Using the MODBUS® Protocol with Omega® CN8200, CN8240, and CN8260 Controllers

OMEGA®

Table of Contents

1.

2. Introduction

2.1 About This Manual

This manual describes the Omega implementation of the MODBUS protocol for the controllers: CN8200, CN8240, and CN8260. The Omega MODBUS implementation enables a MODBUS master to read every configuration parameter, setpoint, and status value stored in the controllers' databases, as well as to write to every configuration parameter and setpoint in the controllers.[1] This manual contains information about the MODBUS functions used to read and write these values, as well as the addresses in the MODBUS register map used to read and write values to the controllers' databases.

Although this manual contains some general information about how the MODBUS protocol is used, the focus of this manual is the Omega implementation of the MODBUS protocol with controllers. When writing this manual we have assumed that you are familiar with the MODBUS protocol.

This manual contains tables that list every parameter, setpoint, and status value stored in the controllers and the valid values for each.

Information about the purpose of every configuration parameter, about the relationships between parameters, and about the effects of setting specific values, is in the *(CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*.

Diagrams that show the terminals to use when wiring the controllers to a MODBUS network are in the installation manual supplied with each Omega controller.

[1] To be able to communicate with a MODBUS master, an Omega controller must contain an RS-485 communication option card and the Omega MODBUS communication firmware.

2.2 About Omega's Implementation of the MODBUS Protocol

2.2.1 MODBUS Function Codes Supported

2.2.1.1 Functions to Access the Controller Databases

The MODBUS protocol provides a means for a master (host) device to communicate with slave devices (in this case controllers) on the network. The network can contain other MODBUS-compliant devices in addition to Omega controllers.[2]

Every message sent from a MODBUS master to a slave on the network contains a function code that represents the action the slave device should take in response to the message. So that a MODBUS master can read and write all configuration parameters, setpoints, and status values stored in an Omega controller, the Omega implementation of the MODBUS protocol supports the following function codes:

- **function 03 ($03)[3] –** This function is used to read to one or more contiguous "holding registers" (database locations that are next to one another in the register map). See 2.2 for more information.
- **function 06 ($06)** – This function is used to write to a single holding register. A message containing function 06 can be broadcast to all controllers on the network simultaneously by sending the message to the controller ID of 0 (zero), instead of the controller ID of a specific device. See 2.3 for more information.
- **function 16 ($010)** – This function is used to write to a single register or to multiple contiguous holding registers, that is, to two or more registers that are next to one another in the register map. A message containing function 16 ($10) can be broadcast to all controllers on the network simultaneously by sending the message with the controller ID of 0 (zero), instead of the controller ID of a specific device. See 2.4 for more information.

2.2.1.2 What Happens if the Controller Can Do What the MODBUS Master Tells It to Do

If the MODBUS slave device (controller) is able to act on a message (sent to a single controller) that contains a 03 read function, or a 06 or 16 ($10) write function, then the controller will send a reply to the master.

- In the case of the 03 read function, this reply will contain the requested information.
- If a write function 06 or 16 ($10) message is sent to a specific controller, this reply will consist of a confirmation of the quantity of holding registers to which data was written in response to the message.

[2] Peer-to-peer communications are not supported by the MODBUS protocol. This means that the controllers cannot communicate with one another, nor with other MODBUS slave devices. The MODBUS protocol does not allow slave devices to initiate data exchanges. The slave devices can only respond to messages from the MODBUS master.

[3] Throughout this manual hexadecimal numbers are preceded by the $ symbol. (All the numbers in a MODBUS message are expressed in hexadecimal format.) Any number in this manual without the $ symbol is a decimal number.

If a message containing a 06 or 16 ($10) write function is broadcast to all devices on the network, then the controllers will write the data to their databases as requested, but the controllers will not send a confirmation message to the master.

2.2.1.3 What Happens if the Controller Cannot Do What the MODBUS Master Tells It to Do

Sometimes a controller cannot act on a message because communication between the master and the controller is not possible (for example, because of a fault in the network or because the controller has not been powered up). However, in some cases the controller will not act on a message sent by the master because the master sent a faulty message to the controller; the result depends on the type of fault.

- **If a read or write message from the master contains the address of an invalid register,** the controller will send a MODBUS error code 03 message back to the master. See 2.2.6, 2.3.5 and 2.4.6.
- **If a write message from the master instructs a controller to write invalid (out of range) data to a valid register, or to write to a read-only register,** the controller will send a MODBUS error code 02 message back to the master. See 2.3.6 and 2.4.7.
- **If a 03 read message or a 16 ($10) write message from the master instructs the controller to read or write too many words**, the message will be ignored and no error message will be sent to the master. The maximum number of words supported by MODBUS functions 03 and 16 ($10) consists of 24 registers. Depending on the type of value being handled, this can equate with 24 or 12 values. See 1.2.6 and Section 3 for information about the types of values, which can be read from and written to the controllers.
- **If the message contains a MODBUS function code other than 03, 06, 08 subfunction 00** (see 1.2.1.4)**, or 16 ($10),** then the controllers will not act on the message from the MODBUS. The controller receiving a message that contains an unsupported function code will not send an error message back to the master.

2.2.1.4 Diagnostic Function

So that you can test communication over the MODBUS network, Omega also supports the diagnostic **function 08 ($08), subfunction 00 ($00)** to perform a loopback test; see 2.5 for more information. When a loopback test is performed, the MODBUS master sends a message to a controller and the controller sends the same message back to the master. (No changes are made to the controller database as a result of processing the message.) If the master does not receive a reply, it is time to troubleshoot the network or the controller's communication setup.

2.2.2 MODBUS Transmission Mode Supported

The Omega Implementation of the MODBUS protocol supports RTU (Remote Terminal Mode) only. RTU mode permits faster data throughput than does ASCII mode at the same baud rate.

2.2.3 Baud Rates Supported

The Omega controllers can communicate with the MODBUS master at the following baud rates:

- 300
- 600
- 1200
- 2400
- 4800
- 9600; this is the default rate for all Omega controllers equipped with Omega MODBUS communication firmware.

The baud rate used by a controller can be changed using a MODBUS message (see 1.3). The baud rate of a CN8200, CN8240, or CN8260 controller can also be changed using the front panel to access and change the value of the **SerL** (serial) menu **bAud** item.[4]

2.2.4 Serial Data Formats Supported by the Controllers

Every Omega controller that is equipped with MODBUS communication firmware supports the following serial data formats:

- 1 start bit, 8 data bits, no parity (0 bits), 1 stop bit; this is the default
- 1 start bit, 8 data bits, even parity (1 bit), 1 stop bit
- 1 start bit, 8 data bits, odd parity (1 bit), 1 stop bit

The data format used by a controller can be changed using a MODBUS message (see 1.3.4.3). The data format for a CN8200, CN8240, CN8260 controller can also be changed using the front panel to access and change the value of the **SerL** (serial) menu **PAr** (parity) item. The menu selections are **none** (the default), **EuEn** (even), and **odd**.

2.2.5 Addresses Supported by the Controllers

Each controller on the MODBUS network must be assigned a unique ID number used as the device address when the master sends out a message. Valid addresses for Omega controllers using the MODBUS protocol are 1 to 247.

The address of a CN8200, CN8260, or CN8260 controller can be set using the front panel as described in the *(CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual.* The address of a controller is set using the DIP switches as described in the installation manual supplied with the controller. (The address of 255 has a special meaning for a controller. Setting the DIP switches to 255 returns the controller's communication parameters to their defaults.)

[4] The controller does not have a display and keypad. Instructions for using the CN8200, CN8240 and CN8260 controllers' front panel to change a configuration parameter setting are in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*.

2.2.6 Data Types Used by Controllers

2.2.6.1 Principle

Two types of values are used by Omega controllers as configuration parameter values, setpoints, statuses, etc.:

- values that can be only integers
- values that can include fractional values, that is, numbers that can include decimal places (such as 123.6, 9.75, and –3.6)

2.2.6.2 Examples

Examples of values used by the controller that can be only integers are the configuration parameter value that represents the input type (0 = B thermocouple, 1 = C thermocouple, 2 = E thermocouple, etc.) and the value that represents the controller mode (1 = manual, 2 = standby, 3 = automatic, etc.).

Examples of values that can include decimal values are the setpoint, process value, and proportional band.

2.2.6.3 How These Data Types Are Transmitted in MODBUS Registers

Because each MODBUS register contains a 16-bit value, reading or writing a parameter value for which the controller always uses an integer is straightforward. The integer can be transmitted in a single register. (Negative integers are transmitted in two's complement format.)

Transmitting a value that can include a fractional value is a greater challenge. The Omega MODBUS implementation provides three methods of transmitting each controller value that can include decimal places.

- in a single register as an integer. Sometimes the fractional part of the number stored in the controller's database is rounded off; sometimes the fractional part of the number is included without the decimal point. See 3.7 for information about the circumstances that determine how the fractional value is handled. (Negative integers are transmitted in two's complement format.)
- in a single register as an integer that is 10 times the value stored in the controller's database. This is called the "10X mirror". Using the 10X mirror enables you to access a fractional value rounded to the nearest tenth, as described in 3.8. (Negative integers are transmitted in two's complement format.)
- in two registers as a true 32-bit IEEE floating point value that includes the fractional value (if any) stored in the controller's database. This is called the "32-bit IEEE mirror" (see 3.9).

The register region you choose to use to transmit a parameter value that can contain a fractional value usually depends on the type of values the MODBUS master can handle, but is also affected by the type of input the controller uses (see 3.6.2). The subject of transmitting integers and values that can include decimal places is covered in greater detail in Section 3. That section also describes the regions of the MODBUS register map used for each type of value.

2.2.7 IEEE Register Ordering

2.2.7.1 Principle

Each value read from and written to the 32-bit IEEE mirror region of the MODBUS register map uses two 16-bit registers to transmit the value. The default for all Omega controllers using MODBUS is to transmit the lower order register before the higher order register, which is the MODBUS standard. However, to accommodate masters that cannot use this standard sequence, the sequence in which these two register values are transmitted is configurable.

The IEEE register sequence used by a controller can be changed using a MODBUS message (see 1.3). The IEEE register sequence for a CN8200, CN8240, or CN8260 controller can also be changed using the instrument's front panel to access and change the value of the **`SerL`** (serial) menu **`nnOd`** (MODBUS) item. The menu selections are **`yes`** for MODBUS standard sequence (the default: low order register before high order) and **`no`** for non-standard sequence (high order register before low order).

2.2.7.2 Example

The 32-bit IEEE representation of 250.0 (expressed in hex) is $437A 0000. Using the MODBUS standard sequence, the registers will be transmitted in this order: $0000 437A. If the non-standard sequence is used, the registers will be transmitted in this order: $437A 0000.

Note that the IEEE register ordering does <u>not</u> affect the sequence in which the bytes <u>within the register</u> are transmitted. Within the register, the most significant (high order) byte is always transmitted before the least significant (low order) byte.

2.2.8 Timing and Latency Issues

2.2.8.1 Separating Messages

The MODBUS specification requires that all MODBUS messages be separated by an idle time (lull in the transmission) that has a duration that is at least 3.5 character times (that is, the time needed to transmit one character multiplied by 3.5). This necessary idle time is easily implemented at the host end by waiting at least four character times prior to sending any request. (This timing is automatically implemented on the controller side.)

2.2.8.2 Allowing the Controller Time to Process a Request

The master must wait sufficient time for a controller to process a request. This time is called “latency time”. The time needed by the controllers to process a request depends on the function code in the message and on the number of registers to be read or written. The table below shows the minimum and maximum latency time for each function.

Function Code	Minimum Latency Time	Maximum Latency Time
03 ($03)	5 milliseconds per register	100 milliseconds per register
06 ($06)	25 milliseconds	180 milliseconds
08 ($08)	0 milliseconds	100 milliseconds
16 ($10)	25 milliseconds per register	180 milliseconds per register

2.2.8.3 Applying the Timing Requirements

The following diagram illustrates timing relationships in Omega MODBUS transactions.

T1T2T3T4	**Request**	**T1T2T3T4**	**T5**	**Response**	**T1T2T3T4**	**Request**

T1T2T3T4 are the 4 characters of idle time required by the MODBUS protocol. **Request** and **Response** are the request messages sent by the host and the response message returned by the controllers.

T5 is the latency time required by each controller to process each request.

The master should expect the start of the acknowledgement after the **minimum T5** has elapsed, but should time out and assume an error if the **maximum T5** has elapsed and the start of the acknowledgement has not been received. Following this guideline will yield the fastest system possible.

T1T2T3T4 is dependent on the BAUD rate and data format used. The formula for calculating **T1T2T3T4** follows.

$$T1T2T3T4 = 4 \times \left(\frac{1000}{baud} \times CharBits \right) \text{ milliseconds}$$

where:

CharBits = 10 if the data format is 8 data bits, no parity, 1 stop bit
11 if the data format is 8 data bits, even parity, 1 stop bit
11 if the data format is 8 data bits, odd parity, 1 stop bit

2.2.8.4 Example

Suppose the network is operating at 9600 baud and 8, N, 1, and you want to read two registers (in a single message). In this case, **T1T2T3T4**, the delay needed to separate messages, is:

4 x (1000/9600 x 10) = 4.17 milliseconds

If the master sends a request to read two registers, then the **minimum T5**, the minimum time required for the controller to respond is:

5 milliseconds x 2 = 10 milliseconds

Therefore, the host should wait at least **T1T2T3T4** plus the **minimum T5** before it expects the start of the response from the controllers:

4.17 milliseconds + 10 milliseconds = 14.17 milliseconds

However, if the master does not receive the start of the response by the time **T1T2T3T4** plus the **maximum T5** has elapsed, the master should assume an error. In this case, the maximum T5 is:

100 milliseconds x 2 = 200 milliseconds

Therefore the master should assume an error if it has not received the start of the response in:

4.17 milliseconds + 200 milliseconds = 204.17 milliseconds

2.3 Preparing Controllers for Use with MODBUS

2.3.1 Introduction

The controllers should be installed and networked in accordance with the instructions in the installation manual supplied with the controllers.

Warning **All wiring should be done by an experienced technician and be installed in accordance with national and local electrical codes. To avoid serious personal injury and damage to equipment, follow all warnings and cautions provided in the controller installation manuals.**

In addition, to respond to messages from a MODBUS master, the controller must satisfy three requirements.

- It must contain an RS-485 communication option card and the appropriate communication firmware; see 1.3.2.
- It must be configured to have a unique controller ID between 1 and 247; see 1.3.3.
- Its communication parameter settings must match those of the MODBUS master (host); see 1.3.4.

2.3.2 MODBUS Communication Option

To see if a new controller contains an RS-485 communication option card and the MODBUS communication firmware on its processor board, compare the model number on the controller's label with the model number information in the installation manual supplied with the controller.

2.3.3 Assigning a Unique Controller ID

With the exception of special orders, every Omega controller is shipped with its controller ID set to 1. If your MODBUS network will contain more than one controller, you must assign each device on the network a unique ID number.

- For CN8200, CN8240, and CN8260 models the controller ID is configured using the `SerL` (serial) menu `Id.no` (ID number) item. Instructions for using the front panel display and keypad to access and change parameter values are in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*. [5]

[5] If the controller is not new, you have one more task to perform with the front panel. Every new controller is configured at the factory to be ready to accept parameter values for the options its hardware supports. However, if you have used the procedure in 8.2 to load all parameter defaults, then you must use the front panel of the controller to configure the `Card` parameter in the `Optn` (option) menu. This prepares the controller to receive the appropriate option parameter values. The only exception is in the case of an CN8240 or CN8260 equipped with only the serial communication option. The CN8240 and CN8260 models are always ready to receive serial communication parameter values.

- For controllers the controller ID is configured using the same set of DIP switches that are used to set the communication parameters to the defaults. Instructions for setting the DIP switches are in the installation manual supplied with the controller.

Warning

Be sure to remove power from the controller before removing the chassis from the case to access the DIP switches. Do not power up the controller while the chassis is out of the case. Failure to observe this precaution can result in exposure to a potentially lethal shock hazard.

2.3.4 Configuring the Controller Communication Parameters to Match the MODBUS Master

2.3.4.1 Communication Defaults

Any Omega controller that contains MODBUS communication firmware is shipped with its communication defaults set to:

Baud Rate = 9600
Parity = none
Controller ID = 1
MODBUS standard register sequence for 32-bit IEEE values = yes

If the configuration settings in a used controller have been changed from these defaults, you can change the communication parameter values using a MODBUS message as described below. The communication settings in a CN8200, CN8240, or CN8260 controller can also be changed using the front panel to access and change the value of the **`SerL`** (serial) menu parameters. Instructions for accessing configuration menus and changing parameter values are in the *(CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*.

The communication parameters in a controller can be returned to their defaults using DIP switches as described in the installation manual supplied with the controller.

Be aware that the CN8200, CN8240, and CN8260 controllers do not have a function that will return only the communication parameters to their defaults. Using the **`supr`** (supervisor) menu **`Ld.dp`** (load defaults) item in one of these controllers will return all configuration parameters and setpoints to their defaults.

2.3.4.2 Methods of Changing the Controller Configuration Parameters

If you plan to operate your network with communication characteristics that are different from the default settings, you have two choices for changing the communication settings in the CN8200, CN8240, and CN8260 controllers:

- You can use the controller front panel and the **`SerL`** (serial) menu to change the communication settings in each controller.
- You can use serial communications to change the communication parameters settings in the controllers.

Because the communication parameters in a can be set to the defaults, but not set to other values, using the DIP switches, the second method (using serial communications) is the only way to change the communication parameter values in a unit.

Of course, if the MODBUS master can operate only with communication characteristics that are different from the controller defaults, then you must use the front panel to change the communication parameter settings for the CN8200, CN8240, and CN8260 models.[6]

However, if the MODBUS master can operate with communication characteristics that are different from the controller defaults, then you can set the master to match these defaults temporarily, change the communication parameters in the controllers by means of MODBUS messages containing write functions, then set the master to use the communication settings you actually want to use for the network.

It is possible to change all the controllers' communication parameter values from the defaults, even if the master does not use the standard MODBUS register sequence for 32-bit IEEE values, because all the communication parameters are stored as integers in the controllers. Therefore, the parameter values can be written to the controllers using MODBUS integer registers. As a result, the temporary incompatibility between the host's register sequence for 32-bit IEEE values and the sequence used by the controllers will not prevent you from making the necessary changes to the controllers' configuration parameter values.

2.3.4.3 Example of Changing the Controllers' Communication Parameter Values Using MODBUS Messages

Here is a scenario where the master can communicate at the defaults of 9600 baud, 1 start bit, 8 data bits, no parity (0 bits), 1 stop bit and standard register order for 32-bit IEEE values. However, other non-Omega devices on the network cannot use these settings. Therefore, you want to operate your network at 300 baud, 1 start bit, 8 data bits, even parity (1 bit), 1 stop bit, and non-standard sequencing of 32-bit IEEE values.

To change the new Omega controllers' communication parameters from their defaults, you could use the following procedure.

1) Configure the MODBUS master for 9600, 8, N, 1. (The master's setting for 32-bit IEEE ordering does not matter, because you use only integer registers to transmit new configuration parameter values to the controllers.)

[6] If your MODBUS host cannot use the Omega communication defaults, contact your Omega sales representative or a member of the Omega Technical Support team for assistance in changing the communication parameter values in a controller.

2) To change the controllers' IEEE register ordering, write a value of 0 (zero) to the register at relative address 4084 in each controller.[7] Communication with the controllers will be maintained. Your ability to write to the other communication parameters will not be affected because they are transmitted using integer registers.

3) To change the parity to even, write a value of 1 to the register at relative address 4083 in the first controller. You can broadcast the message to all the devices on the network if you take the non-Omega devices out of service temporarily. (You should not allow the non-Omega devices to receive the broadcast of the change to relative register 4083 because that register is probably used for a different purpose in the register mapping of non-Omega devices.)

4) Wait 150 milliseconds.

5) Ignore the reply from the controller because it will most likely be garbage due to the parity change. If you did not broadcast the message in step 3, repeat steps 3 and 4 for each controller to be changed.

6) Configure the MODBUS master for 9600, 8, E, 1.

7) To change the baud rate to 300, write a value of 2 to the register at relative address 4082 in the controllers.

8) Wait 1 second; it takes longer to transmit at 300 baud.

9) Ignore the reply. If you did not broadcast the message in step 7, repeat steps 7 and 8 for each controller to be changed.

10) Configure the MODBUS master for 300, 8, E, 1. You are now ready to configure the other parameters in the controllers' databases.

[7] See the table in 5.2 for the register addresses of all the communication parameters, as well as all valid values for these registers.

2.4 Importance of Sequence in Which Configuration Parameters Are Written

The Omega of controllers are versatile instruments that are capable of using many types of input values and implementing several types of control strategies. To support this versatility, the controllers are capable of storing values for many configuration parameters. Interrelationships exist between the parameters. Therefore, it is important that you specify values for the configuration parameters in the correct sequence.

Once you have your MODBUS network up and running, if you plan to use a thermocouple or RTD input and if you do not plan to use degrees Fahrenheit (the default unit of measure), the first step is always to specify the unit to be used (Celsius or Kelvin). The controller uses this unit of measure for internal operations, as well as for external communications. When you change the units of measure for temperature inputs, the controller recalculates any values that have already been specified. For example, if you want the setpoint to be 100 °C, then you must change the units from the default F to C before you write the setpoint of 100 to the controller. If you change the units after you write the setpoint of 100 to the controller, the controller will convert the 100 °F setpoint to 37.8 °C. In this case, you would have to reconfigure the setpoint to 100 °C to implement the control needed by your process.

After the unit of measure has been changed, if necessary, from degrees Fahrenheit to your choice of Celsius or Kelvin, the next step is always to specify the type of input that each controller will receive. That means that you must specify the type of thermocouple or RTD that will provide the input to the controller, or, in the case of a linear input, the range and units of the input (0 to 20 mA, 0 to 5 V, 1 to 5 V, etc.) The type of input specified affects how the controller processes the input signal and calculates the output needed to achieve the setpoint.

Generally, the parameters should be configured in the sequence in which they are presented in Section 4. See 4.1.2 for more information about configuration sequence.

In addition to being aware of the sequence in which parameters should be configured, you should also remember that not all parameters apply to all applications. For example, if you specify that the input type is a thermocouple or RTD, then you do not have to write a value to the low scale and high scale parameters. (You can write the values, but the controller will ignore them.) However, if you use a linear input, then you must specify scaling values, or accept the factory defaults.[8]

For more information about the interrelationships between parameters and about the effects of setting specific values, see the *(CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*.

[8] The database values in new ("out of the box") , CN8200, CN8240, and CN8260 controllers are always the default values shown in the tables in Section 4. Instructions for using the controller front panel to return all database values in the CN8200, CN8240, and CN8260 controllers to their default values are in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*. You can also use MODBUS function 16 ($10) to write a command to special registers in the , CN8200, CN8240, and CN8260 controllers to set all the database values to their defaults (excluding the address) as described in Section 7.

2.5 Numbering Conventions Used in This Manual

The decimal numbers in this manual are not identified with any special symbol. The hexadecimal numbers are preceded by $. For example, the function to write to multiple registers is function 16 ($10).

All the numbers in a MODBUS message are expressed in hexadecimal format because they are transmitted as 16-bit words (each consisting of two bytes).

3. MODBUS Functions Supported

3.1 Overview

Every message sent from a MODBUS master to a slave on the network contains a function code that represents the action the slave device should take in response to the message. So that a MODBUS master can read and write all configuration parameters, setpoints, and status values stored in an Omega , CN8200, CN8240, or CN8260 controller, the Omega implementation of the MODBUS protocol supports the following function codes:

- **function 03 ($03)[9] –** This function is used to read to one or more contiguous "holding registers" (database locations that are next to one another in the register map). See 2.2 for more information.
- **function 06 ($06)** – This function is used to write to a single holding register. A message containing function 06 can be broadcast to all controllers on the network simultaneously by sending the message to the controller ID of 0 (zero), instead of to the controller ID of a specific device. See 2.3 for more information.
- **function 16 ($010)** – This function is used to write to a single register or to multiple contiguous holding registers, that is, to two or more registers that are next to one another in the register map. A message containing function 16 ($10) can be broadcast to all controllers on the network simultaneously by sending the message to the controller ID of 0 (zero), instead of to the controller ID of a specific device. See 2.4 for more information.

So that you can test communication over the MODBUS network, Omega also supports the diagnostic **function 08 ($08), subfunction 00 ($00)** to perform a loopback test; see 2.5 for more information. When a loopback test is performed, the MODBUS master sends a message to a controller and the controller sends the same message back to the master. (No changes are made to the controller database as a result of processing the command.) If the master does not receive a reply, it is time to troubleshoot the network or the controller's communication setup.

The controllers will not take any action, nor will they reply in response to any messages they cannot interpret. This includes any messages that contain a function code other than 03, 06, 16 ($10), or 08 subfunction 00.

Throughout this section the following abbreviations are used:

[9] Throughout this manual hexadecimal numbers are preceded by the $ symbol. (All the numbers in a MODBUS message are expressed in hexadecimal format.) Any number in this manual without the $ symbol is a decimal number.

MSB = most significant byte (high byte)
LSB = least significant byte (low byte)
CRC = cyclical redundancy check[10]

3.2 Function 03 ($03): Read One or More Holding Registers

3.2.1 Introduction

This function is used to read the value of one or more contiguous holding registers (that is, registers that are next to one another). Broadcasting a function 03 message is not supported by the MODBUS protocol.

When using the 03 read function, you must specify the following data:

- the controller ID of the device containing the values to be read
- the function code $03
- the relative address of the first register to be read[11]
- the quantity of words to be read, beginning at the specified register address

The master must append a CRC value to the message.

The register address of every value that can be read from the Omega controllers is listed in Section 4.

3.2.2 Allowable Number of Words to Be Read in a Function 03 Request

Each integer register (including the registers used to transmit a fractional value as an integer) uses one word. Therefore, the value of the **Number Of Words To Read** field in the request message should equal the number of registers requested for database values that are transmitted as integers. (For convenience, we refer to these as "non-IEEE" registers.)

Each 32-bit IEEE floating point register uses two words. Therefore, the value of the **Number of Words to Read** field should equal the number of registers needed multiplied by two. For example, to request a read of two 32-bit IEEE registers, the **Number Of Words To Read** field must contain the value four.

IMPORTANT ➡ **The maximum number of words allowed for function code $03 is 24. This equates to 24 non-IEEE registers and 12 of the 32-bit IEEE registers. Thus, the value in the Number Of Words To Read field cannot exceed 24 ($18).**

[10] If you are not familiar with the function of a CRC, refer to the MODBUS specification available at the modicon.com web site.

[11] All register relative addresses are offset from 40001. See Section 3 for more information about register address regions used by the Omega implementation of the MODBUS protocol. See Sections 4 and 5 for the relative address of specific configuration parameters and other values in the controller databases.

3.2.3 Function 03 Request

The format for a function 03 request is shown below.

Device Address	Function Code 03	First Register Relative Address		Number Of Words To Read		CRC	
1 byte	1 byte containing $03	MSB	LSB	MSB	LSB	MSB	LSB

3.2.4 Function 03 Examples

Suppose you want to read four non-IEEE registers starting at relative address 0 in the controller with controller ID 1. The master should send the message shown below.

Device Address	Function Code 03	First Register Relative Address		Number Of Words To Read		CRC	
$01	$03	$00	$00	$00	$04	$44	$09

To read two 32-bit IEEE registers starting at relative address of 8000 ($1F40) in the same controller, the master should send the message shown below.

Device Address	Function Code 03	First Register Relative Address		Number Of Words To Read		CRC	
$01	$03	$1F	$40	$00	$04	$42	$09

3.2.5 Function 03 Normal Reply

If the controller is able to interpret the message and read the requested registers, the device will respond to the master with a normal reply message.

The reply will consist of the following data:

- the controller ID of the device responding
- the function code $03
- the quantity of bytes read
- the value of the first word read
- the value of the second word read, and so on until
- the value of the last word read

The controller will append a CRC value to the message.

The format for a function 03 normal reply is shown below.

Device Address	Function Code 03	Number Of Bytes Read	Value Of First Word Read		….	Value Of Last Word Read		CRC	
1 byte	1 byte containing $03	1 byte	MSB	LSB	….	MSB	LSB	MSB	LSB

The block containing **…** represents the values between the first and last value. Each 2-byte word's value must be included in the message.

3.2.6 Reply to Function 03 Request Containing Illegal Register Address: 02 ($02) Error Code

The controllers will issue an error reply containing the MODBUS error code of $02 if either of the following conditions is true:

- **First Register Relative Address** field does not point to a valid Omega MODBUS register, or
- **First Register Relative Address** field contains an odd value that is located in the 32-bit IEEE register address region. All 32-bit IEEE register addresses must be even.

A code $02 error message will be returned only if the **First Register Relative Address** field contains an address of an invalid register. Otherwise, reading will continue to the last register even in the event of invalid register addresses occurring after the first register. Garbage data will be returned for invalid register addresses.

The format for an error code $02 response to a function $03 message is shown below. Note that the controller adds $80 to function code $03 in the reply.

Device Address	Function Code	Error Code	CRC	
1 byte	$83	$02	MSB	LSB

3.2.7 Circumstances Under Which No Reply Is Sent in Response to a Function 03 Message

The CN8200, CN8240, CN8260 will not send a reply to a function code $03 message if any of the following conditions exist:

- The message was a broadcast.
- The value of the CRC field does not equal the CRC calculated by the controller for the message received.
- The value in the **Number of Words to Read** field is greater than the maximum of 24 ($18).
- Value in **Number of Words to Read** field is not even when addressing 32-bit IEEE registers. Each 32-bit IEEE register consists of two bytes; therefore their byte counts must be even.

3.3 Function 06 ($06): Write to a Single Holding Register

3.3.1 Introduction

This function is used to set the value of a single holding register. The function 06 write function can be broadcast.

When using the 06 write function, you must specify the following data:

- the controller ID of the device to which the value is to be written
- the function code $06
- the relative address of the register to be written
- the value to be written to the specified register

The master must append a CRC value to the message.

This function can be used to write to any integer register (including the registers used to transmit a fractional value as an integer). Because each 32-bit IEEE value uses two registers, the 06 write function cannot be used for 32-bit IEEE values. (Instead, use the 16 ($10) write function for 32-bit IEEE floating point values.) The controller will respond with a $02 (illegal address) error message if you try to use function 06 to write to a 32-bit IEEE register.

3.3.2 Function 06 Request

The format for a function 06 request is shown below.

Device Address	Function Code 06	Register Relative Address		Data		CRC	
1 byte	1 byte containing $06	MSB	LSB	MSB	LSB	MSB	LSB

3.3.3 Examples

Suppose you want to write the value 50 ($32) to the non-IEEE register at relative address 4009 ($FA9) in the instrument with controller ID 156 ($9C). The master should send the message shown below.

Device Address	Function Code 06	Register Relative Address		Data		CRC	
$9C	$06	$0F	$A9	$00	$32	$C7	$66

Suppose you want to write the value 5 ($05) to the non-IEEE register at relative address 4082 ($1E2) in all the controllers on the network (that is, broadcast the message). The master should send the message shown below.

Device Address	Function Code 06	Register Relative Address		Data		CRC	
$00	$06	$01	$E2	$00	$05	$E9	$D2

3.3.4 Function 06 Normal Reply

If the controller is able to interpret the message and write the data as requested, the device will respond to the master with a normal reply message <u>if the message was not broadcast</u>. If the message <u>was</u> broadcast, then the controller will write the data to the specified registers, but will not send a reply.

The reply to a message sent to a single device will be an echo of the write message. That means that the format and content of the reply will be exactly the same as the format and content of the message sent by the master.

Device Address	Function Code 06	Register Relative Address		Data		CRC	
1 byte	1 byte containing $06	MSB	LSB	MSB	LSB	MSB	LSB

3.3.5 Reply to Function 06 Request Containing Illegal Register Address: 02 ($02) Error Code

The controllers will issue an error reply containing the MODBUS error code of $02 if either of the following conditions is true:

- **Register Relative Address** field does not point to a valid Omega MODBUS register, or
- **Register Relative Address** field contains an address that is located in the 32-bit IEEE register address region. The 06 write function can write to only one register, and every 32-bit IEEE value requires two registers.

The format for an error code $02 response to a function $06 message is shown below. Note that the controller adds $80 to function code $06 in the reply.

Device Address	Function Code	Error Code	CRC	
1 byte	$86	$02	MSB	LSB

3.3.6 Reply to Function 06 Request Containing Illegal Value in Data Field: 03 ($03) Error Code

An error reply containing the MODBUS error code $03 will be issued by the controllers if either of the following conditions is true:

- **Data** field contains a value that is not a valid value for the destination (target) register, or
- **Register Relative Address** field contains an address that is a read-only value.

The format for an error code $03 response to a function $06 message is shown below. Note that the controller adds $80 to function code $06 in the reply.

Device Address	Function Code	Error Code	CRC	
1 byte	$86	$03	MSB	LSB

3.3.7 Circumstances Under Which No Reply Is Sent in Response to a Function 06 Message

The controllers will not reply to a function code $06 message if the following condition exists:

- The message was a broadcast. However, in this case the controllers do write the value, even though they do not reply.

The controllers will not reply, nor act in response to a function code $06 message if the condition listed below exists. If this condition exists, the controllers do not write the value to their databases.

- The value of the CRC field does not equal the CRC calculated by the controller based on the message received.

3.4 Function 16 ($10): Write to Multiple Registers

3.4.1 Introduction

This function is used to write specified values to a single register or to multiple contiguous holding registers (that is, registers that are next to one another). Broadcasting a function 16 ($10) message is supported by the MODBUS protocol.

When using the 16 ($10) write function, you must specify the following data:

- the controller ID of the device to which the values are to be written
- the function code $10
- the relative address of the first register to be written
- the quantity of words to be written
- the quantity of bytes to be written
- the value of the first word to be written, followed by the second word, and so forth until the last word of data to be written

The master must append a CRC value to the message.

3.4.2 Function 16 ($10) Request

The format for a function 16 ($10) request is shown below.

Device Address	Function Code 16	First Register Relative Address (2 bytes)		Word Count (2 bytes)		Byte Count	Value of First Word to Be Written		…	Value of Last Word to Be Written		CRC	
1 byte	1 byte containing $10	MSB	LSB	MSB	LSB	1 byte	MSB	LSB	…	MSB	LSB	MSB	LSB

The block containing **…** represents the words between the first and last word of the values to be written. Each byte of each word of the values to be written must be included in the message.

3.4.3 Allowable Number of Words to Be Written in Function 16 ($10) Request

Each integer register (including the registers used to transmit a fractional value as an integer) uses one word. Therefore, the value of the **Word Count** field in the request message should equal the number of registers requested for database values that are transmitted as integers.

Each 32-bit IEEE floating point register uses two words. Therefore, the value of the **Word Count** field should be equal to the number of registers to be written multiplied by two. For example, to request a write of two 32-bit IEEE registers, the **Word Count** field must contain the value four.

The value in the **Byte Count** field must be two times the value in the **Word Count** field.

IMPORTANT **The maximum number of words allowed for function code $10 is 24. This equates to 24 non-IEEE registers and 12 of the 32-bit IEEE registers. Thus, the value in the Word Count field cannot exceed 24 ($18).**

3.4.4 Function 16 ($10) Examples

Suppose you want to write the following values to non-IEEE registers in controller 73 ($49):

value of 2 ($02) to relative address 4076 ($FEC)
value of 1 ($01) to relative address 4077 ($FED)
value of 100 ($64) to relative address 4078 ($FEE)
value of 200 ($C8) to relative address 4079 ($FEF)

The master should send the message shown below.

Device Address	Function Code 16	First Register Relative Address		Word Count		Byte Count	4 Words (8 Bytes) of Data	CRC	
$49	$10	$0F	$EC	$00	$04	$08	$00 02 $00 01 $00 64 $00 C8	$26	$E4

Suppose you want to write the value 250.0 to one 32-bit IEEE holding register at relative address of 8002 ($1F42) in controller 1. The master should send this message.

Device Address	Function Code 16	First Register Relative Address		Word Count		Byte Count	2 Words (4 Bytes) of Data	CRC	
$01	$10	$1F	$42	$00	$02	$04	$43 7A $00 00	$CE	$2B

3.4.5 Function 16 ($10) Normal Reply

If the controller is able to interpret the message and write to the requested registers, the device will respond to the master with a normal reply message.

The reply will consist of the following data:

- the controller ID of the device to which the values are to be written
- the function code $10
- the relative address of the first register that was written
- the quantity of words that were written

The controller will append a CRC value to the message.

Device Address	**Function Code 16**	**First Register Relative Address**		**Word Count (Number of Words Written)**		**CRC**	
1 byte	1 byte containing $10	MSB	LSB	MSB	LSB	MSB	LSB

The value in **Word Count** field will equal the **actual** number of words written. If a write operation fails part way through the register list (either due to address or data error), then the write operation will terminate at the point of failure and the number of words written will be less than the number of words in the write command message. For example, suppose that addresses $00, $01, and $03 are valid while address $02 is not. If the host issues a request to write four non-IEEE registers starting at address $00, then the value in the **Word Count** field in the reply message will be two instead of the requested four.

In this case the controller will not send an invalid address 02 error code, because the first address in the write request was valid (see 2.4.6).

3.4.6 Reply to Function 16 ($10) Request Containing Illegal Register Address: 02 ($02) Error Code

The controllers will issue an error reply containing the MODBUS error code of $02 only if the following condition is true:

- **First Register Relative Address** field does not point to a valid Omega MODBUS register.

The format for an error code $02 response to a function $10 message is shown below. Note that the controller adds $80 to function code $10 in the reply.

Device Address	**Function Code**	**Error Code**	**CRC**	
1 byte	$90	$02	MSB	LSB

A code $02 message will be returned only if the **First Register Relative Address** field contains an address of an invalid register. When an invalid address occurs after the first register, the write operation stops at the first invalid address and the actual number of words written is sent to the host in the **Word Count** field of the normal reply.

3.4.7 Reply to Function 16 ($10) Request Containing Illegal Value in Data Field: 03 ($03) Error Code

An error reply containing the MODBUS error code $03 will be issued if either of the following conditions is true:

- The **Data** for the first register to be written contains a value that is not a valid value for the destination (target) register, or
- The **First Register Relative Address** field contains an address that is a read-only value.

The format for an error code $03 response to a function $10 message is shown below. Note that the controller adds $80 to function code $10 in the reply.

Device Address	Function Code	Error Code	CRC	
1 byte	$90	$03	MSB	LSB

A code $03 message will be returned only if the data to be written to the first register is invalid. If the write operation fails after writing to the first register, the write operation stops at the first invalid data and the actual number of words written is sent to the host in the **Word Count** field of the normal reply.

When the **Word Count** field in a function code $10 reply message does not match the **Word Count** in the request, it is an indication that a write failure occurred after writing to the first register.

3.4.8 Circumstances Under Which No Reply Is Sent in Response to a Function 16 ($10) Request

The controllers will not reply in response to a function code $10 message if the following condition exists:

- The message was a broadcast. However, in this case the controllers do write the value, even though they do not reply.

The controllers will not reply, nor act in response to a function code $10 message if any of the conditions listed below exist. If any of these conditions exist, the controller does not write the values to its database.

- The value of the CRC field does not equal the CRC calculated by the controller based on the message received, or
- The value in the **Word Count** field is greater than the maximum of 24 ($18), or
- The value in the **Byte Count** field is not equal to two multiplied by the value in the **Word Count** field, or
- Number of data bytes in the request does not equal the number indicated in the **Byte Count** field, or
- The value in the **Word Count** field is not an even value when addressing 32-bit IEEE registers. Each 32-bit IEEE register is two words long and thus, the **Word Count** field must contain an even value.

3.5 Function 08 ($08): Loopback Test

3.5.1 Introduction

So that you can test communication over the MODBUS network, Omega supports the diagnostic **function 08 ($08), subfunction 00 ($00)** to perform a loopback test. When a loopback test is performed, the MODBUS master sends a message to a controller and, if the message is received without errors, the controller sends the same message back to the master. (No changes are made to the controller database as a result of processing the command.) If the master does not receive a reply, it is time to troubleshoot the network or the controller's communication setup.

The MODBUS specification does not support broadcasting function 08.

When using the 08 ($08) loopback diagnostic function, you must specify the following data:

- the controller ID of the device to which the values are to be written
- the function code $08
- the diagnostic subfunction code $00
- any two bytes of data

The master must append a CRC value to the message.

3.5.2 Function 08 Request

The format for a function 08 subfunction 00 request is shown below.

Device Address	Function Code 08	Diagnostic Subfunction Code 00	Loopback Data (2 Bytes)		CRC	
1 byte	1 byte containing $08	2 bytes containing $00 $00	MSB	LSB	MSB	LSB

The **Diagnostic Subfunction Code** field must contain the value of zero in both bytes.

The **Loopback Data** field can contain any value.

3.5.3 Function 08 Example

Here is an example of a loopback test message sent to controller 56 ($38).

Device Address	Function Code 08	Diagnostic Code 00 (2 bytes)		Loopback Data (Two Bytes)		CRC	
$38	$08	$00	$00	$AA	$BB	DB	B1

3.5.4 Function 08 Normal Reply

If the controller receives the message and the CRC is correct, the controller will reply by sending back an echo of the request from the master. That means that the format and content of the reply will be exactly the same as the format and content of the message sent by the master.

Device Address	**Function Code 08**	**Diagnostic Subfunction Code 00**	**Loopback Data (2 Bytes)**		**CRC**	
1 byte	1 byte containing $08	2 bytes containing $00 $00	MSB	LSB	MSB	LSB

3.5.5 Circumstances Under Which No Reply Is Sent in Response to a Function 08 Request

The controllers will not reply to a function code $08 message if any of the following conditions exist:

- The message was a broadcast, or
- The value of the CRC field does not equal the CRC calculated by the controller based on the message received, or
- The value in the **Diagnostic Subfunction Code** field does not equal 0.

4. MODBUS Register Ranges and Data Types Used by Omega

4.1 Introduction

This section describes:

- the types of values used by the controllers for configuration parameter values, setpoints, etc. (see 3.2)
- how the controllers store different types of values, depending on the type of input the controller receives (see 3.3 and 3.4)
- what MODBUS register regions are available to transmit values to and from the controllers (see 3.5 to the end of this section).

4.2 Types of Values Used by the Controllers

4.2.1 Overview

Two types of values are used by Omega controllers as configuration parameter values, setpoints, etc.:

- values that can be only integers
- values that can include fractional values, that is, numbers that can include decimal places (such as 123.6, 9.75, and –3.6); the number of decimal places supported is configurable (see 4.4.2). If the controller receives its input from a thermocouple or RTD, zero or one decimal place can be used. If the controller receives a linear input, the controller can use zero, one, two, or three decimal places.

4.2.2 Examples

Examples of values used by the controllers that can be only integers are the configuration parameter values that represent the input type (0 = B thermocouple, 1 = C thermocouple, 2 = E thermocouple, etc.) and the values that represent the controller mode (1 = manual, 2 = standby, 3 = automatic, etc.).

Examples of values that can include decimal values are the setpoint, process value, and proportional band.

4.3 How the Controller Stores Values That Are Always Integers

4.3.1 Principle

Controller values that are always used as integers are stored as integers. This makes transmission of these values straightforward. Each value can be transmitted as a single register as described in 3.5. Determining the value to write from the MODBUS host (or interpreting the value read from the controller) is also uncomplicated: the value written (or read) using a MODBUS register is the value used by the controller.

4.3.2 What Is Displayed on the Controller Front Panel

If the integer represents a numerical value, such as the fixed output percentage used in manual mode, then the integer stored in the controller is the same as the value displayed on the controller front panel.

However, in most cases, a parameter value that is always used and stored as an integer represents something else. For example, consider the value stored for the input type. If 0 is stored in the controller's database as the value of the input parameter, the controller will display **`b`** (B thermocouple) as the value of the input type parameter if the configuration menus are accessed using the front panel of the controller. If the value stored for the input parameter is 1, the controller will display **`C`** (C thermocouple), etc.

4.4 How the Controller Stores Values That Can Include Fractional Values

4.4.1 Principles

In the case of values that can include decimal places, such as the setpoint, process value, or proportional band, the controller does not always store the value that it actually uses.

When using a MODBUS register to write and read any value that can include a fractional value, the value transmitted will be based on the stored value.[12] If the stored value is not the same as the value the controller uses, you must take that into account when writing or reading the parameter using a MODBUS register.

The value stored depends on several factors that must be considered when determining the value to write to one of these parameters (or when interpreting a value read from the controller).

- Some parameters always store the decimal point as part of the value. These parameters are marked **FV** in the **Type** column of the tables in Section 4. For example, the tuning parameter used to store the rate (derivative action) in Proportional-Integral-Derivative (PID) control is marked **FV** in the table in 4.6.2. This means that if the controller uses a rate of 6.5, that is the value stored in the controller's database. The 6.5 will also be the value on which the value transmitted in a MODBUS register is based.
- Some parameters store the decimal point if the input is a thermocouple or an RTD, but not if the input is linear. These parameters (the majority) are marked **FV∗** in the tables in Section 4. For example, the setpoint is marked **FV∗** in the **Type** column of the table in 4.14.2. This means that if the controller uses a setpoint of 150.5 and the input is from a thermocouple or RTD, the setpoint stored in the controller's database will be 150.5. However, if the controller uses a setpoint of 150.5 and the input is linear, the value stored will be 1505.[13] The controller interprets this stored value correctly by inserting the decimal point in the correct position, based on the value of a configuration parameter that specifies the decimal position.

4.4.2 What Is Displayed on the Controller Front Panel

The value displayed on the controller front panel for a parameter that can include a fractional value is the value used by the controller, within the limits imposed by a four-character display.

For example, if the rate tuning parameter used by the controller is 6.5, then that is the value stored, as well as the value displayed.

[12] The value transmitted will be based on the value stored in the controller. Depending on the register range used, the value transmitted may be exactly the same as the value stored in the controller, or it may be a calculated value based on the value stored. These concepts will be explained later in this section.

[13] A "linear" input is a current, voltage, or millivolt input that varies in direct proportion to the measured process variable.

If the setpoint is 150.5, then that is what is displayed, regardless of the type of input. If the input is an RTD or thermocouple, the controller displays the value stored in its database: 150.5. If the input is linear, the controller's processor reads the 1505 stored in its database, inserts the decimal point based on the configured maximum number of decimal positions, and displays the value used by the controller: 150.5.

If the setpoint is 1200, then that is what is stored in the controller's database, regardless of the type of input. The 1200 is also the value displayed.

What about a setpoint of 1200.5? A five-digit value cannot be configured using the controller front panel, because the display supports a maximum of four characters. However, using a MODBUS host you can transmit a five-digit value and the controller can store and use it, as described later in this section.

4.5 Using MODBUS to Transmit Controller Values That Are Always Integers: Register Addresses 4000 to 4999

4.5.1 Principles

Because each MODBUS register contains a 16-bit value, reading from (and writing to) a parameter for which the controller can use only an integer is straightforward. The integer can be transmitted in a single register as a 16-bit integer value. Negative integers are transmitted in two's complement format.

Parameters that are always integers are marked `I` in the **Type** column in the tables in Section 4.

4.5.2 Examples

Examples of integer values that are used by the controller are the value that represents the input type (0 = B thermocouple, 1 = C thermocouple, 2 = E thermocouple, etc.) and the value that represents the controller mode (1 = manual, 2 = standby, 3 = automatic, etc.).

To specify that the input type will be a C thermocouple, find the relative address for the input type parameter in the table in 4.3. Use this relative address 4049 ($FD1) in a function 06 ($06) or 16 ($10) request to write the value 1 ($01) to the controller.

4.6 Overview of Using MODBUS to Transmit Controller Values That Can Include Fractional Values

4.6.1 Three Register Ranges Available

Transmitting a value that can include a fractional value is a greater challenge than transmitting a value that is always an integer. The Omega MODBUS implementation provides three methods of transmitting each controller value that can include decimal places.

- in a single register as an integer using the region 0000 to 0999; what happens to the fractional value depends on how the value is stored in the controller; see 3.7 for details on using the 0000 to 0999 region
- in a single register as an integer that is 10X the value stored in the controller, using the register region 1000 to 1999 as described in 3.8
- in two registers as a 32-bit IEEE representation of a floating point value in a single read or write operation using the register region 8000 to 9999; when using this region, you can transmit exactly the value stored in the controller as described in 3.9

<u>Regardless of the range used, the value transmitted is based on the value stored in the controller</u>. As mentioned earlier in this section, the value stored is not always the value <u>used</u> by the controller in its calculations. If the controller does not store the decimal point in its database, then the decimal point will not be included in the value read from the controller, even if you read the value as a 32-bit IEEE floating point value.

4.6.2 Which Region to Use

4.6.2.1 First Choice for All Parameters and All Input Types: IEEE Floating Point Region 8000 to 9999

If the MODBUS master can handle 32-bit floating point values, then use the 8000 to 9999 range.

If the controller stores the decimal point in the value, then the value transmitted using two registers (per value) in the 32-bit IEEE region will not require any manipulation in the host (other than interpreting the 32-bit IEEE representation of the value). The controller stores the decimal point for parameters marked **FV** in the tables in Section 4 <u>regardless of input type</u>, as well as for parameters marked **FV∗** parameters, <u>if a thermocouple or RTD input is used</u>.

Even if the controller does not store the decimal point in the value (which means that the host must insert the decimal point in the correct position when interpreting a value read from the controller), there is still an advantage to using the 32-bit IEEE floating point region. If you use the 32-bit IEEE region to write a value to the controller, you are not constrained by the four-character display.

For example, using the front panel of the controller, it is impossible to configure the setpoint as 120.75, because the controller cannot display five characters. However, using MODBUS and the 32-bit IEEE floating point region, you can write a setpoint to the controller that contains a fractional value, and has more than four digits.

Use of the 32-bit IEEE floating point region is described in greater detail in 3.9.

4.6.2.2 Second Choice with Temperature Inputs: 10X Region 1000 to 1999

If the MODBUS master can only handle integers, then your best choice for transmitting **FV** (regardless of input type) and **FV∗** controller values <u>when temperature inputs are used</u> is the 10X region 1000 to 1999.

When you use this region, the value transmitted is ten times the value stored in the controller. This enables you to transmit a fractional value as an integer with greater accuracy than you can using the base region 0000 to 0999.

For example, the setpoint is marked **FV∗** in the table in 4.14.2. This means that if the controller uses a temperature input (thermocouple or RTD), the decimal point is stored in the controller's database with the value. Therefore, if the controller uses a setpoint of 150.5, the setpoint stored in the controller's database will be 150.5. When this value is transmitted using the 10X region, the value is transmitted as 1505. (The maximum number of decimal places allowed with temperature inputs is one.)

The 10X region is not recommended for **FV∗** parameters when a linear input is used. In the case of **FV∗** parameters when a linear input is used, the controller does not store the decimal point in its database, but the controller does store the digits in the fractional value. If the controller uses a linear input, up to three decimal places are supported. However, ten times a value with that many digits can exceed the range of values that can be transmitted as an integer in a single register: –32768 to 32767.

Using the registers in the10X region (addresses 1000 to 1999) is described in greater detail in 3.8.

4.6.2.3 Second Choice with Linear Inputs: Base Region 0000 to 0999

If the MODBUS master can handle only integers, then your best choice for transmitting **FV∗** controller values <u>when a linear input is used</u> is the base region 0000 to 0999.

For **FV∗** parameters when a linear input is used, the controller does not store the decimal point in its database, but the controller does store the digits in the fractional value. As a result, a value such as 75.75 is stored in the controller's database as an integer 7575. When the controller uses this value, the controller checks the Decimal Position parameter, sees that the decimal position is 2, and inserts the decimal point in the correct position, manipulating 7575 into 75.75. Up to three decimal places are allowed if the controller uses a linear input.[14]

The MODBUS master must parallel this operation. Using a register in the 0000 to 0999, the master must be programmed to read the integer stored in the controller's database, then read the Decimal Position, and insert the decimal point in the correct place. When writing values, the MODBUS master must be programmed to check the number of decimal positions supported, then write the appropriate integer to the controller using the 0000 to 0999 region.

Using the base region 0000 to 0999 is described in greater detail in 3.7.

[14] When a linear input is used and the Decimal Position is not 0 (zero), the range of permitted values is reduced for both **FV** and **FV∗** parameters. See 5.1.3.3 for more information.

4.7 Base Fractional Value Region: Register Addresses 0000 to 0999

4.7.1 General Principles

4.7.1.1 Introduction

The most convenient way to transmit any controller value that includes a fractional value is using the registers in the 32-bit IEEE region (addresses 8000 to 9999). If your process requires use of fractional values and the MODBUS master can handle 32-bit IEEE values, skip to 3.9.

If your MODBUS master can handle only integers, then your options for transmitting values that include decimal places are the registers in the 0000 to 0999 region described here and the registers in the 1000 to 1999 10X mirror of the base region described in 3.8. Read on.

4.7.1.2 Base Region Uses One Register Per Value Transmitted

When a controller value that includes a decimal value is transmitted in the 0000 to 0999 region, the value is transmitted in a single MODBUS register as an integer.

The ways the fractional portion of the value is handled when read as an integer using the 0000 to 0999 region depend on the type of parameter and, sometimes, the type of controller input.

4.7.1.3 Range of Values That Can Be Transmitted in Base Region

The limits for a value transmitted as an integer are – 32768 to 32767. The controller values that can be represented within that range depend on the number of decimal places used and the value stored.

4.7.1.4 If Controller Is Configured to Support Decimal Places, Then the Controller's Storage Method for a Parameter Affects Values Read and Written

Many parameters in the controllers can use fractional values. However, that does not mean that you are forced to use fractional values. Every controller permits you to configure a Decimal Position parameter. By default, the controllers are set to use zero decimal places. For many temperature applications, fractional values are not needed.

However, if the controller is configured to use decimal places, the fractional value is not stored the same way for every parameter that can use a fractional value. The value stored by the controller is not always the value used by the controller.

When a configuration parameter (or other controller value that can include decimal values) is read using the registers in the 0000 to 0999 base region, the value returned to the MODBUS master in the controller's reply message depends on how the controller handles the fractional portion of the value for that particular parameter. If the controller is configured to support decimal places for parameters that can use fractional values, then the MODBUS master must also take into account how the controller stores a particular parameter when attempting to read from or write to the controller using the registers in the 0000 to 0999 region. See 3.7.2 and 3.7.3 for more information.

If your process does not require the precision of fractional values, then leave the Display menu Decimal Places parameter (register 4068 for RTD and thermocouple inputs; register 4069 for linear inputs) at the default 0 (zero). Use the base region relative address provided for each parameter in Section 4 to transmit as integers parameter values that can support fractional values. You can skip the rest of this chapter (although the summary in 3.10 is useful).

4.7.2 Interpreting Fractional Values Stored with the Decimal Point Transmitted in Base Region 0000 to 0999

4.7.2.1 Principles

If the **Type** column for a parameter in the table in Section 4 contains **FV∗**, and the input is a thermocouple or an RTD, then the controller stores the decimal point, as well as the digits in the fractional value for this parameter.

If the **Type** column for a parameter in the table in Section 4 contains **FV,** then the controller always stores the decimal point, as well as the digits in the fractional value for this parameter, regardless of input type.

If you read or write one of these parameter values using the registers in the 0000 to 0999 region, the value will be rounded to the nearest integer.

- Values of .1, .2, .3. and .4 are rounded downward.
- Values of .5, .6, .7, .8, and .9 are rounded upward.

4.7.2.2 Examples

For example, suppose the input is not linear, and the controller is configured to use one decimal place (the maximum allowed for temperature input types). In this case, an Alarm 1 Process Alarm Setpoint (**FV∗**) of 120.5 would be stored by the controller as 120.5. If you used a MODBUS 03 ($03) function to read this alarm setpoint as an integer using register 36, the read would return the value rounded to 121.

Similarly, if you tried to read the PID Derivative (Rate) **FV** value of 2.1 using register 11, the read would return 2.

If the input is not linear and you use a MODBUS 06 ($06) or 16 ($10) write function to configure an **FV∗** parameter such as the Alarm 1 Process Alarm Setpoint (register 36), then your only option when using the 0000 to 0999 region is to configure the parameter with an integer value. Regardless of whether the input type is linear or temperature, your only option when using the 0000 to 0999 region to write to an **FV** parameter is to configure the parameter with an integer value.

4.7.3 Interpreting Fractional Values Stored Without Decimal Point Transmitted in Base Region 0000 to 0999

4.7.3.1 Principles

For some parameters if the input is linear the controller stores the value with the decimal point removed, but the digits in the fractional value retained. When the controller uses one of these values for controller operations, the controller inserts the decimal point as appropriate.

The number of decimal places used by each controller is determined by the value assigned to the Decimal Position parameter. The relative address for the Decimal Position parameter is 4069 when the input is linear.

If the **Type** column for a parameter in the table in Section 4 contains **FV∗**, then the method used by the controller to store a fractional value for this parameter depends on the type of input used by the controller. In the case of linear inputs, the controller drops the decimal point, but retains the digits in the decimal value.

If you read one of these values using a register in the 0000 to 0999 region, you must program the MODBUS master to also read the value assigned to the linear Decimal Position parameter (register 4069) and use the information to insert the decimal in the correct position before using the value read.

4.7.3.2 Examples

For example, suppose the controller is configured to use two decimal places and the input is linear. In this case, an Alarm 1 Process Alarm Setpoint (**FV∗**) of 95.75 would be stored by the controller as 9575. Therefore, if you used a MODBUS 03 ($03) function to read this alarm setpoint as an integer using register 36, you would have to program the MODBUS master to insert the decimal point based on the Decimal Position parameter value read using integer register 4069.

Similarly, if you used a MODBUS 06 ($06) or 16 ($10) write function to configure the Alarm 1 Process Alarm Setpoint as 95.75 using register 36, you would have to write the value as 9575.

4.8 10X Mirror of Base Fractional Value Region: Register Addresses 1000 to 1999

4.8.1 Introduction

The value of every parameter (or other controller value) that can include a decimal value can be transmitted using any of three different regions.[15] As described in 3.7, the registers in the base region of 0000 to 0999 can be used to transmit the value actually stored in the controller database only when a linear input is used and the parameter is marked **FV∗** in the tables in Section 4. In this special case, the controller actually stores the entire value as an integer with the decimal point dropped, so reading (or writing) an integer in a single register can transmit the entire value, including the fractional portion of the value. (The controller inserts the decimal point at the appropriate location when the controller uses such a value in its calculations. The MODBUS master must do the same.)

In all other cases (**FV∗** with non-linear input, or **FV** with either linear or temperature input), the value transmitted using the registers in the 0000 to 0999 region will be rounded to the nearest integer.

The 10X mirror region has been provided to enable you to transmit a fractional value for an **FV** parameter or an **FV∗** parameter with a temperature input as an integer with greater accuracy.

<u>If the MODBUS master can handle 32-bit IEEE floating point values, skip this sub-section</u>. Go directly to 3.9.

4.8.2 General Principles

4.8.2.1 10X Region Uses One Register Per Value Transmitted

<u>Each of the 16-bit integer values transmitted using a single register in the 10X mirror region is a rounded off value calculated by multiplying the value stored in the controller by 10, then rounding the result to the nearest integer</u>.

4.8.2.2 If Controller Is Configured to Support Decimal Places, Then the Controller's Storage Method for a Parameter Affects Values Read and Written

Many parameters in the controllers can use fractional values. However, that does not mean that you are forced to use fractional values. Every controller permits you to configure a Decimal Position parameter. By default, the controllers are set to use zero decimal places. For many temperature applications, fractional values are not needed.

[15] There is no need to mirror the region used to read and write values that are stored in the controllers' databases as integers.

However, if the controller is configured to use decimal places, the fractional value is not stored the same way for every parameter that can use a fractional value. The value stored by the controller is not always the value used by the controller. When a configuration parameter (or other controller value that can include decimal values) is read using the registers in the 1000 to 1999 10X region, the value returned to the MODBUS master in the controller's reply message depends on how the controller handles the fractional portion of the value for that particular parameter. If the controller is configured to support decimal places for parameters that can use fractional values, then the MODBUS master must also take into account how the controller stores a particular parameter when attempting to read from or write to the controller using the registers in the 1000 to 1999 region. See 3.8.3 and 3.8.4 for more information.

4.8.2.3 Relative Addresses in 10X Region Are 1000 More Than the Corresponding Address in the Base Region

There is a one-to-one correspondence between the addresses in the 0000 to 0999 region and the 1000 to 1999 region. The controller values are stored in the two regions in precisely the same sequence. Therefore, the address of a 10X mirror register can be calculated by adding 1000 to the address of a floating point register in the 0000 to 0999 region.

4.8.2.4 Because of Range Limits, the 10X Region Not Recommended for FV∗ Parameters When a Linear Input Is Used

The 10X region is not recommended for **FV∗** parameters when a linear input is used. In the case of **FV∗** parameters when a linear input is used, the controller does not store the decimal point in its database, but the controller does store the fractional value. If the controller uses a linear input, up to three decimal places are supported. However, ten times a value with that many digits can exceed the range of values that can be transmitted as an integer in a single register: –32768 to 32767. If ten times the stored value is outside of this range, the value returned by the read operation will be –32768 or 32767.

4.8.3 Interpreting Fractional Controller Values Stored With Decimal Point Transmitted in the 10X Region 1000 to 1999

4.8.3.1 Principles

For some parameters the controller stores the decimal point, as well as the digits in the fractional value. The parameters that fall in this category are:

- parameters marked **FV** in the tables in Section 4 (regardless of input type), and
- parameters marked **FV∗** when the input is a thermocouple or an RTD.

When reading one of these values using a register in the 10X region, the value returned is 10 times the stored value. The product is rounded to the nearest integer.

The exception is when ten times the stored value exceeds the range of values that can be transmitted as an integer in a single register: –32768 to 32767. If ten times the stored value is outside of this range, the value returned by the read operation will be –32768 or 32767.

To determine the stored value (to the closest tenth):

1. Read the value using the appropriate register in the 10X region.
2. Divide the value read in Step 1 by 10.

<u>When writing one of these values using the 10X mirror region, the MODBUS master must be programmed to write a value that is ten times the desired value (rounded to the nearest tenth)</u>. To do this:

1. Decide what value you want the controller to use.
2. Express that value to the closest tenth.
3. Multiply the value obtained in Step 2 by 10.
4. Write the product obtained in Step 3 using the appropriate register in the 10X region.

4.8.3.2 Examples

4.8.3.2.1 Reading an **FV∗** Parameter with Temperature Input

Suppose the controller uses a J thermocouple input and the alarm 1 process alarm setpoint used by the controller is 150.5. To read this value using MODBUS and the 10X region:

1. Check the table in Section 4. You will see that the alarm setpoint is designated **FV∗**. Therefore, you know that if the input comes from an RTD or thermocouple, this value is stored in the controller's database with its decimal point.
2. Read the input type from the integer register with the relative address 4049. The value is 3, which represents a J thermocouple, so you know that the controller is storing the value with the decimal point. Only one division will be necessary to obtain the value (in contrast to values that are not stored with the decimal point; see 3.8.4).
3. Read the alarm 1 process alarm setpoint value using the register with the relative address of 1036 in the 10X region. This is 1505. (150.5 x 10 = 1505.)
4. Divide 1505 by 10 to get the value stored in the controller. This quotient is 150.5.

4.8.3.2.2 Reading an **FV** Parameter

Now suppose you want to read the Derivative (Rate) tuning parameter from this same controller. The controller is currently configured to use a rate of 1.7. To read this value using the 10X region:

1. Check the table in Section 4. You will see that the Derivative (Rate) is designated **FV**. Therefore, you know that the type of input does not matter. The value for this configuration parameter is always stored with its decimal point.
2. Read the Derivative (Rate) value using the register with the relative address of 1011 in the 10X region. This is 17. (1.7 x 10 = 17.)
3. Divide 17 by 10 to calculate the value stored in the controller. This quotient is 1.7.

4.8.3.2.3 Writing to an **FV** Parameter

Suppose you want to change the Derivative (Rate) tuning parameter in this controller to 2.2 using the 10X region.

1. Multiply 2.2 times 10. This is 22.
2. Because you remember that the Derivative (Rate) is designated **FV**, you do not have to check the table in Section 4 again to know that the controller stores this tuning parameter with the decimal point, so you know that no more multiplication is needed to arrive at the value stored in the controller. Write 22 to the controller using the register with the relative address of 1011 in the 10X region.
3. When the controller receives the write command, the controller will recognize that the register is in the 10X region and divide 22 by 10.
4. The controller will store 2.2 as the new Derivative (Rate) tuning parameter value.

4.8.4 Interpreting Fractional Controller Values Stored Without Decimal Point Transmitted in the 10X Region 1000 to 1999

4.8.4.1 Principles

For some parameters (those marked **FV∗** in the tables in Section 4) if the input is linear, the controller stores the configuration value with the decimal point removed, but the digits in the fractional value retained. When one of these parameter values is read using a register in the 1000 to 1999 region, the value returned will be ten times the value stored in the controller. Because the decimal point has been dropped, but the fractional value retained, this is not the same as ten times the value actually used by the controller.

When the controller uses one of the **FV∗** values for controller operations if the input is linear, the controller inserts the decimal point as appropriate, based on the value assigned to the linear Decimal Position parameter. The value displayed and used by the controller is the correct value. For example, suppose the controller is configured to use two decimal places and the input is linear. In this case, an Alarm 1 Process Alarm Setpoint (**FV∗**) stored by the controller as 9575 would be displayed correctly as 95.75. The 95.75 value would also be used by the controller as the alarm setpoint.

When reading one of these values using the 10X mirror region, the MODBUS master must be programmed to read the linear Decimal Position parameter (register 4069) and to calculate the actual value used by the controller. To do this:

1. Read the value using the appropriate register in the 10X region.
2. Divide the value read in Step 1 by 10.
3. Read the decimal position using the 4069 integer register.
4. Insert the decimal point at the appropriate position in the quotient from Step 2 (that is, divide by the appropriate value: 1, 10, 100, or 1000).

When writing one of these values using the 10X mirror region, the MODBUS master must be programmed to write a value that is ten times the desired value without its decimal point. To do this:

1. Read the decimal position using the integer register at relative address 4069.
2. Decide what value you want the controller to use.
3. Express that value with the same number of decimal places the controller uses.
4. Drop the decimal point, but retain the digits after the decimal point (that is, multiply by the appropriate value: 1, 10, 100, or 1000).
5. Multiply the value obtained in Step 4 by 10.
6. Write the product obtained in Step 5 using the appropriate register in the 10X region.

However, the 10X region is not recommended for **FV∗** parameters when a linear input is used, because the decimal point is not stored as part of the value in the controller. If the controller uses a linear input, up to three decimal places are supported. As a result, the value stored in the controller's database can be 1000 times the value actually used by the controller. When multiplied by ten again to transmit the value in the 10X region, the product can exceed the range of values that can be transmitted in a single integer register. The transmitted value would be clipped to the limit (–32768 or 32767), but the MODBUS host would not be aware that the value transmitted is not the correct value. The example in 3.8.4.2.3 demonstrates this situation.

4.8.4.2 Examples

4.8.4.2.1 Reading an **FV∗** Parameter from a Controller with a Linear Input – Range Not Exceeded

Suppose the controller uses a 4 to 20 mA input, the decimal position is 1, and the alarm 1 process alarm setpoint used by the controller is 150.5. To read this value using MODBUS and the 10X region:

1. Check the table in Section 4. You will see that the alarm setpoint is designated **FV∗**. Therefore, you know that if the input is linear, this value is a special case. The decimal point is not stored with the value.
2. Read the input type from the integer register with the relative address 4049. The value is 14, which represents a 4 to 20 mA input, which is linear.
3. Read the alarm 1 process alarm setpoint value using the register with the relative address of 1036 in the 10X region. This value is 15050.
4. Divide 15050 by 10 to get the value stored in the controller. This quotient is 1505.
5. Because you have determined that this is a special case (**FV∗** parameter with a linear input) you know that the value stored in the controller is missing its decimal point. Read the decimal position value in the integer register with the relative address of 4069. It is 1.
6. Because the controller is using one decimal place, you know that you must divide 1505, the quotient obtained in Step 4, by 10. The new quotient is 150.5. This is the alarm 1 process alarm setpoint the controller is currently configured to use.

4.8.4.2.2 Writing an **FV∗** Parameter to a Controller with a Linear Input – Range Not Exceeded

To change the alarm 1 setpoint to 175.90 using the 10X region in the same controller (4 to 20 mA input; decimal position = 1):

1. Read the decimal position using the integer register at relative address 4069.
2. Express the new setpoint with the same number of decimal places the controller uses. This is 175.9.
3. Drop the decimal point, but retain the digits after the decimal point (that is, multiply by 10). This is 1759. This is the value you want the controller to store for the alarm 1 setpoint parameter.

4. Multiply the 1759 (value obtained in Step 3) by 10. This is 17590.
5. Write 17590, the product obtained in Step 4, to the register in the10X region with the relative address of 1036.
6. When the controller receives the write command, the controller will recognize that the register used to transmit the value is in the 10X region, so the controller divides 17590 by 10. The controller writes the quotient, 1759, to its database as the new alarm 1 setpoint.
7. When the controller uses the setpoint, the controller will insert the decimal point at the correct position, yielding 175.9.

4.8.4.2.3 Reading an **FV∗** Parameter from a Controller with a Linear Input – Range Exceeded

In the examples above, we were able to use the 10X region to transmit integers that did not exceed the 32767 limit. However, suppose the controller with a 4 to 20 mA input, uses a decimal position of 3, which is permitted with linear inputs. If we try to read the alarm 1 process alarm setpoint of 7.895 we run into trouble. Repeating the procedure to read the value using MODBUS and the 10X region:

1. Check the table in Section 4. You will see that the alarm setpoint is designated **FV∗**. Therefore, you know that if the input is linear, this value is a special case. The decimal point is not stored with the value.
2. Read the input type from the integer register with the relative address 4049. The value is 14, which represents a 4 to 20 mA input, which is linear.
3. Read the alarm 1 process alarm setpoint value using the register with the relative address of 1036 in the 10X region. The stored setpoint is 7895. When multiplied by 10, the product 78950 exceeds 32767, the top of the range of values that can be transmitted as an integer in a single register. Therefore, the read will return 32767. <u>From here to the end of the procedure, the results of the MODBUS master's manipulation of the returned value yield false results</u>, but the MODBUS master has no way of recognizing that the values are bad.
4. Divide the returned value 32767 by 10 to get the value stored in the controller. This quotient is 3276.7. However, it is obviously <u>not</u> the actual value stored in the controller, because the actual product of multiplying 10 times the stored value exceeded the range of values transmittable in a single register.
5. Because you have determined that this is a special case (**FV∗** parameter with a linear input) you know that the value stored in the controller is missing its decimal point. Read the decimal position value in the integer register with the relative address of 4069. It is 3.
6. Because the controller is using three decimal places, you know that you must divide 3276.7, the quotient obtained in Step 4, by 1000. The new quotient is 3.277.

Although the MODBUS master has performed its calculations correctly, the result is <u>not</u> the alarm 1 process alarm setpoint the controller is currently configured to use, because the range has been exceeded. This example illustrates our reason for recommending that you not use the 10X region for **FV∗** parameters when the input in linear.

4.9 32-bit IEEE Mirror of Base Fractional Value Region: Register Addresses 8000 to 9999

4.9.1 General Principles

4.9.1.1 Introduction

The third region used to transmit the value of configuration parameters (or other controller values) that can include a decimal value is the 32-bit IEEE mirror in region 8000 to 9999.

If the MODBUS master can interpret 32-bit IEEE floating point values, then the 8000 to 9999 region is the most convenient to use.

4.9.1.2 32-Bit IEEE Region Uses Two Registers Per Value Transmitted

Each 32-bit floating point value transmitted in two registers in the 32-bit IEEE region is the 32-bit IEEE representation of the value of the parameter stored by the controller. The value stored is not always the value used by the controller; see 3.9.1.4.

4.9.1.3 Range of Values That Can Be Transmitted in the 32-Bit IEEE Region

The registers in the 32-bit IEEE region can transmit any value that the controller can store.

4.9.1.4 If the Controller Is Configured to Use Decimal Places, Then the Controller's Storage Method for a Parameter Affects Values Read and Written

Many parameters in the controllers can use fractional values. However, that does not mean that you are forced to use fractional values. Every controller permits you to configure a Decimal Position parameter. By default, the controllers are set to use zero decimal places. For many temperature applications, fractional values are not needed.

However, if the controller is configured to use decimal places, the fractional value is not stored the same way for every parameter that can use a fractional value. The value stored by the controller is not always the value used by the controller.

When a configuration parameter (or other controller value that can include decimal values) is read using the registers in the 8000 to 9999 32-bit IEEE region, the value returned to the MODBUS master in the controller's reply message depends on how the controller handles the fractional portion of the value for that particular parameter. If the controller is configured to support decimal places for parameters that can use fractional values, then the MODBUS master must also take into account how the controller stores a particular parameter when attempting to read from or write to the controller using the registers in the 8000 to 9999 region.

If the controller stores the decimal point in the value, then the value transmitted using two registers (per value) in the 32-bit IEEE region will not require any manipulation in the host (other than the interpretation of the 32-bit IEEE representation of the value). The controller stores the decimal point for parameters marked **FV** in the **Type** column of the tables in Section 4 regardless of input type, as well as for parameters marked **FV∗**, if a thermocouple or RTD input is used.

If a linear input is used, then the value of a parameter marked **FV∗**in the **Type** column of the tables in Section 4 is stored in the controller's database without the decimal point, but with the digits in the fractional value (that is, the stored value is 10, 100 or 1000 times the value used by the controller, depending on how many decimal places the controller is configured to use).

4.9.1.5 32-Bit IEEE Region Allows You to Configure the Controller to Use Values with More Than Four Digits

Even if the controller does not store the decimal point in the value (which means that the host must insert the decimal point in the correct position), there is still an advantage to using the 32-bit IEEE floating point region. If you use the 32-bit IEEE region to write a value to the controller, you are not constrained by the four-character display. You can use the 32-bit IEEE region to transmit a five-digit value for any parameter marked **FV** or **FV∗** in Section 4, as long as the five-digit value is within the range of permitted values for the parameter.

For example, using the front panel of the controller, it is impossible to configure the setpoint as 120.75, because the controller cannot display five characters. However, using MODBUS and the 32-bit IEEE floating point region, you can write this five-digit setpoint value to the controller.

4.9.1.6 Relative Addresses in 32-Bit IEEE Region Can Be Calculated from the Relative Address of a Parameter in the Base Region

Because each IEEE value requires two registers, there cannot be a one-to-one correspondence between the addresses in the 0000 to 0999 region and the 8000 to 9999 region. However, the controller values are stored in the two regions in precisely the same sequence. Therefore, the address of the first 32-bit IEEE mirror register for each value can be determined by multiplying the address of a register in the 0000 to 0999 base region by two, then adding 8000 to the product. The address of the second of the pair of registers for the IEEE representation is the next register.

For example, the relative address of the Recipe Setpoint in the base region is 6. To calculate the relative address of the first register used to read the Recipe Setpoint in the 8000 to 9999 region, multiply 6 times 2, then add 8000 to the product. In this case, 6 x 2 = 12. 8000 + 12 = 8012. The relative address of the first register used to read the Recipe Setpoint is 8012; the second register in the pair needed to transmit the 32-bit IEEE representation of the Recipe Setpoint is 8013.

4.9.1.7 A MODBUS Command to Transmit a Value Using the 8000 to 9999 Region Always Addresses an Even-Numbered Relative Address

A 32-bit IEEE value is always addressed at the first of the two registers that are needed to store the value. Therefore, every request from the MODBUS master to read or write a 32-bit IEEE value must address an even-numbered register. Using an odd-numbered relative address in a request to read or write a 32-bit IEEE value will cause the controller to send a MODBUS error code 02 message back to the master.

4.9.1.8 Sequence in Which the Two Registers for a 32-Bit IEEE Value Will Be Transmitted

The two registers will be transmitted in the order of Least Significant Register first if standard MODBUS floating point sequence is used and Most Significant Register first if non-standard ordering is used. Note that the IEEE register ordering does not affect the sequence in which the bytes within the register are transmitted. Within the register, the most significant (high order) byte is always transmitted before the least significant (low order) byte.

The IEEE register ordering used by the controllers is configurable; see 1.2.7 for more information about register ordering for 32-bit IEEE floating point representation of values.

4.9.2 Interpreting Fractional Controller Values Stored With Decimal Point Transmitted in the 32-Bit IEEE Region 8000 to 9999

4.9.2.1 Principles

For some parameters the controller stores the decimal point, as well as the fractional value. The parameters that fall in this category are:

- parameters marked **FV** in the tables in Section 4 (regardless of input type), and
- parameters marked **FV∗** when the input is a thermocouple or an RTD.

When reading one of these values using two registers in the 32-bit IEEE region, the value returned is the stored value, which is also the value used by the controller.

To determine the stored value:

> Read the 32-bit IEEE representation of the value using the appropriate two registers in the 32-bit IEEE region.

When writing one of these values using the 32-bit IEEE region, the MODBUS master must be programmed to write a value that is the same as the value you want the controller to use. To do this:

1. Read the value of the Decimal Position parameter (register 4068 for temperature inputs; 4069 for linear inputs) to determine the number of decimal positions supported by the controller.
2. Decide what value you want the controller to use. Remember that when using the 32-bit IEEE region, you can exceed the four digits supported by the controller display.
3. Express that value with the appropriate number of decimal positions.
4. Write the value from Step 3 using the appropriate pair of registers in the 8000 to 9999 region.

4.9.2.2 Examples

4.9.2.2.1 Reading an **FV∗** Parameter with Temperature Input

Suppose the controller uses a J thermocouple input and the alarm 1 process alarm setpoint used by the controller is 150.5. To read this value using MODBUS and the 32-bit IEEE region:

1. Check the table in Section 4. You will see that the alarm setpoint is designated **FV∗**. Therefore, you know that if the input comes from an RTD or thermocouple, this value is stored in the controller's database with its decimal point.
2. Read the input type from the integer register with the relative address 4049. The value is 3, which represents a J thermocouple, so you know that the controller is storing the value with the decimal point. No division will be necessary to obtain the value used by the controller (in contrast to values that are not stored with the decimal point; see 3.9.3).
3. Read the alarm 1 process alarm setpoint value using the registers with the relative addresses of 8072 and 8073. (The two registers must be read in a single command.) The value returned will be the 32-bit IEEE representation of 150.5.

4.9.2.2.2 Reading an **FV** Parameter

Now suppose you want to read the Derivative (Rate) tuning parameter from this same controller. The controller is currently configured to use a rate of 1.7. To read this value using the 8000 to 9999 region:

1. Check the table in Section 4. You will see that the Derivative (Rate) is designated **FV**. Therefore, you know that the type of input does not matter. The value for this configuration parameter is always stored with its decimal point.
2. In a single read command, read the Derivative (Rate) value using the registers with the relative addresses of 8022 and 8023. This will return the 32-bit IEEE representation of 1.7.

4.9.2.2.3 Writing to an **FV** Parameter

Suppose you want to change the Derivative (Rate) tuning parameter in this controller to 2.2 using the 8000 to 9999 region.

1. Because you remember that the Derivative (Rate) is designated **FV**, you do not have to check the table in Section 4 again to know that the controller stores this tuning parameter with the decimal point. Therefore, you know that no multiplication is needed to arrive at the value stored in the controller. Write the 32-bit IEEE representation of 2.2 to the controller using the registers with the relative addresses of 8022 and 8023.
2. The controller will store 2.2 as the Derivative (Rate) tuning parameter.

4.9.3 Interpreting Fractional Controller Values Stored Without Decimal Point Transmitted in the 32-Bit IEEE Region 8000 to 9999

4.9.3.1 Principles

For some parameters (those marked **FV∗** in the tables in Section 4) if the input is linear, the controller stores the configuration value with the decimal point removed, but the digits in the fractional value retained. When one of these parameter values is read using a pair of registers in the 8000 to 9999 region, the value returned will be the 32-bit IEEE representation of the value stored in the controller. Because the decimal point has been dropped, but the fractional value retained, this stored value is not the same as the value actually used by the controller. The MODBUS master must manipulate the value to determine the value actually used by the controller.

When the controller uses one of the **FV∗** values for controller operations <u>if the input is linear</u>, the controller inserts the decimal point as appropriate, based on the value assigned to the linear Decimal Position parameter. The value used by the controller is the correct value. For example, suppose the controller is configured to use two decimal places and the input is linear. In this case, an Alarm 1 Process Alarm Setpoint (**FV∗**) stored by the controller as 9575 would be displayed correctly as 95.75. The 95.75 value would also be used by the controller as the alarm setpoint.

<u>When reading one of these values using the 32-bit IEEE region, the MODBUS master must be programmed to read the linear Decimal Position parameter (register 4069) and to calculate the actual value used by the controller</u>. To do this:

1. Read the 32-bit IEEE representation of the value using the appropriate two registers in the 8000 to 9999 region.
2. Read the decimal position using the 4069 integer register.
3. Insert the decimal point at the appropriate position in the value read in Step 1 (that is, divide by the appropriate value: 1, 10, 100, or 1000).

<u>When writing one of these values using the 32-bit IEEE region, the MODBUS master must be programmed to write a value that is the desired value without its decimal point</u>. To do this:

1. Read the decimal position using the integer register at relative address 4069.
2. Decide what value you want the controller to use.
3. Express that value with the same number of decimal places the controller uses.
4. Drop the decimal point, but retain the digits after the decimal point (that is, multiply by the appropriate value: 1, 10, 100, or 1000)
5. Write the 32-bit IEEE representation of the value from Step 4 to the appropriate two registers in the 8000 to 9999 region.

4.9.3.2 Examples

4.9.3.2.1 Reading an **FV∗** Parameter from a Controller with a Linear Input

Suppose the controller uses a 4 to 20 mA input, the decimal position is 1, and the alarm 1 process alarm setpoint used by the controller is 150.5. To read this value using MODBUS and the 32-bit IEEE region:

1. Check the table in Section 4. You will see that the alarm setpoint is designated **FV∗**. Therefore, you know that if the input is linear, this value is a special case. The decimal point is not stored with the value.
2. Read the input type from the integer register with the relative address 4049. The value is 14, which represents a 4 to 20 mA input, which is linear.
3. Read the 32-bit IEEE representation of the alarm 1 process alarm setpoint value using the registers with the relative addresses 8072 and 8073. This returns the 32-bit IEEE representation of 1505.

4. Because you have determined that this is a special case (**FV∗** parameter with a linear input) you know that the value stored in the controller is missing its decimal point. Read the decimal position value in the integer register with the relative address of 4069. It is 1.

5. Because the controller is using one decimal place, you know that you must divide 1505, the value read in Step 3, by 10. The quotient is 150.5. This is the alarm 1 process alarm setpoint the controller is currently configured to use.

4.9.3.2.2 Writing an **FV∗** Parameter to a Controller with a Linear Input

To change the alarm 1 setpoint to 175.90 using the 10X region in the same controller (4 to 20 mA input; decimal position = 1):

1. Read the decimal position using the integer register at relative address 4069.

2. Express the new setpoint with the same number of decimal places the controller uses. This is 175.9.

3. Drop the decimal point, but retain the digits after the decimal point (that is, multiply the desired setpoint by 10). This is 1759. This is the value you want the controller to store for the alarm 1 setpoint parameter.

4. Write the 32-bit IEEE representation of 1759 to the registers with the relative addresses of 8072 and 8073.

5. The controller writes 1759 to its database as the new alarm 1 setpoint.

6. When the controller uses the setpoint, the controller will insert the decimal point at the correct position, yielding 175.9.

4.10 Summary

4.10.1.1 Regions of the MODBUS Register Map Used to Transmit Controller Values

The regions (ranges) of register relative addresses used for the various types of representations of controller values are shown in the table below. Each register type used to transmit controller values was described in more detail earlier in this section (see the cross-references in the table). The "factory commands" that use the 7000 to 7999 range are described in Section 7.

Register Address Range	**Region**	**Quantity of MODBUS Registers to Store One Value**	**Type of Value Used by the Controller**	**How Data Is Transmitted in the Register**	**See Subsection**
0000 to 0999	base region for controller values that can include decimal values	1	can include fractional values	16-bit Integer	3.7
1000 to 1999	10X mirror of 0000 to 0999 base region	1	can include fractional values	16-bit Integer	3.8
2000 to 3999	*unused*	*N/A*	*N/A*	*N/A*	---
4000 to 4999	integer	1	integer	16-bit Integer	3.5
5000 to 6999	*unused*	*N/A*	*N/A*	*N/A*	---
7000 to 7999	factory	2	Custom	Custom	Section 7
8000 to 9999	32-bit IEEE mirror of 0000 to 0999 base region	2 (Register addresses must be even.)	can include fractional values	32-bit IEEE Floating Point	3.9

4.10.2 Parameters That Use Integer Values Only

If a configuration parameter value stored and used by the controller is always an integer (Type **I** in the tables in Section 4), then it is possible to accurately transmit the value in a single 16-bit register. For integers, you always read from and write to the 4000 to 4999 region of the MODBUS register map.

4.10.3 Parameters for Which the Controller Can Use a Fractional Value

4.10.3.1 Three Regions Available

There are three methods of reading from or writing to a parameter for which the controller can use a fractional value (**FV** and **FV*** Types in the tables in Section 4).

- You can transmit the value, rounded to an integer, using the 0000 to 0999 base region.
- You can transmit an integer that is equal to ten times the desired controller value (that includes one digit to the right of the decimal) using the 1000 to 1999 10X mirror region.
- You can transmit a 32-bit IEEE floating point representation of the entire value to the 8000 to 9999 32-bit IEEE mirror region.

All of the following examples will set the Derivative (Rate) tuning parameter to 3.

- Writing the value of 3 to relative address 11.
- Writing the value of 30 to relative address 1011 (value 30 = 3 x 10 and address 1011 = 11 + 1000).
- Writing the 4-byte IEEE floating point representation of 3.0 to address 8022 (address 8022 = (2 x 11) + 8000).

If you want to write the Derivative (Rate) tuning parameter as 3.5, then you can use either the 10X or 32-bit IEEE mirror regions.

However, if you want to write the Derivative (Rate) as 3.55, you must use the 32-bit IEEE mirror region.

4.10.3.2 Special Cases: the Significance of the FV and FV∗ Notations

In the case of Derivative (Rate), a parameter marked **FV** in the table in Section 4, the controller stores the decimal point with the value (assuming the controller is configured to use one or more decimal places).

However, the controller does not always store the same parameter value that the controller uses. If a parameter is marked **FV∗** in the tables in Section 4, the way the controller stores the value is affected by the input type (assuming that the controller is configured to use one or more decimal places).

- If the input is from an RTD or thermocouple, the controller stores the decimal point with the value for the **FV∗** parameter, as the controller does with for the parameters marked **FV**. A configuration value of 3.5 is stored in the controller's database as 3.5.
- If the input is linear, then the controller does not store the decimal point for the value of the **FV∗** parameter, but the controller does store the digits to the right of the decimal point in a fractional value. A value of 3.5 is stored in the controller's database as 35 if the controller is configured to use one decimal place, 350 if the controller is configured to use two decimal places, 3500 if the controller is configured to use three decimal places. When the controller applies the value, the controller inserts the decimal point based on the value assigned to the Decimal Position parameter.

In the case of parameters marked **FV∗**, if the controller is configured to use one or more decimal places, then you must consider the type of input used and take into consideration the way the controller stores fractional values for these parameters.

Because **FV∗** parameter values in controllers with linear inputs are stored without the decimal point, you must take into consideration the number of decimal positions the controller is configured to support. When using the base 0000 to 0999 region or 8000 to 9999 region, you may have to write a value that is 10, 100, or even 1000 times larger than the configuration value you want the controller to use. If you use the 10X region 1000 to 1999, you will have to multiply the value by 10 again.

5. Omega MODBUS Register Addresses Arranged by Parameter Function

5.1 Introduction

5.1.1 Importance of Writing Configuration Parameters in Correct Sequence

The Omega, CN8200, CN8240, and CN8260 controllers are versatile instruments that are capable of using many types of input values and implementing several types of control strategies. To support this versatility, the controllers are capable of storing values for many configuration parameters. Interrelationships exist between the parameters. Therefore, it is important that you specify values for the configuration parameters in the correct sequence.

Once you have your MODBUS network up and running, if you plan to use a thermocouple or RTD input, then the first step is to configure the units of measure (unless you plan to use the default, Fahrenheit).[16] Units of measure are not used with linear inputs.

Next, specify the type of input that each controller will receive. That means that you must specify the type of thermocouple or RTD that will provide the input to the controller, or, in the case of a linear input, the range and units of the input (0 to 20 mA, 0 to 5 V, 1 to 5 V, etc.) The type of input specified affects how the controller processes the input signal and calculates the output needed to achieve the setpoint.

The rest of the configuration sequence is outlined in 4.1.2.

In addition to being aware of the sequence in which parameters should be configured, you should also remember that not all parameters apply to all applications. For example, if you specify that the input type is a thermocouple or RTD, then you do not have to write a value to the low scale and high scale parameters. (You can write the values, but the controller will ignore them.) However, if you use a linear input, then you must specify scaling values, or accept the factory defaults.[17]

[16] When you change the units of measure for temperature inputs, the controller recalculates any values that have already been specified. For example, if you want the setpoint to be 100 °C, then you must change the units from the default F to C before you write the setpoint of 100 to the controller. If you change the units after you write the setpoint of 100 to the controller, the controller will convert the 100 °F setpoint to 37.8 °C. In this case, you would have to reconfigure the setpoint to 100 °C to implement the control needed by your process.

[17] The database values in new ("out of the box") CN8200, CN8240, and CN8260 controllers are always the default values shown in the tables in this section. Instructions for using the controller front panel to return all database values in the CN8200, CN8240, and CN8260 controllers to their default values are in the *(CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*. You can also use MODBUS function 16 ($10) to write a command to special registers in any of the controllers to set all the database (excluding the address) values to their defaults as described in Section 7.

For more information about the interrelationships between parameters and about the effects of setting specific values, see the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*.

5.1.2 Arrangement of the Parameters and Other Values in This Section

To make it easy for you to configure the controller parameters in the correct sequence, the controller parameters are grouped by function in this section. With the exception of changing the units of measure (display parameter; see 4.4) first if Fahrenheit is not appropriate, the configuration parameter functional groups should be configured in the sequence in which they are presented in this section:

1st) communication parameters (if these were not already configured using the controller front panel); see 4.2

2nd) *unit of measure for temperature inputs that do not use Fahrenheit (the default),* then input parameters; see 4.3

3rd) display parameters (some of these parameters apply to the , even though it has no display); see 4.4

4th) output parameters; see 4.5

5th) control parameters; see 4.6

6th) alarm parameters; see 4.7

7th) Autotune damping parameter; see 4.8

8th) ramp/soak parameters; see 4.9

9th) parameters for options other than communications; see 4.10

10th) supervisory parameters; see 4.11

The register addresses for the special calibration parameters are in 4.12.

In addition to configuration and calibration parameters, every, CN8200, CN8240, and CN8260 controller can store setpoint values in RAM and on the EEPROM, as well as various status values. The setpoint can be written using the MODBUS 06 ($06) or 16 ($10) function. The process value and status values can be read using the MODBUS 03 ($03) function. The setpoint register addresses follow the configuration parameters in this section; see 4.14. The controller status register addresses are in 4.15.

For your convenience when interpreting messages received from the controllers, all the controller configuration and calibration parameters, as well as all other accessible values stored by the controllers in their databases, are listed in Section 5 arranged by MODBUS register address.

5.1.3 Information Provided in Each Subsection

5.1.3.1 Subsection Introduction

Each subsection provides a brief introduction to the purpose of the controller values, statuses, or configuration parameters in its functional group. More information about the purpose of individual configuration parameters is in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual.*

5.1.3.2 Register Table

Each subsection contains a table that lists all the parameters in the subsection's functional group. The following information is provided for each parameter:

Parameter (or **Value**) **–** Configuration parameter or other value name

Type – Type of value used for this parameter by the controller. There are three categories represented by codes in the Type column:

- **FV** = value controller uses can include fractional value, stored in the controller database with the decimal point. If read using the 0000 to 0999 region, the fractional portion of this value will always be rounded off (250.4 returned as 250 and 250.5 returned as 251) regardless of input used. If 250.4 is read using the 8000 to 9999 region, the value returned will be the 32-bit IEEE representation of 250.4. See Section 3 for more information and more examples.

- **FV∗** = value controller uses can include fractional value, and how the value is stored in the controller's database is affected by input type.

  - If the input is an RTD or thermocouple, then the value is stored in the controller database with the decimal point. Therefore, if the value is read from the 0000 to 0999 region as an integer, the fractional value will be rounded. For example, 250.4 will be returned as 250, and 250.5 will returned as 251. If 250.4 is read using the 10X region, 2504 will be returned. If 250.4 is read using the 8000 to 9999 region, the value returned will be the 32-bit IEEE representation of 250.4.

  - If a linear input is used, then the value stored in the controller database does not include the decimal point, but the digits in the fractional value are retained. Therefore, when the value is read, it will be 1, 10, 100, or 1000 times the value used by the controller (depending on whether the controller is configured to use zero, one, two, or three decimal places). For example, 30.5 will be returned as 305 using the 0000 to 1999 region or 8000 to 9999 region; 305.0 will be returned as 3050, etc. See Section 3 for more information and more examples.[18]

- **I** = integer: This notation indicates that the controller uses (and stores) only an integer as the value for this parameter. Reading from the 4000 to 4999 region returns the same integer value that the controller uses for this configuration parameter. Frequently, this integer represents something else. For example, consider the value stored for the input type. A value of 0 for the input type parameter represents a B thermocouple, a value of 1 represents a C thermocouple, etc.

[18] The 10X region is not recommended for transmission of **FV∗**parameters when a linear input is used.

Register Number – Relative register assigned to this parameter or value, and the absolute address assigned; all addresses are offset from 40000. In the case of parameters that are stored as integers in the controller (Type `I`), the register (relative and absolute) shown in the table is the only register that can be addressed for that parameter. In the case of a value that can include a fractional value (Types **FV** and **FV∗**), the relative and absolute register address shown is for the base range (0000 to 0999). However, this range is mirrored in two additional ranges so that you can more accurately retrieve fractional values. See 4.1.4 for guidelines for calculating the address of registers in the 10X mirror of the 0000 to 0999 range and the 32-bit IEEE mirror of the 0000 to 0999 range.[19]

R/W – Indicates whether the value stored in the controller can be read (**R**) and written (**W**) using the MODBUS protocol.

Supported By – Types of controllers that use this parameter; **all** indicates that the parameter or value applies to the, CN8200, CN8240, and CN8260 controllers, all the controllers that support the MODBUS protocol.

Valid Data Field Value – In the case of parameters that take a numerical input, the range of valid values is indicated. (See 4.1.3.3 for special information about the ranges that apply if the controller uses a linear input.)

In the case of parameters for which a specific value has a particular meaning to the controller, all the valid values are listed.

The default value for each configuration parameter is also shown in this column. The value shown is the value actually used by the controller.

- In the case of integers, reading this value will always return the value listed.
- In the case of configuration parameters that can use fractional values, the value shown in the table is the value read from the base region (0000 to 0999) when the Decimal Position is at the default, 0 (zero). With the Decimal Position set at 0, the value read from the base region is the same as the value used by the controller, regardless of whether the parameter is marked **FV** or **FV∗**. Default values read from the 10X region may be affected by integer overflow, that is, ten times the default value may be outside the range of values that can be transmitted in a single register –32768 to 32767.

Some values (such as statuses, setpoints, process value) have no default, so none is listed in the table.

5.1.3.3 Linear Inputs Affect Range of Valid Values

5.1.3.3.1 If Range Is "Sensor Low to Sensor High", Input Scaling Limits Apply

If the Valid Data Field Value column indicates that the range for a parameter is "sensor low to sensor high", that range applies to a controller using a thermocouple or RTD input only. If a linear input is used, the range of valid values for the parameter is bound by the input scaling limit values transmitted using registers 26 and 27.

[19] The table in 6.3 lists the relative address of each fractional value type parameter in all three regions.

5.1.3.3.2 If Decimal Position Is Non-Zero, Range Is Reduced

If the controller uses a linear input and the Decimal Position (transmitted in register 4069) is not set to 0 (zero), the number of decimal places specified is always imposed on the parameter's value. This has the effect of reducing the range of valid values for every **FV** and **FV∗** parameter when a linear input is used.

For example, suppose the range of valid values for a parameter is –1999 to 9999.

- If a linear input is used and the Decimal Position is 1, then the range for the parameter is reduced to –199.9 to 999.9.
- If a linear input is used and the Decimal Position is 2, then the range is –19.99 to 99.99.
- If a linear input is used and the Decimal Position is 3 (the maximum for linear inputs), then the range is –1.999 to 9.999.

In contrast, when a thermocouple or RTD input is used, the Decimal Position specified is a maximum. This means that the controller applies the specified number of decimal places only if doing so does not limit the value of the parameter. For example, suppose the range of valid values for a parameter is –1999 to 9999.

- If an RTD or thermocouple input is used and the Decimal Position is 1, then the range for the parameter is still –1999 to 9999. If you configure a value of 8000, no decimal places will be stored and displayed, despite the Decimal Position setting of 1.
- If an RTD or thermocouple input is used and the Decimal Position is 2 (the maximum for temperature inputs), then the range for the parameter is still –1999 to 9999. If you configure a value of 8000, no decimal places will be stored and displayed, despite the Decimal Position setting of 2.

5.1.4 Calculating Register Addresses for Parameters That Use Fractional Values

5.1.4.1 Principles

If the Type column in one of the tables in this section contains **FV** or **FV∗** opposite the name of a parameter or other value, that means that the controller can use a fractional value for this parameter. The table provides the relative address for the base 0000 to 0999 range for each parameter (or other value) for which the controller can use a value that includes decimal values. The table also lists the absolute address for these values. (All addresses are offset from 40000.)

However, a parameter for which the controller can use a fractional value can be read and/or written from any of three regions of the register map:

- as an integer (values that include fractional values are rounded to the nearest integer) in the base range 0000 to 0999 for controller fractional values, or
- as an integer in the 1000 to 1999 region, which is the 10X mirror of the base range, or
- as a true 32-bit IEEE floating point value in 8000 to 9999, which is the 32-bit IEEE mirror of the base range.

Remember, however, that the value read is not necessarily the value used by the controller. Review Section 3 if this concept is not clear.

To calculate a register address in the 10X mirror region: add 1000 to the address in the table in this section (or refer to the table in 5.3, which lists all three relative addresses for each parameter).

To calculate a register address in the 32-bit IEEE mirror region: multiply the address in the table in this section by two, then add 8000 to the result. (Alternatively, refer to the table in 5.3, which lists all three relative addresses for each parameter.)

5.1.4.2 Example

The process value has been assigned relative address 0 in the Omega implementation of MODBUS. Therefore, to read the process value, you have a choice. You can send the controller a request to read the PV from any of the following registers:

- as an integer from a single register in the base range for fractional values at the relative address shown in the table in this section, which is 0 (zero), or
- as an integer from a single register in the 10X mirror of the base range at the address calculated by adding 1000 to the relative address shown in the table in this section: 0 + 1000 = 1000, or
- as a floating point value in a single read from two registers in the 32-bit IEEE mirror of the base range; the first of the two registers has an address that is two times the relative address shown in the table in this section, plus 8000:
  (0 x 2) + 8000 = 8000. (To get the complete 32-bit value the single request must read both 8000 and 8001; by definition a 32-bit IEEE value consists of two 16-bit register values.)

Because the process value is a **FV∗** type, the value returned will be affected by the type of input to the controller (temperature or linear) as described in Section 3.

5.2 Communication Parameters

5.2.1 Overview

As described in 1.3.4, you can use MODBUS functions to read and write the communication parameter values stored in the controllers' databases.

These parameters are accessible in the **serL** (serial) menu on the CN8200, CN8240, and CN8260 display.

5.2.2 Communication Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table.

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Communication Protocol	I	**4080**	44081	R	all	1 = Omega Plus 2 = SPI 3 = Arburg 4 = MODBUS
Controller ID	I	**4081**	44082	R/W	all	1 to 247 *default = 1*
Baud Rate	I	**4082**	44083	R/W	all	2 = 300 baud 3 = 600 baud 4 = 1200 baud 5 = 2400 baud 6 = 4800 baud 7 = 9600 baud *default = 7 (see Note 1 below)*
Parity	I	**4083**	44084	R/W	all	0 = none 1 = even 2 = odd *default = 0*
IEEE Register Ordering	I	**4084**	44085	R/W	all	0 = non-standard sequence (most significant register transmitted first) 1 = standard MODBUS sequence (least significant register transmitted first) *default = 1*

Note 1: Different default baud rates may apply to protocols other than MODBUS. See the *(, CN8200, CN8240, CN8260) Configuration and Operation Manual* for details.

5.3 Input Parameters

5.3.1 Overview

As described in 4.1.1, once communication parameters have been set up (and the units of measure for temperature inputs have been changed, if necessary, from the default Fahrenheit), the next step in configuring a controller is specifying the type of input the controller will receive. You can use MODBUS functions to read and write the input parameter values stored in the controllers' databases.

The scaling limits apply only to linear inputs. All the other input parameters apply to all input types. These parameters are accessible in the **`Inp`** (input) menu on the CN8200, CN8240, and CN8260 display.

5.3.2 Input Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV∗** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Input Type	I	**4049**	44050	R/W	all	0 = B thermocouple 1 = C thermocouple 2 = E thermocouple 3 = J thermocouple 4 = K thermocouple 5 = N thermocouple 6 = NNM thermocouple 7 = R thermocouple 8 = S thermocouple 9 = T thermocouple 10 = Platinel II thermocouple 11 = RTD 12 = RTD Decimal 13 = 0 to 20 mA linear 14 = 4 to 20 mA linear 15 = 0 to 10 mV linear 16 = 0 to 50 mV linear 17 = 0 to 100 mV linear 18 = 10 to 50 mV linear 19 = 0 to 1 V linear 20 = 0 to 5 V linear 21 = 0 to 10 V linear 22 = 1 to 5 V linear *default = 3 (J thermocouple)* See Note 1 below.
Input Bias	FV∗	**25**	40026	R/W	all	*thermocouple or RTD input types:* –1000 to 1000 °F –556 to 556 °C and °K *default = 0* *linear input types:* –1000 to 1000 *default = 0*

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Linear Input Scaling Low Limit	FV∗	**26**	40027	R/W	all	–1999 to 9999 *default = –1999*
Linear Input Scaling High Limit	FV∗	**27**	40028	R/W	all	–1999 to 9999 *default = 9999*
Setpoint Low Limit	FV∗	**28**	40029	R/W	all	sensor low limit to sensor high limit *default = –328*
Setpoint High Limit	FV∗	**29**	40030	R/W	all	sensor low limit to sensor high limit *default = 1400*
Input Filter	FV	**30**	40031	R/W	all	1 to 100; each unit (1) represents 0.1 second of filter time *default = 5*

Note 1: The input type is configured at the factory. The type configured depends on the input calibration type ordered. To determine the input calibration ordered for the controller in hand, check the model number on its label. The significance of each character in the model number is in the installation manual supplied with the controller. The table below shows the correlation between the input character and the factory input setting. Not every input calibration character applies to every model.

Input Calibration Model Number Character	Factory Configuration for Input Type (4049)
A	3 (J thermocouple; calibrated for all input types)
B	3 (J thermocouple); calibrated for thermocouple and RTD
C	13 (0 to 20 mA linear)
M	16 (0 to 50 mV linear)
R	11 (RTD)
S	12 (RTD Decimal)
T	3 (J thermocouple)
V	20 (0 to 5 V linear)

5.4 Display Parameters

5.4.1 Overview

You can use MODBUS functions to read and write the input parameter values stored in the controllers' databases. Most are stored as integers; the display filter is stored as a value that can be fractional.

These parameters are accessible in the **dSPL** (display) menu on the CN8200, CN8240, and CN8260 display.

5.4.2 Display Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV*** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Decimal Position for Thermocouple or RTD Input (see Notes 1 and 5 below)	I	**4068**	44069	R/W	all (see note 2 below	0, 1 *default = 0*
Maximum Decimal Position for Linear Input (see Notes 1 and 4 below)	I	**4069**	44070	R/W	all (see Note 2 below	0 to 3 *default = 0*
Display Filter	FV	**35**	40036	R/W	all except	1 to 100 (each unit represents 0.1 second of filter time) *default = 1*
Display Unit of Measure	I	**4070**	**44071**	R/W	all (see Note 3)	1 = Fahrenheit 2 = Celsius 3 = Kelvin *default = 1*
Setpoint Display Blanking	I	**4071**	**44072**	R/W	all except	9 = off 10 to 9999 seconds *default = 9 (off)*

Note 1: These two registers (4068 and 4069) correspond to a single parameter **dEC.P** on the controller display.

Note 2: Even though the has no display, you must configure the appropriate decimal position parameter (or accept the default) because the controller uses this parameter value to determine how many decimal places to store for the PV and SV values.

Note 3: Even though the has no display, if you use a temperature input, you must configure the unit of measure. The controller uses the unit of measure internally and for external communication.

Note 4: The Decimal Position is always imposed on all **FV** and **FV*** parameter values if the controller uses a linear input. This has the effect of reducing the range of valid values for the **FV** and **FV*** parameters if a linear input is used. See 4.1.3.3.2 for more information.

Note 5: The Decimal Position is a maximum that is not always used if the controller uses an RTD or thermocouple input. The Decimal Position does not reduce the range of valid values when a temperature input is used. See 4.1.3.3.2 for more information.

5.5 Output Parameters

5.5.1 Overview

You can use MODBUS functions to read and write the parameters used to specify the type of output (function of the output, not the hardware type) and the parameters associated with each type of output. Not all parameters apply to every type of output.

Also, not all parameters and choices apply to every controller model. Standard outputs can be used for alarm annunciation only on CN8200 and controllers. Use of standard outputs for alarm annunciation on these compact models is allowed because their cases cannot accommodate a supplementary alarm output card if a serial communications card is installed. The larger CN8240 and CN8260 controllers can contain both an optional alarm output card and a serial communications card. Therefore, use of standard outputs for alarm annunciation in the CN8240 and CN8260 is not necessary. (See 4.7 for alarm parameters that control alarm annunciation using the front panel LEDs on CN8200, CN8240, and CN8260 models, and that also control alarm annunciation using the optional output card in CN8240 and CN8260 units.[20])

See the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual* for details about the purpose of each parameter, and the model and control strategy to which they apply.

The controllers store some output parameter values as integers, others can use fractional values.

These parameters are accessible in the **`OutP`** (output) menu on the CN8200, CN8240, and CN8260 display.

[20] The alarm output card can be ordered for CN8200 controllers that do not contain a serial communications card.

5.5.2 Output Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV∗** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Output 1 Type	I	**4050**	44051	R/W	all	*CN8200/:* 1 = inactive (disabled) 2 = PID 3 = *(invalid value)* 4 = on/off 5 = alarm *default = 2 (PID)* *CN8240/CN8260:* 1 = inactive (disabled) 2 = PID 3 = (*invalid value)* 4 = on/off 5 = *(invalid value)* *default = 2 (PID)*
Output 1 Action	I	**4051**	44052	R/W	all	1 = direct 2 = reverse *default = 2 (reverse)*
Output 1 Cycle Time	I	**4056**	44057	R/W	all	0 = 0.2 seconds 1 to 120 seconds *default = 5* (See Note 3 below.)
Output 1 Low Limit	I	**4057**	44058	R/W	all	0 to 100 percent *default = 0*
Output 1 High Limit	I	**4058**	44059	R/W	all	0 to 100 percent *default = 100*
Output 1 Alarm Action	I	**4052**	44053	R/W	/CN8200	1 = off 2 = normal 3 = latched *default = 1 (off)*
Output 1 Alarm Operation	I	**4053**	44054	R/W	/CN8200	1 = process high 2 = process low 3 = deviation high 4 = deviation low 5 = normal band 6 = inverse band *default = 2 (process low)*
Output 1 Alarm Delay	I	**4054**	44055	R/W	/CN8200	0 to 9999 seconds *default = 0*
Output 1 Alarm Inhibit	I	**4055**	44056	R/W	/CN8200	0 to 9999 seconds *default = 0*

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Output 1 Process Alarm Setpoint (see Note 1 below)	FV∗	**31**	40032	R/W	/CN8200	sensor low limit to sensor high limit *default = 77*
Output 1 Deviation, Normal Band, or Inverse Band Alarm Setpoint (see Note 1 below)	FV∗	**32**	40033	R/W	/CN8200	1 to span of sensor *default = 1728*
Output 2 Type	I	**4059**	44060	R/W	all	*CN8200/:* 1 = Inactive/Disabled 2 = PID 3 = *(invalid value)* 4 = On/Off 5 = Alarm *default = 2 (PID)* *CN8240/CN8260:* 1 = Inactive/Disabled 2 = PID 3 = (*invalid value)* 4 = On/Off 5 = *(invalid value)* *default = 2 (PID)*
Output 2 Action	I	**4060**	44061	R/W	all	1 = direct 2 = reverse *default = 1*
Output 2 Cycle Time	I	**4065**	44066	R/W	all	0 = 0.2 seconds 1 to 120 seconds *default = 5* (See Note 3 below.)
Output 2 Low Limit	I	**4066**	44067	R/W	all	0 to 100 percent *default = 0*
Output 2 High Limit	I	**4067**	44068	R/W	all	0 to 100 percent *default = 100*
Output 2 Alarm Action	I	**4061**	44062	R/W	/CN8200	1 = off 2 = normal 3 = latched *default = 1 (off)*
Output 2 Alarm Operation	I	**4062**	44063	R/W	/CN8200	1 = process high 2 = process low 3 = deviation high 4 = deviation low 5 = normal band 6 = inverse band *default = 1 (process high)*
Output 2 Alarm Delay	I	**4063**	44064	R/W	/CN8200	0 to 9999 seconds *default = 0*

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Output 2 Alarm Inhibit	I	**4064**	44065	R/W	/CN8200	0 to 9999 seconds *default = 0*
Output 2 Process Alarm Setpoint (see Note 2 below)	FV∗	**33**	40034	R/W	/CN8200	sensor low limit to sensor high limit *default = 77*
Output 2 Deviation, Normal Band, or Inverse Band Alarm Setpoint (see Note 2 below)	FV∗	**34**	40035	R/W	/CN8200	1 to span of sensor *default = 1728*

Note 1: These two registers (31 and 32) correspond to a single parameter **O1SP** on the controller display.

Note 2: These two registers (33 and 34) correspond to a single parameter **O2SP** on the controller display.

Note 3: The cycle time is configured at the factory. The cycle time configured depends on the output hardware in the controller. To determine the output hardware in the controller in hand, check the model number on its label. The significance of each character in the model number is in the installation manual supplied with the controller. The table below shows the correlation between the output character and the factory cycle time setting. Not every output character applies to every model.

Output Model Number Character	Factory Configuration for Cycle Time (4056 for Output 1; 4065 for Output 2)
0	*not used; no output*
B	5 seconds
C	5 seconds
D	0 (0.2 seconds)
E	0 (0.2 seconds)
F	0 (0.2 seconds)
G	0 (0.2 seconds)
P	0 (0.2 seconds)
S	0 (0.2 seconds)
T	5 seconds
U	5 seconds
V	0 (0.2 seconds)
X	0 (0.2 seconds)
Y	5 seconds

5.6 Control Parameters

5.6.1 Overview

You can use MODBUS functions to read and write the parameter values that tune the control algorithm used when Proportional-Integral-Derivative (PID) is the output type, or the parameter values that prevent “chattering” of on/off outputs.

However, the values of the PID tuning parameters are automatically adjusted during the Autotune procedure. Unless you want to tune the controller manually, do not alter the PID tuning parameters Proportional Band 1, Proportional Band 2, Derivative Action, and Manual Reset or Integral Action Auto Reset (whichever will be used). Consult the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual* for more information about tuning PID control, including the use of Autotune.

The controllers use only integers for some control parameters, others can include fractional values.

These parameters are accessible in the **`CtrL`** (control) menu on the CN8200, CN8240, and CN8260 display.

5.6.2 Control Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV∗** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
PID Proportional Band for Output 1	FV∗	**9**	40010	R/W	all	*thermocouple and RTD input types:* 0.6 to span of sensor *default = 100* *linear input types:* 1 to sensor span (max at 9999) *default = 100*
PID Proportional Band for Output 2	FV∗	**10**	40011	R/W	all	
PID Derivative (Rate) Action (See Note 1 below)	FV	**11**	40012	R/W	all	0.0 to 0.9, 1 to 2400 *default = 0.0*
PID Integral Action: Auto Reset (See Note 1 below)	FV	**12**	40013	R/W	all	0.0 to 0.9, 1 to 9600 *default = 0.0*
PID Integral Action: Manual Reset	FV	**13**	40014	R/W	all	–100 to 100 *default = 0*
On/Off Control Deadband for Output 1	FV∗	**7**	40008	R/W	all	*thermocouple and RTD input types:* negative sensor span to positive sensor span *default = 1* *linear input types:* –1999 to 9999 *default = 1*

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
On/Off Control Hysteresis for Output 1	FV∗	**8**	40009	R/W	all	*thermocouple and RTD input types:* 1 to span of sensor *default = 1* *linear input types:* 1 to 9999 *default = 1*
On/Off Control Deadband for Output 2	FV∗	**14**	40015	R/W	all	*thermocouple and RTD input types:* negative sensor span to positive sensor span *default = 1* *linear input types:* –1999 to 9999 *default = 1*
On/Off Control Hysteresis for Output 2	FV∗	**15**	40016	R/W	all	*thermocouple and RTD input types:* 1 to span of sensor *default = 1* *linear input types:* 1 to 9999 *default = 1*

Note 1: These parameters are special cases. Even if the Linear Decimal Position Parameter (register 4069) is set to 2 or 3, the maximum number of decimal places applied to the PID Derivative (Rate) Action and PID Integral Action (Auto Reset) is 1 (the tenths position).

5.7 Alarm Parameters

5.7.1 Overview

You can use MODBUS functions to read and write the parameters used to control alarm annunciation using the front panel LEDs on the CN8200, CN8240, and CN8260 models, and that also control alarm annunciation using the optional alarm output card in CN8240 and CN8260 units. (See 4.5 in this manual and the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual* for more information about the use of various parameters to control alarm annunciation.)

The controllers store some alarm parameter values as integers, others can include fractional values.

These parameters are accessible in the **ALr** (alarm) menu on the CN8200, CN8240, and CN8260.

5.7.2 Alarm Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV*** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Alarm 1 Action	I	**4072**	44073	R/W	all except	1 = off 2 = normal 3 = latched 4 = event *default = 1 (off)*
Alarm 1 Operation	I	**4073**	44074	R/W	all except	1 = process high 2 = process low 3 = deviation high 4 = deviation low 5 = normal band 6 = inverse band *default = 1 (process high)*
Alarm 1 Delay	I	**4074**	44075	R/W	all except	0 to 9999 seconds *default = 0*
Alarm 1 Inhibit	I	**4075**	44076	R/W	all except	0 to 9999 seconds *default = 0*
Alarm 1 Process Setpoint (see Note 1 below)	FV*	**36**	40037	R/W	all except	sensor low limit to sensor high limit *default = 77*
Alarm 1 Deviation, Normal Band, or Inverse Band Setpoint (see Note 1 below)	FV*	**37**	40038	R/W	all except	1 to span of sensor *default = 1728*
Alarm 2 Action	I	**4076**	44077	R/W	all except	1 = off 2 = normal 3 = latched 4 = event

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
						default = 1 (off)
Alarm 2 Operation	I	**4077**	44078	R/W	all except	1 = process high 2 = process low 3 = deviation high 4 = deviation low 5 = normal band 6 = inverse band *default = 2 (process low)*
Alarm 2 Delay	I	**4078**	44079	R/W	all except	0 to 9999 seconds *default = 0*
Alarm 2 Inhibit	I	**4079**	44080	R/W	all except	0 to 9999 seconds *default = 0*
Alarm 2 Process Setpoint (see Note 2 below)	FV*	**38**	40039	R/W	all except	sensor low limit to sensor high limit *default = 77*
Alarm 2 Deviation, Normal Band, or Inverse Band Setpoint (see Note 2 below)	FV*	**39**	40040	R/W	all except	1 to span of sensor *default = 1728*

Note 1: These two registers (36 and 37) correspond to a single parameter **A1.SP** on the controller display.

Note 2: These two registers (38 and 39) correspond to a single parameter **A2.SP** on the controller display.

5.8 Autotune Damping Parameter

5.8.1 Overview

You can use MODBUS functions to read and write the parameter used to control how aggressively the controller performs its Autotuning operation. This value is stored as an integer in the controller.

This parameter is accessible in the **`tunE`** (Autotune damping) menu on the CN8200, CN8240, and CN8260 display.

5.8.2 Autotune Damping Parameter Register

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table.

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Autotune Damping	I	**4011**	44012	R/W	all	1 = low 2 = normal 3 = high *default = 2 (normal)*

5.9 Ramp/Soak Parameters

5.9.1 Overview

The CN8200, CN8240, and CN8260 controllers can be configured to execute a single-step controlled gradual ramp up to desired setpoint when the controller is powered up, or to execute upon demand a recipe consisting of up to eight ramp and soak segments.

Each segment consists of a ramp time, a soak level, and a soak time. As the controller executes a segment, the controller gradually (over the period of the ramp time) changes the currently used setpoint up or down until the soak level is reached. During the duration of the soak time, the controller maintains the setpoint at the specified soak level. When the soak time has elapsed, the controller executes the next segment of the recipe. If at least one front panel alarm was configured for "events", then you can activate and deactivate the alarm based on the recipe reaching particular events in its execution, such as the start of a numbered soak period. (More details concerning the parameters affecting recipe execution are in the *(CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual.*)

You can use MODBUS functions to read and write to the recipe parameters in the table below. You can also use the MODBUS read function to see the currently used recipe setpoint and segment using registers listed in 4.15.2. Subsection 4.15.2 also lists the register used with the write function to change the controller mode to start running (or to resume running) a recipe and to hold (pause) recipe execution.

The controllers use only integers for some recipe parameters, others can include fractional values.

These parameters are accessible in the **`r-S`** (ramp/soak) menu on the CN8200, CN8240, and CN8260 display.

5.9.2 Ramp/Soak Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV∗** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Recipe Option	I	**4012**	44013	R/W	all	0 = disabled 1 = single-step ramp 2 = multi-step ramp *default = 0 (disabled)*
Single-Step Ramp Time	I	**4013**	44014	R/W	all	1 to 9999 *default = 1*
Holdback	FV∗	**24**	40025	R/W	all	*thermocouple and RTD input types:* 0 = off 0.1 to 100 °F 0.1 to 55.6 °C or °K *default = 10* *linear input types:* 0 = off 1 to 100 *default = 10*
Termination State	I	**4047**	44048	R/W	all	0 = last setpoint 1 = default setpoint 2 = recipe to standby mode *default = 2 (recipe to standby)*
Recycle Number	I	**4046**	44047	R/W	all	0 to 99 100 = continuous *default = 0*
Resume from Power Failure	I	**4048**	44049	R/W	all	1 = resume off 2 = resume on *default = 1*
Ramp Times 1 to 8	I	**4014 to 4021**	44015 to 44022	R/W	all	0 to 9999 minutes *default = 0 (disabled)*
Ramp Events 1 to 8	I	**4022 to 4029**	44023 to 44030	R/W	all except	0 = disabled 1 = event 1 (alarm 1) on 2 = event 1 (alarm 1) off 3 = event 2 (alarm 2) on 4 = event 2 (alarm 2) off *default = 0 (disabled)*
Soak Levels 1 to 8	FV∗	**16 to 23**	40017 to 40024	R/W	all	setpoint low limit to setpoint high limit (see Note 1 below) *default = 77*

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Soak Times 1 to 8	I	**4030 to 4037**	44031 to 44038	R/W	all	0 to 9999 minutes *default = 0 (disabled)*
Soak Events 1 through 8	I	**4038 to 4045**	44039 to 44046	R/W	all except	0 = disabled 1 = event 1 (alarm 1) on 2 = event 1 (alarm 1) off 3 = event 2 (alarm 2) on 4 = event 2 (alarm 2) off *default = 0 (disabled)*

Note 1: These limits are specified using input parameters; see 4.3.

5.10 Parameters for Options

5.10.1 Overview

You can use MODBUS functions to read the parameter that stores the type of option card installed in an CN8240 and CN8260 controller (in addition to the special serial communications card that makes it possible for the controllers to communicate with the MODBUS master).

You can read and write the parameters used to specify how the options are used. No controller supports all of these options simultaneously. This section includes all the option parameters, except the communication parameters, which are in 4.2.

5.10.2 Option Parameters

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV*** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
on `Optn` (option) menu on CN8200, CN8240, CN8260						
Installed Option Card (See Note 1 below)	I	**4088**	44089	R/W	all except	0 = No Option Card 1 = Communication Option 2 = Communication with Contact/Digital Input Option 3 = Remote Analog Setpoint Option 4 = Auxiliary Analog Output Option 5 = Alarm Output Option 6 = Alarm with Contact/Digital Input Option *default for CN8240, CN8260 = 0* *default for CN8200 = 1*
on `Aout` (auxiliary output) menu on CN8200, CN8240, CN8260 – used only when the auxiliary output option card is installed						
Auxiliary Output Variable	I	**4089**	44090	R/W	all except	1 = process value 2 = setpoint *default = 1 (process value)*
Auxiliary Output Scale Low	FV	**52**	40053	R/W	all except	sensor low limit to sensor high limit *default = –328*
Auxiliary Output Scale High	FV	**53**	40054	R/W	all except	sensor low limit to sensor high limit *default = 1400*
on `C-dl`(contact/digital input) menu on CN8200, CN8240, CN8260 – used only if an option card that supports the contact/digital input is installed						
Contact/Digital Input Function	I	**4090**	44091	R/W	all except	1 = disabled 2 = select use of second setpoint (see Note 2 below) 3 = toggle in and out of standby mode 4 = toggle recipe execution between run and hold

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
						default = 1 (disabled)
on **rAS** (remote analog setpoint) menu on CN8200, CN8240, CN8260 – used only if the remote analog setpoint option card is installed (see Note 3 below)						
Remote Analog Setpoint Scale Low	FV*	**54**	40055	R/W	all except	–1999 to 9999 *default = –1999*
Remote Analog Setpoint High	FV*	**55**	40056	R/W	all except	–1999 to 9999 *default = 9999*

Note 1: This parameter is configured at the factory. However, if you use the procedure in 7.2 to load all parameter defaults, then you must use the front panel of the controller to configure the **Card** parameter in the **Optn** (option) menu. This prepares the controller to receive the appropriate option parameter values. The only exception is in the case of an CN8240 or CN8260 equipped with only the serial communication option. The CN8240 and CN8260 models are always ready to receive serial communication parameter values.

Note 2: The second setpoint can be read from and written to RAM using the register with relative address 4, and read from and written to the EEPROM and RAM using register 3 (see 4.14.1.3).

Note 3: The setpoint received from the remote device can be read from the register at relative address 5.

5.11 Supervisor Parameters

5.11.1 Overview

You can use MODBUS functions to read and write to the parameters used to specify what output percentages should be used if the controller detects a problem with the process input (failsafe values), and the length of the time period during which the input should change in response to output action if the input is working normally (loop break time).

You can also use the MODBUS read function to see the highest and lowest process value received by the controller since the readings were reset.

These parameters are accessible in the **`SUPr`** (Supervisor) menu on the CN8200, CN8240, and CN8260 display.[21]

5.11.2 Supervisor Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table. Subsection 4.1.3.3 contains information about the effect of a linear input on the range of valid values for **FV** and **FV∗** parameters**.**

Parameter	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Output 1 Failsafe Output Percentage	I	**4085**	44086	R/W	all	0 to 100 percent *default = 0*
Ouput 2 Failsafe Output Percentage	I	**4086**	44087	R/W	all	0 to 100 percent *default = 0*
Loop Break Time	I	**4087**	44088	R/W	all	3 = off 4 to 9600 seconds *default = 3 (off)*
Highest Reading	FV∗	**40**	40041	R/W	all	sensor low limit to sensor high limit *default = –3566* See Note 1 below.
Lowest Reading	FV∗	**41**	40042	R/W	all	sensor low limit to sensor high limit *default = 18030* See Note 2 below.

Note 1: This highest reading can be reset by writing –1999 to register 40.

Note 2: This lowest reading can be reset by writing 9999 to register 41.

[21] The other function that appears on the **`SUPr`** menu, **`Lddp`** (load default parameters), is accomplished using the MODBUS 16 ($10) write function and factory registers as described in Section 7.

5.12 Calibration Function

5.12.1 Overview

You do not have to calibrate every new controller. When a controller was ordered, you specified an input type for which the unit was calibrated at the factory. This is not the specific type written to the input type register at relative address 4049, such as J thermocouple, or 0 to 20 mA linear. In the context of ordering the controller, "type" refers to these choices: RTD, compressed RTD, thermocouple, millivolt linear, volt linear, or current linear input. The controller was calibrated at the factory for the type of input specified. If you use the controller with a different type of input, you must recalibrate as described in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual* unless you ordered the "Calibrate All" input option.[22]

For example, if you specified when you ordered the controller that you planned to use a thermocouple as the sensor, then you can use the input type register to specify any thermocouple type: B, C, E, J, K, N, NNM, R, S, T, or Platinel II. The controller will be calibrated appropriately at the factory. However, if you ordered thermocouple calibration, but decide to use the controller with an RTD sensor, then you should recalibrate before using the controller.

When calibrating the controller, you can use MODBUS commands to write to special registers in the factory region as described in Section 7 of this manual. (Do not use the factory calibration commands before you have read the calibration instructions in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual.)*.

The values calculated by the controller for the zero offset and the span adjustment are stored in the registers listed in the table below and can be read and written using MODBUS. Different registers are available for each type of input. However, the controller calibrates only for the type of input specified using the input type parameter.

The values in the calibration zero offset and span adjustment registers should not be changed using MODBUS write functions. The zero offset and span adjustment should be changed only by the controller's calibration procedure. However, if you have used the MODBUS master to back up your controller's database, including the zero offset and span adjustment, you can safely restore the values using MODBUS write functions.

[22] To determine whether the controller in hand was calibrated at the factory for all input types, check the model number on the label on the controller. The meaning of each character in the model number is in the installation manual supplied with the controller.

5.12.2 Calibration Zero Offset and Span Adjustment Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table.

Value	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Thermocouple Zero Offset	FV	**42**	40043	R/W	all	–25000 to 25000 *default = 2600*
Thermocouple Span Adjustment	FV	**43**	40044	R/W	all	0.001 to 2.0 *default = 1*
RTD Zero Offset	FV	**44**	40045	R/W	all	–25000 to 25000 *default = 9000*
RTD Span Adjustment	FV	**45**	40046	R/W	all	0.001 to 2.0 *default = 1*
Low-Voltage Zero Offset (See Note 1 below.)	FV	**46**	40047	R/W	all	–25000 to 25000 *default = 0*
Low-Voltage Span Adjustment (See Note 1 below.)	FV	**47**	40048	R/W	all	0.001 to 2.0 *default = 1*
High-Voltage Zero Offset (see Note 2 below.)	FV	**48**	40049	R/W	all	–25000 to 25000 *default = 0*
High-Voltage Span Adjustment (see Note 2 below.)	FV	**49**	40050	R/W	all	0.001 to 2.0 *default = 1*
Current (mA) Zero Offset	FV	**50**	40051	R/W	all	–25000 to 25000 *default = 0*
Current (mA) Span Adjustment	FV	**51**	40052	R/W	all	0.001 to 2.0 *default = 1*
RTD Decimal Zero Offset	FV	**57**	40058	R/W	all	–25000 to 25000 *default = 9000*
RTD Decimal Span Cal	FV	**58**	40059	R/W	all	0.001 to 2.0 *default = 1*
2nd High-Voltage Zero Cal (see Note 3 below.)	FV	**59**	40060	R/W	all	–25000 to 25000 *default = 0*
2nd High-Voltage Span Cal (see Note 3 below.)	FV	**60**	40061	R/W	all	0.001 to 2.0 *default = 1*
0 to 100 mV Zero Cal	FV	**61**	40062	R/W	all	–25000 to 25000 *default = 0*

Value	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
0 to 100 mV Span Cal	FV	**62**	40063	R/W	all	0.001 to 2.0 *default = 1* *CN8200, CN8240, CN8260 default = 0*

Note 1: This “low voltage” register stores a calibration value for the following inputs: 0 to 10 mV, 0 to 50 mV, and 10 to 50 mV inputs.

Note 2: This “high voltage” register stores a calibration value for the following inputs: 0 to 1 V and 0 to 5 V.

Note 3: This “2nd high-voltage” register stores a calibration value for the following inputs: 0 to 10 V and 1 to 5 V.

5.13 Security Parameter

5.13.1 Overview

The CN8200, CN8240, and CN8260 controllers can be configured so that access to their databases by means of the front panel is limited. You can use MODBUS functions to read and write the parameter that controls this access.

5.13.2 Security Parameter Register

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table.

Parameter	**Type**	**Register Number**		**R/W**	**Supported By**	**Valid Data Field Value**
		Rel.	**Abs.**			
Access Level	I	**4005**	44006	R/W	all except	1 = lockout 2 = setpoint and output percentage (manual mode) only 3 = setpoint plus mode 4 = user (setpoint, manual output percentage, mode, plus Autotune and control menus) 5 = configuration (all privileges above plus all configuration menus except calibration) 6 = factory (all of above plus calibration) *default = 6 (factory); this is the value that will be written to the register if you use the procedure in 7.2 to load all parameter defaults. However, controllers are shipped from the factory set to 5 (configuration).*

5.14 Process Value and Setpoints

5.14.1 Overview

5.14.1.1 Introduction

You can use MODBUS functions to read the process value and all setpoints stored in a , CN8200, CN8240, or CN8260 controller. You can use MODBUS functions to change the setpoint in the controller (and the second setpoint, if supported by the controller hardware.)

The PV and setpoints can include fractional values.

The value of the PV and the setpoint currently being used ("active setpoint") are always displayed when a CN8200, CN8240, or CN8260 controller is operating in normal mode.

5.14.1.2 Where Setpoints Are Stored in the Controller

When writing a new setpoint to the controller, you can specify that the setpoint will be stored only in RAM, or in both RAM and on the EEPROM. The controller uses the setpoint stored in RAM. If the setpoint is also stored on the EEPROM, the setpoint will be retained, even when power to the controller is turned off. The setpoint stored on the EEPROM will be written to RAM when the controller is powered up.

However, you can wear out the EEPROM by writing to it too many times. Do not write the setpoint to the EEPROM when you are writing a temporary setpoint to the controller, such as when you are ramping to a final setpoint under the direction of the MODBUS master.

5.14.1.3 Purpose of Second Setpoint

The table below refers to a second setpoint. This second setpoint is not supported by all controllers. Only controllers with optional contact/digital input hardware support the second setpoint. If a controller is equipped with the optional contact/digital input, then you can use the integer register at relative address 4090 to specify what happens when an external device changes the state of this optional contact. One of the choices is to use the second setpoint. Like the primary setpoint, the second setpoint can be written only to RAM, or to both RAM and the EEPROM. Do not write temporary setpoints to the EEPROM.

5.14.1.4 Purpose of Remote Analog Setpoint

The table below also refers to a remote analog setpoint. Use of a remote analog setpoint (RAS) is supported by only controllers that contain the optional RAS card. If use of the remote analog setpoint is enabled (by the contact on the RAS card being closed by an external signal), then the RAS value currently being received by the controller can be read from the register with relative address 5.

5.14.1.5 Purpose of Recipe Setpoint

The , CN8200, CN8240, and CN8260 controllers can each be configured to execute a single ramp to setpoint, or a multi-step ramp and soak recipe. As execution of the ramp or recipe progresses, the setpoint is changed by the controller to achieve the process values specified in the recipe. The value of the setpoint currently being used while the recipe is running can be read from the register at relative address 6. (See 4.9 for information about the registers used to configure the ramp/soak recipes. The recipe segment currently being executed and the recipe state can be read; see 4.15.)

5.14.1.6 Purpose of Active Setpoint

The active setpoint is the setpoint value currently being used for control. This is also the setpoint currently being displayed. This setpoint can come from several sources. The logic flow that determines which setpoint value is displayed is shown below.

```
If a single setpoint ramp or multi-step ramp/soak recipe is active,
    then the active setpoint equals the recipe setpoint.
Else if using the remote analog setpoint is enabled,
    then the active setpoint equals the remote analog setpoint.
Else if using the second setpoint is enabled,
    then the active setpoint equals the second setpoint.
Else active setpoint equals setpoint.
```

5.14.2 PV and SV Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table.

Value	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Process Value	FV∗	**0**	40001	R	all	*range is determined by sensor span* -32768 = Low Error +32767 = High Error
Setpoint (EEPROM and RAM)	FV∗	**1**	40002	R/W	all	setpoint low limit to setpoint high limit (see Note 1 below) *default = 77*
Setpoint (RAM)	FV∗	**2**	40003	R/W	all	setpoint low limit to setpoint high limit (see Note 1 below) *default = 77*
Second Setpoint (EEPROM and RAM)	FV∗	**3**	40004	R/W	all except	setpoint low limit to setpoint high limit (see Note 1 below) *default = 77*
Second Setpoint (RAM)	FV∗	**4**	40005	R/W	all except	setpoint low limit to setpoint high limit (see Note 1 below) *default = 77*
Remote Analog Setpoint	FV∗	**5**	40006	R	all except	remote setpoint limits (see Note 2 below)
Recipe Setpoint	FV∗	**6**	40007	R	all	setpoint low limit to setpoint high limit (see Note 1 below)
Active Setpoint	FV∗	**56**	40057	R/W	all	setpoint low limit to setpoint high limit (see Note 1 below) or –1999 to 9999 when remote analog setpoint is being used

Note 1: Setpoint low limit and setpoint high limit are specified using input parameters; see 4.3.

Note 2: Remote analog setpoint low and high limits are specified using option parameters; see 4.10.

5.15 Controller Information and Status Values

5.15.1 Overview

5.15.1.1 Introduction

You can use the MODBUS read function to read information about the controller (such as firmware version number) and status values (such as the state of the recipe being executed or the recipe segment that is active). You can also use MODBUS write functions to change the controller's operating mode (such as switching from normal to standby, or from recipe run to recipe hold).

The PV and setpoints are stored in the controller as fractional values.

The purpose and function of most of the information and status register in the table below are self-explanatory. However, the following paragraphs explain concepts that may be new to you.

5.15.1.2 Purpose of Contact/Digital Input State

The table below refers to a contact/digital input state. A contact/digital input is not supported by all controllers. Some controllers can accept a discrete input (in addition to the standard analog inputs). This discrete input can be used to trigger use of the second setpoint stored in the controller, to switch the controller to standby mode, or to toggle recipe run and hold. (See 4.10 for the parameter used to specify the function of the discrete input.)

5.15.1.3 Purpose of Resume Exhaustion Flag

A controller can be configured to resume recipe execution after a power failure (see 4.9.2). To be able to resume execution of a recipe after a power failure, the controller must store information about the current state of the recipe on the EEPROM, assuming sufficient space is available on the EEPROM. A MODBUS register can be used to determine whether the controller still has enough storage space on the EEPROM for information about the currently executing recipe.

The integer register at relative address 4094 can be used to read the status of the "resume exhaustion flag". If this register contains a value of 1 (true), then the controller cannot resume recipe execution after a power failure.[23] If the value in register 4094 is 0 (false), then the controller still has storage space available on the EEPROM for recipe status information.

5.15.1.4 Interpreting the Status Byte and the LED Status Indicator Byte

The integer register at relative address 4003 can be used to read a status byte that provides information such as the fact that the controller has detected a possible loop break or that an alarm state has been detected. Similarly, the integer register at relative address 4095 can be used to read a status byte that indicates which LEDs on the controller front panel are lit.

[23] Under normal circumstances, it is not likely that you will consume all the storage space on the EEPROM. However, it is possible if your a site experiences frequent power failures during execution of recipes.

Each status byte consists of eight bits. Each bit has been assigned a meaning. If a bit is set (value is 1), then the condition associated with this bit is true. If a bit is not set (0), then the condition is FALSE.

For example, the following meanings have been assigned to the bits in the LED status byte:

0	unused, always 0
1	unused, always 0
2	F2 on (excluding serial communication activities; always zero for)
3	F1 on (excluding serial communication activities; always zero for)
4	A2 on (CN8200, CN8240, CN8260 only; always zero for)
5	A1 on (CN8200, CN8240, CN8260 only; always zero for)
6	O2 active
7	O1 active

Suppose that LEDs F1 and F2 are lit, and all the other LEDs on the front of a CN8200 controller are not lit. In this case, the bits in positions 2 and 3 are set (value = 1) and all the others are not set (value = 0). Envision the bits as a binary number: 00001100 = 12.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	**0**	**0**	**0**	**1**	**1**	**0**	**0**

If you read the LED status register and the controller returns the value 12, then you know which two bits are set. The set bits must be the bits that represent 4 and 8 in the binary system.

If the controller returns a value of 28, you know that bits 2, 3, and 4 are set, the bits that represent 16, 8, and 4 in the binary system: 00011100 = 28, so F1 and F2 are both lit, as well as A2.

5.15.2 Controller Information and Status Parameter Registers

See 4.1.3.2 for a description of the types of information and the meanings of the abbreviations in this table.

Value	Type	Register Number		R/W	Supported By	Valid Data Field Value
		Rel.	Abs.			
Controller Type	I	**4000**	44001	R	all	2 = CN8200 3 = CN8240/CN8260 *(paired because these two models support the same hardware options)* 4 =
Software Version	I	**4001**	44002	R	all	*Six-digit integer value representing controller firmware version in the form nn.nn.nn. For example, 013100 means version 01.31.00.*
Communications Version	I	**4002**	44003	R	all	*Six-digit integer value representing communication firmware version in the form nn.nn.nn. For example, 013100 means version 01.31.00.*
Status Byte	I	**4003**	44004	R	all	*Data is an 8-bit value of which the bit assignments are as follows:* Bit # Assignment 0 output 2 active 1 output 1 active 2 alarm 2 active (LED lit) 3 alarm 1 active (LED lit) 4 possible loop break detected 5 *unused (always zero)* 6 remote analog setpoint error 7 process input error *If a bit is set (1), then the condition is TRUE.* *If a bit is not set (0), then the condition is FALSE.* *For example, a value of 12 in the data field means that both alarm1 and alarm 2 are active and everything else is inactive. See 4.15.1.4 for detailed instructions for interpreting a status byte.*

Value	Type	Register Number		R/W	Supported By	Valid Data Field Value
Operating Mode	I	**4004**	44005	R/W	all	1 = manual 2 = standby 3 = normal (automatic) 4 = autotune 5 = recipe run/resume 6 = recipe hold (See Note 1 below.) *default =3 (normal/auto mode)*
Contact/Digital Input State	I	**4006**	44007	R	all except	0 = switch inactive 1 = switch active
Output 1 Output Percent	I	**4007**	44008	R	all	0 to 100
Output 2 Output Percent	I	**4008**	44009	R	all	0 to 100
Manual Control Output 1 Percent	I	**4009**	44010	R/W	all	0 to 100 (See Note 2 below.)
Manual Control Output 2 Percent	I	**4010**	44011	R/W	all	0 to 100 (See Note 2 below.)
Autotune State	I	**4091**	44092	R	all	0 = success 1 = aborted 2 = error: no PID output 3 = error: no deviation 4 = error: no output 5 = error: timed out 6 = error: bad tune 7 = waiting for PV to settle 8 = reverse tune in progress 9 = direct tune in progress
Recipe State	I	**4092**	44093	R	all	0 = done 1 = aborted 2 = error: empty recipe 3 = error: no deviation 4 = recipe on hold 5 = ramping 6 = soaking 7 = ramp holdback 8 = soak holdback
Current Recipe Segment	I	**4093**	44094	R	all	0 to 8 *default = 0*
Resume Exhaustion Flag	I	**4094**	44095	R	all	0 = false (resume block available) 1 = true (resume block exhausted) *default = 0*

Value	Type	Register Number		R/W	Supported By	Valid Data Field Value
LED Status Indicator Byte	I	**4095**	44096	R	all	*Data is an 8-bit value. The bit assignments follows:* Bit # Assignment 0 Unused, always 0 1 Unused, always 0 2 F2 On (excluding serial communication activities; always zero for) 3 F1 On (excluding serial communication activities; always zero for) 4 A2 On (CN8200, CN8240, CN8260 only; always zero for) 5 A1 On (CN8200, CN8240, CN8260 only; always zero for) 6 O2 Active 7 O1 Active *If a bit is set (1), then the condition is TRUE.* *If a bit is not set (0), then the condition is FALSE.* *For example, a value of 48 in the data field means that the F1 and F2 LEDs are lit and all others are not lit. See 4.15.1.4 for detailed instructions for interpreting a status byte.*
Watchdog Disable	FV	**63**	40064	R/W		*Write to this register only if a member of the Omega technical support team tells you to write to it. Writing 3.1415 to this register will disable the controller's watchdog circuit temporarily so that the controller can reset itself.*
Controller Ambient Temperature	FV	**64**	40065	R		*Read the controller's ambient temperature in the unit of measure specified using the display units of measure parameter (see 4.4.2).*

Note 1: To terminate execution of a recipe before all segments of all cycles have been run, change the mode to normal, manual, or standby.

Note 2: There is no default for this value. The controllers are designed to provide "bumpless transfer" from auto to manual mode. That means that when a controller is switched to manual, initially it uses the output percentages used most recently in normal (auto) mode.

6. Controller Parameters Arranged by Register Address

6.1 Overview

For your convenience when interpreting messages returned by controllers to the MODBUS master, this section includes two tables, each in numerical order by register.

- The first table lists all the registers for parameters for which the controller can use only integer values (marked **I** in Section 4).
- The second table lists all the registers used for controller values that can include fractional values (parameters marked **FV** and **FV∗** in Section 4). This table lists the relative address of each parameter in all three regions that can be used to access a fractional value in the controllers.

To see the ranges for the **FV** and **FV∗** parameters and the meaning of specific values for the integers, see the appropriate subsections in Section 4.

6.2 Integer Registers

6.2.1 Introduction

The table in 5.2.2 lists the MODBUS registers used to transmit values for configuration parameters for which the controller can use only integers.

6.2.2 Register List

Register Relative Address	Parameter or Value	R/W	Supported By	See Subsection
4000	Controller Type	R	all	4.15 Controller Information and Status Values
4001	Software Version	R	all	4.15 Controller Information and Status Values
4002	Communications Version	R	all	4.15 Controller Information and Status Values
4003	Status Byte	R	all	4.15 Controller Information and Status Values
4004	Operating Mode	R/W	all	4.15 Controller Information and Status Values
4005	Access Level	R/W	all except	4.13 Security Parameter

Register Relative Address	Parameter or Value	R/W	Supported By	See Subsection
4006	Contact/Digital Input State	R	all except	4.15 Controller Information and Status Values
4007	Output 1 Output Percent	R	all	4.15 Controller Information and Status Values
4008	Output 2 Output Percent	R	all	4.15 Controller Information and Status Values
4009	Manual Control Output 1 %	R/W	all	4.15 Controller Information and Status Values
4010	Manual Control Output 2 %	R/W	all	4.15 Controller Information and Status Values
4011	Autotune Damping	R/W	all	4.8 Autotune Damping Parameter
4012	Recipe Option	R/W	all	4.9 Ramp/Soak Parameters
4013	Single Setpoint Ramp Time	R/W	all	4.9 Ramp/Soak Parameters
4014 to 4021	Ramp Time 1 to Ramp Time 8	R/W	all	4.9 Ramp/Soak Parameters
4022 to 4029	Ramp Event 1 to Ramp Event 8	R/W	all except	4.9 Ramp/Soak Parameters
4030 to 4037	Soak Time 1 to Soak Time 8	R/W	all	4.9 Ramp/Soak Parameters
4038 to 4045	Soak Event 1 to Soak Event 8	R/W	all except	4.9 Ramp/Soak Parameters
4046	Recycle Number	R/W	all	4.9 Ramp/Soak Parameters
4047	Termination State	R/W	all	4.9 Ramp/Soak Parameters
4048	Power Fail Resume Enable	R/W	all	4.9 Ramp/Soak Parameters
4049	Input Type	R/W	all	4.3 Input Parameters
4050	Output 1 Type	R/W	all	4.5 Output Parameters
4051	Output 1 Action	R/W	all	4.5 Output Parameters
4052	Output 1 Alarm Action	R/W	/CN8200	4.5 Output Parameters
4053	Output 1 Alarm Operation	R/W	/CN8200	4.5 Output Parameters
4054	Output 1 Alarm Delay	R/W	/CN8200	4.5 Output Parameters
4055	Output 1 Alarm Inhibit	R/W	/CN8200	4.5 Output Parameters
4056	Output 1 Cycle Time	R/W	all	4.5 Output Parameters
4057	Output 1 Low Limit	R/W	all	4.5 Output Parameters

Register Relative Address	Parameter or Value	R/W	Supported By	See Subsection
4058	Output 1 High Limit	R/W	all	4.5 Output Parameters
4059	Output 2 Type	R/W	all	4.5 Output Parameters
4060	Output 2 Action	R/W	all	4.5 Output Parameters
4061	Output 2 Alarm Action	R/W	/CN8200	4.5 Output Parameters
4062	Output 2 Alarm Operation	R/W	/CN8200	4.5 Output Parameters
4063	Output 2 Alarm Delay	R/W	/CN8200	4.5 Output Parameters
4064	Output 2 Alarm Inhibit	R/W	/CN8200	4.5 Output Parameters
4065	Output 2 Cycle Time	R/W	all	4.5 Output Parameters
4066	Output 2 Low Limit	R/W	all	4.5 Output Parameters
4067	Output 2 High Limit	R/W	all	4.5 Output Parameters
4068	TC/RTD Decimal Position	R/W	all	4.4 Display Parameters
4069	Linear Decimal Position	R/W	all	4.4 Display Parameters
4070	Display Unit	R/W	all	4.4 Display Parameters
4071	Display Blanking	R/W	all	4.4 Display Parameters
4072	Alarm 1 Action	R/W	all except	4.7 Alarm Parameters
4073	Alarm 1 Operation	R/W	all except	4.7 Alarm Parameters
4074	Alarm 1 Delay	R/W	all except	4.7 Alarm Parameters
4075	Alarm 1 Inhibit	R/W	all except	4.7 Alarm Parameters
4076	Alarm 2 Action	R/W	all except	4.7 Alarm Parameters
4077	Alarm 2 Operation	R/W	all except	4.7 Alarm Parameters
4078	Alarm 2 Delay	R/W	all except	4.7 Alarm Parameters
4079	Alarm 2 Inhibit	R/W	all except	4.7 Alarm Parameters
4080	Communication Protocol	R	all	4.2 Communication Parameters
4081	Controller ID	R/W	all	4.2 Communication Parameters
4082	Baud Rate	R/W	all	4.2 Communication Parameters
4083	Parity	R/W	all	4.2 Communication Parameters
4084	IEEE Register Ordering	R/W	all	4.2 Communication Parameters
4085	Output 1 Failsafe Output Percent	R/W	all	4.11 Supervisor Parameters
4086	Output 2 Failsafe Output Percent	R/W	all	4.11 Supervisor Parameters

Register Relative Address	Parameter or Value	R/W	Supported By	See Subsection
4087	Loop Break Time	R/W	all	4.11 Supervisor Parameters
4088	Installed Option Card	R	all except	4.10 Parameters for Options
4089	Auxiliary Output Variable	R/W	all except	4.10 Parameters for Options
4090	Contact/Digital Switch Function	R/W	all except	4.10 Parameters for Options
4091	Autotune State	R	all	4.15 Controller Information and Status Values
4092	Recipe State	R	all	4.15 Controller Information and Status Values
4093	Current Recipe Segment	R	all	4.15 Controller Information and Status Values
4094	Resume Exhaustion Flag	R	all	4.15 Controller Information and Status Values
4095	LED Status Indicator	R	all	4.15 Controller Information and Status Values

6.3 Registers for Parameters That Can Use Fractional Values

6.3.1 Introduction

The table in 5.2.2 lists the MODBUS registers used to access all parameters for which the controller can use a fractional value. The relative address in all three regions is given for each parameter.

6.3.2 Register List

The table below contains the relative address in the base region, 10X region, and 32-bit IEEE region (two registers per value) that can be used to transmit a controller value that can include a fractional value.

The ∗ symbol following the names of some parameters or values indicates that the type of input used by the controller affects the way the controller stores the fractional value, and, thus the way you must read and write the value in the base region and 10X region. See 3.7 and 3.8 for information and examples.

Relative Address			Parameter or Value	R/W	Supported By	See Subsection
Base	10X	IEEE				
0	1000	8000 8001	Process Value∗	R	all	4.14 Process Value and Setpoints
1	1001	8002 8003	Setpoint (EEPROM) ∗	R/W	all	4.14 Process Value and Setpoints
2	1002	8004 8005	Setpoint (RAM) ∗	R/W	all	4.14 Process Value and Setpoints
3	1003	8006 8007	Second Setpoint (EEPROM) ∗	R/W	all except	4.14 Process Value and Setpoints
4	1004	8008 8009	Second Setpoint (RAM) ∗	R/W	all except	4.14 Process Value and Setpoints
5	1005	8010 8011	Remote Analog Setpoint∗	R	all except	4.14 Process Value and Setpoints
6	1006	8012 8013	Recipe Setpoint∗	R	all	4.14 Process Value and Setpoints
7	1007	8014 8015	Output 1 Deadband ∗	R/W	all	4.6 Control Parameters
8	1008	8016 8017	Output 1 Hysteresis ∗	R/W	all	4.6 Control Parameters
9	1009	8018 8019	Output 1 Proportional Band ∗	R/W	all	4.6 Control Parameters

Relative Address			Parameter or Value	R/W	Supported By	See Subsection
Base	10X	IEEE				
10	1010	8020 8021	Output 2 Proportional Band ∗	R/W	all	4.6 Control Parameters
11	1011	8022 8023	Rate/Derivative Action	R/W	all	4.6 Control Parameters
12	1012	8024 8025	Reset/Integral Action	R/W	all	4.6 Control Parameters
13	1013	8026 8027	Manual Reset/Integral Action	R/W	all	4.6 Control Parameters
14	1014	8028 8029	Output 2 Deadband∗	R/W	all	4.6 Control Parameters
15	1015	8030 8031	Output 2 Hysteresis∗	R/W	all	4.6 Control Parameters
16 to 23	1016 to 1023	8032/8033 to 8046/8047	Soak Level 1 to Soak Level 8∗	R/W	all	4.9 Ramp/Soak Parameters
24	1024	8048 8049	Holdback Band	R/W	all	4.9 Ramp/Soak Parameters
25	1025	8050 8051	Input Bias∗	R/W	all	4.3 Input Parameters
26	1026	8052 8053	Linear Input Low Scale∗	R/W	all	4.3 Input Parameters
27	1027	8054 8055	Linear Input High Scale∗	R/W	all	4.3 Input Parameters
28	1028	8056 8057	Lower Setpoint Limit ∗	R/W	all	4.3 Input Parameters
29	1029	8058 8059	Upper Setpoint Limit ∗	R/W	all	4.3 Input Parameters
30	1030	8060 8061	Input Filter	R/W	all	4.3 Input Parameters
31	1031	8062 8063	Output 1 Process Alarm Setpoint ∗	R/W	/CN8200	4.5 Output Parameters
32	1032	8064 8065	Output 1 Deviation, Normal Band, or Inverse Band Alarm Setpoint ∗	R/W	/CN8200	4.5 Output Parameters
33	1033	8066 8067	Output 2 Process Alarm Setpoint ∗	R/W	/CN8200	4.5 Output Parameters
34	1034	8068 8069	Output 2 Deviation, Normal Band, or Inverse Band Alarm Setpoint ∗	R/W	/CN8200	4.5 Output Parameters
35	1035	8070 8071	Display Filter	R/W	all	4.4 Display Parameters

Relative Address			Parameter or Value	R/W	Supported By	See Subsection
Base	10X	IEEE				
36	1036	8072 8073	Alarm 1 Process Setpoint∗	R/W	all except	4.7 Alarm Parameters
37	1037	8074 8075	Alarm 1 Deviation, Normal Band, or Inverse Band Setpoint∗	R/W	all except	4.7 Alarm Parameters
38	1038	8076 8077	Alarm 2 Process Setpoint∗	R/W	all except	4.7 Alarm Parameters
39	1039	8078 8079	Alarm 2 Deviation, Normal Band, or Inverse Band Setpoint∗	R/W	all except	4.7 Alarm Parameters
40	1040	8080 8081	Highest Reading ∗	R/W	all	4.11 Supervisor Parameters
41	1041	8082 8083	Lowest Reading ∗	R/W	all	4.11 Supervisor Parameters
42	1042	8084 8085	T/C Zero Offset	R/W	all	4.12 Calibration Function
43	1043	8086 8087	T/C Span Adjustment	R/W	all	4.12 Calibration Function
44	1044	8088 8089	RTD Zero Offset	R/W	all	4.12 Calibration Function
45	1045	8090 8091	RTD Span Adjustment	R/W	all	4.12 Calibration Function
46	1046	8092 8093	Low-Voltage Zero Offset	R/W	all	4.12 Calibration Function
47	1047	8094 8095	Low-Voltage Span Adjustment	R/W	all	4.12 Calibration Function
48	1048	8096 8097	High-Voltage Zero Offset	R/W	all	4.12 Calibration Function
49	1049	8098 8099	High-Voltage Span Adjustment	R/W	all	4.12 Calibration Function
50	1050	8100 8101	Current Zero Offset	R/W	all	4.12 Calibration Function
51	1051	8102 8103	Current Span Adjustment	R/W	all	4.12 Calibration Function
52	1052	8104 8105	Auxiliary Output Scale Low	R/W	all except	4.10 Parameters for Options
53	1053	8106 8107	Auxiliary Output Scale High	R/W	all except	4.10 Parameters for Options
54	1054	8108 8109	RAS Scale Low ∗	R/W	all except	4.10 Parameters for Options

Relative Address			Parameter or Value	R/W	Supported By	See Subsection
Base	10X	IEEE				
55	1055	8110 8111	RAS Scale High ∗	R/W	all except	4.10 Parameters for Options
56	1056	8112 8113	Active Setpoint ∗	R/W	all	4.14 Process Value and Setpoints
57	1057	8114 8115	RTD Decimal Zero Cal	R/W	all	4.12 Calibration Function
58	1058	8116 8117	RTD Decimal Span Cal	R/W	all	4.12 Calibration Function
59	1059	8118 8119	2nd Hi Volt Zero Cal	R/W	all	4.12 Calibration Function
60	1060	8120 8121	2nd Hi Volt Span Cal	R/W	all	4.12 Calibration Function
61	1061	8122 8123	0 to 100 mV Zero Cal	R/W	all	4.12 Calibration Function
62	1062	8124 8125	0 to 100 mV Span Cal	R/W	all	4.12 Calibration Function
63	1063	8126 8127	Watchdog Disable	R/W		4.15 Controller Information and Status Values
64	1064	8128 8129	Ambient Temperature	R		4.15 Controller Information and Status Values

7. Troubleshooting

7.1 Introduction

This section identifies some of the potential mistakes or misunderstandings that can occur when you use MODBUS functions with the controllers.

The symptoms of these mistakes or misunderstandings are:

- The controller does not reply to a MODBUS request; see 6.2.
- The controller sends an error code 02 ($02) message to the MODBUS master; see 6.3.
- The controller sends an error code 03 ($03) message to the MODBUS master; see 6.4.
- The controller reply to a function 16 ($10) write message indicates that not as many words were written as the command specified; see 6.5.
- The controller reply to a function 03 ($03) read message starts with good data, but the good data is followed by garbage; see 6.6.
- The master receives a reply from more than one controller, or the reply is scrambled; see 6.7.

7.2 No Reply from Controller

When the host does not receive a reply from a controller in response to a request, it is not necessarily a sign that there is a problem. Controllers do not reply to every broadcast message. However, in the case of a broadcast function 06 ($06) or 16 ($10) message. the controllers <u>do</u> write the specified value(s) to their databases.

If a controller does not reply to a message addressed to only that one controller, check the following:

- Does the message contain a function code not supported by the Omega implementation of MODBUS? The message must contain one of the following functions: 03 ($03), 06 ($06), 16 ($10), or 08 ($08) subfunction 00 ($00).
- Is the master appending an accurate CRC to the message? The controller will not respond if the CRC calculated for the message by the controller does not equal the CRC appended to the message by the MODBUS master.
- Are you trying to read or write more than 24 words to a non-IEEE register in a single message? More than 12 words to 32-bit IEEE registers in a single message? The controller will ignore messages that exceed these limits.
- In a function 16 ($10) message, did you use a byte count that is not two times the number of words specified in the message? The controller recognizes that this is an invalid request.

- Are you trying to read or write an odd number of words when addressing a 32-bit IEEE register? Because every 32-bit IEEE value requires two registers, all addresses in the IEEE region of the register map are even.
- Did you send a function 08 ($08) message that included a subfunction code other than 00 (in both bytes)?
- Can the MODBUS master communicate with the controller? Test communications by issuing a function 08 ($08) subfunction 00 ($00) loopback message. If a return message that is an echo of the query is not received by the master, make sure the controller is powered up and in an operating mode. Check the integrity of the network.
- Is the master waiting long enough for a reply? See 1.2.8 for guidelines for calculating the length of time the master should wait before assuming that the controller has not sent a reply.

7.3 Controller Sends an Error Code $02 Message

The controller sends an error code 02 $02 (invalid address) message to the MODBUS master in the following situations.

- The register relative address in a function 06 ($06) message or the first register relative address in a function 03 ($03) or 16 ($10) message is not a valid register in the Omega MODBUS register map, or
- The register relative address in a function 06 ($06) message contains an address in the 32-bit IEEE region of the register map. Because every 32-bit IEEE floating point values requires two registers to write, the 06 function (writing to a single register) cannot be used for this region.

7.4 Controller Sends an Error Code $03 Message

The controller sends an error code 03 ($03) (invalid data) message to the MODBUS master in the following situations.

- The register relative address in a function 06 ($06) or the first register relative address in a 16 ($10) message contains a read-only value, or
- The data in a function 06 ($06) message contains a value that is not valid for the destination (target) register, or
- The data for the first register to be written by a function 16 ($10) message is not valid for the destination (target) register.

7.5 Controller Reply to a Function 16 ($10) Write Message Indicates Too Few Words Were Written

If a controller sends the master a reply to a function 16 ($10) write message that indicates that fewer words were written than the master's message specified, this is an indication that the write function encountered a problem after the first register was written. When the controller encounters a problem writing the specified data, the controller stops trying and sends a reply that indicates how many words the controller was able to write before the problem was encountered.

7.6 Controller Reply to a Function 03 ($03) Read Message Starts with Good Data, But Garbage Follows

Sometimes a controller sends the master a reply to a function 03 ($03) read message (for multiple registers) that starts out with good data, but the good data is followed by garbage. This is a sign that the first register the master asked the controller to read was the address of a valid register, but that one or more invalid register addresses followed. If the first register had been invalid (that is if the **First Register Relative Address** field contained an address of an invalid register), then a code $02 error message would have been returned.

7.7 Master Receives Reply from More Than One Controller, or the Reply Is Scrambled

If the master receives a reply from more than one controller, or the reply is scrambled, it is possible that more than one MODBUS slave device is configured with the same address. By default all CN8200, CN8240, and CN8260 controllers have a controller ID of 1. If you do not assign each device a unique address, communications will not work.

8. Factory Commands

8.1 Introduction

There are a few commands that are rarely used in the field. These “factory commands” are issued to a controller by writing special values to special registers using function 16 ($10) write messages.

The factory commands supported are:

- load all parameter defaults; see 7.2
- perform zero calibration; see 7.3
- perform span calibration; see 7.3
- clear all latched alarms; see 7.4

8.2 Load All Parameter Defaults

8.2.1 Principles

The purpose of the command is to return all database values in the controller (with the exception of the controller ID) to the factory defaults.[24]

To execute this command, you must write the value 85 ($55) to register relative address 7000 ($1B58) and 92 ($5C) to register relative address 7001 ($1B59).

You must write these values in a single message using function 16 ($10).

If the controller receives the message and is able to write the values, the controller will send a normal reply.

8.2.2 Example

To set all parameter values to the factory defaults, send the message shown below to the appropriate controller. In this example, we have assumed that the controller address is 01 ($01). The command could be sent to any , CN8200, CN8240, or CN8260 controller on the network.

Device Address	**Function Code 16**	**First Register Relative Address**		**Word Count**		**Byte Count**	**2 Words (4 Bytes) of Data**	**CRC**	
$01	$10	$1B	$58	$00	$02	$04	$00 55 $00 5C	$59	$EC

[24] The address must be configured using DIP switches.

8.3 Perform Zero and Span Calibration

8.3.1 Introduction

You do not have to calibrate every new controller. When a controller was ordered, you specified an input type for which the unit was calibrated at the factory. This is not the specific type written to the input type register at relative address 4049, such as J thermocouple, or 0 to 20 mA linear. In the context of ordering the controller, “type” refers to these choices: RTD, compressed RTD, thermocouple, millivolt linear, volt linear, or current linear input. The controller was calibrated at the factory for the type of input specified. If you use the controller with a different type of input, you must recalibrate as described in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual* unless you ordered the “Calibrate All” input option.[25]

For example, if you specified when you ordered the controller that you planned to use a thermocouple as the sensor, then you can use the input type register to specify any thermocouple type: B, C, E, J, K, N, NNM, R, S, T, or Platinel II. The controller will be calibrated appropriately at the factory. However, if you ordered thermocouple calibration, but decide to use the controller with an RTD sensor, then you should recalibrate before using the controller.

When calibrating the controller, you can use MODBUS commands to write to special registers in the factory region as described in this section. (Do not use the factory calibration commands before you have read the calibration instructions in the *(, CN8200, CN8240, CN8260) Controller Configuration and Operation Manual.)*.

The values calculated by the controller for the zero offset and the span adjustment are stored in the registers listed in the table in 4.12.2, and can be read and written using MODBUS. Different registers are available for various types of input. However, the controller calibrates only for the type of input specified using the input type parameter.

The values in the calibration zero offset and span adjustment registers should not be changed using MODBUS write functions. The zero offset and span adjustment should be changed only by the controller’s calibration procedure. However, if you have used the MODBUS master to back up your controller’s database, including the zero offset and span adjustment, you can safely restore the values using MODBUS write functions.

Before issuing the commands described below, you must prepare the controller as described in the configuration and operation manual. This includes providing a simulated input at the appropriate value for the type of input used, and for the operation to be performed (that is, different values for zero calibration and span calibration). The appropriate simulated input values are in the configuration and operation manual.

[25] To determine whether the controller in hand was calibrated at the factory for all input types, check the model number on the label on the controller. The meaning of each character in the model number is in the installation manual supplied with the controller.

8.3.2 Principles

The purposes of the commands to perform zero and span calibration are to clear the existing zero offset and span adjustment values stored in the controller for the type of input used, and then initiate the low cal (zero) and high cal (span) operations.

To execute the low cal command, you must write the value 85 ($55) to register relative address 7002 ($1B5A) and 92 ($5C) to register relative address 7003 ($1B5B).

To execute the high cal command, you must write the value 85 ($55) to register relative address 7004 ($1B5C) and 92 ($5C) to register relative address 7005 ($1B5D).

When the controller has been prepared, you must write the low cal command values in a single message using function 16 ($10). When the controller has been prepared for the high cal operation, send another function 16 ($10) message.

If the controller receives the message and is able to write the values, the controller will send a normal reply in response to each message. This does not necessarily mean that the calibration will be valid. A normal reply indicates that the controller did its part of the calibration procedure. However, if you did not prepare the controller properly, the new calibration values will not be valid. See 7.3.4 for guidelines for confirming that the calibration was done.

8.3.3 Example

8.3.3.1 Zero (Low) Cal

To initiate the zero (low) cal operation, send the message shown below to the appropriate controller. In this example, we have assumed that the controller address is 01 ($01). The command could be sent to any , CN8200, CN8240, or CN8260 controller on the network.

Device Address	**Function Code 16**	**First Register Relative Address**		**Word Count**		**Byte Count**	**2 Words (4 Bytes) of Data**	**CRC**	
$01	$10	$1B	$5A	$00	$02	$04	$00 55 $00 5C	$D8	$35

8.3.3.2 Span (High) Cal

To initiate the span (high) cal operation, send the message shown below to the appropriate controller. In this example, we have assumed that the controller address is 01 ($01). The command could be sent to any , CN8200, CN8240, or CN8260 controller on the network.

Device Address	**Function Code 16**	**First Register Relative Address**		**Word Count**		**Byte Count**	**2 Words (4 Bytes) of Data**	**CRC**	
$01	$10	$1B	$5C	$00	$02	$04	$00 55 $00 5C	$58	$1F

8.3.4 Confirming That the Calibration Was Performed

A normal reply message from the controller in response to the write function messages containing the low cal and high cal commands does not mean that the calibration procedure was carried out successfully. It simply indicates that the calibration request message was received in good form and that the controller will try to do the calibration procedure. For the calibration to be successful, you must prepare the controller as described in the *(, CN8200, CN8240, and CN8260) Controller Configuration and Operation Manual*.

You can confirm that the low cal and high cal operations were done. The calibration procedure takes about two seconds to complete. If the zero offset register (in the 32-bit IEEE region of the register map) contains a non-zero value at least two seconds after the master sent the low cal command, then the calibration procedure took place. Assuming that you prepared the controller correctly, then the zero calibration was successful.

Similarly, if the value of the span adjustment value in the 32-bit IEEE region is not exactly 1.000 at least two seconds after the master sent the high cal command, then the calibration procedure was carried out.

For example, suppose the currently selected input type is a thermocouple. In this case, two seconds after the low cal command message is sent, the register at relative address of 8084 (the IEEE mirror for relative address 42) will contain a non-zero value if calibration was done and zero if the calibration was not performed.

8.4 Clear All Latched Alarms

8.4.1 Principles

The purpose of the command is to clear latched alarms (LEDs and alarm outputs).

To execute this command, you must write the value 85 ($55) to register relative address 7006 ($1B5E) and 92 ($5C) to register relative address 7007 ($1B5F).

You must write these values in a single message using function 16 ($10).

If the controller receives the message and is able to write the values, the controller will send a normal reply.

8.4.2 Example

To clear the latched alarm outputs, send the message shown below to the appropriate controller. In this example, we have assumed that the controller address is 01 ($01). The command could be sent to any , CN8200, CN8240, or CN8260 controller on the network.

Device Address	**Function Code 16**	**First Register Relative Address**		**Word Count**		**Byte Count**	**2 Words (4 Bytes) of Data**	**CRC**	
$01	$10	$1B	$5E	$00	$02	$04	$00 55 $00 5C	$D9	$C6

12. Index