# Modbus Slave RTU
# Serial Driver Help

# Table of Contents

# Modbus Slave RTU Serial Driver Help

Help version 1.031

**CONTENTS**



## Overview

The Modbus Slave RTU Serial Driver provides an easy and reliable way to connect Modbus Slave RTU Serial devices to OPC Client applications, including HMI, SCADA, Historian, MES, ERP and countless custom applications. It simulates up to 255 Modbus slave devices on a serial communications network. Other devices or PCs can communicate with each simulated Modbus slave device using the Modbus protocol.

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

## Device Setup

### Supported Devices
Modbus compatible devices.

### Communication Protocol
Modbus RTU Protocol.

### Supported Communication Parameters*
Baud Rate: 1200, 2400, 9600, 19200
Parity: Odd, Even, None
Data Bits: 8
Stop Bits: 1, 2

*Not all devices support the listed configurations.

### Device ID (PLC Network Address)
Modbus Serial devices are assigned Device IDs in the range 1 to 255.

### Flow Control
When using an RS232/RS485 converter, the type of flow control that is required will depend upon the needs of the converter. Some converters do not require any flow control and others will require RTS flow. Consult the converter's documentation in order to determine its flow requirements. We recommend using an RS485 converted that provides automatic flow control.

**Note:** When using the manufacturer's supplied communications cable, it is sometimes necessary to choose a flow control setting of **RTS** or **RTS Always** under the Channel Properties.

### Supported Function Codes

- Read Coil Status-code 01H
- Read Input Status-code 02H
- Read Holding Registers-code 03H
- Read Internal Registers-code 04H
- Force Single Coil-code 05H
- Preset Single Register-code 06H
- Diagnostic Loopback-code 08H
- Force Multiple Coils-code 0FH
- Preset Multiple Registers-code 10H

**Note:** For all other function codes, the driver will return an exception code 01H (function not implemented) to the requesting device.

### Accessible Memory Locations

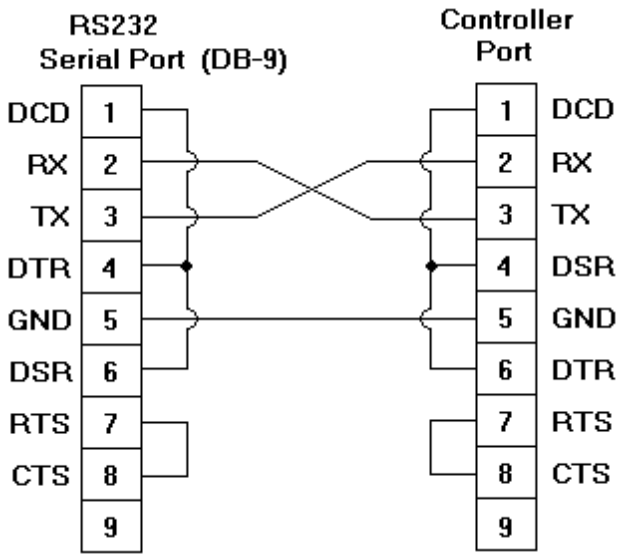
- Output coils-00001 to 065536
- Input coils-10001 to 165536
- Internal registers-30001 to 365536
- Holding registers-40001 to 465536

**Note:** These settings are configurable. For more information, refer to **Memory**.

### Broadcast Commands
The Modbus Slave RTU Serial driver has the ability to receive broadcast write messages. Broadcast messages are defined by using a station ID of 0. When the driver receives a write message (Function 05H, 06H, 0FH, or 10H), with a station ID of 0 the value to be written is placed in all devices defined under the channel on which the command was received. Essentially the broadcast command can be used to send a single piece of data to every device that has been configured in the driver at the same time.

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

## Cable Diagram




## Modem Setup

This driver supports modem functionality. For more information, please refer to the topic "Modem Support" in the OPC Server Help documentation.

## Memory

### Allowed Address Range

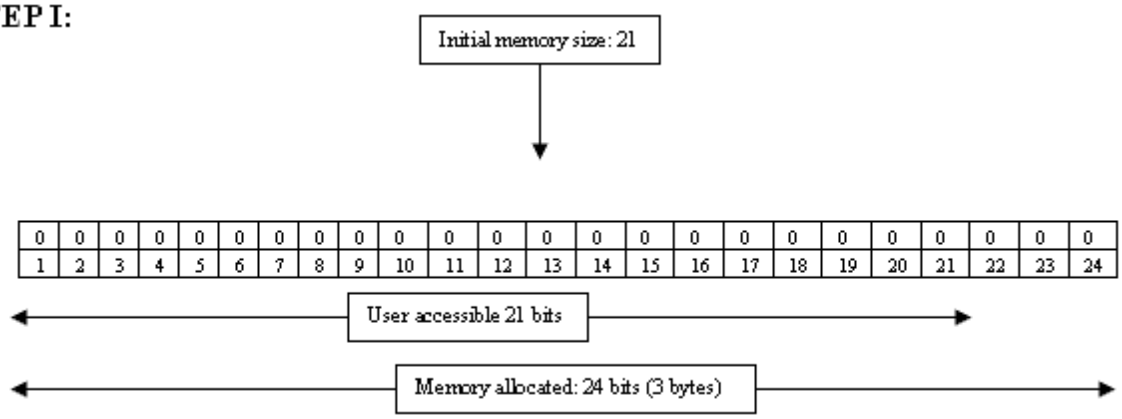
The address range of coils and registers can be configured, and any value between 9999 and 65536 can be specified. The tags can then be addressed up to and including the specified value. Note the following:

1. The address range cannot be changed when the tags are being processed.
2. When the address range is changed, it is possible that a remote request (from a Modbus master) will get rejected because the requested memory address is outside the new address range.
3. When the address range is changed, and the new upper limit is greater than the old one, all old data is preserved and the remaining memory is initialized to '0'. If the new upper limit is smaller then the old one, however, only the data equal to the new memory size is preserved and the remaining data will be lost. There may be an exception to this when dealing with Boolean memory type.

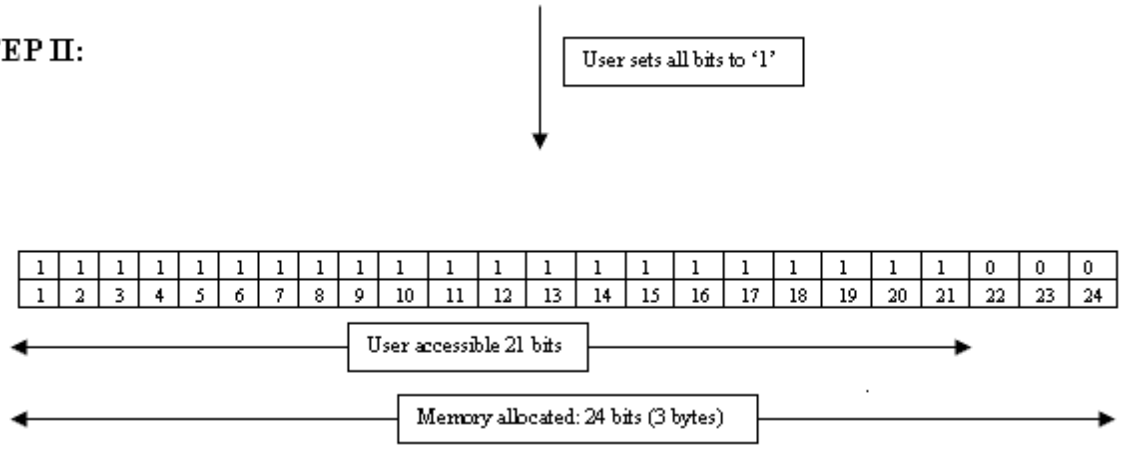
   For example, assume that the initial memory size is 21. This translates into 3 bytes (byte aligned) of memory for Boolean types. Now, say the memory size is changed to 12 (which is 2 bytes). The smaller of the two memory sizes is 2 bytes and that is the amount of data that is preserved. Although users may think only 12 bits have been preserved, 16 bits (2 bytes) have been preserved. Normally this would not be noticed, because with a memory size of 12, memory can be accessed up to index 12 only. If the memory size is increased to 22 (which is 3 bytes) the amount of data that will be preserved from the old memory is 2 bytes (smaller of the two). Even though 12 bits were manipulated earlier (with a memory size of 12), users will have old data for bits 13-16 (which may have been initialized to some values the first time around when the memory size was 21).

The images below apply a coil memory type to the example above for a diagrammatic explanation.

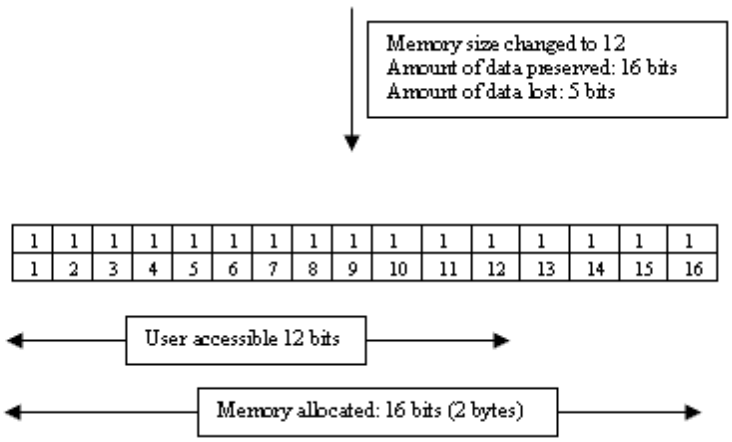
**STEP I:**



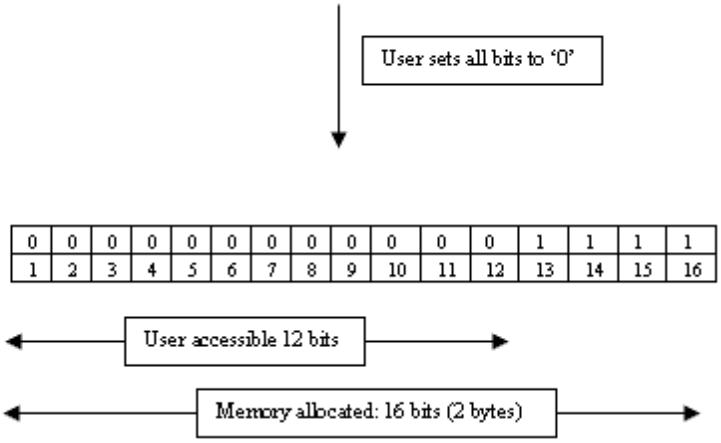

**STEP II:**



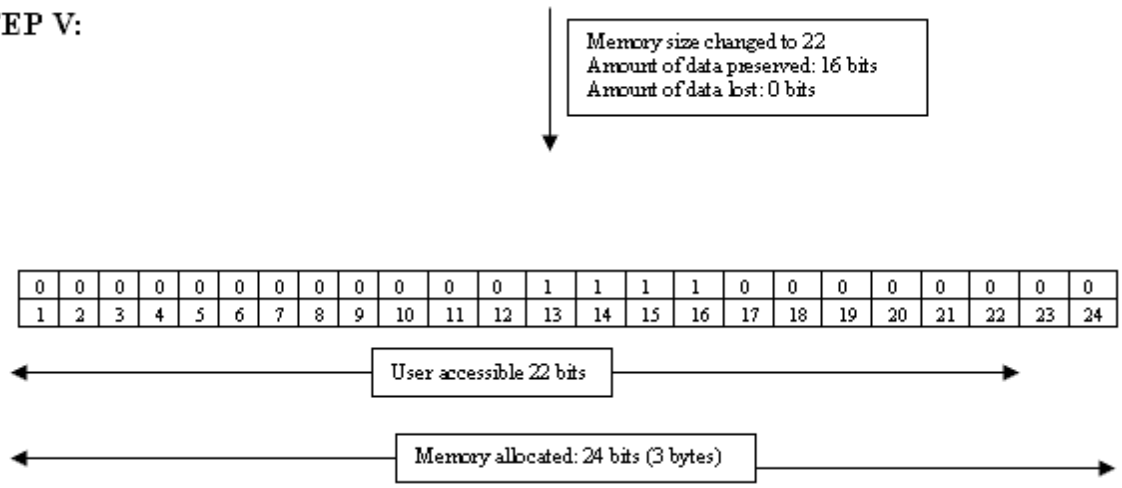

**STEP III:**



STEP IV:




STEP V:




Step V above shows old data for bits 13-16 that were initialized in Step II. Users may expect all the bits in Step V to be '0,' but bits 13-16 have been carried over from Step II and are still set to '1'.

### Zero vs. One Based Addressing

If the address numbering convention for the device starts at one as opposed to zero, users can specify so when defining the device's parameters. By default, addresses entered by users will have one subtracted when frames are constructed to communicate with a Modbus device. If the device doesn't follow this convention, uncheck **Use zero based addressing** in Device Properties. For the appropriate application to obtain information on setting device properties, refer to the online help. The default behavior follows the convention of the Modicon PLCs.

**Note:** 'Use zero based addressing' must be unchecked when using the Daniels/Enron Device.

### First word low in 32 bit data types

Two consecutive register addresses in a Modbus device are used for 32-bit data types. Users can specify whether the driver should assume the first word is the low or the high word of the 32-bit value. If **First word low in 32 bit data types**is checked (the default), first word low will be assumed, which follows the convention of the Modicon Modsoft programming software.

**Note:** 'First word low in 32 bit data types' must be unchecked when using the Daniels/Enron Device.

### First DWord low in 64 bit data types

Four consecutive register addresses (in two groups of two each) are used for 64-bit data types. Users can specify whether the driver should assume the first pair (i.e., first DWord) is the low or the high DWord of the 64-bit value. If **First DWord low in 64 bit data types** is checked, the first DWord low will be assumed; if unchecked, the second DWord low will be assumed.

**Note:** 'First DWord low in 64 bit data types' must be unchecked when using the Daniels/Enron Device.

### OPC Quality Bad Until Write

This option controls the initial OPC quality of tags attached to this driver. When unchecked, all tags will have an initial value of 0 and good OPC quality. This is the default condition. When checked, all tags will have an initial value of 0 and bad OPC quality. The quality of a tag will remain bad until all coils or registers referenced by the tag have been written to by a Modbus master or a client application. For example, a tag with address 400001 and data type DWord references two holding registers: 400001 and 400002. This tag would not show good quality until both holding registers had been written to.

## Timeout

### Communications Timeout

This parameter is used to specify the amount of time that the driver will wait for an incoming request before setting all unsolicited device tags on the channel to a bad quality. After a communications timeout has occurred, the only way to reset the timeout and allow all tags be processed normally is to reestablish communications with the device or disable the timeout by setting **Communications Timeout to 0** (zero) in the **Timeout** tab under Channel Properties.

Disabled: 0
Enabled: 1-->64,800 seconds (18 hours)

### Request Timeout

This parameter is used to specify the amount of time that the driver will wait for a complete request frame to be received. The elapsed time is calculated starting from the instant the first byte of a new request is received. If a complete request frame is not received during this time, the driver will flush its received data buffers and assume the next received byte is the start of a new request. This setting should be chosen carefully.

**Values for the Request Timeout** setting may range from 0 to 30,000 ms, with a default of 0. When 0 is entered, the driver will compute a reasonable timeout through the use of the following formula:

Tdefault = 1000*(Bits per Byte)*512*3 / Baud

This is three times the amount of time required to transmit a frame of 512 bytes. The number of bits per byte includes the start bit and the number of data and stop bits specified in the **Communications** tab under **Channel Properties**. For example, a baud rate of 9600 and 8 data bits, and 1 stop bit, will result in a default timeout of 1600 ms. If the hardware sends relatively short request frames and would retry a failed request in less than the default calculation (1600 ms in this example), try configuring a shorter **Request Timeout**.

The Request Timeout should never be shorter than the amount of time it takes to receive the longest request frame sent by any device on the channel. This can be computed using the following formula:

Tmin = 1000*(Bits per Byte)*(max frame length) / Baud

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

## Data Types Description

| Data Type | Description |
|---|---|
| Boolean | Single bit |
| Word | Unsigned 16 bit value<br><br>bit 0 is the low bit<br>bit 15 is the high bit |
| Short | Signed 16 bit value<br><br>bit 0 is the low bit<br>bit 14 is the high bit<br>bit 15 is the sign bit |
| DWord | Unsigned 32 bit value<br><br>bit 0 is the low bit<br>bit 31 is the high bit |
| Long | Signed 32 bit value<br><br>bit 0 is the low bit<br>bit 30 is the high bit<br>bit 31 is the sign bit |
| BCD | Two byte packed BCD<br><br>Value range is 0-9999. Behavior is undefined for values beyond this range. |
| LBCD | Four byte packed BCD<br><br>Value range is 0-99999999. Behavior is undefined for values beyond this range. |
| String | Null terminated ASCII string<br><br>Supported within the Holding Register Range, includes HiLo LoHi byte order selection. |
| Double* | 64 bit floating point value<br><br>The driver interprets four consecutive registers as a double precision value by making the last two registers the high DWord and the first two registers the low DWord. |
| Double Example | If register 40001 is specified as a double, bit 0 of register 40001 would be bit 0 of the 64 bit data type and bit 15 of register 40004 would be bit 63 of the 64 bit data type. |
| Float* | 32 bit floating point value<br><br>The driver interprets two consecutive registers as a single precision value by making the last register the high word and the first register the low word. |
| Float Example | If register 40001 is specified as a float, bit 0 of register 40001 would be bit 0 of the 32 bit data type and bit 15 of register 40002 would be bit 31 of the 32 bit data type. |

*The descriptions assume the default-first DWord low data handling of 64 bit data types, and first word low data handling of 32 bit data types.

## Address Descriptions

Address specifications vary depending on the model in use. Select a link from the following list to obtain specific address information for the model of interest.

**Modbus Addressing**
**Daniels/Enron Addressing**

## Modbus Addressing

The following address descriptions apply to the client application's access to each simulated Modbus slave device. The client application controls the memory of the simulated Modbus slave device; therefore, all areas have Read/Write access.

Default data types for dynamically defined tags are shown in **bold** where appropriate.

| Address | Range* | Data Type | Access |
|---|---|---|---|
| Output Coils | 000001-065536 | **Boolean** | Read/Write |
| Input Coils | 100001-165536 | **Boolean** | Read/Write |
| Internal Registers | 300001-365536<br>300001-365535<br>3xxxxx.0-3xxxxx.15 | **Word**, Short, BCD<br>Float, DWord, Long, LBCD<br>Boolean, Double | Read/Write |
| Internal Registers As String with HiLo Byte Order<br><br>[Function Codes (decimal): 04] | 300001.2H-365536.240H<br><br>.Bit is string length, range 2 to 240 bytes. | **String** | Read Only |
| Internal Registers As String with LoHi Byte Order<br><br>[Function Codes (decimal): 04] | 300001.2L-365536.240L<br><br>.Bit is string length, range 2 to 240 bytes. | **String** | Read Only |
| Holding Registers | 400001-465536<br>400001-465535<br>4xxxxx.0-4xxxxx.15 | **Word**, Short, BCD<br>Float, DWord, Long, LBCD<br>Boolean, Double | Read/Write |
| Holding Registers As String with HiLo Byte Order | 400001.2H-465536.240H<br><br>.Bit is string length, range 2 to 240 bytes. | **String** | Read/Write |
| Holding Registers As String with LoHi Byte Order | 400001.2L-465536.240L<br><br>.Bit is string length, range 2 to 240 bytes. | **String** | Read/Write |

*The maximum range is determined by the value set in the Memory device property. For more information, refer to **Memory**.

### Array Support

Arrays are supported for internal and holding register locations for all data types except for Boolean. Arrays are also supported for input and output coils (Boolean data types). There are two methods of addressing an array. Examples are given using holding register locations.

4xxxx [rows] [cols]

4xxxx [cols] this method assumes rows is equal to one
For Word, Short and BCD arrays, the base address + (rows *cols) cannot exceed 65536.
For Float, DWord, Long and Long BCD arrays, the base address + (rows *cols *2) cannot exceed 65535.

### String Support

The Modbus model supports reading and writing holding register memory as an ASCII string. When using holding registers for string data, each register will contain two bytes of ASCII data. The order of the ASCII data within a given register can be selected when the string is defined. The length of the string can be from 2 to 240 bytes and is entered in place of a bit number. The length must be entered as an even number. The byte order is specified by appending either a "H" or "L" to the address.

### String Examples

1. To address a string starting at 400200 with a length of 100 bytes and HiLo byte order, enter:
400200.100H

2. To address a string starting at 400500 with a length of 78 bytes and LoHi byte order, enter:
400500.78L

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

## Daniels/Enron Addressing

The following address descriptions apply to the client application's access to each simulated Daniels/Enron slave device. The client application controls the memory of the simulated slave device; therefore, all areas have Read/Write access.

The default data types for dynamically defined tags are shown in **bold** where appropriate. The following table assumes that the slave device has been configured for the maximum allowed address range of 0 to 65535. For more information, refer to **Memory**.

| Address | Range* | Data Type | Access |
|---|---|---|---|
| Output Coils | 000000-065535 | **Boolean** | Read/Write |
| Input Coils | 100000-165535 | **Boolean** | Read/Write |
| Internal Registers | 300000-365535<br>300000-365534<br>300000-365532<br>300000.0-365535.15 | **Word**, Short, BCD<br>Float, DWord, Long, LBCD<br>Double<br>Boolean | Read/Write |
| Holding Registers | 400000-405000<br>406000-407000<br>408000-465535<br><br>400000-404999<br>406000-406999<br>408000-465534<br><br>400000-404999<br>405001-405999<br>406000-406999<br>408000-465534<br><br>400000-404999<br>406000-406999<br>407001-407999<br>408000-465534<br><br>400000-404997<br>406000-406997<br>408000-465532 | **Word**, Short, BCD<br><br><br><br>DWord, LBCD<br><br><br><br>Long<br><br><br><br><br>Float<br><br><br><br><br>Double | Read/Write |
| Holding Registers<br>as Booleans | 400000.xx-405000.xx<br>405001.yy-405999.yy<br>406000.xx-465535.xx<br><br>xx is the bit number from 0-15<br>yy is the bit number from 0-31 | **Boolean** | Read/Write |
| Holding Registers As<br>String with HiLo Byte Order | 400000.xxxH-405000. xxxH<br>406000.xxxH-407000. xxxH<br>408000.xxxH-465535. xxxH<br><br>xxx is string length,<br>range 2 to 240 bytes. | **String** | Read/Write |
| Holding Registers As<br>String with LoHi Byte Order | 400000.xxxL-405000.xxxL<br>406000.xxxL-407000.xxxL<br>408000.xxxL-465535.xxxL<br><br>xxx is string length,<br>range 2 to 240 bytes. | **String** | Read/Write |

*The maximum range is determined by the value set in the Memory device property. For more information, refer to **Memory**.

### Array Support

Arrays are supported for internal and holding register locations for all data types except for Boolean. Arrays are also supported for input and output coils (Boolean data types). There are two methods of addressing an array. Examples are given using holding register locations.

4xxxx [rows] [cols]

4xxxx [cols] this method assumes rows is equal to one
For Word, Short and BCD arrays, the base address + (rows *cols) cannot exceed 65535.
For Float, DWord, Long and Long BCD arrays, the base address + (rows *cols *2) cannot exceed 65534.

### String Support

The Modbus model supports reading and writing holding register memory as an ASCII string. When using holding registers for string data, each register will contain two bytes of ASCII data. The order of the ASCII data within a given register can be selected when the string is defined. The length of the string can be from 2 to 240 bytes and is entered in place of a bit number. The length must be entered as an even number. The byte order is specified by appending either a "H" or "L" to the address.

### String Examples

1. To address a string starting at 400200 with a length of 100 bytes and HiLo byte order, enter:
400200.100H

2. To address a string starting at 400500 with a length of 78 bytes and LoHi byte order, enter:
400500.78L

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

## Error Descriptions

The following error/warning messages may be generated. Click on the link for a description of the message.

### Address Validation

**Missing address**
**Device address '<address>' contains a syntax error**
**Address '<address>' is out of range for the specified device or register**
**Data Type '<type>' is not valid for device address '<address>'**
**Device address '<address>' is Read Only**
**Array size is out of range for address '<address>'**
**Array support is not available for the specified address: '<address>'**

### Serial Communications

**COMn does not exist**
**Error opening COMn**
**COMn is in use by another application**
**Unable to set comm parameters on COMn**
**Communications error on '<channel name>' [<error mask>]**

### Device Status Messages

**Device '<device name>' is not responding**
**Unable to write to '<address>' on device '<device name>'**

### Device Specific Messages

**Address size for device '<device name>' changed from '<old address size>' to '<new address size>'**

### See Also:

**Modbus Exception Codes**

## Modbus Exception Codes

The following data is from Modbus Application Protocol Specifications documentation.

| Code Dec/Hex | Name | Description |
|---|---|---|
| 01/0x01 | ILLEGAL FUNC-TION | The function code received in the query is not an allowable action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type. For example, because it is unconfigured and is being asked to return register values. |
| 02/0x02 | ILLEGAL DATA ADDRESS | The data address received in the query is not an allowable address for the server (or slave). More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed, a request with offset 96 and length 5 will generate exception 02. |
| 03/0x03 | ILLEGAL DATA VALUE | A value contained in the query data field is not an allowable value for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expec-tation of the application program, since the Modbus protocol is unaware of the sig-nificance of any particular value of any particular register. |
| 04/0x04 | SLAVE DEVICE FAILURE | An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action. |
| 05/0x05 | ACKNOWLEDGE | The slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the master. The master can next issue a Poll Program Complete message to determine if processing is completed. |
| 06/0x06 | SLAVE DEVICE BUSY | The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free. |

| | | |
|---|---|---|
| 07/0x07 | NEGATIVE ACKNOWLEDGE | The slave cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 decimal. The master should request diagnostic or error information from the slave. |
| 08/0x08 | MEMORY PARITY ERROR | The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device. |
| 10/0x0A | GATEWAY PATH UNAVAILABLE | Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. This usually means that the gateway is misconfigured or overloaded. |
| 11/0x0B | GATEWAY TARGET DEVICE FAILED TO RESPOND | Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network. |

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

## Address Validation

The following error/warning messages may be generated. Click on the link for a description of the message.

**Address Validation**
**Missing address**
**Device address '<address>' contains a syntax error**
**Address '<address>' is out of range for the specified device or register**
**Data Type '<type>' is not valid for device address '<address>'**
**Device address '<address>' is Read Only**
**Array size is out of range for address '<address>'**
**Array support is not available for the specified address: '<address>'**

## Missing address

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically has no length.

**Solution:**
Re-enter the address in the client application.

## Device address '<address>' contains a syntax error

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically contains one or more invalid characters.

**Solution:**
Re-enter the address in the client application.

## Address '<address>' is out of range for the specified device or register

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically references a location that is beyond the range of supported locations for the device.

**Solution:**
Verify the address is correct; if it is not, re-enter it in the client application.

## Data Type '<type>' is not valid for device address '<address>'

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically has been assigned an invalid data type.

**Solution:**
Modify the requested data type in the client application.

## Device address '<address>' is Read Only

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically has a requested access mode that is not compatible with what the device supports for that address.

**Solution:**
Change the access mode in the client application.

## Array size is out of range for address '<address>'

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically is requesting an array size that is too large for the address type.

**Solution:**
Re-enter the address in the client application to specify a smaller value for the array or a different starting point.

## Array support is not available for the specified address: '<address>'

**Error Type:**
Warning

**Possible Cause:**
A tag address that has been specified dynamically contains an array reference for an address type that doesn't support arrays.

**Solution:**
Re-enter the address in the client application to remove the array reference or correct the address type.

## Serial Communications

The following error/warning messages may be generated. Click on the link for a description of the message.

**Serial Communications**
**COMn does not exist**
**Error opening COMn**
**COMn is in use by another application**
**Unable to set comm parameters on COMn**
**Communications error on '<channel name>' [<error mask>]**

## COMn does not exist

**Error Type:**
Fatal

**Possible Cause:**
The specified COM port is not present on the target computer.

**Solution:**
Verify that the proper COM port has been selected.

## Error opening COMn

**Error Type:**
Fatal

**Possible Cause:**
The specified COM port could not be opened due an internal hardware or software problem on the target computer.

**Solution:**
Verify that the COM port is functional and may be accessed by other Windows applications.

## COMn is in use by another application

**Error Type:**
Fatal

**Possible Cause:**
The serial port assigned to a device is being used by another application.

**Solution:**
Verify that the correct port has been assigned to the channel.

## Unable to set comm parameters on COMn

**Error Type:**
Fatal

**Possible Cause:**
The serial parameters for the specified COM port are not valid.

**Solution:**
Verify the serial parameters and make any necessary changes.

## Communications error on '<channel name>' [<error mask>]

**Error Type:**
Serious

**Error Mask Definitions:**
**B** = Hardware break detected.
**F** = Framing error.
**E** = I/O error.
**O** = Character buffer overrun.
**R** = RX buffer overrun.
**P** = Received byte parity error.
**T** = TX buffer full.

**Possible Cause:**
1. The serial connection between the device and the Host PC is bad.
2. The communications parameters for the serial connection are incorrect.

**Solution:**
1. Verify the cabling between the PC and the device.
2. Verify that the specified communications parameters match those of the device.

## Device Status Messages

The following error/warning messages may be generated. Click on the link for a description of the message.

**Device Status Messages**
**Device '<device name>' is not responding**

Unable to write to '<address>' on device '<device name>'

## Device '<device name>' is not responding

**Error Type:**
Serious

**Possible Cause:**
1. The serial connection between the device and the Host PC is broken.
2. The communications parameters for the serial connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.
4. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

**Solution:**
1. Verify the cabling between the PC and the device.
2. Verify the specified communications parameters match those of the device.
3. Verify the Network ID given to the named device matches that of the actual device.
4. Increase the Request Timeout setting so that the entire response can be handled.

## Unable to write to '<address>' on device '<device name>'

**Error Type:**
Serious

**Possible Cause:**
1. The serial connection between the device and the Host PC is broken.
2. The communications parameters for the serial connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.

**Solution:**
1. Verify the cabling between the PC and the device.
2. Verify the specified communications parameters match those of the device.
3. Verify the Network ID given to the named device matches that of the actual device.

## Device Specific Messages

The following error/warning messages may be generated. Click on the link for a description of the message.

**Device Specific Messages**
Address size for device '<device name>' changed from '<old address size>' to '<new address size>'

## Address size for device '<device name>' changed from '<old address size>' to '<new address size>'

**Error Type:**
Information

**Possible Cause:**
This is not an error but a confirmation that the address size for the specified device has been changed.

**Solution:**
N/A

# Index

## F



## L



## M



## N



## O



## P



## S

## T



## U



## W