Modbus Ethernet Driver Help

Table of Contents

Modbus Ethernet Driver Help

Help version 1.090

CONTENTS

Overview

The Modbus Ethernet driver provides a reliable way to connect Modbus Ethernet devices to OPC Client applications; including HMI, SCADA, Historian, MES, ERP, and countless custom applications. Users must install TCP/IP properly to use this driver. For more information on setup, refer to the Windows documentation.

Note: The driver will post messages when a failure occurs during operation.

Channel Setup

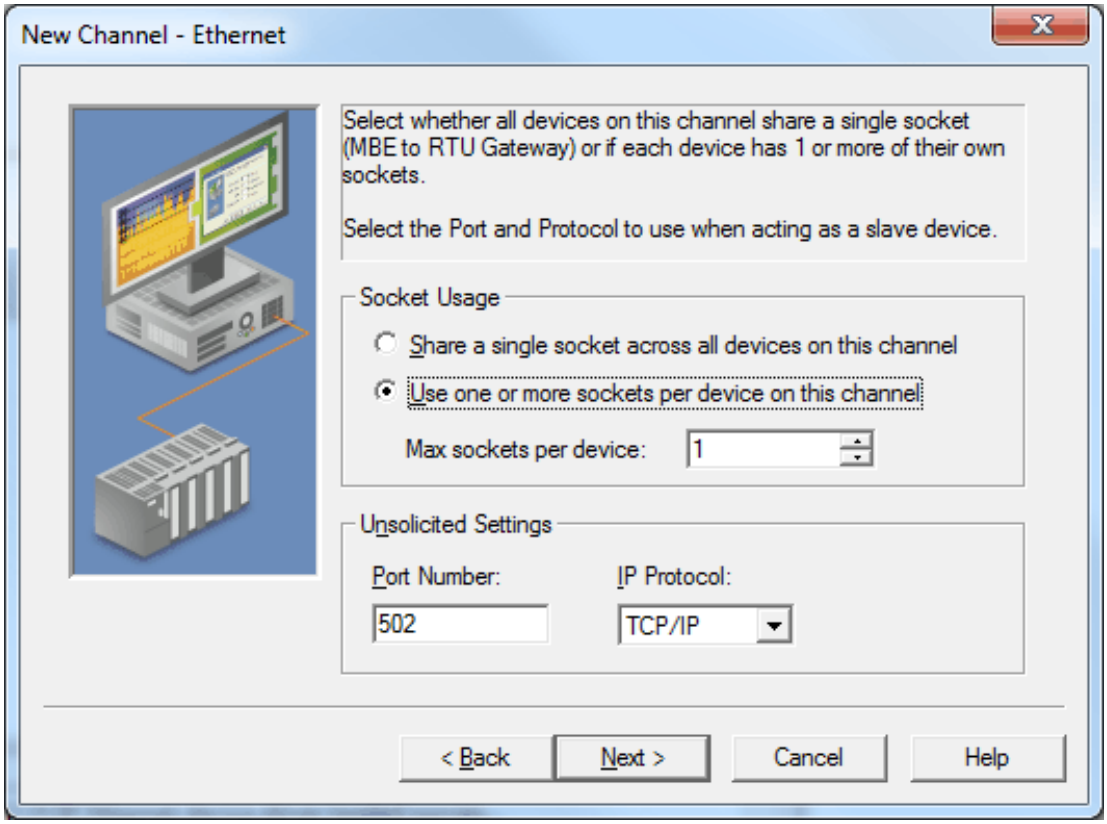
Communication Serialization

The Modbus Ethernet driver supports Communication Serialization, which specifies whether data transmissions should be limited to one channel at a time. For more information, refer to "Channel Properties - Advanced" in the server help file.

Note: When Channel Serialization is enabled, both Unsolicited communications and the "Max sockets per device" parameter will be disabled. The Mailbox Model is unavailable for Channel Serialization.

Ethernet

This dialog is used to specify channel-level Ethernet settings. Channel-level settings apply to all devices that have been configured on the channel.



Descriptions of the parameters are as follows:

- **Share a single socket across all devices on this channel:** When checked, the Modbus Ethernet driver will be forced to use only a single Windows socket for all active devices on the current channel. In this mode, the driver will use the same Windows socket for all communications. A socket will close and reopen each time the driver switches to a new target device. In this way, each device will only see a single connection.

  Notes:

  1. In some cases, it is undesirable for the Modbus Ethernet driver to maintain a connection if the device has a limited number of connections available. The target device usually has a few or even a single port available for connections. If a product like the Modbus Ethernet driver is using that port, then no other system may access the target device. This parameter is useful in these cases.

  2. The ability to put the Modbus Ethernet driver into single socket mode is important when using the Modbus Ethernet driver to talk with a Modbus-Ethernet-to-Modbus-RTU bridge product. Most of these products allow users to connect multiple RS-485 serial-based devices to a single Modbus-Ethernet-to-Modbus-RTU bridge. This parameter must be unchecked when a gateway is handling a number of serial devices.

- **Use one or more sockets per device on this channel:** When checked, the Modbus Ethernet driver will use a Windows socket for each device on the network and maintain that socket as an active connection. Because the driver does not re-establish a connection each time it reads or writes data to a given device, this provides a high level of performance. The default setting is checked.
- **Max sockets per device:** This parameter specifies the maximum number of sockets per device. The default setting is 1.
- **Port Number:** This parameter specifies the port number that the driver will use when listening for unsolicited requests. The valid range is 0 to 65535. The default setting is 502.
- **IP Protocol:** This parameter specifies the protocol that the driver will use when listening for unsolicited request. Options include User Datagram Protocol (UDP) or Transfer Control Protocol (TCP/IP). The default setting is TCP/IP.

Unsolicited Settings

When the Modbus Ethernet driver is in Master mode, it has the ability to accept unsolicited requests. The driver starts a listening thread for unsolicited data once the driver is loaded by the OPC server. This thread is global to all channels configured in the OPC server. For example, if an OPC server project has three channels defined and either setting is changed in one channel, that same change made will be made to the other two channels. The listening thread will be restarted once the change is applied. The Event Log will post an event for the restart.

Note: The Modbus Ethernet driver requires Winsock V1.1 or higher.

Device Setup

Supported Device Models

For more information on a specific device model, select a link from the list below.

Applicom

This model supports Applicom addressing syntax for Generic Modbus, TSX Premium, and TSX Quantum devices.

Ethernet to Modbus Plus Bridge

The driver also has the ability to talk to Modbus Plus devices via an Ethernet to Modbus Plus Bridge. The Device ID used should be the IP address of the bridge along with the Modbus Plus Bridge Index. For example, Bridge IP 205.167.7.12, Bridge Index 5 equates to a Device ID of 205.167.7.12.5. Consult the Modicon/Schneider Automation distributor on obtaining and setting up a MBE to MBP Bridge.

CEG

This model supports the extended block size of CEG devices.

Fluenta

This model supports the non-standard Modbus mapping of the Fluenta FGM 100/130 Flow Computer.

Instromet

This model supports the non-standard Modbus mapping of Instromet devices.

Mailbox

This model affects the way unsolicited requests are handled. By defining a mailbox device, the driver does not act like a PLC on the network. Instead, it acts as a storage area for every mailbox device that is defined. When the driver receives an unsolicited command, the driver detects the IP address the message came from and places the data in the storage area allocated for the device. If the message comes from a device with an IP address that has not been defined as a mailbox device, the message is not processed. Any client application that reads or writes to this type of device will read or write to the storage area are contained in the driver and not the physical device. For information on sending unsolicited requests to the Modbus Ethernet driver, consult the Modicon Documentation on the MSTR instruction.

Note: Modbus Mailbox does not support function code 22 (0x16). Only 0x10 (Holding Reg Write Multiple) and 0x6 (Holding Reg Write Single) are supported. Users can write to a single bit by turning off the "Use holding register bit mask writes" option, which is located in the Settings tab of Device Properties. This forces it to use the Read/Modify/Write sequence instead of directly writing to the bit. Only the Master Modbus device (not the Mailbox) has to change its setting to get this to work.

See Also: Mailbox Client Privileges for Mailbox Device Model

Modbus Master

Most projects will be configured to function as a Modbus Master. In this mode, the driver will access a physical device (such as the TSX Quantum or any other Modbus Open Ethernet compatible device).

Modbus Unsolicited

The Modbus Ethernet driver will act as a device on the network when Modbus is the selected model and is configured with a Device ID equivalent to the host machine's IP address. The driver will accept all unsolicited commands that are received and will attempt to process them as if it were just another PLC. Any Modbus master on the network can communicate with this simulated device using its IP address.

The Device ID for a slave device is specified as *YYY.YYY.YYY.YYY.XXX*. The *YYY* can either be the loopback address or the local IP address of the PC that is running the Modbus Ethernet driver. The *XXX* designates the slave's Station ID and can be in the range 0 to 255.

Multiple slave devices can have the same Station ID. In this scenario, all the devices that share the Station ID will point to one common simulated device. If the remote master requests data from a slave device (Station ID) that does not exist, then the response will contain data from station '0'. Once a slave device is created in the project, that slave is enabled and will stay enabled until the server is shut down. Changing the Station ID will enable a new slave device that will stay enabled until the server is shut down.

Addresses 1 to 65536 are implemented for output coils, input coils, internal registers, and holding registers. In Unsolicited Mode, the driver will respond to any valid request to read or write these values from external devices (Function Codes [decimal] 01, 02, 03, 04, 05, 06, 15, and 16). Furthermore, loopback (also known as Function code 08, sub code 00) has been implemented in this driver. These locations can be accessed locally by the Host PC as tags assigned to the slave device.

Note: Write Only access is not allowed for unsolicited devices.

Roxar
This model supports the non-standard Modbus mapping of the Roxar RFM Water Cut meter.

Maximum Number of Channels and Devices
The maximum number of supported channels is 256. The maximum number of supported devices is 8192.

Device ID (PLC Network Address)
The Device ID is used to specify the device IP address along with a Modbus Bridge Index on the Ethernet network.

Device IDs are specified as *<HOST>.XXX*, where *HOST* is a standard UNC/DNS name or an IP address. The *XXX* designates the Modbus Bridge Index of the device and can be in the range of 0 to 255. If no bridge is used, the index should be set to 0. Depending on the model and Device ID, a device could be configured to act as an unsolicited or master device. For more information on unsolicited mode, refer to **Modbus Unsolicited**.

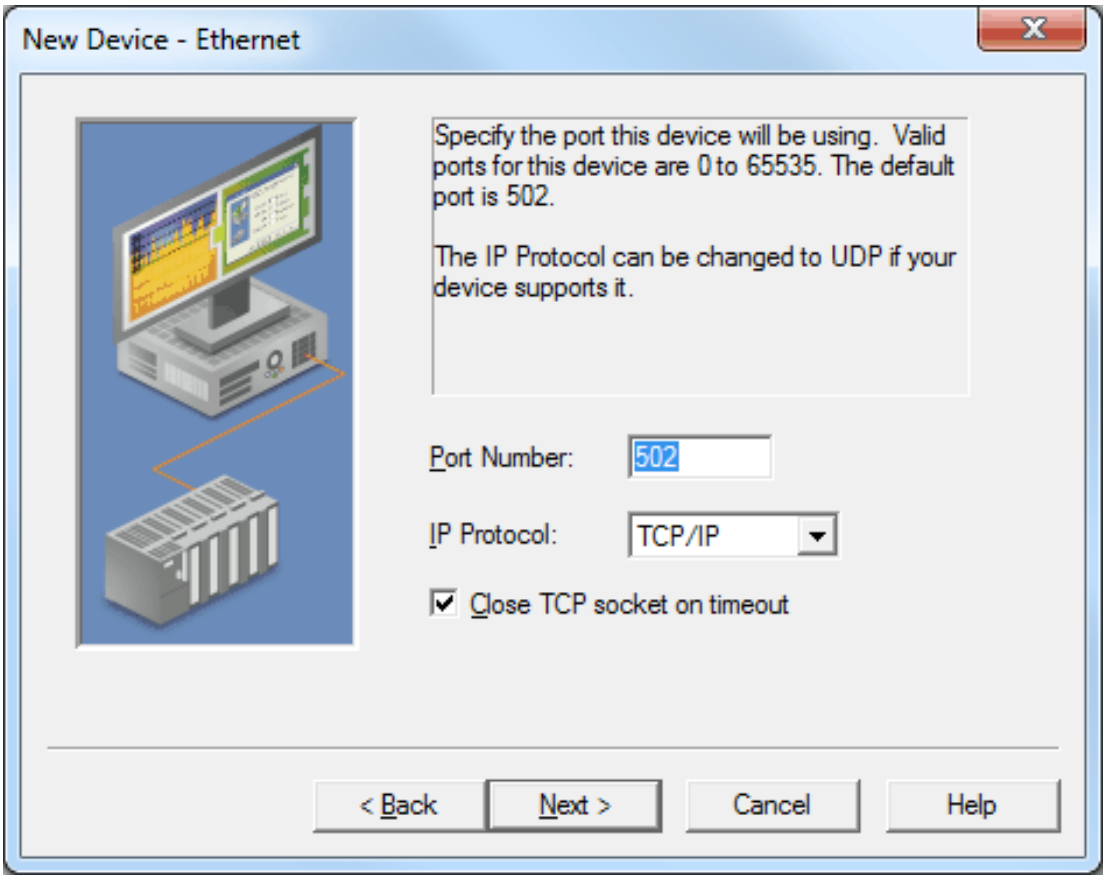
Examples

1. When requesting data from a Modicon TSX Quantum device with IP address 205.167.7.19, the Device ID should be entered as 205.167.7.19.0.
2. When requesting data from a Modbus Plus device connected to bridge index 5 of a Modbus Ethernet Bridge with an IP address of 205.167.7.50, the Device ID should be entered as 205.167.7.50.5.

See Also: **Cable Diagrams** and **Modbus Master & Modbus Unsolicited Considerations**.

Ethernet

This dialog is used to specify device-level Master/Slave solicited communications settings.



Descriptions of the parameters are as follows:

- **Port Number:** This parameter specifies the port number that the remote device is configured to use. The valid range is 0 to 65535. The default setting is 502. The Modbus Ethernet driver will use this port number when making solicited requests to a device.

  Note: If the port system tag is used, the port number setting will be changed. For more information, refer to **Driver System Tag Addresses**.
- **IP Protocol:** This parameter specifies whether the driver should connect to the remote device using the User Datagram Protocol (UDP) or Transfer Control Protocol (TCP). The master and slave settings must match. For example, if the slave's IP protocol setting is TCP/IP, then the master's IP protocol setting for that device must also be TCP/IP.

  Note: This driver requires Winsock V1.1 or higher.
- **Close TCP Socket on Timeout:** This option specifies whether the driver should close a TCP socket connection if the device does not respond within the timeout. When checked, the driver will close the TCP socket connection on timeout. When unchecked, the driver will continue to use the same TCP socket until an error is received, the physical device closes the socket, or the driver is shutdown. The default setting is checked.

  Note: The Modbus Ethernet driver will always close the socket connection on a socket error.

Settings

----- Data Access Group -----

Zero vs. One Based Addressing

If the address numbering convention for the device starts at one as opposed to zero, its value can be specified when defining the device's parameters. By default, user-entered addresses will have one subtracted when frames are constructed to communicate with a Modbus device. If the device does not 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 documentation. The default behavior follows the convention of the Modicon PLCs.

Zero vs. One Based Bit Addressing within Registers

Memory types that allow bits within Words can be referenced as a Boolean. The addressing notation for doing this is *<address>.<bit>,* where *<bit>* represents the bit number within the Word. Zero Based Bit Addressing within registers provides two ways of addressing a bit within a given Word; Zero Based and One Based. Zero Based Bit addressing within registers means that the first bit begins at 0. One Based Bit Addressing within registers means that the first bit begins at 1.

Zero Based Bit Addressing within Registers (Default Setting/Checked)

Data Type	Bit Range
Word	Bits 0–15

One Based Bit Addressing within Registers (Unchecked)

Data Type	Bit Range
Word	Bits 1–16

Holding Register Bit Mask Writes

When writing to a bit location within a holding register, the driver should only modify the bit of interest. Some devices support a special command to manipulate a single bit within a register (Function code hex 0x16 or decimal 22). If the device does not support this feature, the driver will need to perform a Read/Modify/Write operation to ensure that only the single bit is changed.

Check this box if the device supports holding register bit access. The default setting is unchecked. When checked, the driver will use function code 0x16, irrespective of the setting for **Use Modbus function 06 for single register writes**. When unchecked, the driver will use either function code 0x06 or 0x10 depending on the selection for **Use Modbus function 06 for single register writes**.

Note: When Modbus byte order is deselected, the byte order of the masks sent in the command will be Intel byte order.

Use Modbus Function 06 or 16

The Modbus driver has the option of using two Modbus protocol functions to write Holding register data to the target device. In most cases, the driver switches between these two functions based on the number of registers being written. When writing a single 16-bit register, the driver will generally use the Modbus function 06. When writing a 32-bit value into two registers, the driver will use Modbus function 16. For the standard Modicon PLC the use of either of these functions is not a problem. There are, however, a large number of third party devices that have implemented the Modbus protocol. Many of these devices support only the use of Modbus function 16 to write to Holding registers regardless of the number of registers to be written.

The **Use Modbus function 06** selection forces the driver to use only Modbus function 16 if needed. This selection is checked by default, allowing the driver to switch between 06 and 16 as needed. If a device requires all writes to be done using only Modbus function 16, uncheck this selection.

Note: For bit within word writes, the **Holding Register Bit Mask Writes** property takes precedence over this property **Use Modbus Function 06**. If **Holding Register Bit Mask Writes** is selected, then function code 0x16 is used no matter what the selection for this property. If **Holding Register Bit Mask Writes** is not selected, either function code 0x06 or 0x10 will be used for bit within word writes.

Use Modbus Function 05 or 15

The Modbus driver has the option of using two Modbus protocol functions to write output coil data to the target device. In most cases, the driver switches between these two functions based on the number of coils being

written. When writing a single coil, the driver will use the Modbus function 05. When writing an array of coils, the driver will use Modbus function 15. For the standard Modicon PLC the use of either of these functions is not a problem. There are many third party devices that have implemented the Modbus protocol, however: many of these devices only support the use of Modbus function 15 to write to output coils regardless of the number of coils to be written.

The **Use Modbus function 05** selection forces the driver to use only Modbus function 15 if needed. This selection is checked by default, allowing the driver to switch between 05 and 15 as needed. If a device requires all writes to be done using only Modbus function 15, uncheck this selection.

Mailbox Client Privileges for Mailbox Device Model

- **Read Only:** Client applications can only read from a mailbox memory map.
- **Memory Map Writes:** Client applications can read and write to the mailbox memory map.
- **Device Writes:** Client applications can only write to a device; reads are from the memory map.

----- Data Encoding Group -----

Modbus Byte Order
The byte order used by the Ethernet driver can be changed from the default Modbus byte ordering to Intel byte ordering by using this selection. This selection will be checked by default, which is the normal setting for Modbus compatible devices. If the device uses Intel byte ordering, deselecting this selection will enable the Modbus driver to properly read Intel formatted data.

First Word Low in 32-Bit Data Types
Two consecutive registers' addresses in a Modbus device are used for 32-bit data types. It can be specified whether the driver should assume the first word is the low or the high word of the 32-bit value. The default, first word low, follows the convention of the Modicon Modsoft programming software.

First DWord Low in 64-Bit Data Types
Four consecutive registers' addresses in a Modbus device are used for 64-bit data types. It can be specified whether the driver should assume the first DWord is the low or the high DWord of the 64-bit value. The default, first DWord low, follows the default convention of 32-bit data types.

Use Modicon Bit Ordering
When checked, the driver will reverse the bit order on reads and writes to registers to follow the convention of the Modicon Modsoft programming software. For example, a write to address 40001.0/1 will affect bit 15/16 in the device when this option is enabled. This option is disabled (unchecked) by default.

Note: For the following example, the 1st through 16th bit signifies either 0-15 bits or 1-16 bits depending on if the driver is set at Zero Based or One Based Bit Addressing within registers.

MSB = Most Significant Bit
LSB = Least Significant Bit

Use Modicon Bit Ordering Checked

MSB															LSB
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Use Modicon Bit Ordering Unchecked (Default Setting)

MSB															LSB
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Treat Longs as Double Precision Unsigned Decimal
When checked, the driver encodes/decodes Long and DWORD data types as values that range from 0 to 99999999. This format specifies that each word represents a value between 0 and 9999.

Values read above the specified range are not clamped, but the behavior is undefined. All read values are decoded using the following formula:
[Read Value] = HighWord * 10000 + LowWord.

Written values greater than 99999999 are clamped to the maximum value. All written values are encoded using the following formula:
Raw Data = [Written Value]/10000 + [Written Value] % 10000.

Data Encoding Options Details

The following summarizes usage of the Data Encoding options.

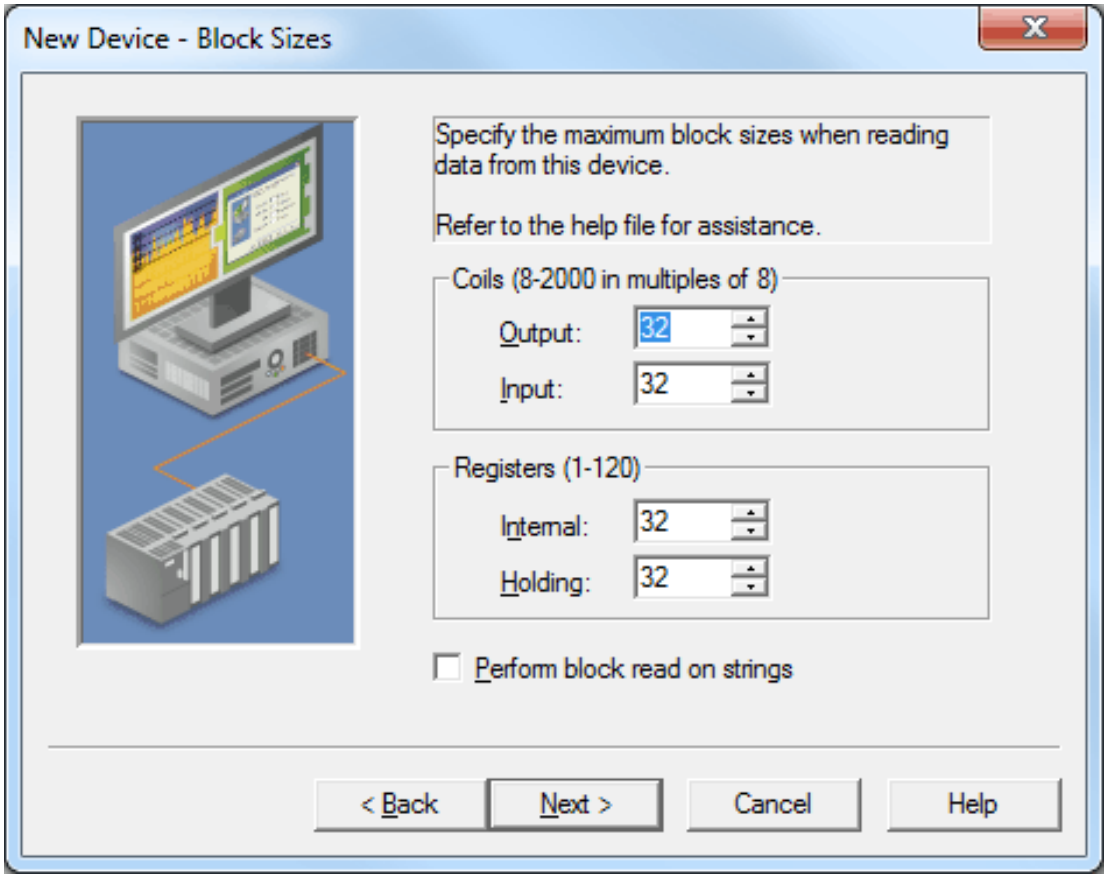
- Use default Modbus byte order option sets the data encoding of each register/16-bit value.
- First word low in 32-bit data types option sets the data encoding of each 32-bit value and each double word of a 64-bit value.
- First DWord low in 64-bit data types option sets the data encoding of each 64-bit value.

Data Types	Use Default Modbus Byte Order Applicable	First Word Low in 32-Bit Data Types Applicable	First DWord Low in 64-Bit Data Types Applicable
Word, Short, BCD	Yes	No	No
Float, DWord, Long, LBCD	Yes	Yes	No
Double	Yes	Yes	Yes

If needed, use the following information and the device's documentation to determine the correct settings of the Data Encoding options. The default settings are acceptable for the majority of Modbus devices.

Data Encoding Group Option	Data Encoding	
Use default Modbus byte order Checked	High Byte (15..8)	Low Byte (7..0)
Use default Modbus byte order Unchecked	Low Byte (7..0)	High Byte (15..8)
First word low in 32-bit data types Unchecked	High Word (31..16) High Word(63..48) of Double Word in 64-bit data types	Low Word (15..0) Low Word (47..32) of Double Word in 64-bit data types
First word low in 32-bit data types Checked	Low Word (15..0) Low Word (47..32) of Double Word in 64-bit data types	High Word (31..16) High Word (63..48) of Double Word in 64-bit data types
First DWord low in 64-bit data types Unchecked	High Double Word (63..32)	Low Double Word (31..0)
First DWord low in 64-bit data types Checked	Low Double Word (31..0)	High Double Word (63..32)

Block Sizes



Descriptions of the parameters are as follows:

- **Coil Block Sizes:** This parameter specifies the output and input coil block sizes. Coils can be read from 8 to 2000 points (bits) at a time. The default setting for both Output and Input Coils is 32.
- **Register Block Sizes:** This parameter specifies the internal and external register block sizes. From 1 to 120 standard Modbus registers (16 bit) can be read at a time. The default setting for both Internal and Holding Registers is 32.

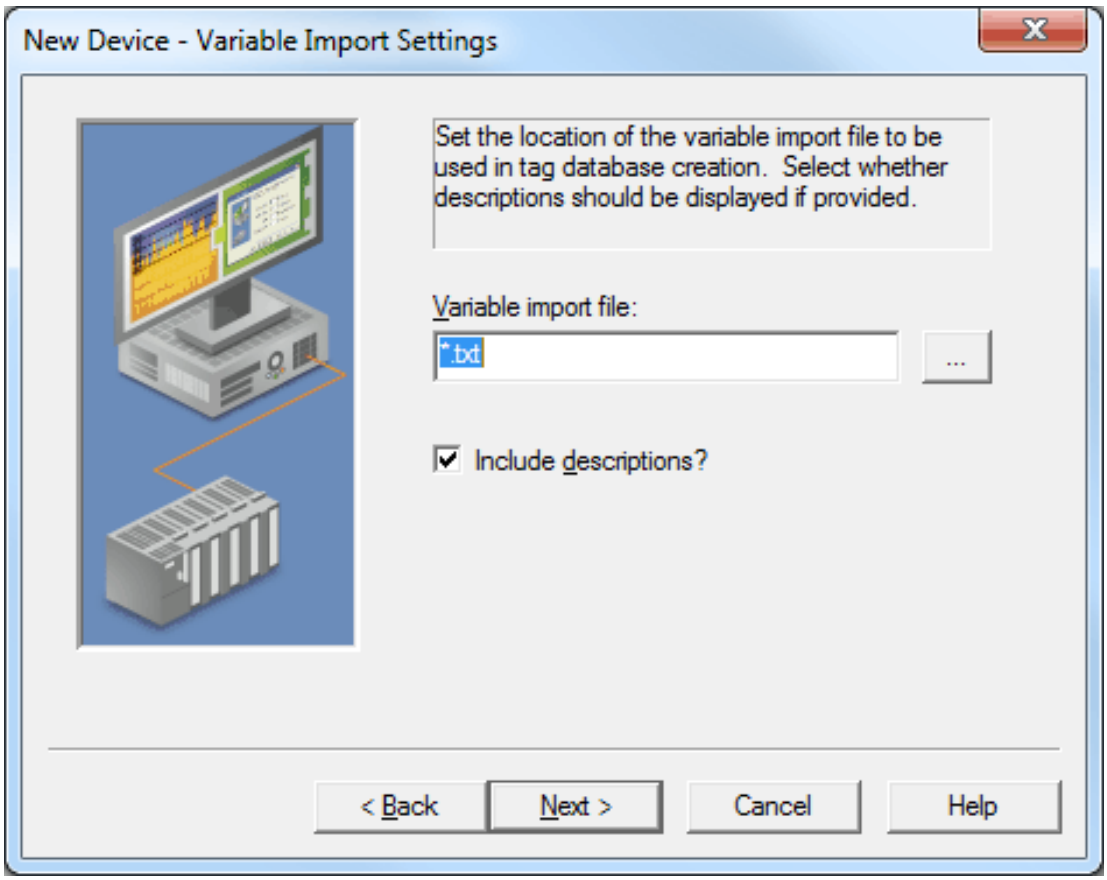
  Notes:
  1. The Instromet, Roxar, and Fluenta models (which support 32-bit and 64-bit registers) require special consideration. The Modbus protocol constrains the block size to be no larger than 256 bytes. This translates to a maximum of block size of 64 32-bit registers, or 32 64-bit registers for these models.
  2. The CEG model supports coil block sizes between 8 and 8000 in multiples of 8 and register block sizes between 1 and 500. This model must only be used with CEG devices.
  3. A "Bad address in block error" could occur if the Register Block sizes value is set above 120 and a 32 or 64-bit data type is used for any tag. To prevent this from occurring, decrease the block size value to 120.
- **Perform Block Read on Strings:** When checked, this option will block read string tags, which are normally read individually. String tags will be grouped together depending on the selected block size. Block reads can only be performed for Modbus model string tags.

Reasons to Change the Default Block Sizes

1. The device may not support block Read/Write operations of the default size. Smaller Modicon PLCs and non-Modicon devices may not support the maximum data transfer lengths supported by the Modbus Ethernet network.

2. The device may contain non-contiguous addresses. If this is the case and the driver attempts to read a block of data that encompasses undefined memory, the device will probably reject the request.

Variable Import Settings

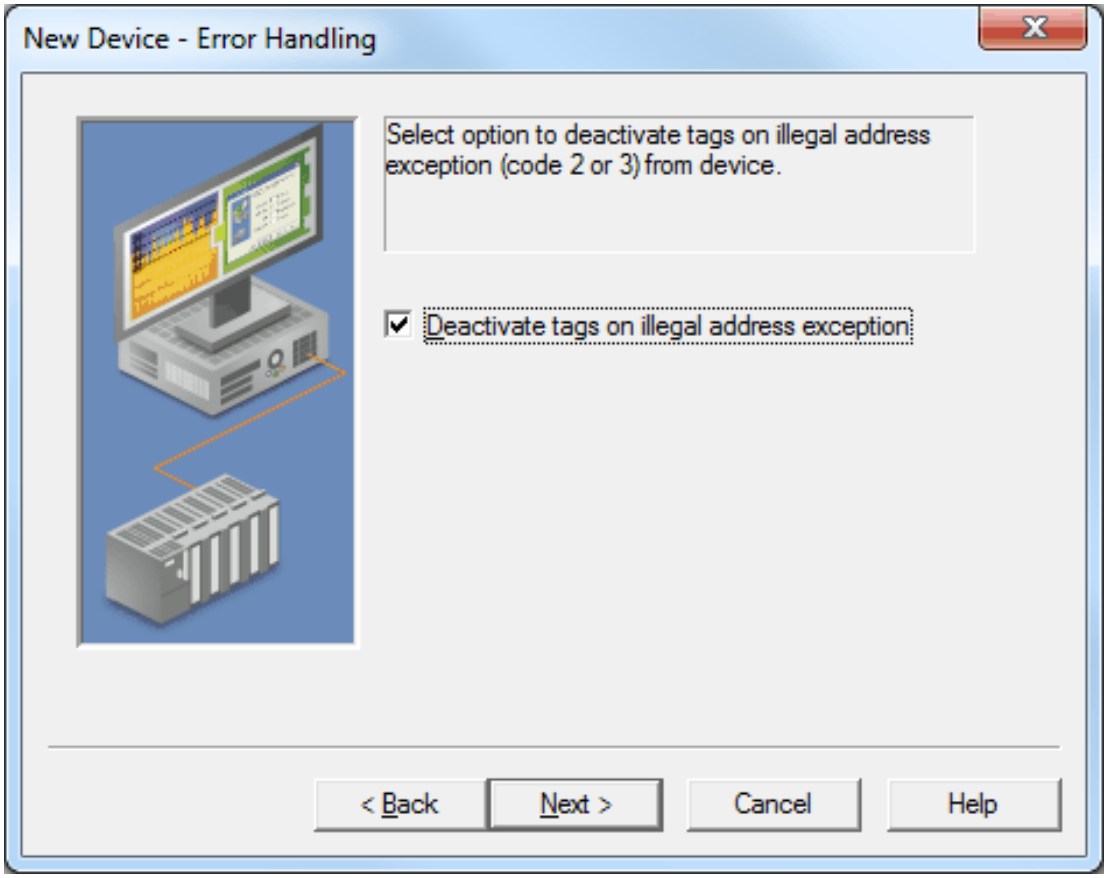


Descriptions of the parameters are as follows:

- **Variable Import File:** This parameter specifies the exact location of the Concept or ProWORX variable import file that the driver should use when the Automatic Tag Database Generation feature is enabled.
- **Display Descriptions:** When checked, this option will use imported tag descriptions (if present in file).

Note: For more information on configuring the Automatic Tag Database Generation feature (and how to create a variable import file), refer to **Automatic Tag Database Generation**.

Error Handling



Description of the parameter is as follows:

- **Deactivate tags on illegal address exception:** When checked, the driver will stop polling for a block of data if the device returns Modbus exception code 2 (illegal address) or 3 (illegal data, such as number of points) in response to a Read of that block. When unchecked, the driver will continue to poll that data block. The default setting is checked.

Unsolicited

Descriptions of the parameters are as follows:

- **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 an OPC quality of Good. This is the default condition. When checked, all tags will have an initial value of 0 and an OPC quality of Bad. The tag's quality 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 will not show Good quality until both holding registers have been written to.

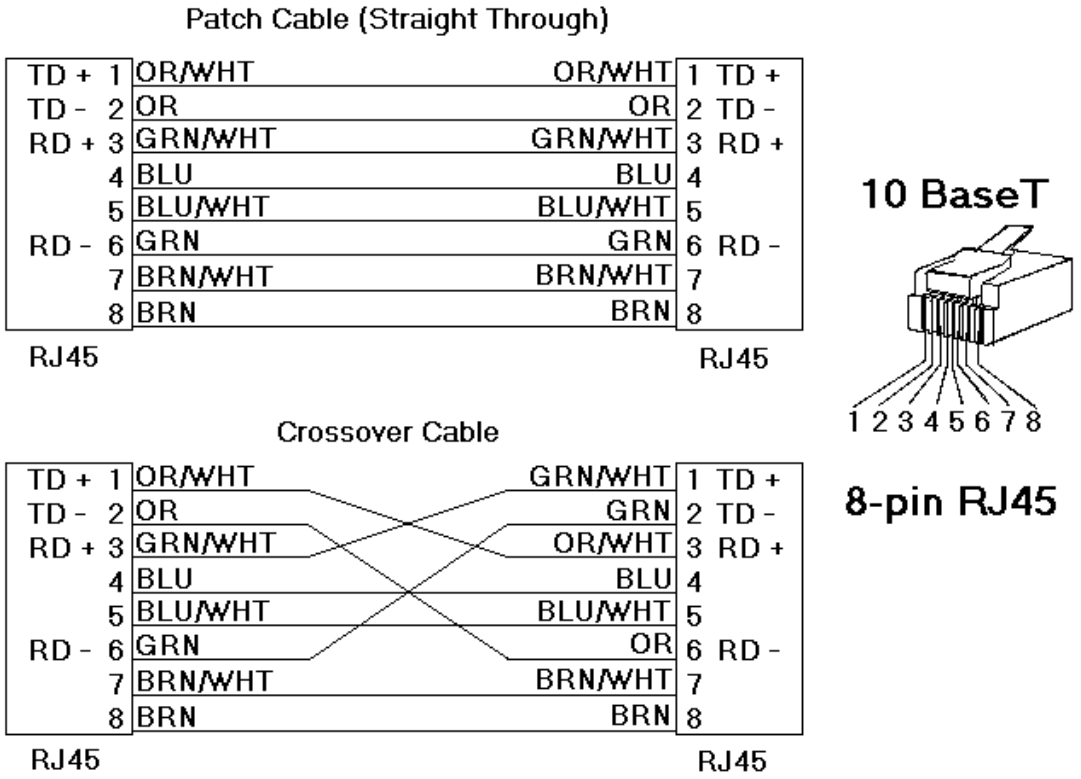
  Note: If the device is not in unsolicited mode, this option will be grayed out.
- **Communications Timeout: ___ Seconds:** The communications timeout parameter sets the amount of time the driver will wait for an incoming request before setting the device's tag quality to bad. After the timeout has occurred, the only way to reset the timeout and allow all the tags to be processed normally is to re-establish communications with the remote master or disable the communications timeout by setting it to 0. When enabled, the valid range is 1 to 64,800 seconds (18 hours).

  Note: If an incoming request comes for a slave device (Station ID) that does not exist, the request is always directed to station '0'. In this case, the timeout for a slave device with Station ID '0' will not occur even if it does not explicitly receive any remote communications for the timeout period.

Notes:

1. Unsolicited devices require the model to be Modbus, and the Device ID to be *IP_Address.yyy*, where *IP_Address* can be the local IP address of the PC running the driver. For example, 127.xxx.xxx.xxx, where *xxx*=0-255, and *yyy* (Station ID)=0-255.
2. When the first unsolicited request for a slave device is received, the Event Log will display the following informational message: "<date>__<time>__<level>__<source>__<event>". For example, "2/4/2011__4:53:10 PM Information__Modbus TCP/IP Ethernet__Created Memory for Slave Device <Slave Number>".
3. For this driver, the terms Slave and Unsolicited are used interchangeably.

Cable Diagrams



Modbus Master & Modbus Unsolicited Considerations

The following notes pertain to both Modbus Master and Modbus Unsolicited devices.

- It is not recommended that a Mailbox device and a Modbus device be on the same machine. Because a master will only get data from one of these devices at a time, it is uncertain from which it will get data.
- It is recommended that master and unsolicited devices be placed on separate channels in the server project for optimal unsolicited device tag processing.
- When a client is connected, the Device ID can only be changed if it does not result in change of mode (master to slave or slave to master) of the device. The mode is changed by changing the loopback or local IP address to a different IP address and vice versa. The loopback address and the local IP address (of the PC running the driver) indicates slave (unsolicited) mode and any other IP address indicates master mode of the device. When no client is connected, the mode can be changed in any manner (such as master to master, master to slave, slave to slave, or slave to master).

  Note: Any address in the format 127.xxx.xxx.xxx, where *xxx* is in the range 0-255 is loopback address.
- The Data Encoding Group settings must be the same in master and slave devices. For example, when a device configured as a Modbus master is communicating with the device setup as a Modbus slave.
- For this driver, the terms Slave and Unsolicited are used interchangeably.

Automatic Tag Database Generation

This driver supports the server's Automatic Tag Database Generation feature, which enables drivers to automatically create tags that access data points used by the device's ladder program. Depending on the configuration, tag generation may start automatically when the server project starts or be initiated manually at some other time. The server's Event Log will show when tag generation started, any errors that occurred while processing the variable import file, and when the process completed. For more information, refer to the server help documentation.

Although it is sometimes possible to query a device for the information needed to build a tag database, this driver must use a Variable Import File instead. Variable import files can be generated using the Concept and ProWORX device programming applications. The import file must be in semicolon delimited Concept ".txt" format, which is the default export file format of the Concept device programming application. The ProWORX programming application can also export variable data in this format. For application-specific information on creating the variable import file, refer to **Exporting Variables from Concept** and **Exporting Variables from ProWORX**.

See Also: **Importing from Custom Applications**

Importing from Custom Applications

Custom tags can be imported using the following CSV file format:

[Record Type] ; [Variable Name] ; [Data Type] ; [Address] ; [Set Value] ; [Comment] where:

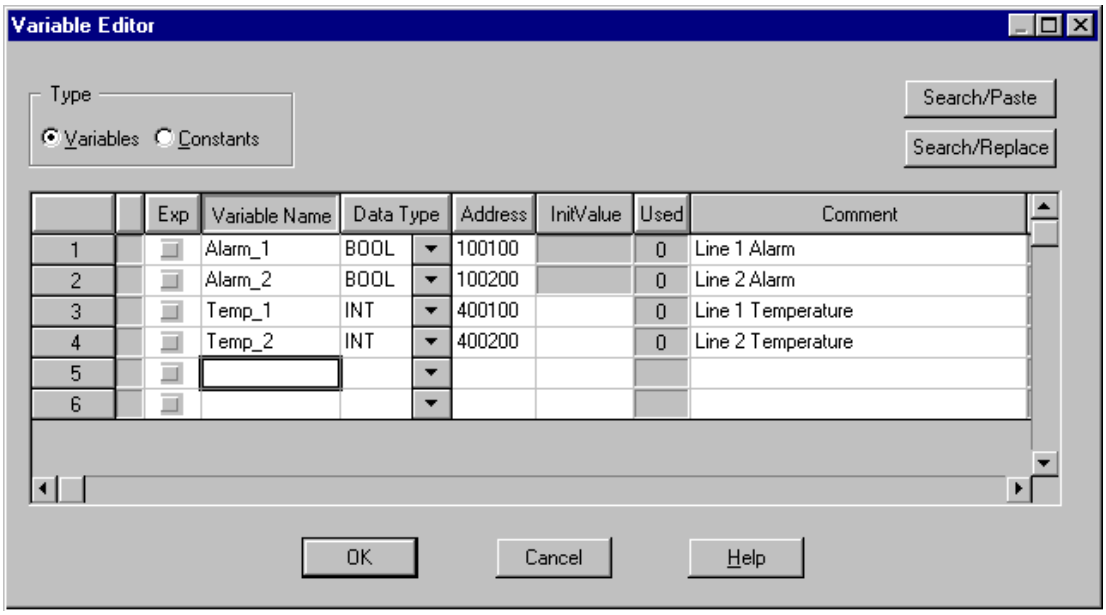
- **Record Type:** This is a flag used in the Concept software, which is another way to import tags. It can be N or E: both flags are treated the same.
- **Variable Name:** This is the name of the Static Tag in the server. It can be up to 256 characters in length.
- **Data Type:** This is the tag's data type. Supported data types are as follows:
    - BOOL
    - DINT
    - INT
    - REAL (32-bit Float)
    - UDINT
    - UINT
    - WORD
    - BYTE
    - TIME (treated as a DWord)
    - STRING
- **Address:** This is the tag's Modbus address. It can be up to 16 characters in length.
- **Set Value:** This is ignored, and should be kept blank.
- **Comment:** This is the description of the tag in the server. It can be up to 255 characters in length.

Examples

- N;Amps;WORD;40001;;Current in
- N;Volts;WORD;40003;;Volts in
- N;Temperature;REAL;40068;;Tank temp

Exporting Variables from Concept

As the ladder program is created, symbolic names for the various data points referenced using the Variable Editor can be defined. Additional symbols and constants that are not used by the ladder program can also be defined.



Note: Although Concept allows variable names to be defined that begin with an underscore, such names are not allowed by the OPC server. The driver will modify invalid imported tag names as needed and will make note of any such name changes in the server's event log.

User defined data types are not currently supported by this driver. Records in the export file containing references to such types will be ignored. The following data types are supported:

Concept Data Type	Generated Tag Data Type
Bool	Boolean
Byte	Word
Dint	Long
Int	Short
Real	Float
Time	DWord
Udint	DWord
Uint	Word
Word	Word
String	String

Notes:

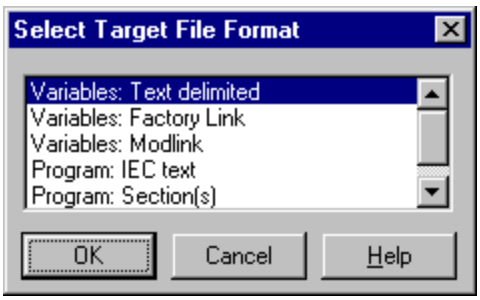
1. Unlocated variables, which are those that do not correspond to a physical address in the device, will be ignored by the driver.
2. Comments are allowed and included as the generated tag descriptions. For more information, refer to **Variable Import Settings**.

Exporting Data from Concept

Once the variables have been defined, the data must be exported from Concept.

1. To start, click **File** | **Export** and then select the **Variables: Text delimited** format.

2. Click **OK.**



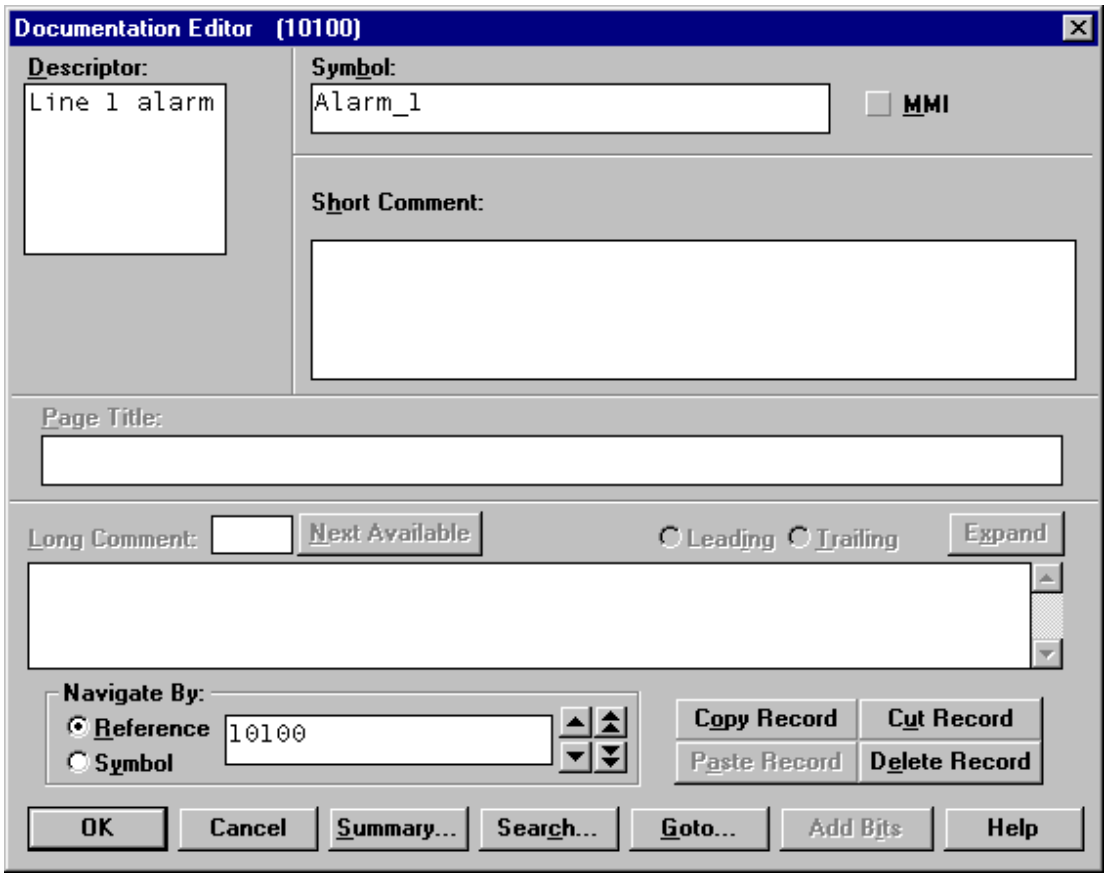
3. Specify the filter and separator settings.

   Note: Choose the filter settings as desired, but remember that this driver will only be able to read the exported data if the default semicolon separator is used.

4. Click **OK** to generate the file.

Exporting Variables from ProWORX

For ProWORX to export the necessary variable information, the **Symbols** option must be checked under **File** | **Preferences**. Symbolic names for various data points referenced can be defined while creating the ladder program by using the Document Editor.



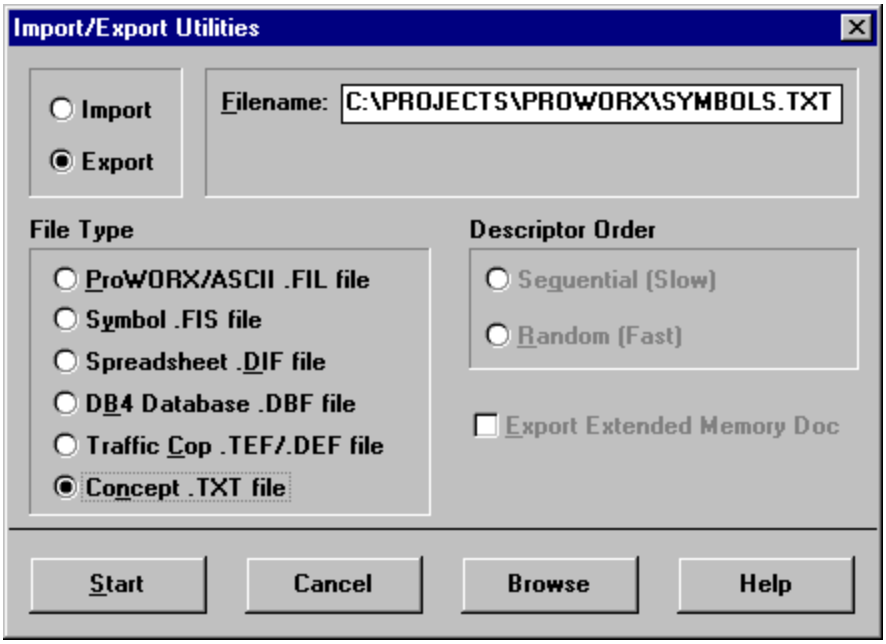
Notes:

1. Although ProWORX does not place many restrictions on variable names, the OPC server requires that tag names consist of only alphanumeric characters and underscores. It also requires that the first character cannot be an underscore. The driver will modify invalid imported tag names as needed, and any such name changes will be noted in the server's Event Log.
2. ProWORX will assign a data type of either Bool or Int to the exported variables. The driver will create tags of type Boolean and Short respectively. To generate tags with other data types, the exported file must be manually edited and use any of the supported Concept data types. For a list of supported types, refer to **Exporting Variables from Concept**.

Exporting Data From ProWorx

Once the variables have been defined, the data must be exported from ProWORX.

1. To start, select **File** | **Utilities** | **Import/Export**.
2. Next, select the **Export** and the **Concept .TXT** file formats.
3. Descriptors are allowed, and can be included as the generated tag descriptions. For more information, refer to **Variable Import Settings**.

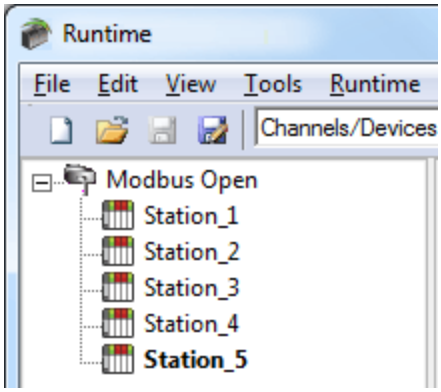


4. Click **OK** to generate the file.

Optimizing Modbus Ethernet Communications

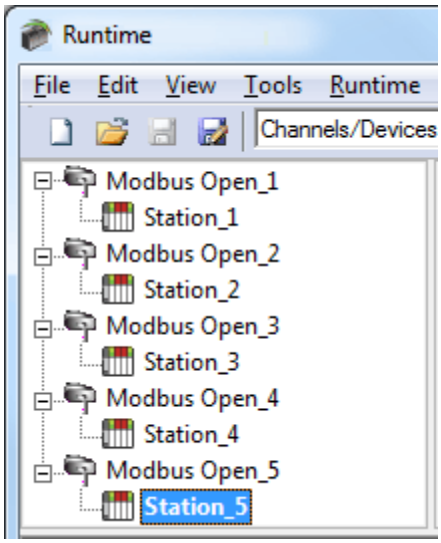
The Modbus Ethernet Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the Modbus Ethernet Driver is fast, there are a couple of guidelines that can be used to control and optimize the application and gain maximum performance.

The server refers to communications protocols like Modbus Ethernet as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must then be defined under that channel. Each of these devices represents a single Modbus controller from which data will be collected. While this approach to defining the application will provide a high level of performance, it won't take full advantage of the Modbus Ethernet Driver or the network. An example of how the application may appear when configured using a single channel is shown below.



Each device appears under a single Modbus Ethernet channel. In this configuration, the driver must move from one device to the next as quickly as possible to gather information at an effective rate. As more devices are added or more information is requested from a single device, the overall update rate begins to suffer.

If the Modbus Ethernet Driver could only define one single channel, then the example shown above would be the only option available; however, the Modbus Ethernet Driver can define up to 256 channels. Using multiple channels distributes the data collection workload by simultaneously issuing multiple requests to the network. An example of how the same application may appear when configured using multiple channels to improve performance is shown below.



Each device has now been defined under its own channel. In this new configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has 256 or fewer devices, it can be optimized exactly how it is shown here.

The performance will improve even if the application has more than 256 devices. While 256 or fewer devices may be ideal, the application will still benefit from additional channels. Although by spreading the device load across all 256 channels will cause the server to move from device to device again, it can now do so with far less devices to process on a single channel.

Block Size

Block size is another parameter that can affect the performance of the Modbus Ethernet Driver. The block size parameter is available on each device being defined (on the OPC server screen, right-click on the device, choose Properties and click the Blocks tab). The block size refers to the number of registers or bits that may be requested from a device at one time. The driver's performance can be refined by configuring the block size to 1 to 120 registers and 8 to 2000 bits.

An additional performance gain can be realized by increasing the **Maximum outstanding requests per socket** value. For more information, refer to **Ethernet**.

Data Types Description

Data Type	Description
Boolean	Single bit
Word	Unsigned 16-bit value bit 0 is the low bit bit 15 is the high bit
Short	Signed 16-bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
DWord	Unsigned 32-bit value bit 0 is the low bit bit 31 is the high bit
Long	Signed 32-bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
BCD	Two byte packed BCD Value range is 0-9999. Behavior is undefined for values beyond this range.
LBCD	Four byte packed BCD Value range is 0-99999999. Behavior is undefined for values beyond this range.
String	Null terminated ASCII string Supported on Modbus Model, includes Hi-Lo Lo-Hi byte order selection.
Double*	64-bit floating point value 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 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; that is, 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.

Applicom Addressing
CEG Addressing
Fluenta Addressing
Instromet Addressing
Mailbox Addressing
Modbus Addressing
Roxar Addressing

Driver System Tag Addressing

Port Tag

The Port system tag allows a client application to read and write the Port Number setting. Writes to this tag will cause the driver to disconnect from the device and attempt to reconnect to the specified port. It will also modify the project: the server will prompt a save on modified projects on shutdown.

Note: The Device Port Number setting is not used by the driver for unsolicited communications.

- **Address:** Port. It is not case sensitive.
- **Data Types:** Word, Short, DWord, and Long.
- **Access:** Read/Write.

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

See Also: **Ethernet**

Function Codes Description

The Function Codes displayed in the table below are supported by the Modbus and Applicom device models.

Decimal	Hexadecimal	Description
01	0x01	Read Coil Status
02	0x02	Read Input Status
03	0x03	Read Holding Registers
04	0x04	Read Internal Registers
05	0x05	Force Single Coil
06	0x06	Preset Single Register
15	0x0F	Force Multiple Coils
16	0x10	Preset Multiple Registers
22	0x16	Masked Write Register

Applicom Addressing

Applicom devices support three Applicom sub-models. For address information, select a link from the list below.

Generic Modbus
TSX Premium
TSX Quantum

Generic Modbus

All Function Codes are displayed in decimal. For more information, refer to **Function Codes Description**.

Output Coils

Address	Range	Data Type	Access	Function Code
Bxxxxx	0-65535	Boolean	Read/Write	01, 05, 15

Array Support
Arrays are supported for the output coil addresses. The syntax for declaring an array is as follows:

Bxxxxx_cols with assumed row count of 1.
Bxxxxx_rows_cols.

The base address+(rows*cols) cannot exceed 65535. The total number of coils being requested cannot exceed the output coil block size that was specified for this device.

Input Coils

Address	Range	Data Type	Access	Function Code
BIxxxxx	0-65535	Boolean	Read Only	02

Array Support
Arrays are supported for the input coil addresses. The syntax for declaring an array is as follows:

BIxxxxx_cols with assumed row count of 1.
BIxxxxx_rows_cols.

The base address+(rows*cols) cannot exceed 65535. The total number of coils being requested cannot exceed the input coil block size that was specified for the device.

Internal Registers

The default data types are shown in **bold**.

Note: For slave devices, Read Only locations are Read/Write.

Address	Range	Data Type	Access	Function Code
WIxxxxx	0-65535 0-65534 0-65532	**Word**, Short, BCD Float, DWord, Long, LBCD Double	Read Only	04
WIxxxxx.bb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read Only	04
WIxxxxx:Xbb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read Only	04
DIxxxxx	0-65534	**DWord**	Read Only	04
FIxxxxx	0-65534	**Float**	Read Only	04
WIxxxxx_S	0-65535	**Short**	Read Only	04
WIxxxxx_B	0-65535	**BCD**	Read Only	04
WIxxxxx_A**	0-65535	**String**	Read Only	04
WIxxxxx_X<1, 2, 3>***	0-65535 0-65534	**Word**, Short, BCD Float, DWord, Long, LBCD	Read Only	04
DIxxxxx_S	0-65534	**Long**	Read Only	04
DIxxxxx_B	0-65534	**LBCD**	Read Only	04
DIxxxxx_X<1, 2, 3>***	0-65534	**DWord**	Read Only	04
FIxxxxx_X<1, 2, 3>***	0-65534	**Float**	Read Only	04
M_WIxxxxx_n(H) String with HiLo Byte Order (H optional)	xxxxx=0-65535 n is string length range is 1 to 120 words	**String**	Read Only	04
M_WIxxxxx_nL String with LoHi Byte Order	xxxxx=0-65535 n is string length range is 1 to 120 words	**String**	Read Only	04

*For more information, refer to "Use 0-Based Bit Addressing" under **Settings**.
**The length of the string is 2 bytes.
***For more information, refer to **Byte Switching Suffixes**.

Array Support
Arrays are supported for the internal register addresses. The syntax for declaring an array is as follows:

WIxxxxx_cols with assumed row count of 1.
WIxxxxx_rows_cols.

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.
For all arrays, the total number of registers being requested cannot exceed the internal register block size that was specified for the device.

Holding Registers

The default data types are shown in **bold**.

Note: For slave devices, Read Only locations are Read/Write.

Address	Range	Data Type	Access	Function Code
Wxxxxx	0-65535 0-65534 0-65532	**Word**, Short, BCD Float, DWord, Long, LBCD Double	Read/Write	03, 06, 16
Wxxxxx.bb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
Wxxxxx:Xbb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
Dxxxxx	0-65534	**DWord**	Read/Write	03, 06, 16
Fxxxxx	0-65534	**Float**	Read/Write	03, 06, 16
Wxxxxx_S	0-65535	**Short**	Read/Write	03, 06, 16
Wxxxxx_B	0-65535	**BCD**	Read/Write	03, 06, 16
Wxxxxx_A**	0-65535	**String**	Read Only	03, 16
Wxxxxx_X<1, 2, 3>***	0-65535 0-65534	**Word**, Short, BCD Float, DWord, Long, LBCD	Read/Write	03, 06, 16
Dxxxxx_S	0-65534	**Long**	Read/Write	03, 06, 16
Dxxxxx_B	0-65534	**LBCD**	Read/Write	03, 06, 16
Dxxxxx_X<1, 2, 3>***	0-65534	**DWord**	Read/Write	03, 06, 16
Fxxxxx_X<1, 2, 3>***	0-65534	**Float**	Read/Write	03, 06, 16
M_Wxxxxx_n(H) String with HiLo Byte Order (H optional)	xxxxx=0-65535 n is string length range is 1 to 120 words	**String**	Read/Write	03, 16
M_Wxxxxx_nL String with LoHi Byte Order	xxxxx=0-65535 n is string length range is 1 to 120 words	**String**	Read/Write	03, 16

*For more information, refer to "Use 0-Based Bit Addressing" under **Settings**.
**The length of the string is 2 bytes.
***For more information, refer to **Byte Switching Suffixes**.

Array Support
Arrays are supported for the holding register addresses. The syntax for declaring an array using decimal addressing is as follows.

Wxxxxx_cols with assumed row count of 1.
Wxxxxx_rows_cols.

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.
For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for the device.

String Support
The Applicom 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 length of the string can be from 1 to 120 words. For more information on performing a block read on string tags, refer to **Block Sizes**.

Note: String length may be limited by the maximum size of the write request allowed by the device. If the error message "Unable to write to address <address> on device<device>: Device responded with exception code 3" is received in the server event window, the device does not support the string length. To fix this, shorten the string to a supported length.

Byte Switching Suffixes

These suffixes are used to switch the bytes that compose data of type 16-bit Word, 32-bit DWord, or 32-bit Float. The byte switching is applied after the device-level settings for Modbus Byte Order and First Word Low in 32-bit Data Types are applied. For more information, refer to **Settings**.

Byte Switching Suffixes can only be used with Internal Registers and Holding Registers. For information on the various types of switching that depend on the suffix and data type of the item, refer to the table below.

Suffix	16-Bit Data Types (Word, Short, BCD)	32-Bit Data Types (DWord, Long, LBCD, Float)
_X1	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O4 O3 O2 O1 (Byte switching)
_X2	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O3 O4 O1 O2 (Word switching)
_X3	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O2 O1 O4 O3 (Switching bytes in the words)

TSX Quantum

All Function Codes are displayed in decimal. For more information, refer to **Function Codes Description**.

Output Coils

Address	Range	Data Type	Access	Function Code
0xxxxx	1-65536	Boolean	Read/Write	01, 05, 15

Array Support
Arrays are supported for the output coil addresses. The syntax for declaring an array is as follows:

0xxxxx_cols with assumed row count of 1.
0xxxxx_rows_cols.

The base address+(rows*cols) cannot exceed 65536. The total number of coils being requested cannot exceed the output coil block size that was specified for the device.

Input Coils

Address	Range	Data Type	Access	Function Code
1xxxxx	1-65536	Boolean	Read Only	02

Array Support
Arrays are supported for the input coil addresses. The syntax for declaring an array is as follows:

1xxxxx_cols with assumed row count of 1.
1xxxxx_rows_cols.

The base address+(rows*cols) cannot exceed 65536. The total number of coils being requested cannot exceed the input coil block size that was specified for the device.

Internal Registers

The default data types are shown in **bold**.

Note: For slave devices, Read Only locations are Read/Write.

Address	Range	Data Type	Access	Function Code
3xxxxx	1-65536 1-65535 1-65533	**Word**, Short, BCD Float, DWord, Long, LBCD Double	Read Only	04
3xxxxx.bb	xxxxx=1-65536 bb=0/1-15/16*	**Boolean**	Read Only	04
3xxxxx:Xbb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read Only	04
D3xxxxx	1-65535	**DWord**	Read Only	04
F3xxxxx	1-65535	**Float**	Read Only	04
3xxxxx_S	1-65536	**Short**	Read Only	04

Address	Range	Data Type	Access	Function Code
3xxxxx_B	1-65536	**BCD**	Read Only	04
3xxxxx_A**	1-65536	**String**	Read Only	04
3xxxxx_X<1, 2, 3>***	1-65536 1-65535	**Word**, Short, BCD Float, DWord, Long, LBCD	Read Only	04
D3xxxxx_S	1-65535	**Long**	Read Only	04
D3xxxxx_B	1-65535	**LBCD**	Read Only	04
D3xxxxx_X<1, 2, 3>***	1-65535	**DWord**	Read Only	04
F3xxxxx_X<1, 2, 3>***	1-65535	**Float**	Read Only	04
M_3xxxxx_n(H) String with HiLo Byte Order (H optional)	xxxxx=1-65536 n is string length range is 1 to 120 words	**String**	Read Only	04
M_3xxxxx_nL String with LoHi Byte Order	xxxxx=1-65536 n is string length range is 1 to 120 words	**String**	Read Only	04

*For more information, refer to "Use 0-Based Bit Addressing" under **Settings**.
**The length of the string is 2 bytes.
***For more information, refer to **Byte Switching Suffixes**.

Array Support
Arrays are supported for the internal register addresses. The syntax for declaring an array is as follows:

3xxxxx_cols with assumed row count of 1.
3xxxxx_rows_cols.

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.
For all arrays, the total number of registers being requested cannot exceed the internal register block size that was specified for the device.

Holding Registers

The default data types are shown in **bold**.

Note: For slave devices, Read Only locations are Read/Write.

Address	Range	Data Type	Access	Function Code
4xxxxx	1-65536 1-65535 1-65533	**Word**, Short, BCD Float, DWord, Long, LBCD Double	Read/Write	03, 06, 16
4xxxxx.bb	xxxxx=1-65536 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
4xxxxx:Xbb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
D4xxxxx	1-65535	**DWord**	Read/Write	03, 06, 16
F4xxxxx	1-65535	**Float**	Read/Write	03, 06, 16
4xxxxx_S	1-65536	**Short**	Read/Write	03, 06, 16
4xxxxx_B	1-65536	**BCD**	Read/Write	03, 06, 16
4xxxxx_A**	1-65536	**String**	Read Only	03, 16
4xxxxx_X<1, 2, 3>***	1-65536 1-65535	**Word**, Short, BCD Float, DWord, Long, LBCD	Read/Write	03, 06, 16
D4xxxxx_S	1-65535	**Long**	Read/Write	03, 06, 16
D4xxxxx_B	1-65535	**LBCD**	Read/Write	03, 06, 16
D4xxxxx_X<1, 2, 3>***	1-65535	**DWord**	Read/Write	03, 06, 16
F4xxxxx_X<1, 2, 3>***	1-65535	**Float**	Read/Write	03, 06, 16
M_4xxxxx_n(H) String with HiLo	xxxxx=1-65536 n is string length	**String**	Read/Write	03, 16

Address	Range	Data Type	Access	Function Code
Byte Order (H optional)	range is 1 to 120 words			
M_4xxxxx_nL String with LoHi Byte Order	xxxxx=1-65536 n is string length range is 1 to 120 words	**String**	Read/Write	03, 16

*For more information, refer to "Use 0-Based Bit Addressing" under **Settings**.
**The length of the string is 2 bytes.
***For more information, refer to **Byte Switching Suffixes**.

Array Support
Arrays are supported for the holding register addresses. The syntax for declaring an array using decimal addressing is as follows.

4xxxxx_cols with assumed row count of 1.
4xxxxx_rows_cols.

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.
For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for the device.

String Support
The Applicom 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 length of the string can be from 1 to 120 words. For information on performing a block read on string tags, refer to **Block Sizes**.

Note: String length may be limited by the maximum size of the write request allowed by the device. If the error message "Unable to write to address <address> on device <device>: Device responded with exception code 3" is received in the server event window, the device does not support the string length. To fix this, shorten the string to a supported length.

Byte Switching Suffixes
These suffixes are used to switch the bytes that compose data of type 16-bit Word, 32-bit DWord, or 32-bit Float. The byte switching is applied after the device-level settings for Modbus Byte Order and First Word Low in 32-bit Data Types are applied. For more information, refer to **Settings**.

Byte Switching Suffixes can only be used with Internal Registers and Holding Registers. For information on the various types of switching that depend on the suffix and data type of the item, refer to the table below.

Suffix	16-Bit Data Types (Word, Short, BCD)	32-Bit Data Types (DWord, Long, LBCD, Float)
_X1	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O4 O3 O2 O1 (Byte switching)
_X2	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O3 O4 O1 O2 (Word switching)
_X3	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O2 O1 O4 O3 (Switching bytes in the words)

TSX Premium
All Function Codes are displayed in decimal. For more information, refer to **Function Codes Description**.

Output Coils

Address	Range	Data Type	Access	Function Code
%MXxxxxx	0-65535	Boolean	Read/Write	01, 05, 15
%Mxxxxx	0-65535	Boolean	Read/Write	01, 05, 15

Array Support
Arrays are supported for the output coil addresses. The syntax for declaring an array is as follows:

%MXxxxxx_cols with assumed row count of 1.
%MXxxxxx_rows_cols.

The base address+(rows*cols) cannot exceed 65535. The total number of coils being requested cannot exceed the output coil block size that was specified for the device.

Holding Registers

The default data types are shown in **bold**.

Note: For slave devices, Read Only locations are Read/Write.

Address	Range	Data Type	Access	Function Code
%MWxxxxx	0-65535 0-65534 0-65532	**Word**, Short, BCD Float, DWord, Long, LBCD Double	Read/Write	03, 06, 16
%MWxxxxx.bb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
%MWxxxxx:Xbb	xxxxx=0-65535 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
%DWxxxxx or %MDxxxxx	0-65534	**DWord**	Read/Write	03, 06, 16
%FWxxxxx or %MFxxxxx	0-65534	**Float**	Read/Write	03, 06, 16
%MWxxxxx_S	0-65535	**Short**	Read/Write	03, 06, 16
%MWxxxxx_B	0-65535	**BCD**	Read/Write	03, 06, 16
%MWxxxxx_A**	0-65535	**String**	Read Only	03, 16
%MWxxxxx_X<1, 2, 3>***	0-65535 0-65534	**Word**, Short, BCD Float, DWord, Long, LBCD	Read/Write	03, 06, 16
%DWxxxxx_S	0-65534	**Long**	Read/Write	03, 06, 16
%DWxxxxx_B	0-65534	**LBCD**	Read/Write	03, 06, 16
%DWxxxxx_X<1, 2, 3>*** or %MDxxxxx_X<1, 2, 3>***	0-65534	**DWord**	Read/Write	03, 06, 16
%FWxxxxx_X<1, 2, 3>*** or %MFxxxxx_X<1, 2, 3>***	0-65534	**Float**	Read/Write	03, 06, 16
M_%MWxxxxx_n(H) String with HiLo Byte Order (H optional)	xxxxx=0-65535 n is string length range is 1 to 120 words	**String**	Read/Write	03, 16
M_%MWxxxxx_nL String with LoHi Byte Order	xxxxx=0-65535 n is string length range is 1 to 120 words	**String**	Read/Write	03, 16

*For more information, refer to "Use 0-Based Bit Addressing" under **Settings**.
**The length of the string is 2 bytes.
***For more information, refer to **Byte Switching Suffixes**.

Array Support
Arrays are supported for the holding register addresses. The syntax for declaring an array using decimal addressing is as follows:

%MWxxxxx_cols with assumed row count of 1.
%MWxxxxx_rows_cols.

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.
For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for the device.

String Support
The Applicom 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 length of the string can be from 1 to 120 words. For more information on performing block read on string tags, refer to **Block Sizes**.

Note: String length may be limited by the maximum size of the write request allowed by the device. If the error message "Unable to write to address <address> on device<device>: Device responded with exception code 3" is received in the server event window, the device does not support the string length. To fix this, shorten the string to a supported length.

Byte Switching Suffixes

These suffixes are used to switch the bytes that compose data of type 16-bit Word, 32-bit DWord, or 32-bit Float. The byte switching is applied after the device-level settings for Modbus Byte Order and First Word Low in 32-bit Data Types are applied. For more information, refer to **Settings**.

Byte Switching Suffixes can only be used with Internal Registers and Holding Registers. For information on the various types of switching that depend on the suffix and data type of the item, refer to the table below.

Suffix	16-Bit Data Types (Word, Short, BCD)	32-Bit Data Types (DWord, Long, LBCD, Float)
_X1	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O4 O3 O2 O1 (Byte switching)
_X2	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O3 O4 O1 O2 (Word switching)
_X3	O1 O2 -> O2 O1 (Byte switching)	O1 O2 O3 O4 -> O2 O1 O4 O3 (Switching bytes in the words)

CEG Addressing

Addressing for the CEG device model is the same as that for the Modbus device model. For more information, refer to **Modbus Addressing**.

Fluenta Addressing

The default data types are shown in **bold**.

Address	Range	Data Type	Access
System	400000-409999	**Float**, Double	Read/Write
Output	410000-410999 420000-420999 430000-430999	**Float**, Double	Read Only
User	411000-411999 421000-421999 431000-431999	**Float**, Double	Read/Write
Service	412000-412999 422000-422999 432000-432999	**Float**, Double	Read/Write
Accumulation	413000-413999 423000-423999 433000-433999	**Float**, Double	Read Only

Instromet Addressing

The default data types are shown in **bold**.

Address	Range	Data Type	Access
Short Integers	400000-400199	**Word**, Short	Read Only
Long Integers	400200-400399	**DWord**, Long	Read Only
Floats	400400-400599	**Float**	Read Only

Mailbox Addressing

The default data types are shown in **bold**.

Decimal Addressing

Address	Range	Data Type	Access
4xxxxx	1-65536	Word, Short, BCD	Read/Write
4xxxxx.bb	xxxxx=1-65536 bb=0-15	**Boolean**	Read/Write
4xxxxx	1-65535	Float, DWord, Long, LBCD	Read/Write

Hexadecimal Addressing

Address	Range	Data Type	Access
H4yyyyy	1-10000	Word, Short, BCD	Read/Write
H4yyyyy.c	yyyyy=1-10000 c=0-F	**Boolean**	Read/Write
H4yyyy	1-FFFF	Float, DWord, Long, LBCD	Read/Write

Note: Modbus Mailbox does not support function code 22 (0x16). Only 0x10 (Holding Reg Write Multiple) and 0x6 (Holding Reg Write Single) are supported. It is possible to write to a single bit by turning off **Use holding register bit mask writes** in Device Properties under the settings tab. This forces it to use the Read/Modify/Write sequence instead of directly writing to the bit. Only the **Master Modbus** device (not the Mailbox) has to change its setting to get this to work.

Arrays

Arrays are also supported for the holding register addresses. The syntax for declaring an array (using decimal addressing) is as follows:

4xxxx[cols] with assumed row count of 1.
4xxxx[rows][cols].

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.

For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for this device.

Modbus Addressing

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

5-Digit Addressing vs. 6-Digit Addressing

In Modbus addressing, the first digit of the address specifies the primary table. The remaining digits represent the device's data item. The maximum value of the data item is a two-byte unsigned integer (65,535). Internally, this driver requires six digits to represent the entire address table and item. It is important to note that many Modbus devices may not support the full range of the data item. To avoid confusion when entering an address for such a device, this driver "pads" the address (adds a digit) according to what was entered in the address field. If a primary table type is followed by up to 4 digits (example: 4x, 4xx, 4xxx or 4xxxx), the address stays at or pads, with extra zeroes, to five (5) digits. If a primary table type is followed by five (5) digits (example: 4xxxxx), the address does not change. Internally, addresses entered as 41, 401, 4001, 40001 or 400001 are all equivalent representations of an address specifying primary table type 4 and data item 1.

Primary Table	Description
0	Output Coils
1	Input Coils
3	Internal Registers
4	Holding Registers

Modbus Addressing in Decimal Format

The Function Codes are displayed in decimal. For more information, refer to **Function Codes Description**.

Address Type	Range	Data Type	Access*	Function Codes
Output Coils	000001-065536	Boolean	Read/Write	01, 05, 15
Input Coils	100001-165536	Boolean	Read Only	02
Internal Registers	300001-365536 300001-365535 300001-365533 xxxxx=1-65536 bb=0/1-15/16**	**Word**, Short, BCD Float, DWord, Long, LBCD Double **Boolean**	Read Only Read Only Read Only Read Only	04 04 04 04

Address Type	Range	Data Type	Access*	Function Codes
	300001.2H-365536.240H***	String	Read Only	04
	300001.2L-365536.240L***	String	Read Only	04
Holding Registers	400001-465536	**Word**, Short, BCD	Read/Write	03, 06, 16
	400001-465535	Float, DWord, Long, LBCD	Read/Write	03, 06, 16
	400001-465533	Double	Read/Write	03, 06, 16
	xxxxx=1-65536 bb=0/1-15/16*	**Boolean**	Read/Write	03, 06, 16, 22
	400001.2H-465536.240H***	**String**	Read/Write	03, 16
	400001.2L-465536.240L***	**String**	Read/Write	03, 16

*For slave devices, Read Only locations are Read/Write.
For more information, refer to "Zero vs. One Based Addressing" in **Settings.
***.Bit is string length, range 2 to 240 bytes.

Modbus Addressing in Hexadecimal Format

Address Type	Range	Data Type	Access*
Output Coils	H000001-H010000	Boolean	Read/Write
Input Coils	H100001-H110000	Boolean	Read Only
Internal Registers	H300001-H310000	**Word**, Short, BCD	Read Only
	H300001-H30FFFF	Float, DWord, Long, LBCD	Read Only
	H300001-H30FFFD	Double	Read Only
	yyyyy=1-10000 cc=0/1-F/10	**Boolean**	Read Only
	H300001.2H-H3FFFF.240H**	String	Read Only
	H300001.2L-H3FFFF.240L**	String	Read Only
Holding Registers	H400001-H410000	**Word**, Short, BCD	Read/Write
	H400001-H40FFFF	Float, DWord, Long, LBCD	Read/Write
	H400001-H40FFFD	Double	Read/Write
	yyyyy=1-10000 cc=0/1-F/10	**Boolean**	Read/Write
	H400001.2H-H4FFFF.240H	**String**	Read/Write
	H400001.2L-H4FFFF.240L	**String**	Read/Write

*For slave devices, Read Only locations are Read/Write.
**.Bit is string length, range 2 to 240 bytes.

Packed Coils

The Packed Coil address type allows access to multiple consecutive coils as an analog value. This feature is available for both Input Coils and Output Coils when in Polled Mode only. It is not available to devices that are configured to access the unsolicited memory map or that are in Mailbox Mode. The decimal syntax is 0*xxxxx#nn*, where:

- *xxxxx* is the address of the first coil (with a range of 000001-065521).
- *nn* is the number of coils that will be packed into an analog value (with a range of 01-16).

The hexadecimal syntax is H0*yyyyy#nn*, where:

- *yyyyy* is the address of the first coil (with a range of H000001-H000FFF1).
- *nn* is the number of coils that will be packed into an analog value (with a range of 01-16).

Notes:

1. The only valid data type is Word. Output Coils have Read/Write access, whereas Input Coils have Read Only access. In decimal addressing, Output Coils support Function Codes 01 and 15, whereas Input Coils support Function Code 02.
2. The bit order will be such that the start address will be the Least Significant Bit (LSB) of analog value.

Write Only Access

All Read/Write addresses may be set as Write Only by prefixing a "W" to the address such as "W40001", which will prevent the driver from reading the register at the specified address. Any attempts by the client to read a Write Only tag will result in obtaining the last successful write value to the specified address. If no successful writes have occurred, then the client will receive 0/NULL for numeric/string values for an initial value.

Caution: Setting the Write Only tags client access privileges to Read Only will cause writes to these tags to fail and the client to always receive 0/NULL for numeric/string values.

Mailbox Mode

Only Holding Registers are supported in Mailbox Mode. When read from a client, the data is read locally from a cache, not from a physical device. When written to from a client, the data is written to both the local cache and the physical device as determined by the Device ID routing path. For more information, refer to **Mailbox Mode**.

Note: The Double data type is not supported.

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. Appending either an "H" or "L" to the address specifies the byte order.

Note: For more information on performing block reads on string tags for the Modbus model, refer to **Block Sizes**.

Examples

1. To address a string starting at 40200 with a length of 100 bytes and HiLo byte order, enter "40200.100H".
2. To address a string starting at 40500 with a length of 78 bytes and LoHi byte order, enter "40500.78L".

Note: String length may be limited by the maximum size of the write request allowed by the device. If the error message "Unable to write to address <address> on device<device>: Device responded with exception code 3" is received in the server event window, the device did not like the length of the string. If possible, try shortening the string.

Array Support

Arrays are supported both for internal and holding register locations (including all data types except Boolean and String) and for input and output coils (Boolean data types). There are two ways to address an array. The following examples apply to holding registers:

4*xxxx* [rows] [cols]
4*xxxx* [cols] with assumed row count of 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. For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for this device.

Roxar Addressing

The default data types are shown in **bold**.

Address	Range	Data Type	Access
Short Integers	403000-403999	**Word**, Short	Read/Write
Floats	407000-407999	**Float**	Read/Write
Floats	409000-409999	**Float**	Read Only

Statistics Items

Statistical items use data collected through additional diagnostics information, which is not collected by default. To use statistical items, Communication Diagnostics must be enabled. To enable Communication Diagnostics, right-click on the channel in the Project View and click **Properties | Enable Diagnostics**. Alternatively, double-click on the channel and select **Enable Diagnostics**.

Channel-Level Statistics Items

The syntax for channel-level statistics items is *<channel>._Statistics*.

Note: Statistics at the channel level are the sum of those same items at the device level.

Item	Data Type	Access	Description
_CommFailures	DWord	Read/Write	The total number of times communication has failed (or has run out of retries).
_ErrorResponses	DWord	Read/Write	The total number of valid error responses received.
_ExpectedResponses	DWord	Read/Write	The total number of expected responses received.
_LastResponseTime	String	Read Only	The time at which the last valid response was received.
_LateData	DWord	Read/Write	The total number of times that a driver tag's data update occurred later than expected (based on the specified scan rate).
_MsgResent	DWord	Read/Write	The total number of messages sent as a retry.
_MsgSent	DWord	Read/Write	The total number of messages sent initially.
_MsgTotal	DWord	Read Only	The total number of messages sent (both _MsgSent + _MsgResent).
_PercentReturn	Float	Read Only	The proportion of expected responses (Received) to initial sends (Sent) as a percentage.
_PercentValid	Float	Read Only	The proportion of total valid responses received (_TotalResponses) to total requests sent (_MsgTotal) as a percentage.
_Reset	Bool	Read/Write	Resets all diagnostic counters. Writing to the _Reset Tag causes all diagnostic counters to be reset at this level.
_RespBadChecksum*	DWord	Read/Write	The total number of responses with checksum errors.
_RespTimeouts	DWord	Read/Write	The total number of messages that failed to receive any kind of response.
_RespTruncated	DWord	Read/Write	The total number of messages that received only a partial response.
_TotalResponses	DWord	Read Only	The total number of valid responses received (_ErrorResponses + _ ExpectedResponses).

* The _RespBadChecksum statistic is not implemented; packet checksums are handled by the TCP protocol.

Statistical items are not updated in simulation mode *(see device general properties)*.

Device-Level Statistics Items

The syntax for device-level statistics items is *<channel>.<device>._Statistics*.

Item	Data Type	Access	Description
_CommFailures	DWord	Read/Write	The total number of times communication has failed (or has run out of retries).
_ErrorResponses	DWord	Read/Write	The total number of valid error responses received.
_ExpectedResponses	DWord	Read/Write	The total number of expected responses received.
_LastResponseTime	String	Read Only	The time at which the last valid response was received.
_LateData	DWord	Read/Write	The total number of times that a driver tag's data update occurred later than expected (based on the specified scan rate).
_MsgResent	DWord	Read/Write	The total number of messages sent as a retry.
_MsgSent	DWord	Read/Write	The total number of messages sent initially.
_MsgTotal	DWord	Read Only	The total number of messages sent (both _MsgSent + _MsgResent).
_PercentReturn	Float	Read Only	The proportion of expected responses (Received) to initial sends (Sent) as a percentage.
_PercentValid	Float	Read Only	The proportion of total valid responses received (_TotalResponses) to total requests sent (_MsgTotal) as a percentage.
_Reset	Bool	Read/Write	Resets all diagnostic counters. Writing to the _Reset Tag causes all diagnostic counters to be reset at this level.
_RespBadChecksum*	DWord	Read/Write	The total number of responses with checksum errors.
_RespTimeouts	DWord	Read/Write	The total number of messages that failed to receive any kind of response.
_RespTruncated	DWord	Read/Write	The total number of messages that received only a partial response.
_TotalResponses	DWord	Read Only	The total number of valid responses received (_ErrorResponses + _ ExpectedResponses).

* The _RespBadChecksum statistic is not implemented; packet checksums are handled by the TCP protocol.

Statistical items are not updated in simulation mode *(see device general properties)*.

Error Descriptions

The following classes of messages may be generated. Click on a link for a list of messages.

Address Validation

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

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

Error Type:
Warning

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

Solution:
Verify that the address is correct; if it is not, re-enter it 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 statically is requesting an array size that is too large for the address type or block size of the driver.

Solution:
Re-enter the address in the client application to specify either 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 statically 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.

Data Type <type> is not valid for device address <address>.

Error Type:
Warning

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

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

Device address <address> contains a syntax error.

Error Type:
Warning

Possible Cause:
An invalid tag address has been specified in a dynamic request.

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

Device address <address> is not supported by model <model name>.

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically references a location that is valid for the communications protocol but not supported by the target device.

Solution:
Verify that the address is correct; if it is not, re-enter it in the client application. Verify also that the selected model name for the device is correct.

Device address <address> is read only.

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically 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.

Missing address

Error Type:
Warning

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

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

Device Status Messages

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

All channels are subscribed to a virtual network, stopping unsolicited communication.
Device <device name> is not responding.
Failed to resolve host <host name> on device <device name>.
Modbus TCP/IP Ethernet channel <channel name> is in a virtual network, all devices reverted to use one socket per device.
Starting unsolicited communication using TCP protocol through port <port>.

All channels are subscribed to a virtual network, stopping unsolicited communication.

Error Type:
Information

Possible Cause:
Channel Serialization was enabled for all Modbus Ethernet channels.

Solution:
To enable unsolicited communications, add at least one channel that is not in a virtual network.

Note:
Unsolicited communications will be disabled when all Modbus Ethernet channels are in a virtual network.

Starting unsolicited communication using TCP protocol through port <port>.

Error Type:
Information

Possible Cause:
Channel serialization has been disabled on at least one channel.

Solution:
N/A

Device <device name> is not responding.

Error Type:
Serious

Possible Cause:

1. The connection between the device and the Host PC is broken.
2. The communication parameters for the 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 that the specified communication parameters match those of the device.
3. Verify that 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.

Failed to resolve host <host name> on device <device name>.

Error Type:
Fatal

Possible Cause:

The device is configured to use a DNS host name rather than an IP address. The host name cannot be resolved by the server to an IP address.

Solution:
Verify that the device is online and registered with the domain.

Modbus TCP/IP Ethernet channel <channel name> is in a virtual network, all devices reverted to use one socket per device.

Error Type:
Information

Possible Cause:
The channel that contains the device was configured to use channel serialization.

Solution:
If more than one socket is required per device, disable channel serialization.

Note:
Channels/devices that are in a virtual network can only be configured to use one socket per device.

Socket error <code> occurred on <device name>. Operation <operation name> failed because <reason>.

Error Type:
Warning

Possible Cause:
Communication with <device name> failed during the specified socket operation. Possible operations include:

- Connect
- Wait for send data (test socket before sending)
- Send
- Wait for receive data (test socket before receiving)
- Receive

Both error <code> and detailed <reason> are provided by the operating system. This error is posted when the <device name> is in an error state (_Error is true).

Solution:
Follow the guidance in the <reason>, which details why the error occurred and suggests a remedy when appropriate.

Unable to bind to adapter: <network adapter name>. Connect failed.

Error Type:
Warning

Possible Cause:
Since the specified network adapter cannot be located in the system device list, it cannot be bound to for communications. This usually occurs when a project is moved from one PC to another (and when the project specifies a network adapter rather than using the default). The server falls back to the default adapter.

Solution:
Change the Network Adapter property to Default (or select a new adapter) and then save the project and retry.

Unable to create a socket connection for device <device name>.

Error Type:
Warning

Possible Cause:
The server was unable to establish a TCP/IP socket connection to the specified device. It will continue to attempt connection.

Solution:

1. Verify that the device is online.
2. Verify that the device IP is within the subnet of the IP to which the server is bound. Alternatively, verify that a valid gateway is available that allows a connection the other network.

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

Error Type:
Serious

Possible Cause:

1. The named device may not be connected to the network.
2. The named device may have been assigned an incorrect network ID.
3. The named device is not responding to write requests.
4. The address does not exist in the PLC.

Solution:

1. Check the PLC network connections.
2. Verify the network ID given to the named device matches that of the actual device.

Unable to write to address <address> on device <device>: Device responded with exception code <code>.

Error Type:
Warning

Possible Cause:
See **Modbus Exception Codes** for a description of the exception code.

Solution:
See **Modbus Exception Codes**.

Device Specific Messages

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

Bad address in block [x to y] on device <device name>.
Bad array spanning [<address> to <address>] on device <device name>.
Bad received length [x to y] on device <device name>.
Cannot change device ID <device ID> from <current mode> to <new mode> with a client connected.
Failure to initiate winsock.dll.
Failure to start unsolicited communications.
Unsolicited mailbox access for undefined device (IP: <device IP>.<device index>)... Closing socket.
Unsolicited mailbox memory allocation error (IP: <device IP>)
Unsolicited mailbox unsupported request received (IP: <device IP>).

Bad address in block [x to y] on device <device name>.

Error Type:
Fatal addresses falling in this block.

Possible Cause:
This error is reported when the driver attempts to read a location in a PLC that does not exist. For example, in a PLC that only has holding registers 40001 to 41400, requesting address 41405 would generate this error. Once

this error is generated, the driver will not request the specified block of data from the PLC again. Any other addresses being requested that are in this same block will also go invalid.

Solution:
The client application should be modified to ask for addresses within the range of the device.

See Also:
Error Handling

Bad array spanning [<address> to <address>] on device <device name>.

Error Type:
Fatal

Possible Cause:
An array of addresses was defined that spans past the end of the address space.

Solution:
Verify the size of the device's memory space and then redefine the array length accordingly.

Bad received length [x to y] on device <device name>.

Error Type:
Fatal addresses falling in this block.

Possible Cause:
The driver attempted to read a block of memory in the PLC. The PLC responded with no error, but did not provide the driver with the requested block size of data.

Solution:
Ensure that the range of memory exists for the PLC.

Cannot change device ID <device ID> from <current mode> to <new mode> with a client connected.

Error Type:
Warning

Possible Cause:
When a client is connected, the Device ID can only be changed if it does not result in change of mode (master to slave or slave to master) of the device. The mode is changed by changing the loopback* or local IP address to a different IP address and vice versa. The loopback address and the local IP address (of the PC running the driver) indicates slave (unsolicited) mode, and any other IP address indicates master mode of the device.

*Any address in the format 127.xxx.xxx.xxx, where xxx is in the range 0-255, is the loopback address.

Solution:
To change the Device ID that results in change of mode (master to slave OR slave to master), disconnect all the clients.

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

Failure to initiate winsock.dll.

Error Type:
Fatal

Possible Cause:
Could not negotiate with the operating systems winsock 1.1 functionality.

Solution:
Verify that the winsock.dll is properly installed on the system.

Failure to start unsolicited communications.

Error Type:
Fatal

Possible Cause:
The driver was not able to create a listen socket for unsolicited communications.

Solution:

1. Verify that the port defined at the driver's channel level is not being used by another application on the system.
2. Call a Technical Support representative.

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

Unsolicited mailbox access for undefined device (IP: <device IP>.<device index>)... Closing socket.

Error Type:
Warning

Possible Cause:
A device with the specified IP address attempted to send a mailbox message to the server. The message did not pass validation, due to one of the following reasons:

1. There is no device with that IP configured in the Mailbox Project.
2. Although a device is configured, there are no clients requesting data from it.

Solution:
For the server to accept mailbox messages, the specified Device IP must be configured in the project. At least one data item from the device must be requested by a client.

Unsolicited mailbox memory allocation error (IP: <device IP>).

Error Type:
Fatal

Possible Cause:
An attempt made to allocate memory for the specified IP address failed.

Solution:
The server was unable to increase the working memory set for additional Mapped Memory addresses. This will occur if there is no more RAM or virtual RAM available for the server to use. To check for available memory, use the Task Manager. Update the machine to accommodate the demands of the project, if necessary.

Unsolicited mailbox unsupported request received (IP: <device IP>).

Error Type:
Warning

Possible Cause:
An unsupported request was received from the specified Device IP. The format of the device's request was invalid and not within Modbus specification.

Solution:
Verify that the devices configured to send Mailbox data are sending the correct requests.

Automatic Tag Database Generation Messages

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

Description truncated for import file record number <record>.
Error parsing import file record number <record>, field <field>.

Description truncated for import file record number <record>.

Error Type:
Warning

Possible Cause:
The tag description given in specified record is too long.

Solution:
The driver will truncate the description as needed. To prevent this error in the future, edit the variable import file to change the description (if possible).

Error parsing import file record number <record>, field <field>.

Error Type:
Serious

Possible Cause:
The specified field in the variable import file could not be parsed because it is longer than expected or invalid.

Solution:
Edit the variable import file to change the offending field if possible.

File exception encountered during tag import.

Error Type:
Serious

Possible Cause:
The variable import file could not be read.

Solution:
Regenerate the variable import file.

Imported tag name <tag name> is invalid. Name changed to <tag name>.

Error Type:
Warning

Possible Cause:
The tag name encountered in the variable import file contained invalid characters.

Solution:
The driver will construct a valid name based on the one from the variable import file. To prevent this error in the future and to maintain name consistency, change the name of the exported variable if possible.

Tag <tag name> could not be imported because data type <data type> is not supported.

Error Type:
Warning

Possible Cause:
The data type specified in the variable import file is not one of the types supported by this driver.

Solution:
If possible, change the data type specified in variable import file to one of the supported types. If the variable is for a structure, manually edit the file to define each tag required for the structure. Alternatively, manually configure the required tags in the OPC server.

See Also:
Exporting Variables from Concept

Tag import failed due to low memory resources.

Error Type:
Serious

Possible Cause:
The driver could not allocate memory required to process variable import file.

Solution:
Shut down all unnecessary applications and retry.

Modbus Exception Codes

The following data is from Modbus Application Protocol Specifications documentation.

Code Dec/Hex	Name	Meaning
01/0x01	ILLEGAL FUNCTION	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 expectation of the application program, since the Modbus protocol is unaware of the significance 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. This usually means that the device is not present on the network.

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

Index

A

B

C

D

E

F

G

H

I

M

O

P

R

S

T

U

V