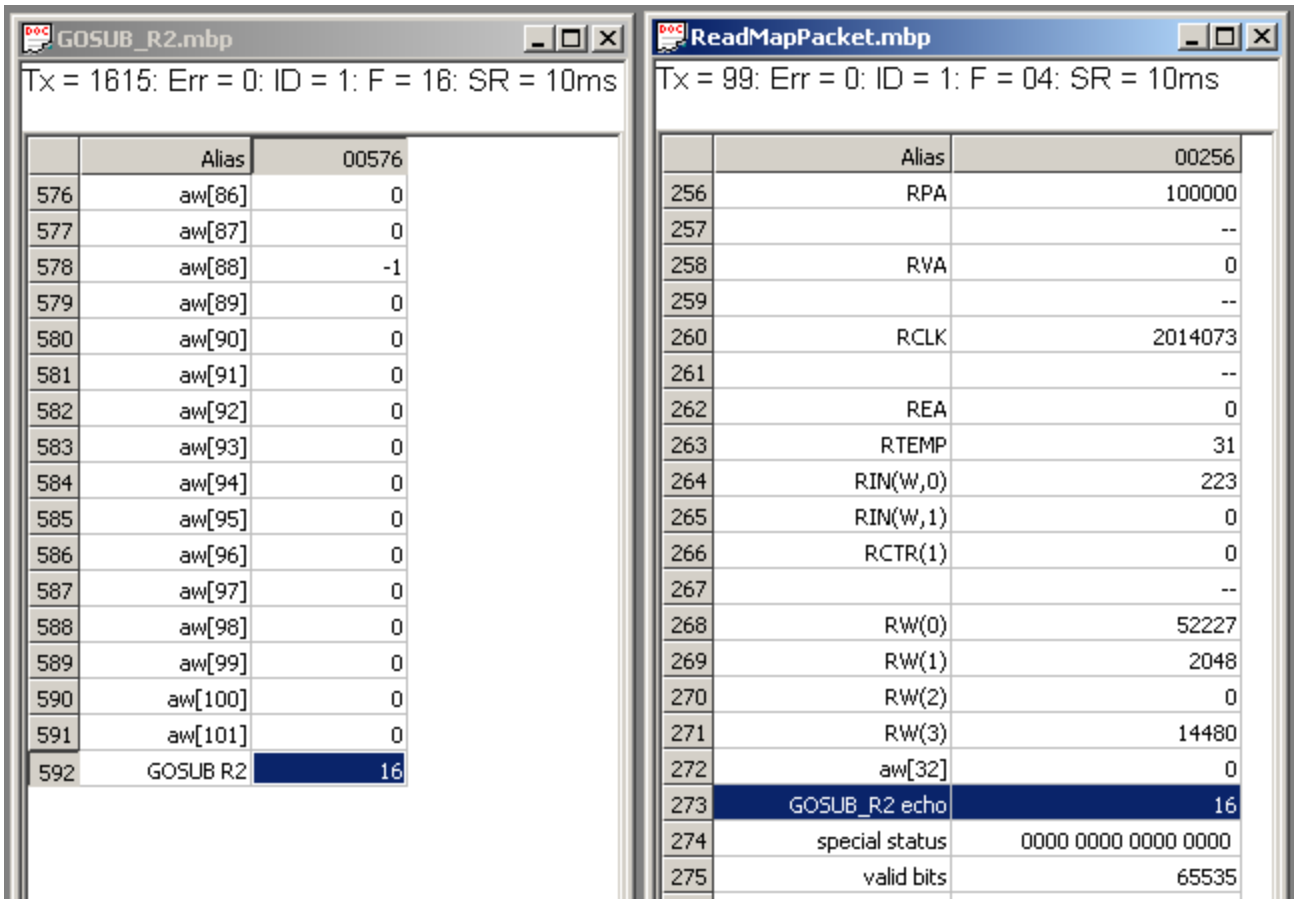
   Wait until the reported value is -1, then proceed with the next step.
3. Load Modbus register locations 576-591, as desired, to prepare the input data for the subroutine. Any subrange of registers can be written within that range.
4. Set 592 (GOSUB R2) to the desired subroutine (16 in this example).
5. Check the value of the GOSUB R2 echo from that location that is mapped.

   Wait until the reported value is 16, then proceed with the next step.
6. Repeat from step 1 to call the GOSUB again.

Example: GOSUB R2

These figures provide an example of GOSUB R2. The write packet is in the left pane, and the read packet is in the right pane.

PLC simulator software was used for this example, which can be obtained from: https://www.modbustools.com/modbus_poll.html.



GOSUB_R2.mbp

Tx = 1615: Err = 0: ID = 1: F = 16: SR = 10ms

	Alias	00576
576	aw[86]	0
577	aw[87]	0
578	aw[88]	-1
579	aw[89]	0
580	aw[90]	0
581	aw[91]	0
582	aw[92]	0
583	aw[93]	0
584	aw[94]	0
585	aw[95]	0
586	aw[96]	0
587	aw[97]	0
588	aw[98]	0
589	aw[99]	0
590	aw[100]	0
591	aw[101]	0
592	GOSUB R2	16

ReadMapPacket.mbp

Tx = 99: Err = 0: ID = 1: F = 04: SR = 10ms

	Alias	00256
256	RPA	100000
257		--
258	RVA	0
259		--
260	RCLK	2014073
261		--
262	REA	0
263	RTEMP	31
264	RIN(W,0)	223
265	RIN(W,1)	0
266	RCTR(1)	0
267		--
268	RW(0)	52227
269	RW(1)	2048
270	RW(2)	0
271	RW(3)	14480
272	aw[32]	0
273	GOSUB_R2 echo	16
274	special status	0000 0000 0000 0000
275	valid bits	65535

GOSUB R2 Write Packet (left) and Read Packet (right)

Also, see the Read Mapping Setup on page 45 for read packet configuration in a SmartMotor user program.

Special status word (16-bits)

This reports conditions of the GOSUB R2 feature and the single write command operation.

Example: Mapping Setup

This SmartMotor code provides an example mapping setup for special status words (16-bit).

```
aw[x]=1359+4096  aw[x+1]=0  x=x+2        ' Special as 16-bit
```

Register Bitwise Description

Refer to the next table for the register bitwise description.

Bit 15 to 4	Bit 3	Bit 2	Bit 1	Bits 0
Reserved (reports as 0)	Modbus write (register 512+) error Set when command activated (by continuous bit or rising edge of one-time event) and that command resulted in error. Clear error bit by clearing continuous enable and one-time bit. Also cleared in continuous mode when a command completes successfully. However, best practice is to clear the enable bit during any changes to mapping info command, index or value.	GOSUB R2 error: unknown program label, etc. Does not report if cyclic protection or nesting protections are preventing call. Only updated when GOSUB R2 invoked, not cleared automatically, i.e., requires a successful call to GOSUB R2 to clear bit.	Modbus write (register 512+) ack Either writing continuously if commanded, or rising edge detected on one-time command bit and waiting for that bit to clear.	GOSUB R2 busy, has not returned yet.

SmartMotor Modbus Register Space

This section provides details about the Modbus register space in the SmartMotor.

Input Registers

This table describes the Input registers, also referred to as "3x space" in Modbus terminology. Note that these 0-based registers are read-only.

Register Address (0-based)		Modbus function code	
Hex	Decimal		
0x0000	0	Read Input register Modbus function code 0x04 (4 decimal)	Status word RW(0)
0x0001	1		Status word RW(1)
0x0002	2		Status word RW(2)
0x0003	3		Status word RW(3)
0x0004	4		Status word RW(4)
0x0005	5		Status word RW(5)
0x0006	6		Status word RW(6)
0x0007	7		Status word RW(7)
0x0008	8		Status word RW(8)
0x0009	9		Status word RW(9)
0x000A	10		Status word RW(10)
0x000B	11		Status word RW(11)
0x000C	12		Status word RW(12)
0x000D	13		Status word RW(13)
0x000E	14		Status word RW(14)
0x000F	15		Status word RW(15)
0x0010	16		Status word RW(16)
0x0011	17		Status word RW(17)
0x0012 - 0x007F	18-127		N/A reports as 0
0x0080 - 0x00FF	*128 - 255*	*Read Input register Modbus function code 0x04 (4 decimal)*	*Not used (returns error if accessed)*
0x0100	**256**	Read Input register Modbus function code 0x04 (4 decimal)	**Read packed data start**
0x0101 - 0x017F	*257-383*	*Read Input register Modbus function code 0x04 (4 decimal)*	*Reserved for read data packet Do not start at one of these addresses; will return error. A read can continue from above address.*
0x0180 - 0xFFFF	*384-65535*	*Read Input register Modbus function code 0x04 (4 decimal)*	*Not used (returns error if accessed)*

Output Registers

This table describes the Output registers, also referred to as "4x space" in Modbus terminology. Note that these registers are 0-based.

Register Address (0-based)		Modbus function code	
Hex	Decimal		
0x0000-0x01FF	*0-511*	*Holding register* *Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)*	*Not used (returns error if accessed)*
0x0200	512	Write Holding register (write-only) Modbus function codes: 0x06, 0x10 (6, 16 decimal)	Write to aw[64] with the write map info.
0x0201	513		Write to aw[65] with the write map index.
0x0202	514		Write data packet start
0x0203-0x0205	515-517		*Reserved for read data packet Do not start at one of these addresses; will return error. A 'write multiple' can continue from*

Register Address (0-based)		Modbus function code	
Hex	Decimal		
			above address.
0x0206-0x023F	*518-575*	*Holding register* *Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)*	*Not used (returns error if accessed)*
0x0240	576	Write Holding register (write-only) Modbus function codes: 0x06, 0x10 (6, 16 decimal)	Write to **aw[86]** for the purpose of GOSUB using this data.
0x0241	577		Same as above, but write to **aw[87]**
0x0242	578		Same as above, but write to **aw[88]**
0x0243	579		Same as above, but write to **aw[89]**
0x0244	580		Same as above, but write to **aw[90]**
0x0245	581		Same as above, but write to **aw[91]**
0x0246	582		Same as above, but write to **aw[92]**
0x0247	583		Same as above, but write to **aw[93]**
0x0248	584		Same as above, but write to **aw[94]**
0x0249	585		Same as above, but write to **aw[95]**
0x024A	586		Same as above, but write to **aw[96]**
0x024B	587		Same as above, but write to **aw[97]**
0x024C	588		Same as above, but write to **aw[98]**
0x024D	589		Same as above, but write to **aw[99]**
0x024E	590		Same as above, but write to **aw[100]**
0x024F	591		Same as above, but write to **aw[101]**
0x0250	592		GOSUB R2
0x0251-0x1FFF	*593-8191*	*Holding register* *Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)*	*Not used (returns error if accessed)*
0x2000-0x2101	8192-8449	Holding register Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)	Variables a-zzz, aw[0]-aw[101]
0x2102-0x8003	*8450-32771*	*Holding register* *Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)*	*Not used (returns error if accessed)*
0x8004	32772	Holding register Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)	GOSUB (deprecated implementation)
0x8005-0xFFFF	*3277365535*	*Holding register* *Modbus function codes: 0x03, 0x06, 0x10 (3, 6, 16 decimal)*	*Not used (returns error if accessed)*

Report Command Codes

This table provides a list of the available report command codes.

Desired report	Mapping info field Command code (includes size field) (aw[0], aw[2], ...aw[30])		Index field (aw[1],aw[3], ...aw[31])	Resulting data size	Notes
	Enabled (bit 12 set) i.e., 4096 added in	Disabled			
RPA	6164	2068	0	32-bit (2 Modbus registers)	
RVA	6212	2116	0	32-bit (2 Modbus registers)	
RCLK	6820	2724	0	32-bit (2 Modbus registers)	
RCTR(0)	6532	2436	0	32-bit (2 Modbus registers)	
RCTR(1)	6532	2436	1	32-bit (2 Modbus registers)	
RPC(0)	6196	2100	0	32-bit (2 Modbus registers)	
RPC(1)	6196	2100	1	32-bit (2 Modbus registers)	
RPC(2)	6196	2100	2	32-bit (2 Modbus registers)	
RIN(W,0)	5138	1042	0	16-bit (1 Modbus register)	Physical inputs 0-15.
RIN(W,1)	5138	1042	1	16-bit (1 Modbus register)	-AD1 option on D-series physical inputs 16-31.
RTEMP as 16-bit	5620	1524	0	16-bit (1 Modbus register)	
RTEMP as 32-bit	6644	2548	0	32-bit (2 Modbus registers)	
REA as 16-bit	5396	1300	0	16-bit (1 Modbus register)	
REA as 32-bit	6420	2324	0	32-bit (2 Modbus registers)	
RW(reg) i.e., RW(0) – RW(17)	5524	1428	reg (0-17)	16-bit (1 Modbus register)	
Raw[reg] i.e., Raw[0] – Raw[101]	5161	1065	156+reg (156-257)	16-bit (1 Modbus register)	Offset required, i.e., aw[0] is index 156.
Ral[reg] i.e., Ral[0] – Ral[50]	6217	2121	78+reg (78-128)	32-bit (2 Modbus registers)	Offset required, i.e., al[0] is index 78.
Ra	6217	2121	0	32-bit (2 Modbus registers)	Examples of the 32-bit "letter variables" Where index is: a=0, b=1, … z=25 aa=26, ... zz=51 aaa=52, … zzz=77
Rb	6217	2121	1	32-bit (2 Modbus registers)	
Raa	6217	2121	26	32-bit (2 Modbus registers)	
Raaa	6217	2121	52	32-bit (2 Modbus registers)	
Rzzz	6217	2121	77	32-bit (2 Modbus registers)	
Special status register	5455	1359	0	16-bit (1 Modbus register)	
GOSUB R2 echo	5455	1359	1	16-bit (1 Modbus register)	
Valid bits/validity report	N/A see notes	8192	0	16-bit (1 Modbus register)	Special case. Does not require enable bit.
RUIA	6612	2516	0	32-bit (2 Modbus registers)	
RUJA	6676	2580	0	32-bit (2 Modbus registers)	
RINA(V1, input_line)	5141	1045	input_line (0-6)	16-bit (1 Modbus register)	Analog 5 volt I/O D-series
RINA(A, input_line)	5173	1077	input_line (0-6)	16-bit (1 Modbus register)	Analog raw format (5 volt I/O D-series)
RINA(A, input_line)	5173	1077	input_line (16-25)	16-bit (1 Modbus register)	Analog raw format (AD1 option 24 volt I/O D-series)
RINA(V, input_line)	5125	1029	input_line (16-25)	16-bit (1 Modbus register)	(AD1 option 24 volt I/O D-series)

Write Command Codes

This table provides a list of the available command codes to write to the SmartMotor.

Desired command	Mapping info field Command code* (aw[64])	Index field (aw[65])	<value> starts at Modbus register address 514 Requires data size	Notes
O=<value>	2436	0	32-bit (2 Modbus registers)	
O(0)=<value>	2436	0	32-bit (2 Modbus registers)	
O(1)=<value>	2436	1	32-bit (2 Modbus registers)	
O(2)=<value>	2436	2	32-bit (2 Modbus registers)	
OSH=<value>	2644	0	32-bit (2 Modbus registers)	
OSH(0)= <value>	2644	0	32-bit (2 Modbus registers)	
OSH(1)= <value>	2644	1	32-bit (2 Modbus registers)	
OSH(2)= <value>	2644	2	32-bit (2 Modbus registers)	
OUT(W,0,mask)=outputstate	2066	0	32-bit (2 Modbus registers)	Modbus register 514 = output state (high/low) for all 16 bits, and Modbus register 515 = output mask for all 16 bits
OUT(W,1,mask)=outputstate	2066	1		
VT=<value>	2132	0	32-bit (2 Modbus registers)	
PT=<value>	2084	0	32-bit (2 Modbus registers)	
PRT=<value>	2260	0	32-bit (2 Modbus registers)	
T=<value>	2420	0	32-bit (2 Modbus registers)	
ADT=<value>	2180	0	32-bit (2 Modbus registers)	
AT=<value>	2196	0	32-bit (2 Modbus registers)	
DT=<value>	2212	0	32-bit (2 Modbus registers)	
MP	3060	20	32-bit (2 Modbus registers)	Write data as 0.
MV	3060	21	32-bit (2 Modbus registers)	Write data as 0.
MT	3060	22	32-bit (2 Modbus registers)	Write data as 0.
G	3060	2	32-bit (2 Modbus registers)	Write data as 0.
X	3060	3	32-bit (2 Modbus registers)	Write data as 0.
OFF	3060	0	32-bit (2 Modbus registers)	Write data as 0.
MTB	3060	1	32-bit (2 Modbus registers)	Write data as 0.
ZS	3060	19	32-bit (2 Modbus registers)	Write data as 0.
Z	3060	25	32-bit (2 Modbus registers)	Write data as 0.
RUN	3060	31	32-bit (2 Modbus registers)	Write data as 0.
END	3060	30	32-bit (2 Modbus registers)	Write data as 0.
GOSUB	-	-	-	See feature GOSUB R2 through Modbus on page 37.
MP(<Value>)	3060	20	32-bit (2 Modbus registers)	Allowed values: 0 or 1
MV(<Value>)	3060	21	32-bit (2 Modbus registers)	Allowed values: 0 or 1
G(<value>)	3060	2	32-bit (2 Modbus registers)	Allowed values: 0, 1, 2
X(<value>)	3060	3	32-bit (2 Modbus registers)	Allowed values: 0, 1, 2
Z(word,bit)	2452	word (0-10)	32-bit (2 Modbus registers)	Bit number is written in data field (Modbus register 514-515)
a=<value>	2121	0	32-bit (2 Modbus registers)	
b=<value>	2121	1	32-bit (2 Modbus registers)	
aa=<value>	2121	26	32-bit (2 Modbus registers)	
aaa=<value>	2121	52	32-bit (2 Modbus registers)	
zzz=<value>	2121	77	32-bit (2 Modbus registers)	
al[reg]= <value> i.e., al[0]=, though al[50]=	2121	78 + reg(78-128)	32-bit (2 Modbus registers)	Example: al[25]=<value>, index field is: 78+25 = 103
aw[reg]=<value> i.e., aw[0]=, though aw[101]=	1065	156 + reg (156-257)	16-bit (1 Modbus register)	Example: aw[39]=<value>, index field is: 156+39 = 195
UO(W,0,mask)=state	2098	0	32-bit (2 Modbus registers)	Modbus register 514 = state (high/low) for all 16 bits, and Modbus register 515 = mask for all 16 bits
UO(W,1,mask)=state	2098	1	32-bit (2 Modbus registers)	

*These values include a size field. However, they do not include write enable bits 12 or 13, or bit 14. Add those in addition to the value shown in the table for command code. I.e., bit 12 is +4096, bit 13 is +8192, bit 14 is +16384.

This section shows a more detailed mapping example.

NOTE: Many other mappings are possible. This is just a demonstration that will be used for the examples shown later in this section.

Read Mapping Setup

This is an example Moog Animatics SmartMotor user program that will set up the read mapping. It is loaded into the motor using the SmartMotor Interface (SMI) software, which is available as a free download from the Moog Animatics website: www.animatics.com/smi.

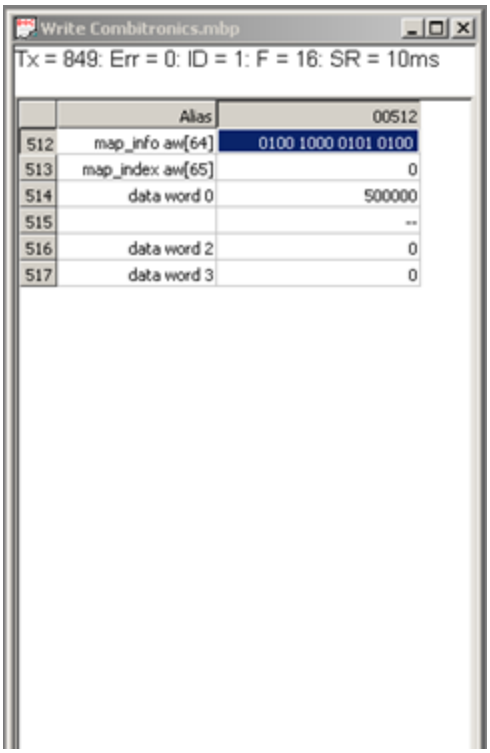
```
x=0
aw[x]=2068+4096 aw[x+1]=0   x=x+2  ' RPA                32-bit
aw[x]=2116+4096 aw[x+1]=0   x=x+2  ' RVA                32-bit
aw[x]=2724+4096 aw[x+1]=0   x=x+2  ' RCLK               32-bit
aw[x]=1300+4096 aw[x+1]=0   x=x+2  ' REA as             16-bit
aw[x]=1524+4096 aw[x+1]=0   x=x+2  ' RTEMP as           16-bit
aw[x]=1042+4096 aw[x+1]=0   x=x+2  ' RIN(W,0)           16-bit
aw[x]=1042+4096 aw[x+1]=1   x=x+2  ' RIN(W,1)           16-bit
aw[x]=2436+4096 aw[x+1]=1   x=x+2  ' RCTR(1)            32-bit
aw[x]=1428+4096 aw[x+1]=0   x=x+2  ' RW(0)              16-bit
aw[x]=1428+4096 aw[x+1]=1   x=x+2  ' RW(1)              16-bit
aw[x]=1428+4096 aw[x+1]=2   x=x+2  ' RW(2)              16-bit
aw[x]=1428+4096 aw[x+1]=3   x=x+2  ' RW(3)              16-bit
aw[x]=1065+4096 aw[x+1]=188 x=x+2  ' aw[32]             16-bit
aw[x]=1359+4096 aw[x+1]=1   x=x+2  ' GOSUB R2 echo      16-bit
aw[x]=1359+4096 aw[x+1]=0   x=x+2  ' special            16-bit
aw[x]=8192      aw[x+1]=0   x=x+2  ' validity report    16-bit
```

NOTE: +4096 sets the enable bit in each case.

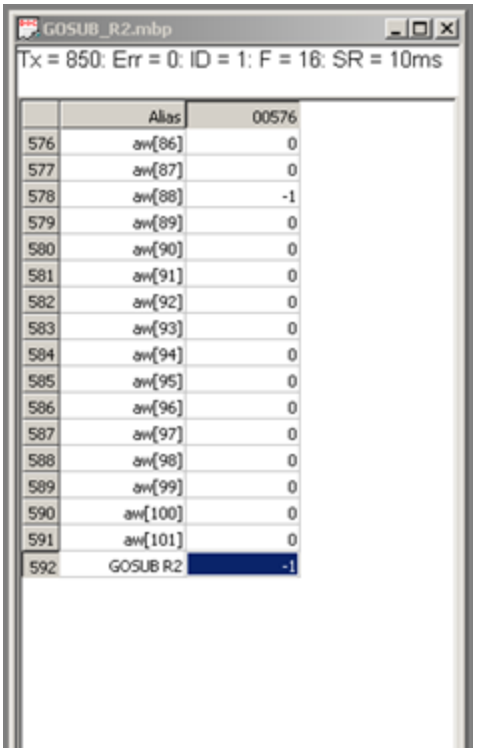
PLC Simulator

The Modbus Poll PLC simulator, available from https://www.modbustools.com/modbus_poll.html, shows a setup configured to cyclically write to the single command code write area (write multiple holding registers 512-517), the GOSUB R2 area (write multiple holding registers 576-592), and reads the mapped/packet data area (input registers starting with 256, data ends at 275 in this case). See the next figures.

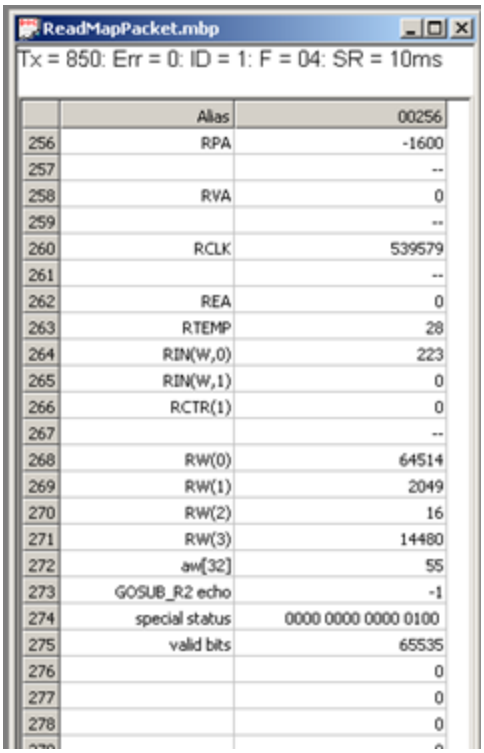
- Write multiple holding registers 512-517:



- Write multiple holding registers 576-592:



- Read input registers 256-275:



	Alias	00256
256	RPA	-1600
257		--
258	RVA	0
259		--
260	RCLK	539579
261		--
262	REA	0
263	RTEMP	28
264	RIN(W,0)	223
265	RIN(W,1)	0
266	RCTR(1)	0
267		--
268	RW(0)	64514
269	RW(1)	2049
270	RW(2)	16
271	RW(3)	14480
272	aw[32]	55
273	GOSUB_R2 echo	-1
274	special status	0000 0000 0000 0100
275	valid bits	65535
276		0
277		0
278		0

Example: Host to Call "VT=100000 G"

The next example describes how to configure the host to call "VT=100000 G". For this example:

- Refer to the above figures where memory locations in the PLC (shown in the left-side column in each window) correspond to the Modbus registers that will be written or read on the next Modbus update cycle. An actual PLC system may employ other memory locations depending on various factors, including multiple Modbus devices, etc.
- The update cycles write/read the entire blocks of addresses as a "write multiple" or "read multiple" operation, as those are typically configured to a fixed range during PLC operation. Therefore, this example works regardless of synchronization between PLC program memory writes/reads and Modbus request/response cycles.

PLC Programming Steps

To program the PLC, use these steps:

1. Write location 512 = 0
2. Write location 513 = 0
3. Read special bits (mapped to location 274 in this particular example), confirm ack (bit 1) is 0.

   If not 0, wait for next update cycle. The fact that location 512 was written to a 0 will eventually be written to the motor, then the next read of the mapped packet back to PLC will have the updated ack bit.

4. Write data 0x86A0 to location 514, write data 0x0001 to location 515. This prepares the value 100000 (decimal)

   It is OK if part or all of this is sent in the next Modbus write cycle. It does not have immediate effect because the enable bits as a part of location 512 are 0.

5. Write location 512 = 26708 (0x6854). This sets the one-time event bit and the "G" flag bit, and the command is VT= (0x854).

6. Read special bits (mapped to location 274), confirm ack (bit 1) is 1. Wait for update write/read exchange until this ack occurs.

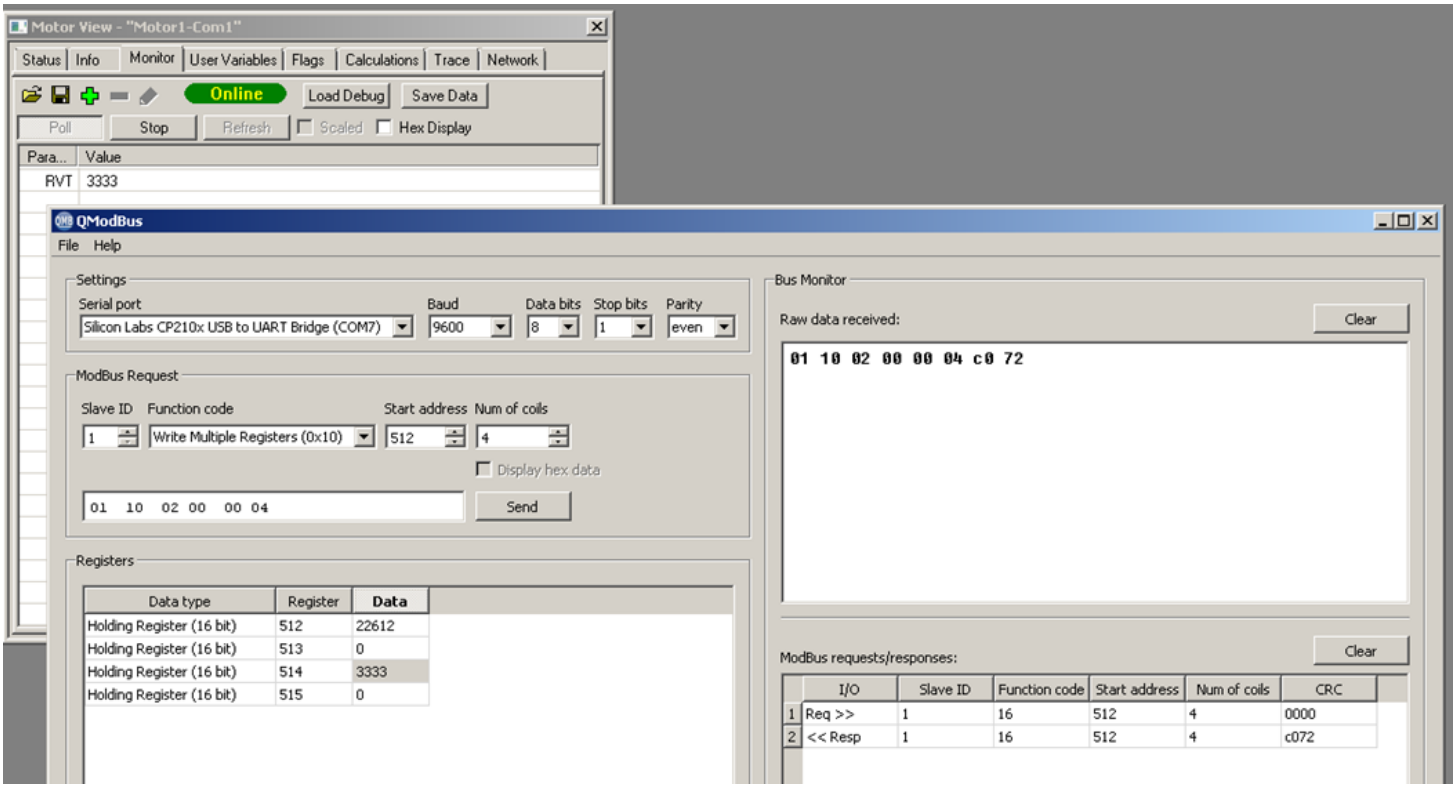
7. Proceed with other commands starting from step 1 above (where location 512 is set to 0).

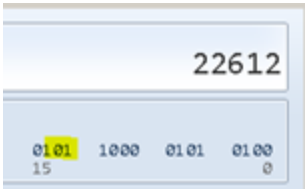
Explicit Command Example

In some systems, the Modbus host/control application may have total control of the time at which commands are written, and may also be able to control the range of registers written to on the fly. Therefore, this example gives a more direct way rather than the "PLC method", which would typically be less flexible in that regard.

When immediate action is desired, only send the "write multiple holding registers" Modbus function at the time the action is desired, and refrain from writing when no action is desired.

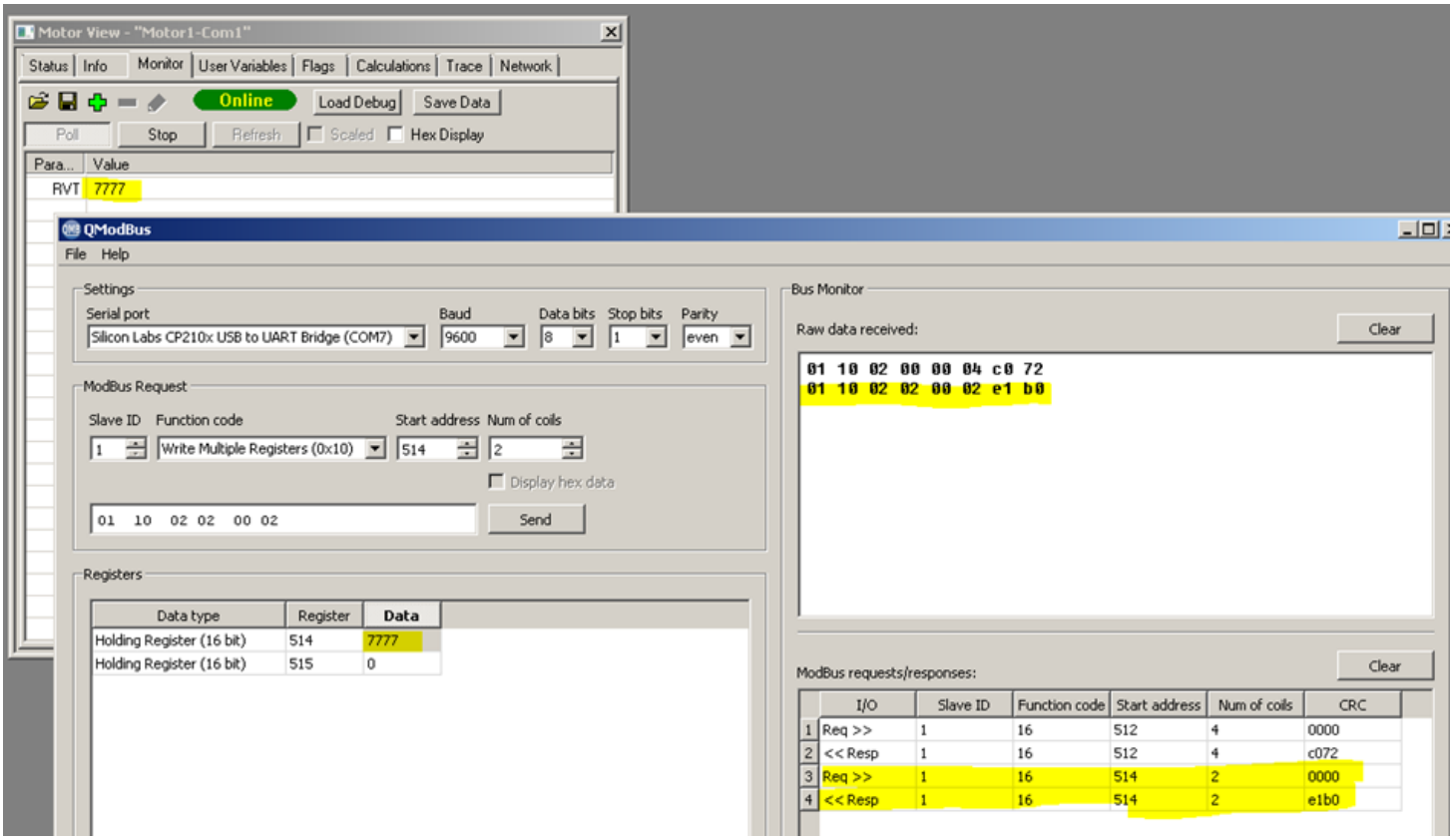
The "enable continuous" bit is provided in the Modbus info register (512), bit 12. This has an immediate effect when data in Modbus register 514+ are written.





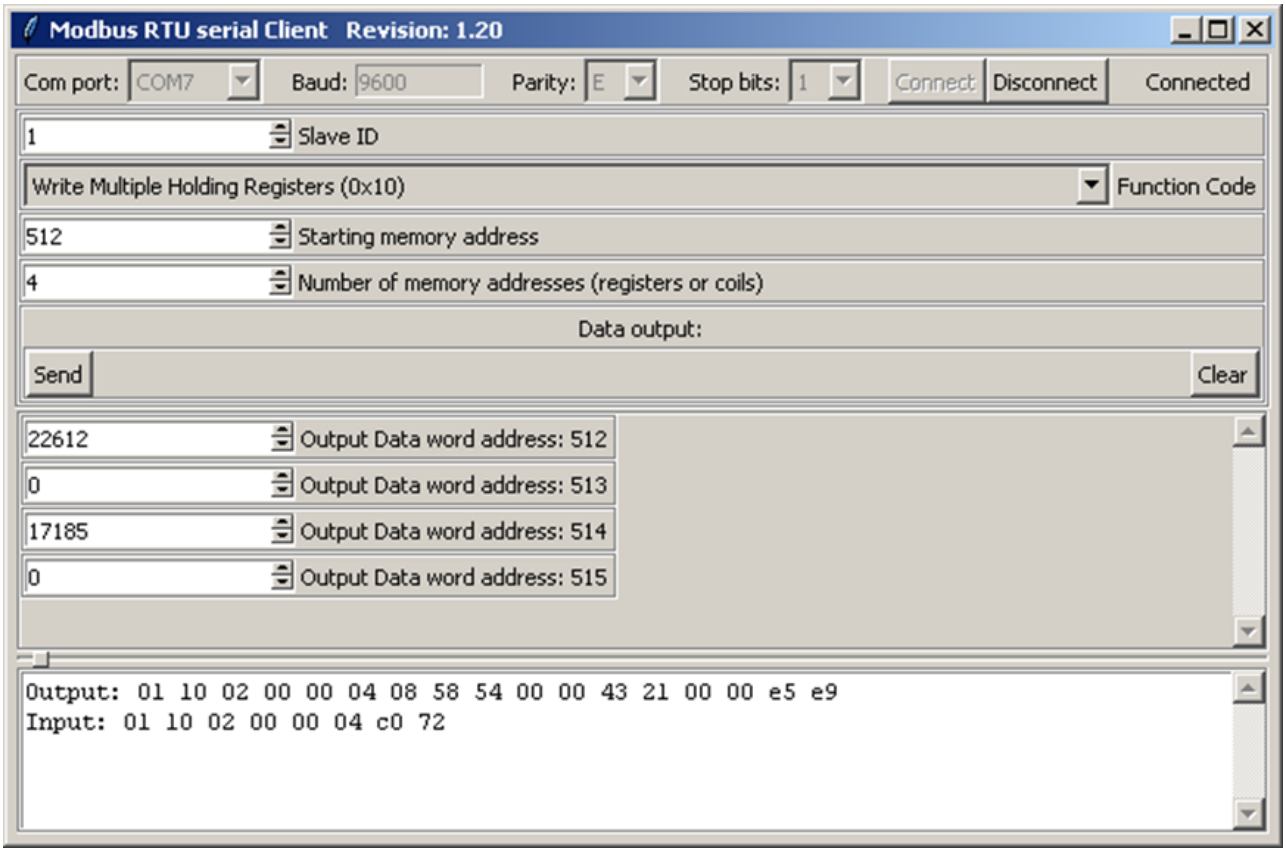
Once the Modbus addresses 512 and 513 are written, it is optionally permitted to simply write the required amount of data to locations 514 – 517. This shortens the amount of data written, and keeps reusing the map information previously written to aw[64], aw[65] (Modbus locations 512 and 513).

For example, in this case, VT= is a 32-bit value. Therefore 2 x 16-bit words must be written at 514 and 515. These must be written in a single "write multiple" Modbus function. Location 514 is the least-significant word. The write cannot begin at 515 or higher. Writing address 514 is the "trigger" to accept the rest of the data and take action.


Motor View - "Motor1-Com1"
Status
Info
Monitor
User Variables
Flags
Calculations
Trace
Network
Online
Load Debug
Save Data
Poll
Stop
Refresh
Scaled
Hex Display
Para...
Value
RVT
7777
QModBus
File
Help
Settings
Serial port
Silicon Labs CP210x USB to UART Bridge (COM7)
Baud
9600
Data bits
8
Stop bits
1
Parity
even
ModBus Request
Slave ID
Function code
Write Multiple Registers (0x10)
Start address
514
Num of coils
2
Display hex data
01 10 02 02 00 02
Send
Registers
Data type
Register
Data
Holding Register (16 bit)
514
7777
Holding Register (16 bit)
515
0
Bus Monitor
Raw data received:
Clear
01 10 02 00 00 04 c0 72
01 10 02 02 00 02 e1 b0
ModBus requests/responses:
Clear
I/O
Slave ID
Function code
Start address
Num of coils
CRC
1 Req >> 1 16 512 4 0000
2 << Resp 1 16 512 4 c072
3 Req >> 1 16 514 2 0000
4 << Resp 1 16 514 2 e1b0

Write Packet and Response

This section provides a more detailed view of the Modbus write packet and response: Write multiple holding registers.

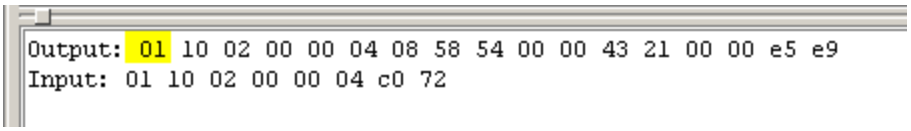


The next sections provide a breakdown of the Output and Input data.

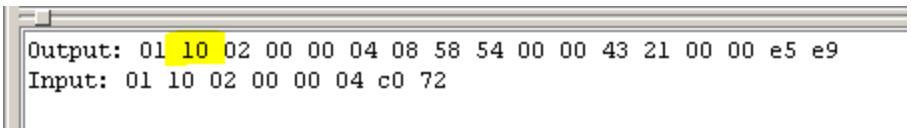
Output Data

This section provides descriptions of the Output data. Each item is highlighted to show the command part it represents.

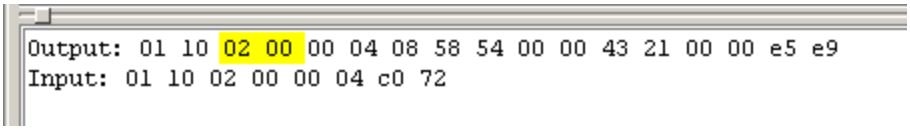
Command: Device address



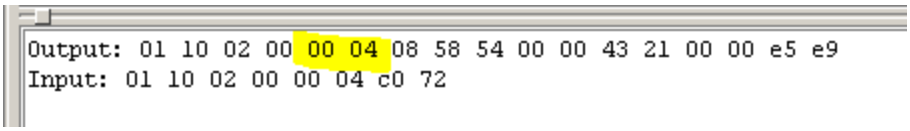
Command: Modbus function code



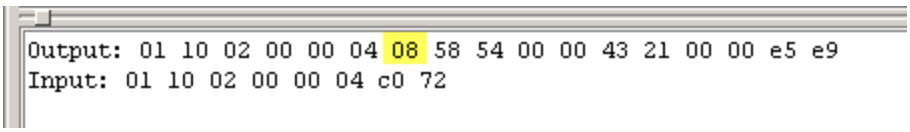
Command: Starting register address



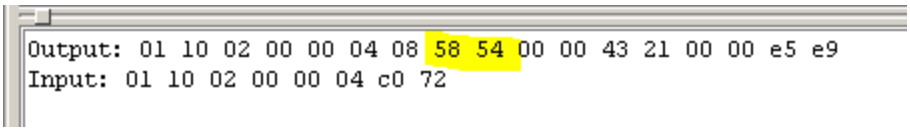
Command: Number of registers



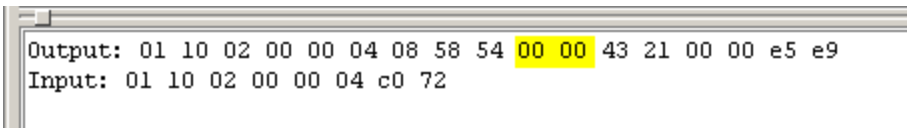
Command: Byte Count



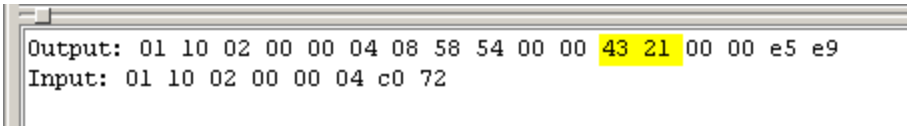
Command: Data word at 512



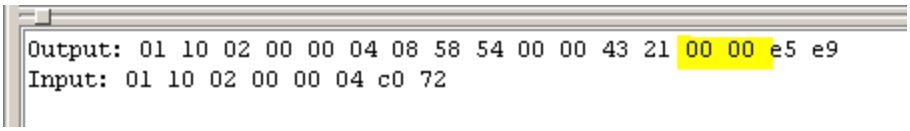
Command: Data word at 513



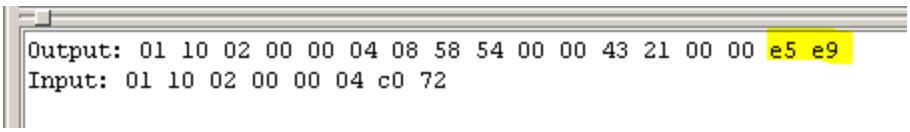
Command: Data word at 514



Command: Data word at 515



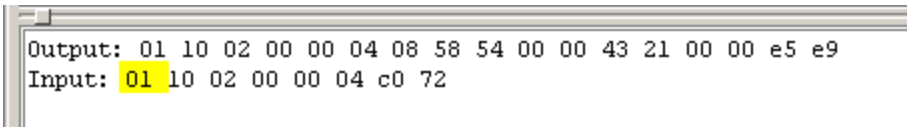
Command: CRC



Input Data

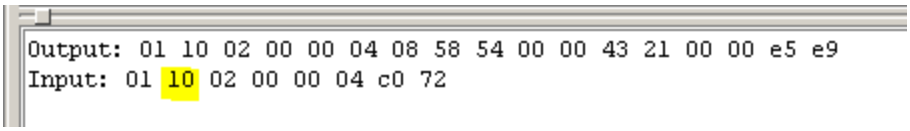
This section provides descriptions of the Input data. Each item is highlighted to show the response part it represents.

Response: From this device address

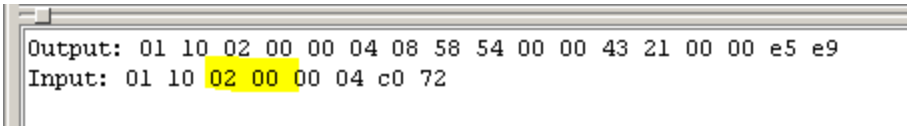


Response: Modbus function code

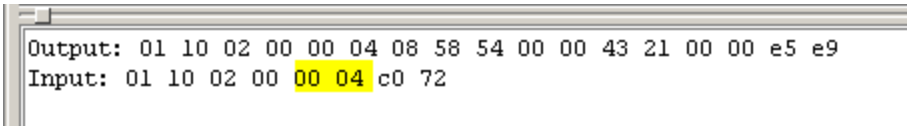
NOTE: This response is echoed back; it is not an error function code.



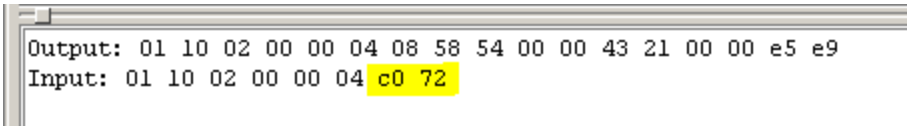
Response: Starting address



Response: Number of registers

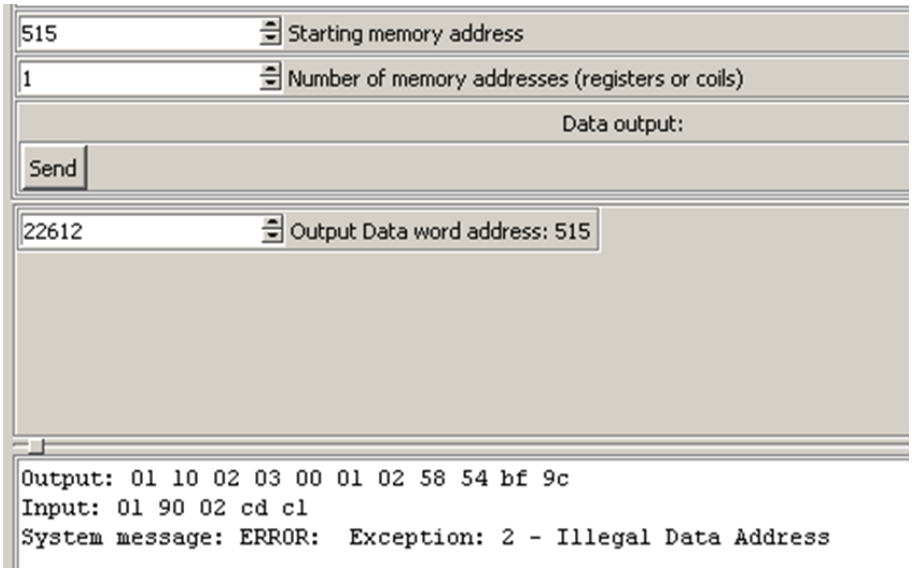


Response: CRC



Write Packet and Response - Invalid Start Address

This section is similar to the previous Modbus write packet and response. However, it shows an erroneous attempt to write to an invalid starting address.



The next sections provide a breakdown of the Input data.

Input Data

This section provides descriptions of the Input data. Each item is highlighted to show the response part it represents.

Response: From this device address

```
Output: 01 10 02 03 00 01 02 58 54 bf 9c
Input: 01 90 02 cd c1
System message: ERROR:  Exception: 2 - Illegal Data Address
```

Response: Modbus function code + 0x80

NOTE: This indicates an error in the Modbus request.

```
Output: 01 10 02 03 00 01 02 58 54 bf 9c
Input: 01 90 02 cd c1
System message: ERROR:  Exception: 2 - Illegal Data Address
```

Response: Error code is number 2

```
Output: 01 10 02 03 00 01 02 58 54 bf 9c
Input: 01 90 02 cd c1
System message: ERROR:  Exception: 2 - Illegal Data Address
```

Response: CRC

```
Output: 01 10 02 03 00 01 02 58 54 bf 9c
Input: 01 90 02 cd c1
System message: ERROR:  Exception: 2 - Illegal Data Address
```

Troubleshooting

This table provides troubleshooting information for solving common problems. For additional support resources, see the Moog Animatics Support page at:

http://www.animatics.com/support.html

Issue	Cause	Solution
Communication and Control Issues		
Motor control power light does not illuminate.	Control power is off, disconnected or incorrectly wired.	Check that control power is connected to the proper pins and turned on. For connection details, see Cable Diagram on page 16.
	Motor has routed drive power through drive-enable pins.	Ensure cabling is correct and drive power is not being delivered through the 15-pin connector.
	Motor is equipped with the DE option.	To energize control power, apply 24-48 VDC to pin 15 and ground to pin 14.
Motor does not communicate with SMI.	Transmit, receive or ground pins are not connected correctly.	Ensure that transmit, receive and ground are all connected properly to the host PC.
	Motor program is stuck in a continuous loop or is disabling communications.	To prevent the program from running on power up, use the Communications Lockup Wizard located on the SMI software Communications menu.
Motor does not communicate with Modbus RTU.	No OCHN command in program.	Verify that the OCHN command is used in program to set communication parameters. Modbus RTU does not have default settings.
	Incorrect baud rate.	Check the settings used for the OCHN command.
	Incorrect Modbus RTU address.	Use SADDR or ADDR= command in program to set the correct address at startup.
Motor stops communicating after power reset, requires re-detection.	Motor does not have its address set in the user program. NOTE: Serial addresses are lost when motor power is off or reset.	Use the SADDR or ADDR= command within the program to set the motor address.
Motor disconnects from SMI sporadically.	COM port buffer settings are too high.	Adjust the COM port buffer settings to their lowest values.
	Poor connection on serial cable.	Check the serial cable connections and/or replace it.
	Power supply unit (PSU) brownout.	PSU may be too high-precision and/or undersized for the application, which causes it to brown-out during motion. Make moves less aggressive, increase PSU size or change to a linear unregulated power supply.

Issue	Cause	Solution
Red PWR SERVO light illuminated.	Critical fault.	To discover the source of the fault, use the Motor View tool located on the SMI software Tools menu.
Common Faults		
Bus voltage fault.	Bus voltage is either too high or too low for operation.	Check servo bus voltage. If motor uses the DE power option, ensure that both drive and control power are connected.
Overcurrent occurred.	Motor intermittently drew more than its rated level of current. Does not cease motion.	Consider making motion less abrupt with softer tuning parameters or acceleration profiles.
Excessive temperature fault.	Motor has exceeded temperature limit of 85°C. Motor will remain unresponsive until it cools down below 80°C.	Motor may be undersized or ambient temperature is too high. Consider adding heat sinks or forced air cooling to the system.
Excessive position error.	The motor's commanded position and actual position differ by more than the user-supplied error limit.	Increase error limit, decrease load or make movement less aggressive.
Historical positive/negative hardware limit faults.	A limit switch was tripped in the past.	Clear errors with the ZS command.
	Motor does not have limit switches attached.	Configure the motor to be used without limit switches by setting their inputs as general use.
Programming and SMI Issues		
Several commands not recognized during compiling.	Compiler default firmware version set incorrectly.	Use the Compiler default firmware version option in the SMI software Compile menu to select a default firmware version closest to the motor's firmware version. In the SMI software, view the motor's firmware version by right-clicking the motor and selecting Properties.

TAKE A CLOSER LOOK

Moog Animatics, a sub-brand of Moog Inc. since 2011, is a global leader in integrated automation solutions. With over 30 years of experience in the motion control industry, the company has U.S. operations and international offices in Germany and Japan as well as a network of Automation Solution Providers worldwide.

Americas - West
Moog Animatics
2581 Leghorn Street
Mountain View, CA 94043
United States

Tel: +1 650-960-4215
Email: animatics_sales@moog.com

Americas - East
Moog Animatics
1995 NC Hwy 141
Murphy, NC 28906
United States

Europe
Moog GmbH
Memmingen Branch
Allgaeustr. 8a
87766 Memmingerberg
Germany

Tel: +49 8331 98 480-0
Email: info.mm@moog.com

Asia
Moog Animatics
Kichijoji Nagatani City Plaza 405
1-20-1, Kichijojihoncho
Musashino-city, Tokyo 180-0004
Japan

Tel: +81 (0)422 201251
Email: mcg.japan@moog.com

For Animatics product information, visit **www.animatics.com**

For more information or to find the office nearest you, email **animatics_sales@moog.com**

Moog Animatics Class 5 and 6 SmartMotor™ Modbus RTU Guide, Rev. E
PN: SC80100014-001