

Modbus TCP/IP Ethernet Driver

© 2025 PTC Inc. All Rights Reserved.

Table of Contents

Modbus TCP/IP Ethernet Driver	1
Table of Contents	2
Modbus TCP/IP Ethernet Driver	5
Overview	5
Supported Device Models	6
Setup	7
Channel Properties – General	8
Tag Counts	8
Channel Properties – Ethernet Communications	9
Channel Properties – Write Optimizations	9
Channel Properties – Advanced	10
Channel Properties – Communication Serialization	10
Channel Properties – Ethernet	11
Device Properties – General	13
Device Properties – Scan Mode	14
Device Properties – Timing	14
Device Properties – Auto-Demotion	15
Device Properties – Tag Generation	16
Device Properties – Variable Import Settings	17
Device Properties – Unsolicited	18
Modbus Client & Modbus Server Considerations	18
Device Properties – Error Handling	19
Device Properties – Ethernet	19
Device Properties – Settings	20
Device Properties – Block Sizes	22
Device Properties – Redundancy	23
Configuration API – Modbus TCP/IP Ethernet Example	24
Enumerations	25
Device Model Enumerations	26
Automatic Tag Database Generation	27
Importing from Custom Applications	27
Optimizing Communications	28
Data Types Description	30
Address Descriptions	31
Driver System Tag Addressing	31
Function Codes Description	31
Applicom Sub-Model and Addressing	32
Generic Modbus Addressing	32
TSX Quantum	35
TSX Premium	38
CEG Addressing	40

Fluenta Addressing	40
Instromet Addressing	40
Mailbox Addressing	40
Modbus Addressing	41
Roxar Addressing	44
Statistics Items	44
Event Log Messages	46
Failure to start winsock communications.	46
Failure to start unsolicited communications.	46
Unsolicited mailbox access for undefined device. Closing socket. IP address = '<address>'.	46
Unsolicited mailbox unsupported request received. IP address = '<address>'.	46
Unsolicited mailbox memory allocation error. IP address = '<address>'.	47
Unable to create a socket connection.	47
Error opening file for tag database import. OS error = '<error>'.	47
Bad array. Array range = <start> to <end>.	47
Bad address in block. Block range = <address> to <address>.	47
Failed to resolve host. Host name = '<name>'.	48
Specified output coil block size exceeds maximum block size. Block size specified = <number> (coils), Maximum block size = <number> (coils).	48
Specified input coil block size exceeds maximum block size. Block size specified = <number> (coils), Maximum block size = <number> (coils).	48
Specified internal register block size exceeds maximum block size. Block size specified = <number> (registers), Maximum block size = <number> (registers).	48
Specified holding register block size exceeds maximum block size. Block size specified = <number> (registers), Maximum block size = <number> (registers).	48
Block request responded with exception. Block range = <address> to <address>, Exception = <code>.	48
Block request responded with exception. Block range = <address> to <address>, Function code = <code>, Exception = <code>.	49
Bad block length received. Block range = <start> to <end>.	49
Tag import failed due to low memory resources.	49
File exception encountered during tag import.	49
Error parsing record in import file. Record number = <number>, Field = <field>.	49
Description truncated for record in import file. Record number = <number>.	50
Imported tag name is invalid and has been changed. Tag name = '<tag>', Changed tag name = '<tag>'.	50
A tag could not be imported because the data type is not supported. Tag name = '<tag>', Unsupported data type = '<type>'.	50
Unable to write to address, device responded with exception. Address = '<address>', Exception = <code>.	50
Ethernet Manager started.	51
Ethernet Manager stopped.	51
Importing tag database. Source file = '<filename>'.	51
A client application has changed the CEG extension via system tag _CEGExtension. Extension = '<extension>'.	51
Starting unsolicited communication. Protocol = '<name>', Port = <number>.	51
Created memory for Modbus server device. Modbus server device ID = <device>.	51
All channels are subscribed to a virtual network or all devices are listening to remote addresses, stop-	51

ping unsolicited communication.

Channel is in a virtual network, all devices reverted to use one socket per device. 51

Cannot change device ID from Modbus client mode to server mode with a client connected. 51

Cannot change device ID from Modbus server mode to client mode with a client connected. 52

Modbus server mode not allowed when the channel is in a virtual network. The device ID cannot contain a loop-back or local IP address. 52

Mailbox model not allowed when the channel is in a virtual network. 52

Modbus Exception Codes 53

Modbus Ethernet Channel Properties 54

Modbus Ethernet Device Properties 54

Modbus Ethernet Tag Properties 55

Index 56

Modbus TCP/IP Ethernet Driver

Help version 1.158

CONTENTS

[Overview](#)

What is the Modbus TCP/IP Ethernet Driver?

[Setup](#)

How do I configure a channel and device for use with this driver?

[Configuration via API](#)

How do I configure a channel and device using the Configuration API?

[Automatic Tag Database Generation](#)

How can I configure tags for the Modbus TCP/IP Ethernet Driver?

[Optimizing Communications](#)

How do I get the best performance from the Modbus TCP/IP Ethernet Driver?

[Data Types Description](#)

What data types does the Modbus TCP/IP Ethernet Driver support?

[Address Descriptions](#)

How do I reference a data location in a Modbus Ethernet device?

[Event Log Messages](#)

What messages does the Modbus TCP/IP Ethernet Driver produce?

Overview

The Modbus TCP/IP Ethernet Driver provides a reliable way to connect Modbus Ethernet devices to 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 posts messages when a failure occurs during operation.

Supported Device Models

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 reads or writes to the storage area in the driver and not the physical device.

• *For information on sending unsolicited requests to the Modbus TCP/IP 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 disabling Holding Register Bit Writes in the device properties. This forces it to use the Read/Modify/Write sequence instead of directly writing to the bit. Only the client Modbus device (not the Mailbox) has to change its setting to get this to work.

• [Mailbox Client Privileges for Mailbox Device Model](#)

Modbus Client

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

Modbus Unsolicited or Server Mode

The Modbus TCP/IP Ethernet Driver acts 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 accepts all unsolicited commands that are received and attempts to process them as if it were just another PLC. Any Modbus client on the network can communicate with this simulated device using its IP address.

The device ID for a Modbus server 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 driver. The *XXX* designates the Modbus server's Station ID and can be in the range 0 to 255.

Multiple Modbus server devices can have the same Station ID. In this scenario, all the devices that share the Station ID point to one common simulated device. If the remote Modbus client requests data from a Modbus server device (Station ID) that does not exist, the response contains data from station 0. Once a Modbus server device is created in the project, that Modbus server is enabled and stays enabled until the server is shut down. Changing the Station ID enables a new Modbus server device that stays 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 responds 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 Modbus server 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.

● **See Also:** [Device Model Enumerations](#) and [Device Properties](#).

Setup

Channel and Device Limits

The maximum number of channels supported by this driver is 1024. The maximum number of devices supported by this driver is 8192 per channel.

● **Tip:** Channel-level settings apply to all devices that have been configured on this channel.

● **Note:** The Modbus TCP/IP Ethernet Driver requires Winsock V1.1 or higher.

Communication Serialization

The Modbus TCP/IP Ethernet Driver supports Communication Serialization, which specifies whether data transmissions should be limited to one channel at a time.

● **For more information, refer to** [Communication Serialization](#).

● **Notes:**

- When Channel Serialization is enabled, Unsolicited communications and the [Max Sockets per Device](#) property is disabled. Mailbox Model is unavailable for Channel Serialization.
- Not all properties are available and applicable for all models.

● **See Also:** [Configuring a device with Configuration API commands](#), [API Example with Modbus](#)

Channel Properties – General

This server supports the use of multiple simultaneous communications drivers. Each protocol or driver used in a server project is called a channel. A server project may consist of many channels with the same communications driver or with unique communications drivers. A channel acts as the basic building block of an OPC link. This group is used to specify general channel properties, such as the identification attributes and operating mode.

Property Groups	General	
	Write Optimizations	
	Advanced	
	Identification	
	Name	
	Description	
	Driver	
	Diagnostics	
	Diagnostics Capture	Disable
	Tag Counts	
Static Tags	10	

Identification

Name: Specify the user-defined identity of this channel. In each server project, each channel name must be unique. Although names can be up to 256 characters, some client applications have a limited display window when browsing the OPC server's tag space. The channel name is part of the OPC browser information. The property is required for creating a channel.

• For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.

Description: Specify user-defined information about this channel.

• Many of these properties, including Description, have an associated system tag.

Driver: Specify the protocol / driver for this channel. Specify the device driver that was selected during channel creation. It is a disabled setting in the channel properties. The property is required for creating a channel.

• **Note:** With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-acquire using the old channel name, the item is not accepted. Changes to the properties should not be made once a large client application has been developed. Utilize proper user role and privilege management to prevent operators from changing properties or accessing server features.

Diagnostics

Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.

• **Note:** This property is not available if the driver does not support diagnostics.

• For more information, refer to *Communication Diagnostics in the server help*.

Tag Counts

Static Tags: Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

Channel Properties – Ethernet Communications

Ethernet Communication can be used to communicate with devices.

Property Groups	Ethernet Settings	
General	Network Adapter	Default
Ethernet Communications		
Write Optimizations		
Advanced		

Ethernet Settings

Network Adapter: Specify the network adapter to bind. When left blank or Default is selected, the operating system selects the default adapter.

Channel Properties – Write Optimizations

The server must ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties to meet specific needs or improve application responsiveness.

Property Groups	Write Optimizations	
General	Optimization Method	Write Only Latest Value for All Tags
Write Optimizations	Duty Cycle	10

Write Optimizations

Optimization Method: Controls how write data is passed to the underlying communications driver. The options are:

- **Write All Values for All Tags:** This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags:** Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are needed to reach the same final output value. In this way, no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.
 - **Note:** This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags:** This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

Duty Cycle: is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

• **Note:** It is recommended that the application be characterized for compatibility with the write optimization enhancements before being used in a production environment.

Channel Properties – Advanced

This group is used to specify advanced channel properties. Not all drivers support all properties; so the Advanced group does not appear for those devices.

Property Groups	☒ Non-Normalized Float Handling	
General	Floating-Point Values	Replace with Zero
Write Optimizations	☒ Inter-Device Delay	
Advanced	Inter-Device Delay (ms)	0

Non-Normalized Float Handling: A non-normalized value is defined as Infinity, Not-a-Number (NaN), or as a Denormalized Number. The default is Replace with Zero. Drivers that have native float handling may default to Unmodified. Non-normalized float handling allows users to specify how a driver handles non-normalized IEEE-754 floating point data. Descriptions of the options are as follows:

- **Replace with Zero:** This option allows a driver to replace non-normalized IEEE-754 floating point values with zero before being transferred to clients.
- **Unmodified:** This option allows a driver to transfer IEEE-754 denormalized, normalized, non-number, and infinity values to clients without any conversion or changes.

Note: This property is disabled if the driver does not support floating-point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM data is not affected by this setting.

For more information on the floating-point values, refer to "How To ... Work with Non-Normalized Floating-Point Values" in the server help.

Inter-Device Delay: Specify the amount of time the communications channel waits to send new requests to the next device after data is received from the current device on the same channel. Zero (0) disables the delay.

Note: This property is not available for all drivers, models, and dependent settings.

Channel Properties – Communication Serialization

The server's multi-threading architecture allows channels to communicate with devices in parallel. Although this is efficient, communication can be serialized in cases with physical network restrictions (such as Ethernet radios). Communication serialization limits communication to one channel at a time within a virtual network.

The term "virtual network" describes a collection of channels and associated devices that use the same pipeline for communications. For example, the pipeline of an Ethernet radio is the client radio. All channels using the same client radio associate with the same virtual network. Channels are allowed to communicate each in turn, in a "round-robin" manner. By default, a channel can process one transaction before handing communications off to another channel. A transaction can include one or more tags. If the controlling channel contains a device that is not responding to a request, the channel cannot release control until the transaction times out. This results in data update delays for the other channels in the virtual network.

Property Groups	☒ Channel-Level Settings	
General	Virtual Network	None
Serial Communications	Transactions per Cycle	1
Communication Serialization	☒ Global Settings	
	Network Mode	Load Balanced

Channel-Level Settings

Virtual Network: Specify the channel's mode of communication serialization. Options include None and Network 1 - Network 500. The default is None. Descriptions of the options are as follows:

- **None:** This option disables communication serialization for the channel.
- **Network 1 - Network 500:** This option specifies the virtual network to which the channel is assigned.

Transactions per Cycle: Specify the number of single blocked/non-blocked read/write transactions that can occur on the channel. When a channel is given the opportunity to communicate, this is the number of transactions attempted. The valid range is 1 to 99. The default is 1.

Global Settings

Network Mode: This property is used to control how channel communication is delegated. In **Load Balanced** mode, each channel is given the opportunity to communicate in turn, one at a time. In **Priority** mode, channels are given the opportunity to communicate according to the following rules (highest to lowest priority):

1. Channels with pending writes have the highest priority.
2. Channels with pending explicit reads (through internal plug-ins or external client interfaces) are prioritized based on the read's priority.
3. Scanned reads and other periodic events (driver specific).

The default is Load Balanced and affects *all* virtual networks and channels.

🔧 Devices that rely on unsolicited responses should not be placed in a virtual network. In situations where communications must be serialized, it is recommended that Auto-Demotion be enabled.

Due to differences in the way that drivers read and write data (such as in single, blocked, or non-blocked transactions); the application's Transactions per cycle property may need to be adjusted. When doing so, consider the following factors:

- How many tags must be read from each channel?
- How often is data written to each channel?
- Is the channel using a serial or Ethernet driver?
- Does the driver read tags in separate requests, or are multiple tags read in a block?
- Have the device's Timing properties (such as Request timeout and Fail after x successive timeouts) been optimized for the virtual network's communication medium?

Channel Properties – Ethernet

Property Groups	Socket Usage	
General	Socket Utilization	One or More Sockets per Device
Ethernet Communications	Max Sockets per Device	1
Write Optimizations	Unsolicited Settings	
Advanced	Port	502
Communication Serialization	IP Protocol	TCP/IP
Ethernet		

Socket Usage

Socket Utilization: Specify if the driver should share a single socket across all devices on this channel or use multiple sockets to communicate with devices.

- **One or More Sockets per Device:** Specifies that the driver uses one or more sockets for each device on the network and maintains that socket as an active connection. This is the default setting and behavior.
- **One Socket per Channel:** Specifies that the driver use only a single socket to provide communications for each device in the channel.

Max. Sockets per Device: Specifies the maximum number of sockets available to the device. The default is 1.

🔧 For more information on how these settings impact performance, refer to [Optimizing Communications](#).

Unsolicited Settings

Unsolicited settings apply to both Modbus client and Modbus server modes.

Port: Specifies the port number that the driver uses when listening for unsolicited requests. The valid range is 0 to 65535. The default is 502.

IP Protocol: Specifies the protocol that the driver uses when listening for unsolicited request. Options include User Datagram Protocol (UDP) or Transmission Control Protocol (TCP/IP). The default is TCP/IP.

 Notes:

1. Modbus client mode is enabled when any external IP is used as the Device ID. When the driver is in client mode, it can 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 is made to the other two channels. The listening thread is restarted once the change is applied. The Event Log posts an event for the restart.
2. Modbus server mode is enabled when the model is set to Modbus and the Device ID is set to IP_Address.yyy, where IP_Address can be the local IP address of the network adapter assigned in the channel or the local loopback IP address of the PC running the driver. For example, 127.0.0.1.1 or 127.xxx.xxx.xxx, where xxx=0-255 and yyy (station ID)=0-255. When the driver is in Modbus server mode, it starts a single listening thread for unsolicited data once the driver is loaded by the OPC server. This thread listens on a single port and is global to all channels configured in the OPC server. Multiple devices in a server project can be set to act as Modbus servers, but while all Modbus server devices in the project can be accessed individually using unique device IDs, all Modbus servers in the same project listen, or are reached, on the same port.

 For more information on unsolicited mode, refer to [Modbus Unsolicited or Server Mode](#).

Device Properties – General

Property Groups	General	
	Scan Mode	
	Timing	
	Auto-Demotion	
	Tag Generation	
	Variable Import Settings	
	Error Handling	
	Ethernet	
	Settings	
	Block Sizes	
	Unsolicited	
	Redundancy	
	Identification	
	Name	Modbus
	Description	
Driver	Modbus TCP/IP Ethernet	
Model	Modbus	
Channel Assignment	Modbus TCP/IP Ethernet	
ID	<10.10.110.122>.0	
Operating Mode		
Data Collection	Enable	
Simulated	No	
Tag Counts		
Static Tags	1	

Identification

Name: User-defined identity of this device.

Description: User-defined information about this device.

Channel Assignment: User-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

• For more information on a specific device model, see [Supported Device Models](#).

Model: The specific version of the device.

ID: 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 Server ID 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 can be configured to act as a client or server.

• For more information on unsolicited mode, refer to [Modbus Unsolicited or Server Mode](#).

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.

Operating Mode

Data Collection: This property controls the device's active state. Although device communications are enabled by default, this property can be used to disable a physical device. Communications are not attempted when a device is disabled. From a client standpoint, the data is marked as invalid and write operations are not accepted. This property can be changed at any time through this property or the device system tags.

Simulated: This option places the device into Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client as valid data. While in Simulation Mode, the server treats all device data as reflective: whatever is written to the simulated device is read back and each OPC item is treated individually. The item's memory map is based on the group Update Rate. The data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

• **Notes:**

1. This System tag (_Simulated) is read only and cannot be written to for runtime protection. The System tag allows this property to be monitored from the client.
2. In Simulation mode, the item's memory map is based on client update rate(s) (Group Update Rate for OPC clients or Scan Rate for native and DDE interfaces). This means that two clients that reference the same item with different update rates return different data.

Simulation Mode is for test and simulation purposes only. It should never be used in a production environment.

See Also: [Configuring a device with Configuration API commands](#), [API Example with Modbus](#)

Device Properties – Scan Mode

The Scan Mode specifies the subscribed-client requested scan rate for tags that require device communications. Synchronous and asynchronous device reads and writes are processed as soon as possible; unaffected by the Scan Mode properties.

Property Groups	☒ Scan Mode	
General	Scan Mode	Respect Client-Specified Scan Rate ▼
Scan Mode	Initial Updates from Cache	Disable

Scan Mode: Specify how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

- **Respect Client-Specified Scan Rate:** This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate:** This mode specifies the value set as the maximum scan rate. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
 - **Note:** When the server has an active client and items for the device and the scan rate value is increased, the changes take effect immediately. When the scan rate value is decreased, the changes do not take effect until all client applications have been disconnected.
- **Request All Data at Scan Rate:** This mode forces tags to be scanned at the specified rate for subscribed clients. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
- **Do Not Scan, Demand Poll Only:** This mode does not periodically poll tags that belong to the device nor perform a read to get an item's initial value once it becomes active. It is the OPC client's responsibility to poll for updates, either by writing to the _DemandPoll tag or by issuing explicit device reads for individual items. *For more information, refer to "Device Demand Poll" in server help.*
- **Respect Tag-Specified Scan Rate:** This mode forces static tags to be scanned at the rate specified in their static configuration tag properties. Dynamic tags are scanned at the client-specified scan rate.

Initial Updates from Cache: When enabled, this option allows the server to provide the first updates for newly activated tag references from stored (cached) data. Cache updates can only be provided when the new item reference shares the same address, scan rate, data type, client access, and scaling properties. A device read is used for the initial update for the first client reference only. The default is disabled; any time a client activates a tag reference the server attempts to read the initial value from the device.

Device Properties – Timing

The device Timing properties allow the driver's response to error conditions to be tailored to fit the application's needs. In many cases, the environment requires changes to these properties for optimum performance. Factors such as electrically generated noise, modem delays, and poor physical connections can influence how many errors or timeouts a communications driver encounters. Timing properties are specific to each configured device.

Property Groups	☒ Communication Timeouts	
General	Connect Timeout (s)	3
Scan Mode	Request Timeout (ms)	1000
Timing	Attempts Before Timeout	3

Communications Timeouts

Connect Timeout: This property (which is used primarily by Ethernet based drivers) controls the amount of time required to establish a socket connection to a remote device. The device's connection time often takes longer than normal communications requests to that same device. The valid range is 1 to 30 seconds. The default is typically 3 seconds, but can vary depending on the driver's specific nature. If this setting is not supported by the driver, it is disabled.

● **Note:** Due to the nature of UDP connections, the connection timeout setting is not applicable when communicating via UDP.

Request Timeout: Specify an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9999999 milliseconds (167 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

Attempts Before Timeout: Specify how many times the driver issues a communications request before considering the request to have failed and the device to be in error. The valid range is 1 to 10. The default is typically 3, but can vary depending on the driver's specific nature. The number of attempts configured for an application depends largely on the communications environment. This property applies to both connection attempts and request attempts.

Timing

Inter-Request Delay: Specify how long the driver waits before sending the next request to the target device. It overrides the normal polling frequency of tags associated with the device, as well as one-time reads and writes. This delay can be useful when dealing with devices with slow turnaround times and in cases where network load is a concern. Configuring a delay for a device affects communications with all other devices on the channel. It is recommended that users separate any device that requires an inter-request delay to a separate channel if possible. Other communications properties (such as communication serialization) can extend this delay. The valid range is 0 to 300,000 milliseconds; however, some drivers may limit the maximum value due to a function of their particular design. The default is 0, which indicates no delay between requests with the target device.

● **Note:** Not all drivers support Inter-Request Delay. This setting does not appear if it is not available.

Property Groups	☒ Timing	
General	Inter-Request Delay (ms)	0
Scan Mode		
Timing		

Device Properties – Auto-Demotion

The Auto-Demotion properties can temporarily place a device off-scan in the event that a device is not responding. By placing a non-responsive device offline for a specific time period, the driver can continue to optimize its communications with other devices on the same channel. After the time period has been reached, the driver re-attempts to communicate with the non-responsive device. If the device is responsive, the device is placed on-scan; otherwise, it restarts its off-scan time period.

Property Groups	☒ Auto-Demotion	
General	Demote on Failure	Enable v
Scan Mode	Timeouts to Demote	3
Timing	Demotion Period (ms)	10000
Auto-Demotion	Discard Requests when Demoted	Disable

Demote on Failure: When enabled, the device is automatically taken off-scan until it is responding again.

● **Tip:** Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

Timeouts to Demote: Specify how many successive cycles of request timeouts and retries occur before the device is placed off-scan. The valid range is 1 to 30 successive failures. The default is 3.

Demotion Period: Indicate how long the device should be placed off-scan when the timeouts value is reached. During this period, no read requests are sent to the device and all data associated with the read requests are set to bad quality. When this period expires, the driver places the device on-scan and allows for another attempt at communications. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds.

Discard Requests when Demoted: Select whether or not write requests should be attempted during the off-scan period. Disable to always send write requests regardless of the demotion period. Enable to discard writes; the server automatically fails any write request received from a client and does not post a message to the Event Log.

Device Properties – Tag Generation

The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the supporting driver) can be browsed from the clients.

⚙ *Not all devices and drivers support full automatic tag database generation and not all support the same data types. Consult the data types descriptions or the supported data type lists for each driver for specifics.*

If the target device supports its own local tag database, the driver reads the device's tag information and uses the data to generate tags within the server. If the device does not natively support named tags, the driver creates a list of tags based on driver-specific information. An example of these two conditions is as follows:

1. If a data acquisition system supports its own local tag database, the communications driver uses the tag names found in the device to build the server's tags.
2. If an Ethernet I/O system supports detection of its own available I/O module types, the communications driver automatically generates tags in the server that are based on the types of I/O modules plugged into the Ethernet I/O rack.

⚙ **Note:** Automatic tag database generation's mode of operation is completely configurable. *For more information, refer to the property descriptions below.*

Property Groups	Tag Generation	
General	On Device Startup	Do Not Generate on Startup
Timing	On Duplicate Tag	Delete on Create
Auto-Demotion	Parent Group	
Tag Generation	Allow Automatically Generated Subgroups	Enable
Communications	Create	Create tags
Redundancy		

On Property Change: If the device supports automatic tag generation when certain properties change, the **On Property Change** option is shown. It is set to **Yes** by default, but it can be set to **No** to control over when tag generation is performed. In this case, the **Create tags** action must be manually invoked to perform tag generation.

On Device Startup: Specify when OPC tags are automatically generated. Descriptions of the options are as follows:

- **Do Not Generate on Startup:** This option prevents the driver from adding any OPC tags to the tag space of the server. This is the default setting.
- **Always Generate on Startup:** This option causes the driver to evaluate the device for tag information. It also adds tags to the tag space of the server every time the server is launched.
- **Generate on First Startup:** This option causes the driver to evaluate the target device for tag information the first time the project is run. It also adds any OPC tags to the server tag space as needed.

⚙ **Note:** When the option to automatically generate OPC tags is selected, any tags that are added to the server's tag space must be saved with the project. Users can configure the project to automatically save from the **Tools | Options** menu.

On Duplicate Tag: When automatic tag database generation is enabled, the server needs to know what to do with the tags that it may have previously added or with tags that have been added or modified after the communications driver since their original creation. This setting controls how the server handles OPC tags that were automatically

generated and currently exist in the project. It also prevents automatically generated tags from accumulating in the server.

For example, if a user changes the I/O modules in the rack with the server configured to **Always Generate on Startup**, new tags would be added to the server every time the communications driver detected a new I/O module. If the old tags were not removed, many unused tags could accumulate in the server's tag space. The options are:

- **Delete on Create:** This option deletes any tags that were previously added to the tag space before any new tags are added. This is the default setting.
- **Overwrite as Necessary:** This option instructs the server to only remove the tags that the communications driver is replacing with new tags. Any tags that are not being overwritten remain in the server's tag space.
- **Do not Overwrite:** This option prevents the server from removing any tags that were previously generated or already existed in the server. The communications driver can only add tags that are completely new.
- **Do not Overwrite, Log Error:** This option has the same effect as the prior option, and also posts an error message to the server's Event Log when a tag overwrite would have occurred.

● **Note:** Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags. Users should avoid adding tags to the server using names that may match tags that are automatically generated by the driver.

Parent Group: This property keeps automatically generated tags from mixing with tags that have been entered manually by specifying a group to be used for automatically generated tags. The name of the group can be up to 256 characters. This parent group provides a root branch to which all automatically generated tags are added.

Allow Automatically Generated Subgroups: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting tags are named with the address value. For example, the tag names are not retained during the generation process.

● **Note:** If, as the server is generating tags, a tag is assigned the same name as an existing tag, the system automatically increments to the next highest number so that the tag name is not duplicated. For example, if the generation process creates a tag named "AI22" that already exists, it creates the tag as "AI23" instead.

Create: Initiates the creation of automatically generated OPC tags. If the device's configuration has been modified, **Create tags** forces the driver to reevaluate the device for possible tag changes. Its ability to be accessed from the System tags allows a client application to initiate tag database creation.

● **Note:** **Create tags** is disabled if the Configuration edits a project offline.

Device Properties – Variable Import Settings

● For more information on CSV files for Modbus Drivers, refer to [Creating CSV Files for Modbus Drivers](#).

Property Groups	Variable Import Settings	
General	Variable Import File	*.txt
Scan Mode	Include Descriptions	Enable
Timing		
Auto-Demotion		
Tag Generation		
Variable Import Settings		
Error Handling		
Ethernet		
Settings		
Block Sizes		
Unsolicited		
Redundancy		

Variable Import File: This parameter specifies the exact location of the variable import file that the driver should use when the Automatic Tag Database Generation feature is enabled.

Include Descriptions: When enabled, this option imports tag descriptions (if present in file).

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](#).

Device Properties – Unsolicited

Property Groups General Scan Mode Timing Auto-Demotion Tag Generation Variable Import Settings Error Handling Ethernet Settings Block Sizes Unsolicited Redundancy	OPC Quality	
	OPC Quality Bad until Write	Disable
	Communications Timeout (s)	0

OPC Quality

OPC Quality Bad until Write: Controls the initial OPC quality of tags attached to this driver. When disabled, all tags have an initial value of 0 and an OPC quality of Good. This is the default condition. When enabled, all tags have an initial value of 0 and an OPC quality of Bad. The tag's quality remains Bad until all coils or registers referenced by the tag have been written to by a Modbus client or a client application. For example, a tag with address 400001 and data type DWord references two holding registers: 400001 and 400002. This tag does not show Good quality until both holding registers have been written.

Note: If the device is not in server mode, this option is grayed out.

Communications Timeout: Sets the amount of time, in seconds, the driver waits 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 client or disable the communications timeout by setting it to 0. When enabled, the valid range is 1 to 64,800 seconds (18 hours).

Notes:

1. If an incoming request comes for a server device (station ID) that does not exist, the request is directed to station 0. In this case, the timeout for a server device with station ID 0 does not occur even if it does not explicitly receive any remote communications for the timeout period.
2. Server devices require the model to be Modbus and the Device ID to be *IP_Address.yyy*, where *IP_Address* can be the IP address of the network adapter assigned in the channel or the local IP loopback IP address of the PC running the driver. For example, 127.0.0.1.1 or 127.xxx.xxx.xxx, where xxx=0-255 and yyy (station ID)=0-255.
3. When the first unsolicited request for a server device is received, the Event Log displays 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 server Device <server Number>".

Modbus Client & Modbus Server Considerations

The following notes pertain to both Modbus Client and Modbus Server devices.

- It is not recommended that a Mailbox device and a Modbus device be on the same machine. Because a client only gets data from one of these devices at a time; it is uncertain from which it gets data.
- It is recommended that client and server devices be placed on separate channels in the server project for optimal server device tag processing.

- When an OPC client is connected, the device ID can only be changed if it does not result in change of mode (client to server or server to client) 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 server (unsolicited) mode and any other IP address indicates client mode of the device. When no OPC client is connected, the mode can be changed in any manner (such as client to client, client to server, server to server, or server to client).
- 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 client and server devices. For example, when a device configured as a Modbus client is communicating with the device setup as a Modbus server.
- The server project as a whole allows a maximum of 255 server devices, one for each unique server ID. The same server ID cannot be used across multiple channels.
- The server sees ANY loopback address (127.x.x.x), or localhost IP as a reference back to itself and creates shared memory space unique to the server ID. The same ID in multiple channels is the same server device using the same register memory.
- If the same server ID must be used more than once in a project, choose tag address ranges that do not coincide with other instances of the same server device IDs. Multiple channels / devices using the same tag address range in the same server ID experience cross-talk and data corruption.
- For this driver, the terms server and unsolicited are used interchangeably.

Device Properties – Error Handling

Property Groups General Scan Mode Timing Auto-Demotion Tag Generation Variable Import Settings Error Handling Ethernet Settings Block Sizes Unsolicited Redundancy	Error Handling	
	Deactivate Tags on Illegal Address	Enable

Deactivate Tags on Illegal Address: Choose Enable for the driver to 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. Choose Disable for the driver to continue polling the data block despite errors. The default is enabled.

Device Properties – Ethernet

Property Groups General Scan Mode Timing Auto-Demotion Tag Generation Variable Import Settings Error Handling Ethernet Settings Block Sizes Unsolicited Redundancy	Ethernet	
	Port	502
	IP Protocol	TCP/IP
	Close Socket on Timeout	Enable

Port: Specifies the port number that the remote device is configured to use. The valid range is 0 to 65535. The default is 502. This port number is used when the device object is in Modbus client mode and making solicited requests to a device.

● If the port system tag is used, the port number setting is changed. For more information, refer to [Driver System Tag Addresses](#).

IP Protocol: Specifies whether the driver should connect to the remote device using the User Datagram Protocol (UDP) or Transmission Control Protocol (TCP/IP). The client and server settings must match. For example, if the server's IP protocol setting is TCP/IP, then the client's IP protocol setting for that device must also be TCP/IP.

● **Note:** This driver requires Winsock V1.1 or higher.

Close Socket on Timeout: Specifies whether the driver should close a TCP socket connection if the device does not respond within the timeout. When enabled, the default, the driver closes the socket connection on timeout. When disabled, the driver continues to use the same TCP socket until an error is received, the physical device closes the socket, or the driver is shutdown.

● **Note:** The Modbus TCP/IP Ethernet Driver closes the socket connection on a socket error.

Device Properties – Settings

Property Groups	Data Access	
General	Zero-Based Addressing	Enable
Scan Mode	Zero-Based Bit Addressing	Enable
Timing	Holding Register Bit Writes	Enable
Auto-Demotion	Modbus Function 06	Enable
Tag Generation	Modbus Function 05	Enable
Variable Import Settings	Data Encoding	
Error Handling	Modbus Byte Order	Enable
Ethernet	First Word Low	Enable
Settings	First DWord Low	Enable
Block Sizes	Modicon Bit Order	Disable
Unsolicited	Treat Longs as Decimals	Disable
Redundancy		

Data Access

Zero-Based Addressing: If the address-numbering convention for the device starts at one as opposed to zero, the value can be specified when defining the device parameters. By default, user-entered addresses have one subtracted when frames are constructed to communicate with a Modbus device. If the device does not follow this convention, choose disable. The default behavior follows the convention of Modicon PLCs.

Zero-Based Bit Addressing: Within registers, memory types that allow bits within Words can be referenced as Booleans. The addressing notation is <address>. <bit>, where <bit> represents the bit number within the Word. This option provides two ways of addressing a bit within a given Word; zero- or one-based. Zero-based means that the first bit begins at 0 (range=0-15); one-based means that the first bit begins at 1 (range=1-16).

Holding Register Bit 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 must perform a Read / Modify / Write operation to ensure that only the single bit is changed. When enabled, the driver uses function code 0x16, regardless of this setting for single register writes. When disabled, the driver uses function code 0x06 or 0x10, depending on the selection for Modbus Function 06 for single register writes. The default is disabled.

● **Note:** When Modbus byte order is disabled, the byte order of the masks sent in the command is Intel byte order.

Modbus Function 06: This driver supports Modbus protocol functions to write holding register data to the target device. In most cases, the driver switches between functions 06 and 16 based on the number of registers being written. When writing a single 16-bit register, the driver generally uses Modbus function 06. When writing a 32-bit value into two registers, the driver uses 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 using the Modbus protocol and many support only Modbus function 16 to write to holding registers. This selection is enabled by default,

allowing the driver to switch between 06 and 16 as needed. If a device requires all writes to use only Modbus function 16, disable this selection.

● **Note:** For bit within word writes, the Holding Register Bit Writes property takes precedence over this option. If Holding Register Bit Writes is enabled, function code 0x16 is used regardless of this property. If not enabled, either function code 0x06 or 0x10 is used for bit within word writes.

Modbus Function 05: This driver supports 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 uses Modbus function 05. When writing an array of coils, the driver uses Modbus function 15. For the standard Modicon PLC, the use of these functions is not a problem. There are, however, many third-party devices that use the Modbus protocol and many only support the use of Modbus function 15 to write to output coils regardless of the number of coils. This selection is enabled by default, allowing the driver to switch between 05 and 15 as needed. If a device requires all writes to use only Modbus function 15, disable this selection.

CEG Extension: The Modbus driver can communicate with CEG devices that support extended block sizes or Modbus devices configured with the CEG model. This property is only available for the CEG model. The default is enabled, indicating the device is a CEG device with extended block sizes. Disabled indicates the device does not support the extended block sizes.

● **Note:** This property can be modified when an active OPC client connection exists. In this situation, disabling the option causes the block size ranges to change. If any of the block size properties exceed the maximum value, they are automatically adjusted to the new maximum value.

Mailbox Client Privileges: The Modbus driver can communicate with Mailbox clients with the following options:

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

Data Encoding

Modbus Byte Order: sets the data encoding of each register / 16-bit value. The byte order for can be changed from the default Modbus byte ordering to Intel byte ordering using this selection. The default is enabled, which is the normal setting for Modbus-compatible devices. If the device uses Intel byte ordering, disable this property to read Intel-formatted data.

First Word Low: sets the data encoding of 32-bit values and the double word of 64-bit values. Two consecutive registers' addresses in a Modbus device are used for 32-bit data types. The driver can read the first word as the low or the high word of the 32-bit value based on this option. The default is enabled, first word low, to follow the convention of the Modicon Modsoft programming software.

First DWord Low: sets the data encoding of 64-bit values. Four consecutive registers' addresses in a Modbus device are used for 64-bit data types. The driver can read the first DWord as the low or the high DWord of the 64-bit value. The default is enabled, first DWord low, to follow the default convention of 32-bit data types.

Modicon Bit Order: when enabled, the driver reverses 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 affects bit 15/16 in the device when this option is enabled. This option is disabled (disabled) by default.

For the following example, the 1st through 16th bit signifies either 0-15 bits or 1-16 bits, depending on the driver using zero-based or one-based bit addressing within registers.

MSB = Most Significant Bit

LSB = Least Significant Bit

Modicon Bit Order Enabled

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

Modicon Bit Order Disabled

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

Treat Longs as Decimals: when enabled, the driver encodes and decodes double-precision unsigned 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 formula $[\text{Read Value}] = \text{HighWord} * 10000 + \text{LowWord}$. Written values greater than 99999999 are clamped to the maximum value. All written values are encoded using the formula $\text{Raw Data} = [\text{Written Value}] / 10000 + [\text{Written Value}] \% 10000$.

Tips on Settings

Data Types	Modbus Byte Order	First Word Low	First DWord Low
Word, Short, BCD	Applicable	N/A	N/A
Float, DWord, Long, LBCD	Applicable	Applicable	N/A
Double	Applicable	Applicable	Applicable

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 Option	Data Encoding	
Modbus Byte Order	High Byte (15..8)	Low Byte (7..0)
Modbus Byte Order	Low Byte (7..0)	High Byte (15..8)
First Word Low	High Word (31..16)	Low Word (15..0)
	High Word (63..48) of Double Word in 64-bit data types	Low Word (47..32) of Double Word in 64-bit data types
First Word Low	Low Word (15..0)	High Word (31..16)
	Low Word (47..32) of Double Word in 64-bit data types	High Word (63..48) of Double Word in 64-bit data types
First DWord Low	High Double Word (63..32)	Low Double Word (31..0)
First DWord Low	Low Double Word (31..0)	High Double Word (63..32)

Device Properties – Block Sizes

Property Groups	☒ Coils (in Multiples of 8)	
General	Output Coils	32
Scan Mode	Input Coils	32
Timing	☒ Registers	
Auto-Demotion	Internal Registers	32
Tag Generation	Holding Registers	32
Variable Import Settings	☒ Blocks	
Error Handling	Block Read Strings	Disable
Ethernet		
Settings		
Block Sizes		
Unsolicted		
Redundancy		

Coils

Output Coils: Specifies the output block size in bits. Coils can be read from 8 to 2000 points (bits) at a time. The default is 32.

Input Coils: Specifies the input block size in bits. Coils can be read from 8 to 2000 points (bits) at a time. The default is 32.

Registers

Internal Registers: Specifies the internal register block size in bits. From 1 to 120 standard 16-bit Modbus registers can be read at a time. The default is 32.

Holding Registers: Specifies the holding register block size in bits. From 1 to 120 standard 16-bit Modbus registers can be read at a time. The default is 32.

Blocks

Block Read Strings: Enables group / block reads of string tags, which are normally read individually. String tags are grouped together depending on the selected block size. Block reads can only be performed for Modbus model string tags.

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 can occur if the register block sizes are set above 120 and a 32- or 64-bit data type is used for any tag. To prevent this, decrease the block size value to 120.
4. Some devices may not support block read operations at the default size. Smaller Modicon PLCs and non-Modicon devices may not support the maximum data transfer lengths supported by the Modbus Ethernet network.
5. Some devices may contain non-contiguous addresses. In this case, and the driver attempts to read a block of data that encompasses undefined memory, the request may be rejected.

Device Properties – Redundancy

Property Groups	Redundancy	
General	Secondary Path	Channel.Device 1 ...
Scan Mode	Operating Mode	Switch On Failure
Timing	Monitor Item	
Auto-Demotion	Monitor Interval (s)	300
Redundancy	Return to Primary ASAP	Yes

Redundancy is available with the Media-Level Redundancy Plug-In.

Consult the website, a sales representative, or the [user manual](#) for more information.

Configuration API – Modbus TCP/IP Ethernet Example

For a list of channel and device definitions and enumerations, access the following endpoints with the REST client or refer to the appendices.

Channel Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<-  
port>/config/v1/doc/drivers/Modbus%20TCP%2FIP%20Ethernet/channels
```

Device Definitions

Endpoint (GET):

```
https://<hostname_or_ip>:<port>/config/v1/doc/drivers/Modbus%20TCP%2FIP%20Ethernet/devices
```

Create Modbus TCP/IP Ethernet Channel

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels
```

Body:

```
{  
  "common.ALLTYPES_NAME": "MyChannel",  
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet"  
}
```

• **See Also:** [Appendix](#) for a list of channel properties.

Create Modbus TCP/IP Ethernet Device

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/MyChannel/devices
```

Body:

```
{  
  "common.ALLTYPES_NAME": "MyDevice",  
  "servermain.DEVICE_ID_STRING": "<IP Address>.<Modbus ID>",  
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet",  
  "servermain.DEVICE_MODEL": <model enumeration>  
}
```

where <IP Address>.<Modbus ID> is the device's IP address and Modbus ID, such as 192.160.0.1.0.

• **See Also:** [Device Model Enumerations](#) and [Device Properties](#).

Device ID Update

Update the Device ID using a "PUT" command from a REST client.

The Endpoint example below references the "demo-project.json" project configuration with "ModbusTCPIP" channel name and "ModbusDevice" device name.

Device ID Example

Endpoint (PUT):

```
https://<hostname_or_ip>:<-  
port>/config/v1/project/channels/ModbusTCPIP/devices/ModbusDevice
```

Body:

```
{
  "project_id": <project_ID_from_GET>,
  "servermain.DEVICE_ID_STRING": "<IP Address>.<Modbus ID>"
}
```

Create Modbus TCP/IP Ethernet Tags

Endpoint (POST):

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/MyChannel/devices/MyDevice/tags
```

Body:

```
[
  {
    "common.ALLTYPES_NAME": "MyTag1",
    "servermain.TAG_ADDRESS": "40001"
  },
  {
    "common.ALLTYPES_NAME": "MyTag2",
    "servermain.TAG_ADDRESS": "40002"
  }
]
```

• **See Also:** [Appendix](#) for a list of tag properties.

• See server and driver-specific help for more information on configuring projects over the Configuration API.

Enumerations

Some properties, such as Device Model, have values that are mapped to an enumeration. A valid list of enumerations and their values can be found by querying the device endpoint with 'content=property_definitions' or the documentation definitions endpoints.

For example, to view the property definitions for a device named "MyDevice" under a channel named "MyChannel", the GET request would be sent to:

```
https://<hostname_or_ip>:<port>/config/v1/project/channels/MyChannel/devices/MyDevice/?content=property_definitions
```

Property definitions are also available for other objects such as channels or tags.

Alternatively, if enabled in the settings for the Configuration API, the channel and device property definitions for the driver can be viewed at:

```
https://<hostname_or_ip>:<port>/config/v1/doc/drivers/<drivername>/Channels
```

```
https://<hostname_or_ip>:<port>/config/v1/doc/drivers/<drivername>/Devices
```

Example Data Type Enumerations

Querying the documentation endpoint for tag data types provides the following enumerations:

```
{
  "Default": -1,
  "String": 0,
  "Boolean": 1,
  "Char": 2,
  "Byte": 3,
  "Short": 4,
```

```

"Word": 5,
"Long": 6,
"DWord": 7,
"Float": 8,
"Double": 9,
"BCD": 10,
"LBCD": 11,
"Date": 12,
"LLong": 13,
"QWord": 14,
"String Array": 20,
"Boolean Array": 21,
"Char Array": 22,
"Byte Array": 23,
"Short Array": 24,
"Word Array": 25,
"Long Array": 26,
"DWord Array": 27,
"Float Array": 28,
"Double Array": 29,
"BCD Array": 30,
"LBCD Array": 31,
"Date Array": 32,
"LLong Array": 33,
" QWord Array": 34
}

```

 **Note:** Supported data types vary by protocol and driver.

Device Model Enumerations

The Device Model property has values mapped to the following enumerations. The below table is for reference only; the information at the device endpoint is the complete and current source of information:

```

https://<hostname_or_ip>:<-
port>/config/v1/doc/drivers/Modbus%20TCP%2FIP%20Ethernet/Channels

```

```

https://<hostname_or_ip>:<port>/config/v1/doc/drivers/Modbus%20TCP%2FIP%20Ethernet/Devices

```

Enumeration	Device Model
0	Modbus
1	Mailbox
2	Instromet
3	Roxar RFM
4	Fluenta FGM
5	Applicom
6	CEG

Automatic Tag Database Generation

This driver supports the Automatic Tag Database Generation, 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 Event Log shows 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 device programming applications, such as Concept and ProWORX. The import file must be in semicolon-delimited .txt format, which is the default export file format of the Concept device programming application.

• See Also: [Importing from Custom Applications](#)

• For specific information on creating the variable import file, consult *Technical Note Creating CSV Files for Modbus Drivers*.

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.

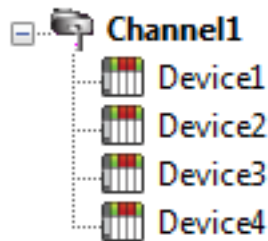
Examples

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

Optimizing Communications

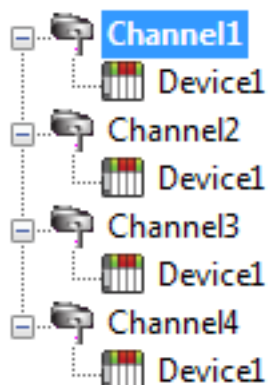
The Modbus TCP/IP Ethernet Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the driver is fast, there are a couple of guidelines that can be used to control and optimize the application and gain maximum performance.

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 is collected. While this approach to defining the application provides a high level of performance, it doesn't take full advantage of the driver or the network. An example of how the application may appear when configured using a single channel is shown below.



Each device is defined 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 TCP/IP Ethernet Driver could only define one single channel, then the example above would be the only option available; however, the driver can define up to 1024 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 can be defined under its own channel. In this configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has 1024 or fewer devices, it can be optimized in this manner.

The performance improves even if the application has more devices. While fewer devices may be ideal, the application still benefits from additional channels. Although by spreading the device load across all channels causes the server to move from device to device again, it can do so with far less devices to process on a single channel.

Block Size

Block size can affect the performance of the Modbus TCP/IP Ethernet Driver. The block size parameter is available on each device, defined under the [Block Size](#) settings for device properties. 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.

Tips:

- Additional performance gain can be realized by enabling the **Close Socket on Timeout** property and by adjusting timeouts and timing properties.
- Performance is also impacted by socket usage, discussed below.

For more information, refer to the [Ethernet properties](#), [Communication Timeouts](#), and [Timing](#).

Socket Usage

Users can specify if the driver should share a single socket across all devices on a channel or use multiple sockets to communicate with devices on a single channel. If the driver is using a port, no other system may access the target device, so balancing access to devices may need to be considered in some environments. If it is undesirable for the driver to maintain a connection or if the device has a limited number of connections available, the user can limit the number of sockets the driver can use.

The ability to put the driver into single-socket mode is important when using the driver to communicate with a Modbus-Ethernet-to-Modbus-RTU bridge product. Most of these products allow connecting multiple RS-485 serial-based devices to a single Modbus-Ethernet-to-Modbus-RTU bridge. This be done by setting One Socket per Channel or by setting One or More Sockets per Device with the Max Sockets per Device set to 1 (see [Channel Ethernet settings](#)).

When more than one socket is configured for use, the driver spreads the data to read or write operations to a target device across all of the available sockets in use with the target device. The driver uses **only** the number of sockets needed to process the data, up to the maximum number specified.

If device objects under a channel target different IP addresses and/or ports, the socket is closed and reopened only as needed. This is recommended if an Ethernet to serial gateway device is handling a number of serial devices. Gateways (and devices) typically limit the number of simultaneous connections to protect against communications conflicts. Avoid exceeding these limits. If these limits are exceeded, the driver posts failure-to-connect messages.

Device response messages may be received by the driver at the same time. The device's responses are processed sequentially by the single thread at the channel-level; however, this processing of data at the channel-level can occur very fast (milliseconds) and therefore, when configured to use more than one socket, a significant performance improvement can be achieved.

 **Tip:** Performance is also impacted by block size, discussed above, which can affect the data transfer across the sockets.

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

Internal Tags

Tag	Description	Data Type	Access
Port	The Port system tag allows a client application to read and write the Port Number setting. Writes to this tag cause the driver to disconnect from the device and attempt to reconnect to the specified port.	Word, Short, DWord, Long	Read/Write

Notes:

- The device port setting is not used by the driver for server communications.
- For this driver, the terms server and unsolicited are used interchangeably.
- Changes to this tag modifies the project, which causes the server to prompt to save the project on shut-down.

System Tags

Tag	Description	Data Type	Access
_CEGExtension	This tag is only used for CEG model devices. It allows the CEG extension device property to be changed from a client application.	Boolean	Read/Write
_InputCoilBlockSize	This tag allows the Input Coils block size property to be changed from a client application.	DWord	Read/Write
_OutputCoilBlockSize	This tag allows the Output Coils block size property to be changed from a client application.	DWord	Read/Write
_InternalRegisterBlockSize	This tag allows the Internal Registers block size property to be changed from a client application.	DWord	Read/Write
_HoldingRegisterBlockSize	This tag allows the Holding Registers block size property to be changed from a client application.	DWord	Read/Write

Note: Changes to these tags modify the project, which causes the server to prompt to save the project on shut-down.

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

Decimal	Hexadecimal	Description
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

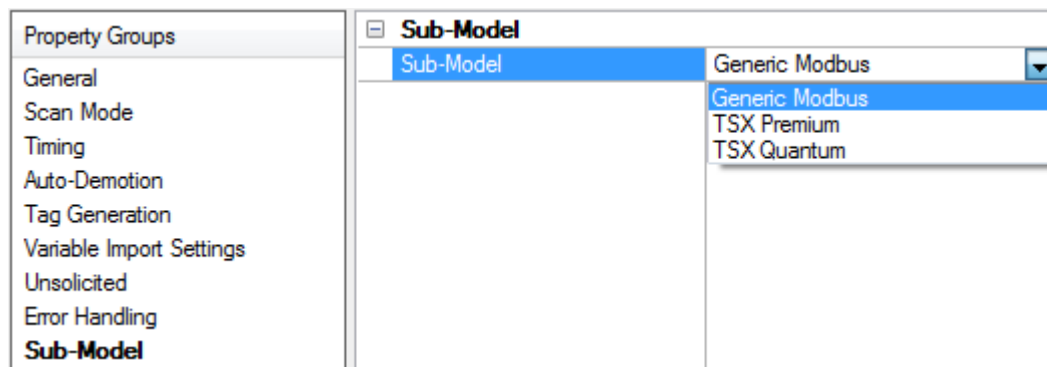
Applicom Sub-Model and Addressing

Applicom devices support three Applicom sub-models. Select the appropriate sub-model for the device being connected. Click on the sub-model link below for address information.

[Generic Modbus](#)

[TSX Premium](#)

[TSX Quantum](#)



Generic Modbus Addressing

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
Blxxxxx	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:

Blxxxxx_cols with assumed row count of 1.

Blxxxxx_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**.

Arrays are supported for internal register locations for all data types except for Boolean and strings.

● **Note:** For server devices, read-only locations are read / write.

Address	Range	Data Type	Access	Function Code
Wlxxxx	0-65535 0-65534 0-65532	Word , Short, BCD Float, DWord, Long, LBCD Double	Read Only	04
Wlxxxx.bb	xxxx=0-65535 bb=0/1-15/16*	Boolean	Read Only	04
Wlxxxx:Xbb	xxxx=0-65535 bb=0/1-15/16*	Boolean	Read Only	04
Dlxxxx	0-65534	DWord	Read Only	04
Flxxxx	0-65534	Float	Read Only	04
Wlxxxx_S	0-65535	Short	Read Only	04
Wlxxxx_B	0-65535	BCD	Read Only	04
Wlxxxx_A**	0-65535	String	Read Only	04
Wlxxxx_X<1, 2, 3>***	0-65535 0-65534	Word , Short, BCD Float, DWord, Long, LBCD	Read Only	04
Dlxxxx_S	0-65534	Long	Read Only	04
Dlxxxx_B	0-65534	LBCD	Read Only	04
Dlxxxx_X<1, 2, 3>***	0-65534	DWord	Read Only	04
Flxxxx_X<1, 2, 3>***	0-65534	Float	Read Only	04
M_Wlxxxx_n(H) String with HiLo Byte Order (H optional)	xxxx=0-65535 n is string length range is 1 to 120 words	String	Read Only	04
M_Wlxxxx_nL String with LoHi Byte Order	xxxx=0-65535 n is string length range is 1 to 120 words	String	Read Only	04

*For more information, refer to Zero-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:

Wlxxxx_cols with assumed row count of 1.

Wlxxxx_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**.

Arrays are supported for holding register locations for all data types except for Boolean and strings.

 **Note:** For server 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	xxxx=0-65535 bb=0/1-15/16*	Boolean	Read / Write	03, 06, 16, 22
Wxxxx:Xbb	xxxx=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)	xxxx=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	xxxx=0-65535 n is string length range is 1 to 120 words	String	Read / Write	03, 16

*For more information, refer to Zero-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 contains 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 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**.

Arrays are supported for internal register locations for all data types except for Boolean and strings.

● **Note:** For server 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
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 Zero-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**.

Arrays are supported for holding register locations for all data types except for Boolean and strings.

● **Note:** For server devices, read-only locations are read / write.

Address	Range	Data Type	Access	Function Code
4xxxxx	1-65536	Word , Short, BCD	Read/Write	03, 06, 16

Address	Range	Data Type	Access	Function Code
	1-65535 1-65533	Float, DWord, Long, LBCD Double		
4xxxx.bb	xxxx=1-65536 bb=0/1-15/16*	Boolean	Read/Write	03, 06, 16, 22
4xxxx:Xbb	xxxx=0-65535 bb=0/1-15/16*	Boolean	Read/Write	03, 06, 16, 22
D4xxxx	1-65535	DWord	Read/Write	03, 06, 16
F4xxxx	1-65535	Float	Read/Write	03, 06, 16
4xxxx_S	1-65536	Short	Read/Write	03, 06, 16
4xxxx_B	1-65536	BCD	Read/Write	03, 06, 16
4xxxx_A**	1-65536	String	Read Only	03, 16
4xxxx_X<1, 2, 3>***	1-65536 1-65535	Word , Short, BCD Float, DWord, Long, LBCD	Read/Write	03, 06, 16
D4xxxx_S	1-65535	Long	Read/Write	03, 06, 16
D4xxxx_B	1-65535	LBCD	Read/Write	03, 06, 16
D4xxxx_X<1, 2, 3>***	1-65535	DWord	Read/Write	03, 06, 16
F4xxxx_X<1, 2, 3>***	1-65535	Float	Read/Write	03, 06, 16
M_4xxxx_n(H) String with HiLo Byte Order (H optional)	xxxx=1-65536 n is string length range is 1 to 120 words	String	Read/Write	03, 16
M_4xxxx_nL String with LoHi Byte Order	xxxx=1-65536 n is string length range is 1 to 120 words	String	Read/Write	03, 16

*For more information, refer to Zero-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.

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 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 contains 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 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**.

Arrays are supported for holding register locations for all data types except for Boolean and strings.

 **Note:** For server 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
%MWxxxx.bb	xxxx=0-65535 bb=0/1-15/16*	Boolean	Read/Write	03, 06, 16, 22
%MWxxxx.Xbb	xxxx=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	0-65534	Float	Read/Write	03, 06, 16

Address	Range	Data Type	Access	Function Code
or %MFxxxxx				
%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 Zero-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 contains 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 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,	Read/Write

Address	Range	Data Type	Access
		LBCD	

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
H4yyyyy	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 **Holding Register Bit 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 client 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:

4xxx[cols] with assumed row count of 1.
4xxx[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 server 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** 300001.2H-365536.240H*** 300001.2L-365536.240L***	Word , Short, BCD Float, DWord, Long, LBCD Double Boolean String String	Read Only Read Only Read Only Read Only Read Only Read Only	04 04 04 04 04 04
Holding Registers	400001-465536 400001-465535 400001-465533 xxxxx=1-65536 bb=0/1-15/16* 400001.2H-465536.240H*** 400001.2L-465536.240L***	Word , Short, BCD Float, DWord, Long, LBCD Double Boolean String String	Read/Write Read/Write Read/Write Read/Write Read/Write Read/Write	03, 06, 16 03, 06, 16 03, 06, 16 03, 06, 16, 22 03, 16 03, 16

*For server devices, read-only locations are read / write.

**For more information, refer to Zero-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 H300001-H30FFFF H300001-H30FFFD yyyyy=1-10000 cc=0/1-F/10 H300001.2H-H3FFFF.240H** H300001.2L-H3FFFF.240L**	Word , Short, BCD Float, DWord, Long, LBCD Double Boolean String String	Read Only Read Only Read Only Read Only Read Only Read Only
Holding Registers	H400001-H410000 H400001-H40FFFF H400001-H40FFFD yyyyy=1-10000 cc=0/1-F/10 H400001.2H-H4FFFF.240H H400001.2L-H4FFFF.240L	Word , Short, BCD Float, DWord, Long, LBCD Double Boolean String String	Read/Write Read/Write Read/Write Read/Write Read/Write Read/Write

*For server 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 `0xxxx#nn`, where:

*For server 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 `0xxxx#nn`, where:

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

The hexadecimal syntax is `H0yyyy#nn`, where:

- `yyyyy` is the address of the first coil (with a range of H000001-H000FFF1).
- `nn` is the number of coils 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 is such that the start address is 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 prevents the driver from reading the register at the specified address. Any attempts by the client to read a write-only tag results in obtaining the last successful write value to the specified address. If no successful writes have occurred, then the client receives 0 / NULL for numeric / string values for an initial value.

⚠ **Caution:** Setting the write-only tags client access privileges to read only causes 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.

🔍 **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 contains 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:

4xxx [rows] [cols]

4xxx [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 tag is read later than expected (based on the specified scan rate). This value does not increase due to a DNR error state. A tag is not counted as late (even if it was) on the initial read after a communications loss. This is by design.
_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.

Item	Data Type	Access	Description
_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 tag is read later than expected (based on the specified scan rate). This value does not increase due to a DNR error state. A tag is not counted as late (even if it was) on the initial read after a communications loss. This is by design.
_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.

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

Event Log Messages

The following information concerns messages posted to the Event Log pane in the main user interface. Consult the OPC server help on filtering and sorting the Event Log detail view. Server help contains many common messages, so should also be searched. Generally, the type of message (informational, warning) and troubleshooting information is provided whenever possible.

 **Tip:** Messages that originate from a data source (such as third-party software, including databases) are presented through the Event Log. Troubleshooting steps should include researching those messages online and in vendor documentation.

Failure to start winsock communications.

Error Type:

Error

Failure to start unsolicited communications.

Error Type:

Error

Possible Cause:

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

Possible Solution:

Verify that the port defined at the channel level is not being used by another application on the system.

Note:

For this driver, the terms Modbus server and unsolicited are used interchangeably.

Unsolicited mailbox access for undefined device. Closing socket. | IP address = '<address>'.

Error Type:

Error

Possible Cause:

1. A device with the specified IP address attempted to send a mailbox message to the server. The message did not pass validation because there is no device with that IP configured in the Mailbox Project.
2. A device with the specified IP address attempted to send a mailbox message to the server. The message did not pass validation because, although a device is configured, there are no clients requesting data from it.

Possible 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 unsupported request received. | IP address = '<address>'.

Error Type:

Error

Possible Cause:

An unsupported request was received from the specified device IP. The format of the request was invalid and not within Modbus specification.

Possible Solution:

Verify that the devices configured to send Mailbox data are sending valid requests.

Unsolicited mailbox memory allocation error. | IP address = '<address>'.

Error Type:

Error

Possible Cause:

1. A device with the specified IP address attempted to send a mailbox message to the server. The message did not pass validation because there is no device with that IP configured in the Mailbox Project.
2. A device with the specified IP address attempted to send a mailbox message to the server. The message did not pass validation because, although a device is configured, there are no clients requesting data from it.

Possible 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.

Unable to create a socket connection.

Error Type:

Error

Possible Cause:

The server was unable to establish a TCP/IP socket connection to the specified device, but will continue to attempt connection.

Possible 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. Verify that a valid gateway is available that allows a connection to the other network.

Error opening file for tag database import. | OS error = '<error>'.

Error Type:

Error

Bad array. | Array range = <start> to <end>.

Error Type:

Error

Possible Cause:

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

Possible Solution:

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

Bad address in block. | Block range = <address> to <address>.

Error Type:

Error

Possible Cause:

The driver attempted to read a location in a PLC that does not exist, perhaps out of range. 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 does not request the specified block of data from the PLC again. Any other addresses being requested from this same block are considered invalid.

Possible Solution:

Update the client application to request addresses within the range of the device.

See Also:

Error Handling

Failed to resolve host. | Host name = '<name>'.

Error Type:

Error

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.

Possible Solution:

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

Specified output coil block size exceeds maximum block size. | Block size specified = <number> (coils), Maximum block size = <number> (coils).

Error Type:

Error

Specified input coil block size exceeds maximum block size. | Block size specified = <number> (coils), Maximum block size = <number> (coils).

Error Type:

Error

Specified internal register block size exceeds maximum block size. | Block size specified = <number> (registers), Maximum block size = <number> (registers).

Error Type:

Error

Specified holding register block size exceeds maximum block size. | Block size specified = <number> (registers), Maximum block size = <number> (registers).

Error Type:

Error

Block request responded with exception. | Block range = <address> to <address>, Exception = <code>.

Error Type:

Warning

Possible Cause:

The device returned an exception code.

Possible Solution:

Consult the exception codes documentation.

See Also:

Modbus Exception Codes

Block request responded with exception. | Block range = <address> to <address>, Function code = <code>, Exception = <code>.

Error Type:

Warning

Possible Cause:

The device returned an exception code.

Possible Solution:

Consult the exception codes documentation.

 See Also:

Modbus Exception Codes

Bad block length received. | Block range = <start> to <end>.

Error Type:

Warning

Possible Cause:

The driver attempted to read a block of memory in the PLC. The PLC responded without an error, but did not provide the driver with the requested block size of data.

Possible Solution:

Ensure that the range of memory exists for the PLC.

Tag import failed due to low memory resources.

Error Type:

Warning

Possible Cause:

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

Possible Solution:

Shut down all unnecessary applications and retry.

File exception encountered during tag import.

Error Type:

Warning

Possible Cause:

The variable import file could not be read.

Possible Solution:

Regenerate the variable import file.

Error parsing record in import file. | Record number = <number>, Field = <field>.

Error Type:

Warning

Possible Cause:

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

Possible Solution:

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

Description truncated for record in import file. | Record number = <number>.

Error Type:

Warning

Possible Cause:

The tag description given in specified record is too long.

Possible Solution:

The driver truncates descriptions as needed. To prevent this error, edit the variable import file to shorten the description.

**Imported tag name is invalid and has been changed. | Tag name = '<tag>',
Changed tag name = '<tag>'.**

Error Type:

Warning

Possible Cause:

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

Possible Solution:

The driver constructs valid names based on the variable import file. To prevent this error and to maintain name consistency, change the name of the exported variable.

A tag could not be imported because the data type is not supported. | Tag name = '<tag>', Unsupported data type = '<type>'.

Error Type:

Warning

Possible Cause:

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

Possible Solution:

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 or manually configure the required tags in the server.

 See Also:

Exporting Variables from Concept

Unable to write to address, device responded with exception. | Address = '<address>', Exception = <code>.

Error Type:

Warning

Possible Cause:

The device returned an exception code.

Possible Solution:

Consult the exception codes documentation.

 See Also:

Modbus Exception Codes

Ethernet Manager started.

Error Type:

Informational

Ethernet Manager stopped.

Error Type:

Informational

Importing tag database. | Source file = '<filename>'.

Error Type:

Informational

A client application has changed the CEG extension via system tag _CEGExtension. | Extension = '<extension>'.

Error Type:

Informational

Possible Cause:

A client application connected to the server changed the CEG extension on the specified device to 0 for Modbus or 1 for CEG.

Possible Solution:

This device property applies only to CEG model devices. Changes do not affect other models. To restrict the client application from changing this property, disable the client's ability to write to system-level tags through the OPC DA settings.

Starting unsolicited communication. | Protocol = '<name>', Port = <number>.

Error Type:

Informational

Created memory for Modbus server device. | Modbus server device ID = <device>.

Error Type:

Informational

All channels are subscribed to a virtual network or all devices are listening to remote addresses, stopping unsolicited communication.

Error Type:

Informational

Channel is in a virtual network, all devices reverted to use one socket per device.

Error Type:

Informational

Cannot change device ID from Modbus client mode to server mode with a client connected.

Error Type:

Informational

Cannot change device ID from Modbus server mode to client mode with a client connected.

Error Type:

Informational

Modbus server mode not allowed when the channel is in a virtual network. The device ID cannot contain a loop-back or local IP address.

Error Type:

Informational

Mailbox model not allowed when the channel is in a virtual network.

Error Type:

Informational

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. 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 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. 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 generates exception 02.
03/0x03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server. 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	SERVER DEVICE FAILURE	An unrecoverable error occurred while the server was attempting to perform the requested action.
05/0x05	ACKNOWLEDGE	The server has accepted the request and is processing it, but a long duration of time is required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed.
06/0x06	SERVER DEVICE BUSY	The server is engaged in processing a long-duration program command. The client should retransmit the message later when the server is free.
07/0x07	NEGATIVE ACKNOWLEDGE	The server 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 client should request diagnostic or error information from the server.
08/0x08	MEMORY PARITY ERROR	The server attempted to read extended memory, but detected a parity error in the memory. The client can retry the request, but service may be required on the server 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 mis-configured 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 server and unsolicited are used interchangeably.

Modbus Ethernet Channel Properties

Below is a full list of all Modbus Ethernet channel-level properties.

```
{
  "common.ALLTYPES_NAME": "MyChannel",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet",
  "servermain.CHANNEL_DIAGNOSTICS_CAPTURE": false,
  "servermain.CHANNEL_UNIQUE_ID": 721923342,
  "servermain.CHANNEL_ETHERNET_COMMUNICATIONS_NETWORK_ADAPTER_STRING": "",
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD": 2,
  "servermain.CHANNEL_WRITE_OPTIMIZATIONS_DUTY_CYCLE": 10,
  "servermain.CHANNEL_NON_NORMALIZED_FLOATING_POINT_HANDLING": 0,
  "servermain.CHANNEL_COMMUNICATIONS_SERIALIZATION_VIRTUAL_NETWORK": 0,
  "servermain.CHANNEL_COMMUNICATIONS_SERIALIZATION_TRANSACTIONS_PER_CYCLE": 1,
  "servermain.CHANNEL_COMMUNICATIONS_SERIALIZATION_NETWORK_MODE": 0,
  "modbus_ethernet.CHANNEL_USE_ONE_OR_MORE_SOCKETS_PER_DEVICE": 1,
  "modbus_ethernet.CHANNEL_MAXIMUM_SOCKETS_PER_DEVICE": 1
}
```

Modbus Ethernet Device Properties

Below is a full list of all Modbus Ethernet device-level properties.

```
{
  "common.ALLTYPES_NAME": "MyDevice",
  "common.ALLTYPES_DESCRIPTION": "",
  "servermain.MULTIPLE_TYPES_DEVICE_DRIVER": "Modbus TCP/IP Ethernet",
  "servermain.DEVICE_MODEL": 0,
  "servermain.DEVICE_UNIQUE_ID": 70949968,
  "servermain.DEVICE_CHANNEL_ASSIGNMENT": "MyChannel",
  "servermain.DEVICE_ID_FORMAT": 0,
  "servermain.DEVICE_ID_STRING": "<0.0.0>.0",
  "servermain.DEVICE_ID_HEXADECIMAL": 0,
  "servermain.DEVICE_ID_DECIMAL": 0,
  "servermain.DEVICE_ID_OCTAL": 0,
  "servermain.DEVICE_DATA_COLLECTION": true,
  "servermain.DEVICE_SIMULATED": false,
  "servermain.DEVICE_SCAN_MODE": 0,
  "servermain.DEVICE_SCAN_MODE_RATE_MS": 1000,
  "servermain.DEVICE_SCAN_MODE_PROVIDE_INITIAL_UPDATES_FROM_CACHE": false,
  "servermain.DEVICE_CONNECTION_TIMEOUT_SECONDS": 3,
  "servermain.DEVICE_REQUEST_TIMEOUT_MILLISECONDS": 1000,
  "servermain.DEVICE_RETRY_ATTEMPTS": 3,
  "servermain.DEVICE_INTER_REQUEST_DELAY_MILLISECONDS": 0,
  "servermain.DEVICE_AUTO_DEMOTION_ENABLE_ON_COMMUNICATIONS_FAILURES": false,
  "servermain.DEVICE_AUTO_DEMOTION_DEMOTE_AFTER_SUCCESSIVE_TIMEOUTS": 3,
  "servermain.DEVICE_AUTO_DEMOTION_PERIOD_MS": 10000,
  "servermain.DEVICE_AUTO_DEMOTION_DISCARD_WRITES": false,
  "servermain.DEVICE_TAG_GENERATION_ON_STARTUP": 0,
  "servermain.DEVICE_TAG_GENERATION_DUPLICATE_HANDLING": 0,
  "servermain.DEVICE_TAG_GENERATION_GROUP": "",
  "servermain.DEVICE_TAG_GENERATION_ALLOW_SUB_GROUPS": true,
  "modbus_ethernet.DEVICE_VARIABLE_IMPORT_FILE": "normal.txt",
  "modbus_ethernet.DEVICE_VARIABLE_IMPORT_INCLUDE_DESCRIPTIONS": 1,
  "modbus_ethernet.DEVICE_DEACTIVATE_TAGS_ON_ILLEGAL_ADDRESS": 1,
  "modbus_ethernet.DEVICE_SUB_MODEL": 1,
}
```

```

"modbus_ethernet.DEVICE_ETHERNET_PORT_NUMBER": 502,
"modbus_ethernet.DEVICE_ETHERNET_IP_PROTOCOL": 1,
"modbus_ethernet.DEVICE_ETHERNET_CLOSE_TCP_SOCKET_ON_TIMEOUT": true,
"modbus_ethernet.DEVICE_ZERO_BASED_ADDRESSING": true,
"modbus_ethernet.DEVICE_ZERO_BASED_BIT_ADDRESSING": true,
"modbus_ethernet.DEVICE_HOLDING_REGISTER_BIT_MASK_WRITES": true,
"modbus_ethernet.DEVICE_MODBUS_FUNCTION_06": true,
"modbus_ethernet.DEVICE_MODBUS_FUNCTION_05": true,
"modbus_ethernet.DEVICE_MODBUS_BYTE_ORDER": true,
"modbus_ethernet.DEVICE_FIRST_WORD_LOW": true,
"modbus_ethernet.DEVICE_FIRST_DWORD_LOW": true,
"modbus_ethernet.DEVICE_MODICON_BIT_ORDER": false,
"modbus_ethernet.DEVICE_TREAT_LONGS_AS_DOUBLE_PRECISION_UNSIGNED_DECIMAL": false,
"modbus_ethernet.DEVICE_OUTPUT_COILS": 32,
"modbus_ethernet.DEVICE_INPUT_COILS": 32,
"modbus_ethernet.DEVICE_INTERNAL_REGISTERS": 32,
"modbus_ethernet.DEVICE_HOLDING_REGISTERS": 32,
"modbus_ethernet.DEVICE_PERFORM_BLOCK_READ_ON_STRINGS": 0
}

```

 **Note:** The servermain.DEVICE_MODEL parameter defaults to the generic Modbus model. If this is not desired, ensure this parameter is defined correctly.

Modbus Ethernet Tag Properties

Below is a full list of all Modbus Ethernet tag properties.

```

{
"common.ALLTYPES_NAME": "MyTag",
"common.ALLTYPES_DESCRIPTION": "",
"servermain.TAG_ADDRESS": "400001",
"servermain.TAG_DATA_TYPE": 5,
"servermain.TAG_READ_WRITE_ACCESS": 1,
"servermain.TAG_SCAN_RATE_MILLISECONDS": 100,
"servermain.TAG_AUTOGENERATED": false,
"servermain.TAG_SCALING_TYPE": 0,
"servermain.TAG_SCALING_RAW_LOW": 0,
"servermain.TAG_SCALING_RAW_HIGH": 1000,
"servermain.TAG_SCALING_SCALED_DATA_TYPE": 9,
"servermain.TAG_SCALING_SCALED_LOW": 0,
"servermain.TAG_SCALING_SCALED_HIGH": 1000,
"servermain.TAG_SCALING_CLAMP_LOW": false,
"servermain.TAG_SCALING_CLAMP_HIGH": false,
"servermain.TAG_SCALING_NEGATE_VALUE": false,
"servermain.TAG_SCALING_UNITS": ""
}

```

Index

A

A client application has changed the CEG extension via system tag _CEGExtension. | Extension = '<extension>'. 51

A tag could not be imported because the data type is not supported. | Tag name = '<tag>', Unsupported data type = '<type>'. 50

Accumulation 40

Address 27

Address Descriptions 31

All channels are subscribed to a virtual network or all devices are listening to remote addresses, stopping unsolicited communication. 51

Allow Sub Groups 17

Applicom 26

Applicom Addressing 32

Array Support 32-34, 38-39, 43

Arrays 41

Attempts Before Timeout 15

Auto-Demotion 15

Automatic Tag Database Generation 27

B

Bad address in block. | Block range = <address> to <address>. 47

Bad array. | Array range = <start> to <end>. 47

Bad block length received. | Block range = <start> to <end>. 49

BCD 30

Block Read Strings 23

Block request responded with exception. | Block range = <address> to <address>, Exception = <code>. 48

Block request responded with exception. | Block range = <address> to <address>, Function code = <code>, Exception = <code>. 49

Block Sizes 22

BOOL 27

Boolean 30

BYTE 27

Byte Switching Suffixes 35, 39

C

Cannot change device ID from Modbus client mode to server mode with a client connected. 51

Cannot change device ID from Modbus server mode to client mode with a client connected. 52

CEG 26

CEG Addressing 40

CEG Extension 21
CEGExtension 31
Channel-Level Settings 10
Channel Assignment 13
Channel is in a virtual network, all devices reverted to use one socket per device. 51
Channel Properties – Advanced 10
Channel Properties – Communication Serialization 10
Channel Properties – Ethernet Communications 9
Channel Properties – General 8
Channel Properties – Write Optimizations 9
Close Socket on Timeout 20
Comment 27
Communications Timeout 18
Communications Timeouts 14
Connect Timeout 15
Create 17
Created memory for Modbus server device. | Modbus server device ID = <device>. 51
CSV 27
Custom tags 27

D

Data Access 20
Data Collection 13
Data Encoding 21
Data Types Description 30
Deactivate Tags on Illegal Address 19
Decimal Addressing 40
Delete 17
Demote on Failure 15
Demotion Period 16
Description 13
Description truncated for record in import file. | Record number = <number>. 50
Device Properties – Auto-Demotion 15
Device Properties – Redundancy 23
Device Properties – Tag Generation 16
Device Properties – Timing 14
Diagnostics 8, 44
DINT 27
Discard Requests when Demoted 16
Do Not Scan, Demand Poll Only 14
Double 30
Driver 13
Driver System Tag Addressing 31

Duty Cycle 9

DWord 30

E

Enumerations 26

Error Handling 19

Error opening file for tag database import. | OS error = '<error>'. 47

Error parsing record in import file. | Record number = <number>, Field = <field>. 49

Ethernet 11, 19

Ethernet Manager started. 51

Ethernet Manager stopped. 51

Ethernet Settings 9

Ethernet to Modbus Plus Bridge 6

Event Log Messages 46

F

Failed to resolve host. | Host name = '<name>'. 48

Failure to start unsolicited communications. 46

Failure to start winsock communications. 46

File exception encountered during tag import. 49

First DWord Low 21

First Word Low 21

Five-Digit Addressing 41

Float 30

Floats 40, 44

Fluenta 6

Fluenta Addressing 40

Fluenta FGM 26

Force Multiple Coils 32

Force Single Coil 32

Function Codes Description 31

G

General 13

Generate 16

Generic Modbus Addressing 32

Global Settings 11

H

Help Contents 5
Hexadecimal Addressing 41
Holding Register Bit Writes 20
Holding Registers 23, 34, 36, 38, 42
HoldingRegisterBlockSize 31

I

ID 13
Identification 8
Imported tag name is invalid and has been changed. | Tag name = '<tag>', Changed tag name = '<tag>'. 50
Importing from Custom Applications 27
Importing tag database. | Source file = '<filename>'. 51
Include Descriptions 17
Initial Updates from Cache 14
Input Coils 23, 32, 35, 42
InputCoilBlockSize 31
Instromet 6, 26
Instromet Addressing 40
INT 27
Inter-Device Delay 10
Internal Registers 23, 33, 35, 42
Internal Tags 31
InternalRegisterBlockSize 31
IP Protocol 12, 20

L

LBCD 30
Load Balanced 11
Long 30
Long Integers 40

M

Mailbox 6, 26
Mailbox Addressing 40
Mailbox Client Privileges 21
Mailbox Mode 43
Mailbox model not allowed when the channel is in a virtual network. 52
Masked Write Register 32

Max Sockets per Device 11
Modbus Addressing 41
Modbus Byte Order 21
Modbus Client 6
Modbus Client & Server Considerations 18
Modbus Exception Codes 53
Modbus Function 05 21
Modbus Function 06 20
Modbus Mailbox 41
Modbus server mode not allowed when the channel is in a virtual network. The device ID cannot contain a loop-back or local IP address. 52
Modbus Unsolicited 6
Model 13
Models 6
Modicon Bit Order 21

N

Name 13
Network 1 - Network 500 10
Network Adapter 9
Network Mode 11
Non-Normalized Float Handling 10

O

On Device Startup 16
On Duplicate Tag 17
On Property Change 16
OPC Quality Bad 18
Optimization Method 9
Optimizing Modbus Ethernet Communications 28
Output 40
Output Coils 23, 32, 35, 38, 42
OutputCoilBlockSize 31
Overview 5
Overwrite 17

P

Parent Group 17
Port 12, 20, 31
Preset Multiple Registers 32

Preset Single Register 32

Priority 11

R

Read Coil Status 31

Read Holding Registers 32

Read Input Status 31

Read Internal Registers 32

REAL 27

Record 27

Redundancy 23

Replace with Zero 10

Request Timeout 15

Respect Tag-Specified Scan Rate 14

Roxar 7

Roxar Addressing 44

Roxar RFM 26

S

Scan Mode 14

Service 40

Set Value 27

Settings 20

Setup 7

Short 30

Short Integers 40, 44

Simulated 13

Six-Digit Addressing 41

Socket Usage 11, 28

Specified holding register block size exceeds maximum block size. | Block size specified = <number> (registers), Maximum block size = <number> (registers). 48

Specified input coil block size exceeds maximum block size. | Block size specified = <number> (coils), Maximum block size = <number> (coils). 48

Specified internal register block size exceeds maximum block size. | Block size specified = <number> (registers), Maximum block size = <number> (registers). 48

Specified output coil block size exceeds maximum block size. | Block size specified = <number> (coils), Maximum block size = <number> (coils). 48

Starting unsolicited communication. | Protocol = '<name>', Port = <number>. 51

Statistics Items 44

String 30

STRING 27

String Support 34, 39, 43

Supported 6

System 40
System Tags 31

T

Tag Counts 8
Tag Generation 16
Tag import failed due to low memory resources. 49
TIME 27
Timeouts to Demote 15
Timing 14
Transactions per Cycle 11
Treat Longs as Decimals 22
TSX Premium 38
TSX Quantum 35

U

UDINT 27
UINT 27
Unable to create a socket connection. 47
Unable to write to address, device responded with exception. | Address = '<address>', Exception = <code>. 50
Unmodified 10
Unsolicited 18
Unsolicited mailbox access for undefined device. Closing socket. | IP address = '<address>'. 46
Unsolicited mailbox memory allocation error. | IP address = '<address>'. 47
Unsolicited mailbox unsupported request received. | IP address = '<address>'. 46
User 40

V

Variable 27
Variable Import Settings 17
Virtual Network 10

W

Word 30
WORD 27
Write-Only Access 43
Write All Values for All Tags 9
Write Only Latest Value for All Tags 9
Write Only Latest Value for Non-Boolean Tags 9

Z

Zero-Based Addressing 20

Zero-Based Bit Addressing 20