Modbus Serial Driver

Table of Contents

Modbus Serial Driver

Help version 1.083

CONTENTS

Overview

The Modbus Serial Driver provides a reliable way to connect Modbus serial devices to OPC client applications, including HMI, SCADA, Historian, MES, ERP, and countless custom applications. It is intended for use with serial devices that support the Modbus RTU protocol. The Modbus Serial Driver has been developed to support a wide range of Modbus RTU compatible devices.

Setup

Communication Protocol

Modbus RTU Protocol

Supported Communication Parameters

Baud Rate: 1200, 2400, 9600, and 19200
Parity: Odd, Even, and None
Data Bits: 8
Stop Bits: 1 and 2

Supported Devices

- Modbus-compatible devices
- Elliott Flow Computer
- Magnetek GPD 515 Drive
- Omni Flow Computer
- Daniel S500 Flow Computer

- Dynamic Fluid Meter (DFM) SFC3
- TSXCUSBMBP USB Adapter

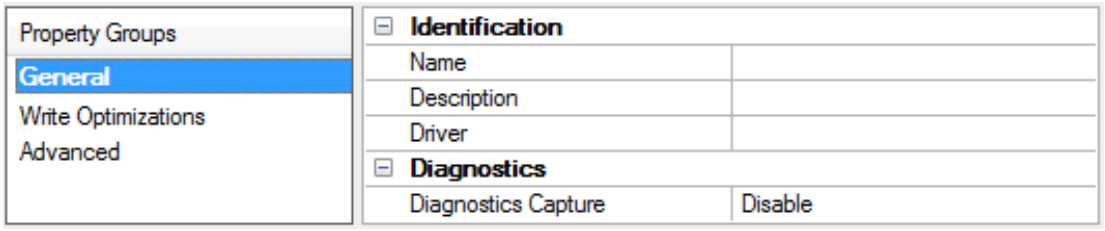
Supported Maximums

The maximum number of channels supported by this driver is 2048.
The maximum number of devices supported per channel is 255.

Note: Not all of the listed configurations may be supported in every device.

Channel Properties — General

This server supports the use of simultaneous multiple 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.



Identification

Name: 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: User-defined information about this channel.
Many of these properties, including Description, have an associated system tag.

Driver: Selected protocol / driver for this channel. This property specifies 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. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to 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.

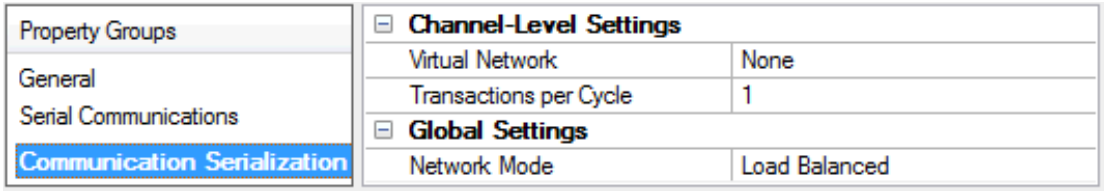
Note: This property is not available if the driver does not support diagnostics.

For more information, refer to "Communication Diagnostics" and "Statistics Tags" in the server help.

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 master radio. All channels using the same master 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.



Channel-Level Settings

Virtual Network: This property specifies 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: This property specifies 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):
  - Channels with pending writes have the highest priority.
  - Channels with pending explicit reads (through internal plug-ins or external client interfaces)

are prioritized based on the read's priority.
- Scanned reads and other periodic events (driver specific).

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

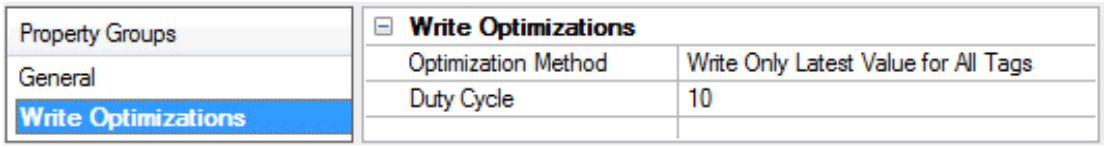
Devices that rely on unsolicited responses should not be placed in a virtual network. In situations where communications must be serialized, it is recommended that Auto-Demotion be enabled.

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 — Write Optimizations

As with any server, writing data to the device may be the application's most important aspect. The server intends to ensure that the data written from the client application gets to the device on time. Given this goal, the server provides optimization properties that can be used to meet specific needs or improve application responsiveness.



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.

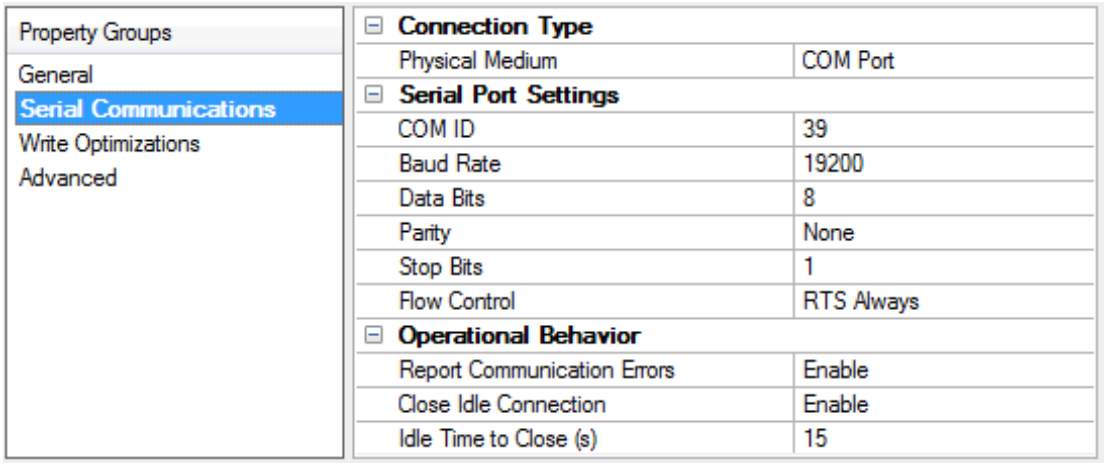
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write 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 — Serial Communications

Serial communication properties are available to serial drivers and vary depending on the driver, connection type, and options selected. Below is a superset of the possible properties.
Click to jump to one of the sections: **Connection Type**, **Serial Port Settings** or **Ethernet Settings**, and **Operational Behavior**.

Note: With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features, as changes made to these properties can temporarily disrupt communications.



Connection Type

Physical Medium: Choose the type of hardware device for data communications. Options include COM Port, None, Modem, and Ethernet Encapsulation. The default is COM Port.

- **None**: Select None to indicate there is no physical connection, which displays the **Operation with no Communications** section.
- **COM Port**: Select Com Port to display and configure the **Serial Port Settings** section.
- **Modem**: Select Modem if phone lines are used for communications, which are configured in the **Modem Settings** section.

- **Ethernet Encap.**: Select if Ethernet Encapsulation is used for communications, which displays the Ethernet Settings section.
- **Shared**: Verify the connection is correctly identified as sharing the current configuration with another channel. This is a read-only property.

Serial Port Settings

COM ID: Specify the Communications ID to be used when communicating with devices assigned to the channel. The valid range is 1 to 9991 to 16. The default is 1.

Baud Rate: Specify the baud rate to be used to configure the selected communications port.

Data Bits: Specify the number of data bits per data word. Options include 5, 6, 7, or 8.

Parity: Specify the type of parity for the data. Options include Odd, Even, or None.

Stop Bits: Specify the number of stop bits per data word. Options include 1 or 2.

Flow Control: Select how the RTS and DTR control lines are utilized. Flow control is required to communicate with some serial devices. Options are:

- **None**: This option does not toggle or assert control lines.
- **DTR**: This option asserts the DTR line when the communications port is opened and remains on.
- **RTS**: This option specifies that the RTS line is high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line is low. This is normally used with RS232/RS485 converter hardware.
- **RTS, DTR**: This option is a combination of DTR and RTS.
- **RTS Always**: This option asserts the RTS line when the communication port is opened and remains on.
- **RTS Manual**: This option asserts the RTS line based on the timing properties entered for RTS Line Control. It is only available when the driver supports manual RTS line control (or when the properties are shared and at least one of the channels belongs to a driver that provides this support).
  RTS Manual adds an **RTS Line Control** property with options as follows:
  - **Raise**: This property specifies the amount of time that the RTS line is raised prior to data transmission. The valid range is 0 to 9999 milliseconds. The default is 10 milliseconds.
  - **Drop**: This property specifies the amount of time that the RTS line remains high after data transmission. The valid range is 0 to 9999 milliseconds. The default is 10 milliseconds.
  - **Poll Delay**: This property specifies the amount of time that polling for communications is delayed. The valid range is 0 to 9999. The default is 10 milliseconds.

Tip: When using two-wire RS-485, "echoes" may occur on the communication lines. Since this communication does not support echo suppression, it is recommended that echoes be disabled or a RS-485 converter be used.

Operational Behavior

- **Report Communication Errors**: Enable or disable reporting of low-level communications errors. When enabled, low-level errors are posted to the Event Log as they occur. When disabled, these same errors are not posted even though normal request failures are. The default is Enable.

- **Close Idle Connection**: Choose to close the connection when there are no longer any tags being referenced by a client on the channel. The default is Enable.
- **Idle Time to Close**: Specify the amount of time that the server waits once all tags have been removed before closing the COM port. The default is 15 seconds.

Ethernet Settings

Note: Not all serial drivers support Ethernet Encapsulation. If this group does not appear, the functionality is not supported.

Ethernet Encapsulation provides communication with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port that converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted, users can connect standard devices that support serial communications to the terminal server. The terminal server's serial port must be properly configured to match the requirements of the serial device to which it is attached. *For more information, refer to "How To... Use Ethernet Encapsulation" in the server help.*

- **Network Adapter**: Indicate a network adapter to bind for Ethernet devices in this channel. Choose a network adapter to bind to or allow the OS to select the default.
  Specific drivers may display additional Ethernet Encapsulation properties. For more information, refer to Channel Properties — Ethernet Encapsulation.

Modem Settings

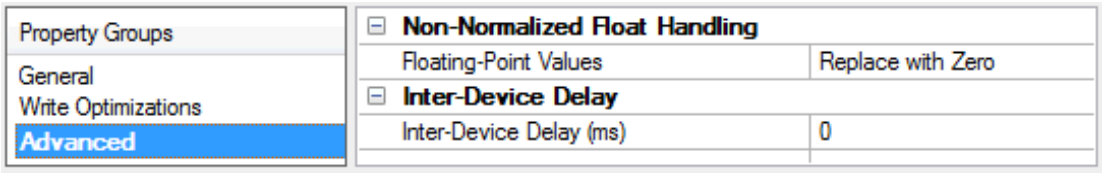
- **Modem**: Specify the installed modem to be used for communications.
- **Connect Timeout**: Specify the amount of time to wait for connections to be established before failing a read or write. The default is 60 seconds.
- **Modem Properties**: Configure the modem hardware. When clicked, it opens vendor-specific modem properties.
- **Auto-Dial**: Enables the automatic dialing of entries in the Phonebook. The default is Disable. *For more information, refer to "Modem Auto-Dial" in the server help.*
- **Report Communication Errors**: Enable or disable reporting of low-level communications errors. When enabled, low-level errors are posted to the Event Log as they occur. When disabled, these same errors are not posted even though normal request failures are. The default is Enable.
- **Close Idle Connection**: Choose to close the modem connection when there are no longer any tags being referenced by a client on the channel. The default is Enable.
- **Idle Time to Close**: Specify the amount of time that the server waits once all tags have been removed before closing the modem connection. The default is 15 seconds.

Operation with no Communications

- **Read Processing**: Select the action to be taken when an explicit device read is requested. Options include Ignore and Fail. Ignore does nothing; Fail provides the client with an update that indicates failure. The default setting is Ignore.

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.



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 not available 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.

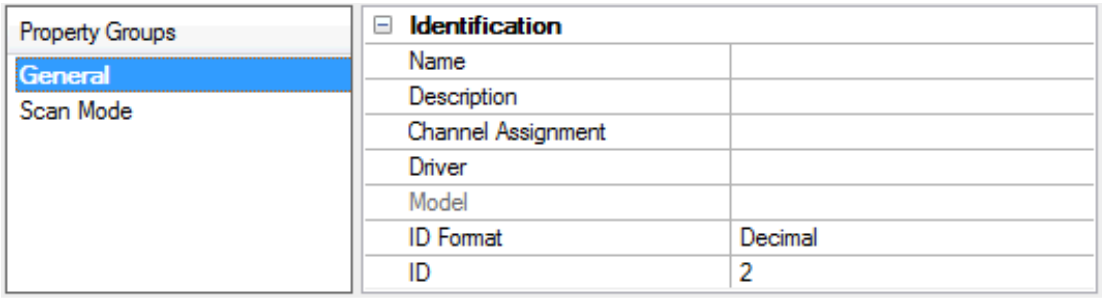
For more information on the floating point values, refer to "How To ... Work with Non-Normalized Floating Point Values" in the server help.

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.

Device Properties — General

A device represents a single target on a communications channel. If the driver supports multiple controllers, users must enter a device ID for each controller.



Identification

Name: This property specifies the name of the device. It is a logical user-defined name that can be up to 256 characters long, and may be used on multiple channels.

Note: Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name

become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".
For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.

Description: User-defined information about this device.
Many of these properties, including Description, have an associated system tag.

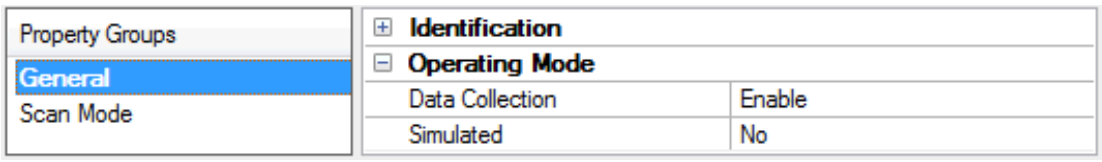
Channel Assignment: User-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

Model: This property specifies the specific type of device that is associated with this ID. The contents of the drop-down menu depends on the type of communications driver being used. Models that are not supported by a driver are disabled. If the communications driver supports multiple device models, the model selection can only be changed when there are no client applications connected to the device.
Note: If the communication driver supports multiple models, users should try to match the model selection to the physical device. If the device is not represented in the drop-down menu, select a model that conforms closest to the target device. Some drivers support a model selection called "Open," which allows users to communicate without knowing the specific details of the target device. For more information, refer to the driver help documentation.

ID: This property specifies the device's driver-specific station or node. The type of ID entered depends on the communications driver being used. For many communication drivers, the ID is a numeric value. Drivers that support a Numeric ID provide users with the option to enter a numeric value whose format can be changed to suit the needs of the application or the characteristics of the selected communications driver. The format is set by the driver by default. Options include Decimal, Octal, and Hexadecimal.
Note: If the driver is Ethernet-based or supports an unconventional station or node name, the device's TCP/IP address may be used as the device ID. TCP/IP addresses consist of four values that are separated by periods, with each value in the range of 0 to 255. Some device IDs are string based. There may be additional properties to configure within the ID field, depending on the driver. *For more information, refer to the driver's help documentation.*

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.

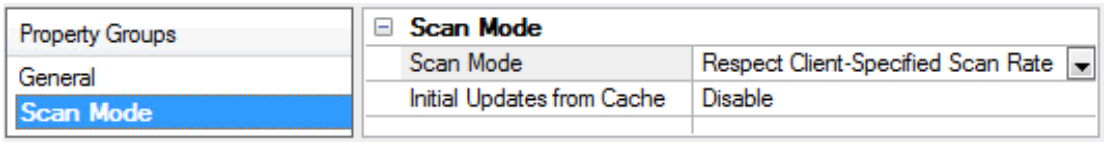
Notes:

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.

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.



Scan Mode: Specifies 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 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.

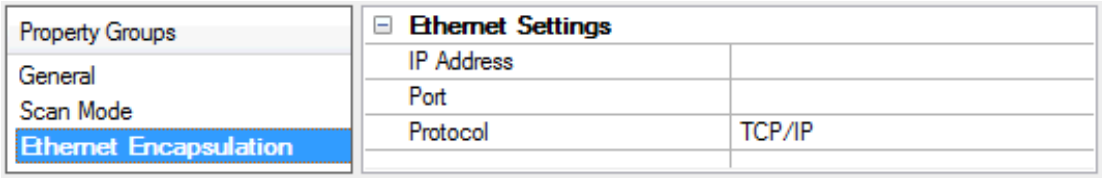
Initial Updates from Cache: When enabled, this option allows the server to provide the first updates for newly activated tag references from stored (cached) data. Cache updates can only be provided when the new item reference shares the same address, scan rate, data type, client access, and scaling properties. 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 — Ethernet Encapsulation

Ethernet Encapsulation is designed to provide communication with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port. The terminal server

converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted to a serial form, users can connect standard devices that support serial communications to the terminal server.

- *For more information, refer to "How to... Use Ethernet Encapsulation" in server help.*
- Ethernet Encapsulation is transparent to the driver; configure the remaining properties as if connecting to the device directly on a local serial port.



IP Address: This property is used to enter the four-field IP address of the terminal server to which the device is attached. IPs are specified as YYY.YYY.YYY.YYY. The YYY designates the IP address: each YYY byte should be in the range of 0 to 255. Each serial device may have its own IP address; however, devices may have the same IP address if there are multiple devices multi-dropped from a single terminal server.

Port: This property is used to configure the Ethernet port to be used when connecting to a remote terminal server.

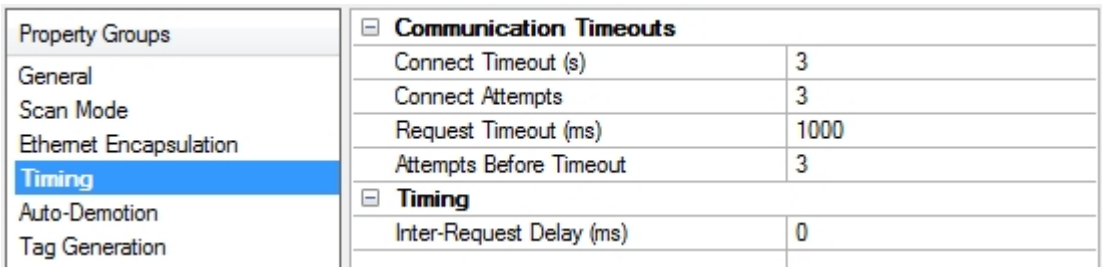
Protocol: This property is used to select either TCP/IP or UDP communications. The selection depends on the nature of the terminal server being used. The default protocol selection is TCP/IP. For more information on available protocols, refer to the terminal server's help documentation.

- **Notes**

1. With the server's online full-time operation, these properties can be changed at any time. Utilize the User Manager to restrict access rights to server features and prevent operators from changing the properties.
2. The valid IP Address range is greater than (>) 0.0.0.0 to less than (<) 255.255.255.255.

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.



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.

Connect Attempts: This property (which is used primarily by some Ethernet Encapsulation based drivers) limits the number of times a connection between the driver and the target device can be attempted. If the limit is reached, the connection request has failed. The Connect Timeout property specifies the time interval between connection attempts. The valid range is 1 to 10 attempts. The default is 3 attempts. If this setting is not supported by the driver, it is disabled.

Request Timeout: This property specifies 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 9,999,999 milliseconds (167.6667 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: This property specifies 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.

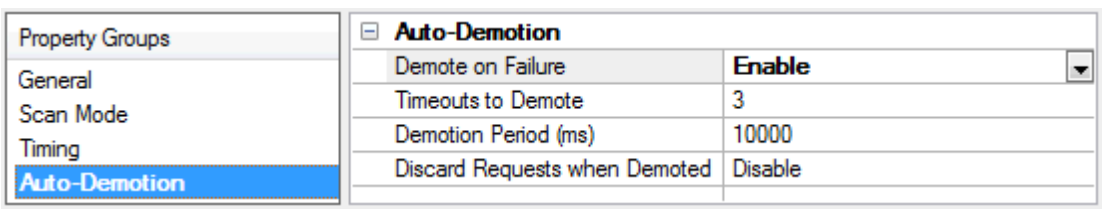
Timing

Inter-Request Delay: This property specifies 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.

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.



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

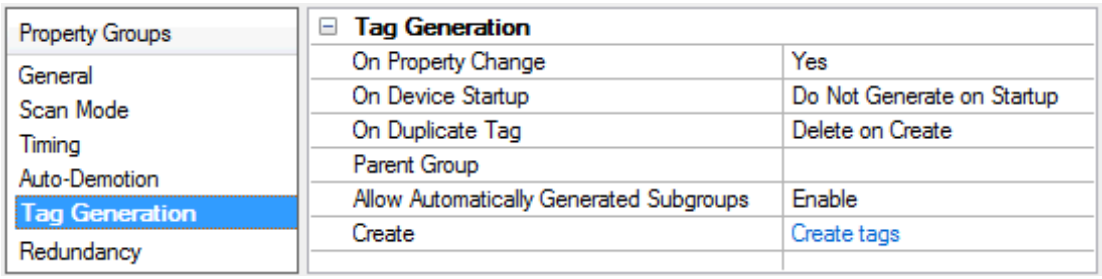
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the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



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: This property specifies 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.

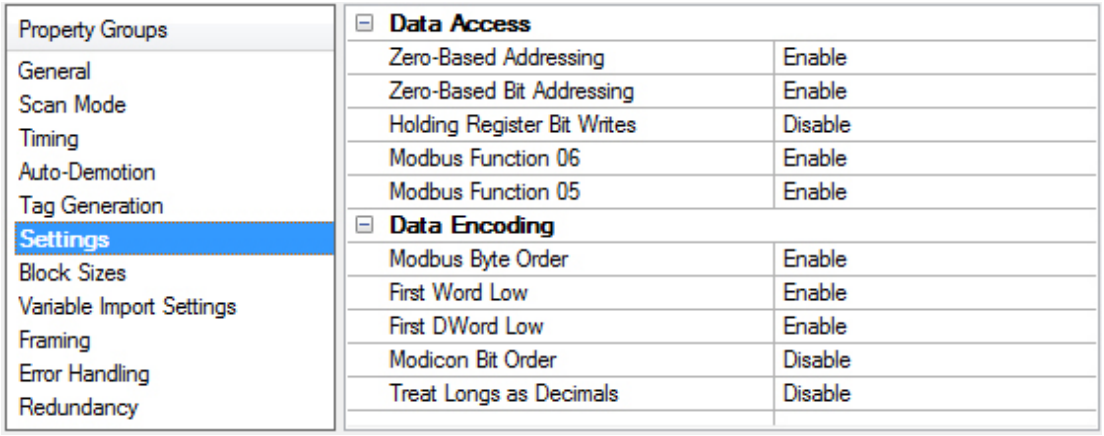
Allow Automatically Generated Subgroups: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting 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 — Settings



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 Mask: 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 setting 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 Mask property takes precedence over this option. If Holding Register Bit Mask 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.

Data Encoding

Modbus Byte Order: sets the data encoding of each register / 16-bit value. The byte order 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.

Note: This setting does not apply to the Omni model. It always uses Modbus byte order.

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.

Note: This setting does not apply to the Omni model. It always uses Modbus byte order.

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.

Note: This setting does not apply to the Omni model. It always uses Modbus byte order.

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

Tips on Settings

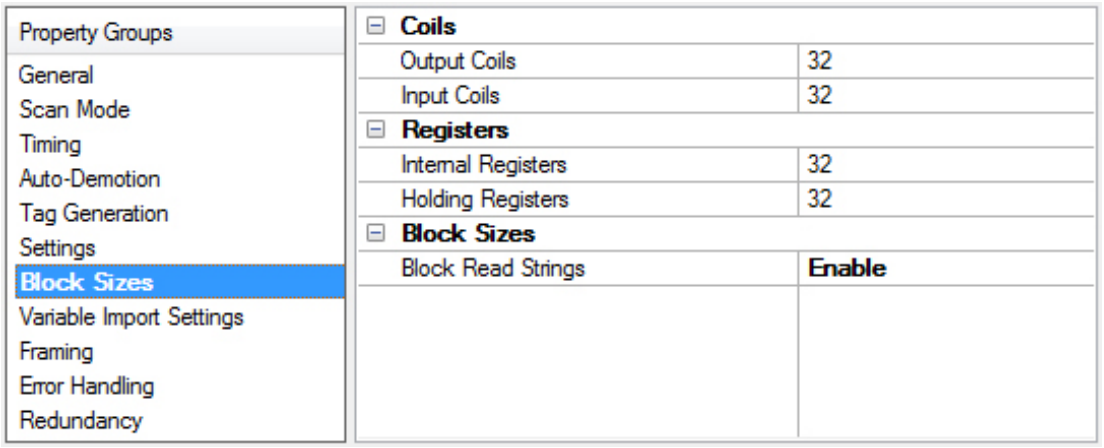
Data Types	Modbus Byte Order	First Word Low	First DWord Low
Word, Short, BCD	Applicable	N/A	N/A
Float, DWord, Long, LBCD	Applicable	Applicable	N/A
Double	Applicable	Applicable	Applicable

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) High Word (63..48) of Double Word in 64-bit data types	Low Word (15..0) Low Word (47..32) of Double Word in 64-bit data types
First Word Low	Low Word (15..0) Low Word (47..32) of Double Word in 64-bit data types	High Word (31..16) High Word (63..48) of Double Word in 64-bit data types
First DWord Low	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



Coils

Output Coils: Specifies the output block size in bits. Coils can be read from 8 to 2000 points (bits) at a time. A higher block size means more points are read from the device in a single request. The block size can be reduced to read data from non-contiguous locations within the device. The default setting is 32.

Input Coils: Specifies the input block size in bits. Coils can be read from 8 to 2000 points (bits) at a time. A higher block size means more points are read from the device in a single request. The block size can be reduced to read data from non-contiguous locations within the device. The default setting is 32.

Registers

Internal Registers: Specifies the internal register block size in bits. From 1 to 125 standard 16-bit Modbus registers can be read at a time. A higher block size means more register values are read from the device in a single request. The block size can be reduced to read data from non-contiguous locations within the device. The default setting is 32.

Holding Registers: Specifies the holding register block size in bits. From 1 to 125 standard 16-bit Modbus registers can be read at a time. A higher block size means more register values are read from the device in a single request. The block size can be reduced to read data from non-contiguous locations within the device. The default setting is 32.

Caution: 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.

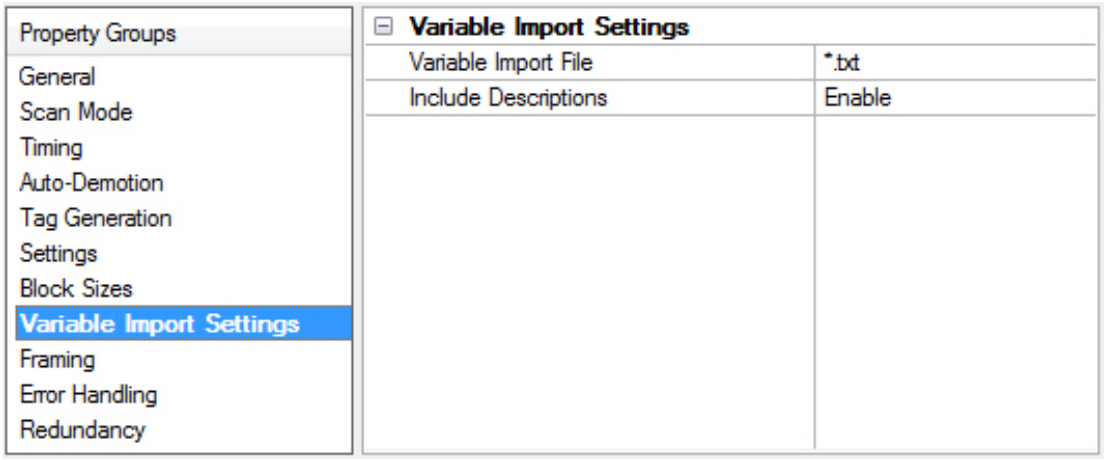
Block Sizes

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. The default setting is disabled.

Device Properties — Variable Import Settings

The Variable Import Settings parameters specify the location of the variable import file to be used for Automatic Tag Database Generation.

*For more information on CSV files for Modbus Drivers, refer to **Creating CSV Files for Modbus Drivers**.*



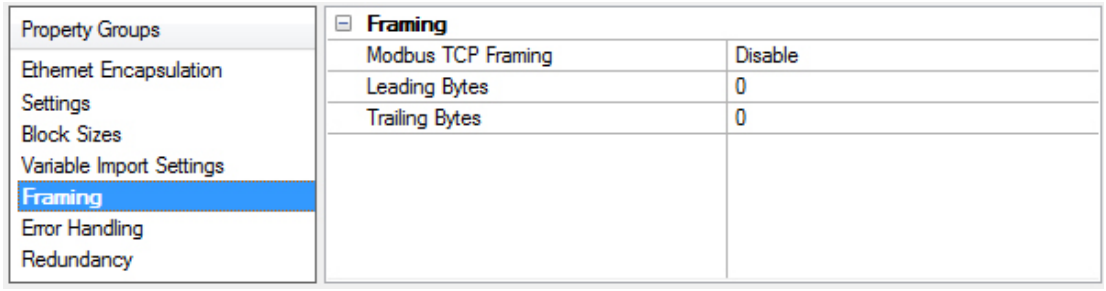
Variable Import File: This parameter is used to browse to the exact location of the variable import file to use for Automatic Tag Database Generation.

Include Descriptions: When enabled, imported tag descriptions are used if present in the 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 — Framing

Some terminal server devices add additional data to Modbus frames; as such, the Framing parameters can be used to configure the driver to ignore the additional bytes in response messages.



Framing

Modbus TCP Framing: Select **Enable** if the driver should use Modbus TCP frames with MBAP headers. The default is disabled.

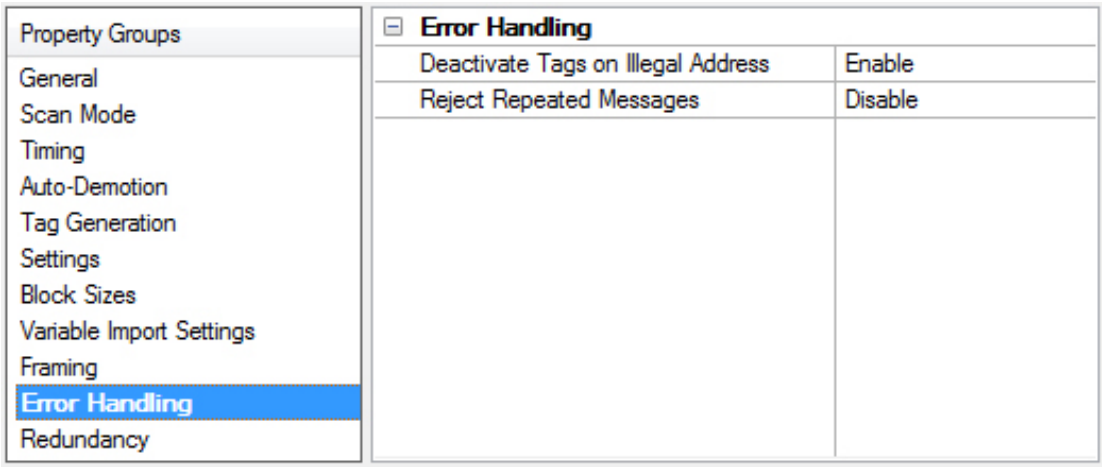
Tip: This setting should be enabled when communicating with native Modbus TCP devices.

Leading Bytes: Specify the number of bytes to be attached to the beginning of Modbus responses. Values may range from 0 to 8.

Trailing Bytes: Specify the number of bytes to be attached to the end of Modbus responses. Values may range from 0 to 8.

Device Properties — Error Handling

The error handling parameters determine how to deal with errors from the device.

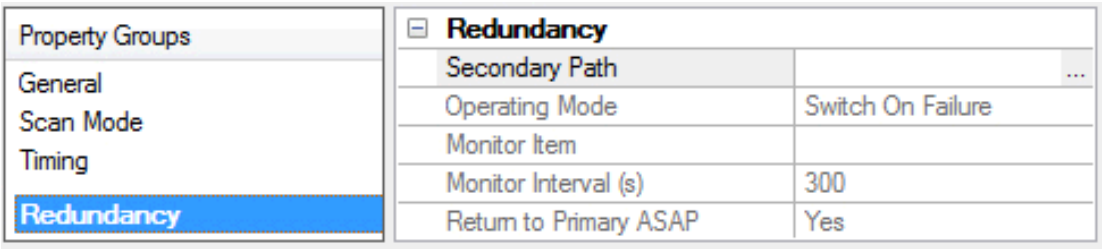


Deactivate Tags on Illegal Address: When enabled, the driver stops polling for a block of data if the device returns Modbus exception code 2 (illegal address) or 3 (illegal data, such as number of points) in response to a read of that block. When disabled, the driver continues to poll that data block. The default setting is enabled.

Reject Repeated Messages: When enabled, the driver interprets a repeated message as an invalid response and retries the request. The default setting is enabled. When disabled, the driver expects repeated messages.

Note: Some message-relay equipment echoes Modbus requests back to the driver.

Device Properties — Redundancy



Redundancy is available with the Media-Level Redundancy Plug-In.
Consult the website, a sales representative, or the user manual for more information.

Automatic Tag Database Generation

The Modbus Serial Driver makes use of automatic tag database generation, which enables drivers to automatically create tags that access data points used by the device's ladder program. While it is sometimes possible to query a device for the information needed to build a tag database, this driver must use a **Variable Import File** instead. Variable import files can be generated using the Concept and ProWORX device programming applications.

Creating the Variable Import File

The import file must be in semicolon-delimited text .TXT format, which is the default export file format of the many device programming applications.

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

This driver requires additional settings in addition to the basic settings that are common to all drivers that support automatic tag database generation. The specialized settings include the name and location of the variable import file, which can be specified during the Variable Import Settings step of the Device Wizard or later by selecting **Device Properties**| **Variable Import Settings**.

For more information, refer to ***Variable Import Settings****.*

Server Configuration

Automatic tag database generation can be customized to fit an application's specific needs. The primary control options can be set during the Database Creation step of the Device Wizard or later by selecting **Device Properties** | **Tag Generation**.

Operation

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 show when the tag generation process started, any errors that occurred while processing the variable import file, and when the process completed.

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

Item	Data Type	Access	Description
			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).

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

Item	Data Type	Access	Description
			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).

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

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

Data Type	Description
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 HiLo LoHi byte order, 8 Byte and 16 Byte Omni Flow Computer string data.
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 first DWord low data handling of 64-bit data types, and first word low data handling of 32-bit data types.

Address Descriptions

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

Modbus Addressing
Magnetek GPD 515 Drive Addressing
Elliott Flow Computer Addressing
Daniels S500 Flow Computer Addressing
Dynamic Fluid Meter Addressing
Omni Flow Computer Addressing
Statistics

See Also: **Function Codes Description**

Modbus Addressing

The default data types for dynamically defined tags are shown in **bold**. The Function Codes are displayed in decimal.

*For more information, refer to **Function Codes Description**.*

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

Address	Range	Data Type	Access*	Function Code
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 3xxxxx.0/1- 3xxxxx.15/16***	**Word**, Short, BCD Float, DWord, Long, LBCD Double Boolean	Read Only	04
Internal Registers As String with HiLo Byte Order	300001.2H-365536.240H .Bit is string length, range 2 to 240 bytes.	**String****	Read Only	04
Internal Registers As String with LoHi Byte Order	300001.2L-365536.240L .Bit is string length, range 2 to 240 bytes.	**String****	Read Only	04
Holding Registers	400001-465536 400001-465535 400001-465533 4xxxxx.0/1- 4xxxxx.15/16***	**Word**, Short, BCD Float, DWord, Long, LBCD Double	Read/Write	03, 06, 16 03, 06, 16,

Address	Range	Data Type	Access*	Function Code
		Boolean		22
Holding Registers As String with HiLo Byte Order	400001.2H-465536.240H .Bit is string length, range 2 to 240 bytes.	**String****	Read/Write	03, 16
Holding Registers As String with LoHi Byte Order	400001.2L-465536.240L .Bit is string length, range 2 to 240 bytes.	**String****	Read/Write	03, 16

*All Read/Write addresses may be set as Write Only by prefixing a "W" to the address such as "W40001." This 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, the client receives 0/NULL for numeric/string values for an initial value.

Caution: Setting the Client Access privileges of Write Only tags to Read Only causes writes to these tags to fail and the client to always receive 0/NULL for numeric/string values.

- ***For more information, refer to **String Support**.*
- ****For more information, refer to Zero-Based Bit Addressing in **Settings**.*

Modbus Addressing in Hexadecimal Format

Address	Range	Data Type	Access	Function Code
Output Coils	H000001-H0FFFF	**Boolean**	Read/Write	01, 05, 15
Input Coils	H100001-H1FFFF	**Boolean**	Read Only	02
Internal Registers	H300001-H310000 H300001-H3FFFF H300001-H3FFFD H3xxxxx.0/1-H3xxxxx.F/10*	**Word**, Short, BCD Float, DWord, Long, LBCD Double Boolean	Read Only	04
Internal Registers As String with HiLo Byte Order	H300001.2H-H3FFFF.240H. Bit is string length, range 2 to 240 bytes.	**String****	Read Only	04
Internal Registers As String with LoHi Byte Order	H300001.2L-H3FFFF.240L. Bit is string length, range 2 to 240 bytes.	**String****	Read Only	04
Holding Registers	H400001-H410000 H400001-H4FFFF H400001-H4FFFD H4xxxxx.0/1-H4xxxxx.F/10*	**Word**, Short, BCD, Float, DWord, Long, LBCD, Double, Boolean	Read/Write	03, 06, 16 03, 06, 16, 22
Holding Registers As String with HiLo Byte Order	H400001.2H-H4FFFF.240H. Bit is string length, range 2 to 240 bytes.	**String****	Read/Write	03, 16
Holding Registers As String with LoHi Byte Order	H400001.2L-H4FFFF.240L. Bit is string length, range 2 to 240 bytes.	**String****	Read/Write	03, 16

- **For more information, refer to Zero-Based Bit Addressing in **Settings**.*
- ***For more information, refer to **String Support**.*

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. The byte order is specified by appending either a "H" or "L" to the address.

String 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: The string's length may be limited by the maximum size of the write request that the device allows. If, while utilizing a string tag, an error message of "Unable to write to address <address> on device <device>: Device responded with exception code 3" is received in the server event window, this means that the device did not like the string's length. If possible, shorten the string.

Normal Address Examples

1. The 255th output coil would be addressed as '0255' using decimal addressing or 'H0FF' using hexa-decimal addressing.
2. Some documentation refers to Modbus addresses by function code and location. For instance, function code 3; location 2000 would be addressed as '42000' or 'H47D0'. The leading '4' represents holding registers or function code 3.
3. Some documentation refers to Modbus addresses by function code and location. For instance, setting function code 5 location 100 would be addressed as '0100' or 'H064'. The leading '0' represents output coils or function code 5. Writing 1 or 0 to this address would set or reset the coil.

Array Support

Arrays are supported for internal and holding register locations for all data types except for Boolean and Strings. Arrays are also supported for input and output coils (Boolean data types). There are two methods of addressing an array. The following examples use holding register locations:

4xxxx [rows] [cols]
4xxxx [cols] this method assumes rows is equal to one.

For arrays, rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register / coil type. For register arrays of 32-bit data types, rows multiplied by cols multiplied by 2 cannot exceed the block size.

Packed Coil Address Type

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, polled mode only. The only valid data type is Word. The syntax is:

Output coils: 0*xxxxx#nn* Word Read/Write
Input coils: 1*xxxxx#nn* Word Read Only

where *xxxxx* is the address of the first coil (decimal and hex values allowed), and *nn* is the number of coils to be packed into an analog value (1-16, decimal only).

The bit order is such that the start address is the LSB (least significant bit) of analog value.

Magnetek GPD 515 Drive Addressing

This table provides the general ranges of data available from the Magnetek GPD 515 Drive. For information on how specific Drive parameters can be accessed using Modbus RTU addressing, refer to the Magnetek Modbus RTU Technical Manual, part number TM4025. In all cases, the letter H (used to signify Hex addressing) should precede the desired address. The default data types for dynamically defined tags are shown in **bold** where appropriate.

Magnetek GPD 515 Addressing Hexadecimal Format

Address	Range	Data Type	Access
Command Registers Bit Level Access	H40001-H4000F H4xxxx.0/1-H4xxxx.F/10*	**Word**, Short Boolean	Read/Write
Monitor Registers Bit Level Access	H40010-H4001A H4xxxx.0/1-H4xxxx.F/10*	**Word**, Short Boolean	Read Only
Drive Parameter Registers (Monitor Only) Bit Level Access	H40020-H40097 H4xxxx.0/1-H4xxxx.F/10*	**Word**, Short Boolean	Read Only
Drive Parameter Registers Bit Level Access	H40100-H4050D H4xxxx.0/1-H4xxxx.F/10*	**Word**, Short Boolean	Read/Write
Special Registers	H4FFDD ACCEPT H4FFFD ENTER	Word, Short	Write Only

*For more information, refer to Zero-Based Bit Addressing in **Settings**.*

Example

To access the driver's Operation Status, address 02BH, enter the following address: H4002B.

Note: When adding a Magnetek Device to the OPC Server project, users must make sure that the setting Zero-Based Addressing is disabled. If this parameter is not set correctly, the Modbus RTU driver offsets all of the Magnetek addresses by 1.

Array Support

Arrays are supported for holding register locations for all data types except Boolean. There are two methods of addressing an array. The following examples use holding register locations:

4xxxx [rows] [cols]
4xxxx [cols] this method assumes rows is equal to one.

Rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register type.

Elliott Flow Computer Addressing

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

Address	Range	Data Type	Access
Output Coils	000001-065536	**Boolean**	Read/Write
Input Coils	100001-165536	**Boolean**	Read Only
Internal Registers	300001-365536 300001-365535 3xxxxx.0/1-3xxxxx.15/16*	**Word**, Short, BCD Float, DWord, Long, LBCD Boolean	Read Only
Holding Registers	400001-465536 400001-465535 4xxxxx.0/1-4xxxxx.15/16*	**Word**, Short, BCD** Float, DWord, Long, LBCD Boolean	Read/Write

*For more information, refer to Zero-Based Bit Addressing in **Settings**.*

****Address ranges 405001 to 405315 and 407001 to 407315 are 32-bit registers. Addresses in the range of 405001 to 405315 use a default data type of Long.

Addresses in the range of 407001 to 407315 use a default data type of Float. Since these address registers are 32 -bit, only Float, DWord, Long, or LBCD data types are allowed. Arrays are not allowed.

Array Support

Arrays are supported for internal and holding register locations for all data types except Boolean. There are two methods of addressing an array. The following examples use holding register locations:

4xxxx [rows] [cols]
4xxxx [cols] this method assumes "rows" is equal to one.

Rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register type. For arrays of 32-bit data types, rows multiplied by cols multiplied by 2 cannot exceed the block size.

Daniels S500 Flow Computer Addressing

The default data types for dynamically defined tags are shown in **bold** where appropriate. The Function Codes are displayed in decimal.

*For more information, refer to **Function Codes Description**.*

Address	Hex Range	Decimal Range	Data Type	Function Codes	Access
Totals	000-0FF	4096-4351	Double	03	Read Only
Calculated /Measured Variables	100-24F	4352-4687	Float	03, 16	Read/Write
Calculation Constants	250-28F	4688-4751	Float	03, 16	Read/Write
Keypad Default Values	290-2AF	4752-4783	Float	03, 16	Read/Write
Alarm and Scaling Constants	2B0-5FF	4784-5631	Float	03, 16	Read/Write
Status /Control	700-7FF	5888-6143	**Boolean**	01, 05	Read/Write
Alarms	800-FFF	6144-8191	**Boolean**	02	Read Only

Dynamic Fluid Meter Addressing

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

Dynamic Fluid Meter Addressing Decimal Format

Address	Range	Data Type	Access
Holding Registers (16 bit)	400000-407000 400000-406999 408001-465535 408001-465534 4xxxxx.0/1-4xxxxx.15/16*	**Word**, Short, BCD Float, DWord, Long, LBCD **Word**, Short, BCD Float, DWord, Long, LBCD Boolean	Read/Write
Holding Registers (32 bit)	407001-408000	**Float**	Read/Write
Holding Registers As String with HiLo Byte Order	400000.2H-407000.240H 408001.2H-465535.240H .Bit is string length, range 2 to 240 bytes.	**String**	Read/Write
Holding Registers As String with LoHi Byte Order	400000.2L-407000.240L 408001.2L-465535.240L .Bit is string length, range 2 to 240 bytes.	**String**	Read/Write

*For more information, refer to Zero-Based Bit Addressing in **Settings**.*

Dynamic Fluid Meter Addressing Hexadecimal Format

Address	Range	Data Type	Access
Holding Registers (16 bit)	H400000-H401B58 H400000-H401B57 H401F41-H40FFFF H401F41-H40FFFE H4xxxxx.0/1-H4xxxxx.F/10*	**Word**, Short, BCD Float, DWord, Long, LBCD **Word**, Short, BCD Float, DWord, Long, LBCD Boolean	Read/Write
Holding Registers (32 bit)	H401B59-H401F40	**Float**	Read/Write
Holding Registers As String with HiLo Byte Order	H400000.2H-H401B58.240H H401F41.2H-H40FFFF.240H .Bit is string length, range 2 to 240 bytes.	**String**	Read/Write
Holding Registers As String with LoHi Byte Order	H400000.2L-H401B58.240L H401F41.2L-H0FFFF.240L	**String**	Read/Write

Address	Range	Data Type	Access
	.Bit is string length, range 2 to 240 bytes.		

*For more information, refer to Zero-Based Bit Addressing in **Settings**.*

Note: This driver requires that all addresses begin with "4" for the Dynamic Fluid Meter model. This 4 may not always be written explicitly in the Dynamic Fluid Meter documentation. For example, users may see a reference to "Unit ID at address 3001". This value must be addressed in the server as "403001".

String Support

The Dynamic Fluid Meter 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. The byte order is specified by appending either a "H" or "L" to the address.

String 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: The string's length may be limited by the maximum size of the write request that the device allows. If, while utilizing a string tag, an error message of "Unable to write to address <address> on device <device>: Device responded with exception code 3" is received in the server event window, this means the device did not like the string's length. If possible, try shortening the string.

Omni Flow Computer Addressing

The default data types for dynamically defined tags are shown in **bold**.

Address	Range	Data Type	Access
Digital I/O Point	1001-1024	**Boolean**	Read/Write
Programmable Boolean Point	1025-1088	**Boolean**	Read/Write
Meter Run Status and Alarm Points	1n01-001n59 1n76-1n99 n=Number of Meter Run	**Boolean**	Read/Write
Micro Motion Alarm Status Points	1n60-1n75 n=Number of Meter Run	**Boolean**	Read/Write
User Scratch Pad Boolean Points	1501-1599 1601 -1649	**Boolean**	Read/Write
User ScratchPad One Shot Points	1650-1699	**Boolean**	Read/Write
Command Boolean Points/Variables	1700-1798	**Boolean**	Read/Write
Meter Station Alarm and Status Points	1801-1899	**Boolean**	Read/Write
Prover Alarm and Status Points	1901-1967	**Boolean**	Read/Write
Meter Totalizer Roll-over Flags	2n01-2n37 n=Number of Meter Run	**Boolean**	Read/Write

Address	Range	Data Type	Access
Misc. Meter Station Alarm and Status	2601-2623	**Boolean**	Read/Write
Station Totalizer Roll-over Flags	2801-2851	**Boolean**	Read/Write
Station Totalizer Decimal Resolution	2852-2862 2865-2999	**Boolean**	Read/Write

16-Bit Integer Data Addresses	Range	Data Type	Access
Custom Data Packet #1	3001-3040	**Short**, Word, BCD	Read/Write
Custom Data Packet #2	3041-3056	**Short**, Word, BCD	Read/Write
Custom Data Packet #3	3057-3096	**Short**, Word, BCD	Read/Write
Misc. 16-bit Integer Data	3097-3099 3737-3799 3875-3899	**Short**, Word, BCD	Read/Write
Meter Run 16-bit Integer Data	3n01-3n52 n=Number of Meter Run	**Short**, Word, BCD	Read/Write
Scratchpad 16-bit Integer Data	3501-3599	**Short**, Word, BCD	Read/Write
User Display #1	3601-3608	**Short**, Word, BCD	Read/Write
User Display #2	3609-3616	**Short**, Word, BCD	Read/Write
User Display #3	3617-3624	**Short**, Word, BCD	Read/Write
User Display #4	3625-3632	**Short**, Word, BCD	Read/Write
User Display #5	3633-3640	**Short**, Word, BCD	Read/Write
User Display #6	3641-3648	**Short**, Word, BCD	Read/Write
User Display #7	3649-3656	**Short**, Word, BCD	Read/Write
User Display #8	3657-3664	**Short**, Word, BCD	Read/Write
Access Raw Data Archive Records	3701-3736	**Short**, Word, BCD	Read/Write
Meter Station 16-bit Integer Data	3800-3842	**Short**, Word, BCD	Read/Write
Meter #1 Batch Sequence	3843-3848	**Short**, Word, BCD	Read/Write
Meter #2 Batch Sequence	3849-3854	**Short**, Word, BCD	Read/Write
Meter #3 Batch Sequence	3855-3860	**Short**, Word, BCD	Read/Write
Meter #4 Batch Sequence	3861-3866	**Short**, Word, BCD	Read/Write
Flow Computer Time/Date	3867-3874	**Short**, Word, BCD	Read/Write
Prover 16-bit Integer Data	3901-3999	**Short**, Word, BCD	Read/Write

8-Character ASCII String Data	Range	Data Type	Access
Meter Run ASCII Data	4n01-4n39 n=Number of Meter Run	**String**	Read/Write
Scatch Pad ASCII Data	4501-4599	**String**	Read/Write
User Display Definition Variables	4601 -4640	**String**	Read/Write
Station Auxiliary Input Variables	4707-4710	**String**	Read/Write
Meter Station ASCII Data	4801-4851	**String**	Read/Write
Meter #1 Batch ID	4852-4863	**String**	Read/Write

8-Character ASCII String Data	Range	Data Type	Access
Meter #2 Batch ID	4864-4875	**String**	Read/Write
Meter #3 Batch ID	4876-4887	**String**	Read/Write
Meter #4 Batch ID	4888-4899	**String**	Read/Write
Prover ASCII String Data	4901-4942	**String**	Read/Write

32-Bit Integer Data	Range	Data Type	Access
Meter Run 32-bit Integer Data	5n01-5n99 n=Number of Meter Run	**Long**, DWord, LBCD, Float	Read/Write
Scratch Pad 32-bit Integer Data	5501-5599	**Long**, DWord, LBCD, Float	Read/Write
Station 32-bit Integer Data	5801-5818	**Long**, DWord, LBCD, Float	Read/Write
Meter #1 Batch Size	5819-5824	**Long**, DWord, LBCD, Float	Read/Write
Meter #2 Batch Size	5825-5830	**Long**, DWord, LBCD, Float	Read/Write
Meter #3 Batch Size	5831-5836	**Long**, DWord, LBCD, Float	Read/Write
Meter #4 Batch Size	5837-5842	**Long**, DWord, LBCD, Float	Read/Write
Additional 32-bit Meter Run Data	5843-5899	**Long**, DWord, LBCD, Float	Read/Write
Prover 32-bit Integer Data	5901-5973	**Long**, DWord, LBCD, Float	Read/Write
Compact Prover TDVOL/TDFMP Pulses	5974-5999	**Long**, DWord, LBCD, Float	Read/Write

32-Bit IEEE Floating Point Data	Range	Data Type	Access
Reserved Data	6001-7000	**Float**, Long, DWord, LBCD	Read/Write
Digital to Analog Outputs	7001-7024	**Float**, Long, DWord, LBCD	Read/Write
User Variables	7025-7088	**Float**, Long, DWord, LBCD	Read/Write
Programmable Accumulator	7089-7099	**Float**, Long, DWord, LBCD	Read/Write
Meter Run Data	7n01 - 7n99 n=Number of Meter Run	**Float**, Long, DWord, LBCD	Read/Write
Scratch Pad Data	7501-7599	**Float**, Long, DWord, LBCD	Read/Write
PID Control Data	7601-7623	**Float**, Long, DWord, LBCD	Read/Write

32-Bit IEEE Floating Point Data	Range	Data Type	Access
Miscellaneous Meter Run Data	7624-7699	**Float**, Long, DWord, LBCD	Read/Write
Miscellaneous Variables	7701-7799	**Float**, Long, DWord, LBCD	Read/Write
Meter Station Data	7801-7899	**Float**, Long, DWord, LBCD	Read/Write
Prover Data	7901-7918	**Float**, Long, DWord, LBCD	Read/Write
Configuration Data for Prover	7919-7958	**Float**, Long, DWord, LBCD	Read/Write
Last Prove Data	7959-7966	**Float**, Long, DWord, LBCD	Read/Write
Data Rejected During Prove	7967-7990	**Float**, Long, DWord, LBCD	Read/Write
Prove Run Data	7991-8050	**Float**, Long, DWord, LBCD	Read/Write
Prove Average Data	8051-8079	**Float**, Long, DWord, LBCD	Read/Write
Prove Run-Master Meter Data	8080-8199	**Float**, Long, DWord, LBCD	Read/Write
Proving Series Data	8200-8223	**Float**, Long, DWord, LBCD	Read/Write
Data of Meter Being Proved	8224-8230	**Float**, Long, DWord, LBCD	Read/Write
Mass Prove Data	8231-8500	**Float**, Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #1	8501-8599	**Float**, Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #2	8601-8699	**Float**, Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #3	8701-8799	**Float**, Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #4	8801-8899	**Float**, Long, DWord, LBCD	Read/Write
Station Previous Batch Average Data	8901-8999	**Float**, Long, DWord, LBCD	Read/Write

16-Bit Integer Configuration Data	Range	Data Type	Access
Meter Run #1	13001-13013	**Short**, Word, BCD	Read/Write
Meter Run #2	13014-13026	**Short**, Word, BCD	Read/Write
Meter Run #3	13027-13039	**Short**, Word, BCD	Read/Write
Meter Run #4	13040-13052	**Short**, Word, BCD	Read/Write
Prover Configuration	13053-13073	**Short**, Word, BCD	Read/Write

16-Bit Integer Configuration Data	Range	Data Type	Access
General Flow Configuration	13074-13084	**Short**, Word, BCD	Read/Write
Serial Port Configuration	13085-13128	**Short**, Word, BCD	Read/Write
PID Configuration	13129-13160	**Short**, Word, BCD	Read/Write
PLC Data	13161-13299	**Short**, Word, BCD	Read/Write
Peer to Peer Setup	13300-13499	**Short**, Word, BCD	Read/Write
Raw Data Archive	13500-13999	**Short**, Word, BCD	Read/Write

16-Character ASCII String Data	Range	Data Type	Access
Flow Computer Configuration	14001-14499	**String**	Read/Write

32-Bit Integer Data	Range	Data Type	Access
Flow Computer Configuration	15001-16999	**Long**, DWord, LBCD, Float	Read/Write

32-Bit IEEE Floating Point Data	Range	Data Type	Access
Flow Computer Configuration	17001-18999	**Float**, Long, DWord, LBCD	Read/Write

Supported Extended Omni Types

Custom Packets
Raw Data Archive
Text Reports
Text Archive

Omni Custom Packets

The Omni Flow Computer allows users to map various ranges of memory to a single data structure that can be read with a single, highly efficient read command. These data structures are called Custom Packets.

Packet Configuration

Each custom packet may contain up to twenty groups of data points. Each group is defined by its starting index and the number of data points. The total size of the custom packet must not exceed 250 bytes. The addresses used to define the custom packets are listed below.

Custom Packet 1 (address 1)
3001 Group 1-Starting index
3002 Group 1-Number of points
to
3039 Group 20-Starting index
3040 Group 20-Number of points

Custom Packet 2 (address 201)
3041 Group 1-Starting index
3042 Group 1-Number of points
to
3055 Group 20-Starting index
3056 Group 20-Number of points

Custom Packet 3 (address 401)
3057 Group 1-Starting index
3058 Group 1-Number of points
to
3095 Group 20-Starting index
3096 Group 20-Number of points

Note: Data is returned from the device as 16-bit registers. Digital I/O must be mapped in blocks of 16 bits.

Custom Packet Address Syntax

Tags can be created to access data at a given offset within a custom packet. The address syntax is as follows. The default data types are shown in **bold.**

Address	Range	Data Type	Access
CPn_o	n = Packet Number (1-3) o = Word offset (0-125)	Word, **Short**, BCD, DWord, Long, LBCD, Float, String	Read Only
CPn_o.b	n = Packet Number (1-3) o = Word offset (0-125) b = Bit number (0/1-15/16)*	**Boolean**	Read Only

*For more information, refer to Zero-Based Bit Addressing in **Settings**.*

Notes:

1. Only 8 character ASCII string data is supported.
2. If a 16 character ASCII string data address is contained in group configuration, then data can be read as two 8 character ASCII string data items.

Example

Define Custom Packet #1 to map to the following:

- 16 bits of digital I/O (1001-1016).
- Fifteen 32-bit integers of Meter Run 1 Batch data (5101 -5115).
- Twelve 32-bit floats of Analog Outputs (7001-7012).
- Four 8-character ASCII strings of Meter Run (4101-4104).
- Six 8-character ASCII strings of Meter Station (4808-4813).
- Two 16-character ASCII strings of Flow Configuration data (14001-14002).

Note: This makes a total of 222 bytes. The custom packet configuration registers would have the following values:

3001 = 1001
3002 = 16
3003 = 5101
3004 = 15
3005 = 7001
3006 = 12
3007 = 4101
3008 = 4
3009 = 4808

3010 = 6
3011 = 14001
3012 = 2

Tags to access the Digital I/O data would have the following addresses (all 16 values contained in word 0):
CP1_0.0 (Word 0 of Custom Packet 1, bit 0-mapped to 1009)
CP1_0.1 (Word 0 of Custom Packet 1, bit 1-mapped to 1010)
...
CP1_0.6 (Word 0 of Custom Packet 1, bit 6-mapped to 1015)
CP1_0.7 (Word 0 of Custom Packet 1, bit 7-mapped to 1016)
CP1_0.8 (Word 0 of Custom Packet 1, bit 8-mapped to 1001)
CP1_0.9 (Word 0 of Custom Packet 1, bit 9-mapped to 1002)
...
CP1_0.14 (Word 0 of Custom Packet 1, bit 14-mapped to 1007)
CP1_0.15 (Word 0 of Custom Packet 1, bit 15-mapped to 1008)

Tags to access the Meter Run 1 Batch data would have the following addresses (each 32-bit value uses 2 words):
CP1_1 (Word 1 of Custom Packet 1-mapped to 5101)
CP1_3 (Word 3 of Custom Packet 1-mapped to 5102)
...
CP1_29 (Word 29 of Custom Packet 1-mapped to 5115)

Tags to access the Analog Output data would have the following addresses (each 32-bit value uses 2 words):
CP1_31 (Word 31 of Custom Packet 1-mapped to 7001)
CP1_33 (Word 33 of Custom Packet 1-mapped to 7002)
...
CP1_53 (Word 53 of Custom Packet 1-mapped to 7012)

Tags to access the Meter Run 8 character ASCII String data would have the following addresses (each String value uses 4 words):
CP1_55 (Word 55 of Custom Packet 1-mapped to 4101)
...
CP1_67 (Word 67 of Custom Packet 1-mapped to 4104)

Tags to access the Meter Station 8 character ASCII String data would have the following addresses (each String value uses 4 words):
CP1_71 (Word 71 of Custom Packet 1-mapped to 4808)
...
CP1_91 (Word 91 of Custom Packet 1-mapped to 4813)

Tags to access the Flow Configuration 16 character ASCII String data would have the following addresses (each String value uses 4 words):
CP1_95 (Word 95 of Custom Packet 1-mapped to 14001 characters 1-8)
CP1_99 (Word 99 of Custom Packet 1-mapped to 14001 characters 9-16)
CP1_103 (Word 103 of Custom Packet 1-mapped to 14002 characters 1-8)
CP1_107 (Word 107 of Custom Packet 1-mapped to 14002 characters 9-16)

Omni Raw Data Archive

The Omni Flow Computer may be configured to map various ranges of memory to a single data structure, and then store that structure in an archive when triggered. Users may configure up to ten archives. There

are two additional fixed format archives for Alarm and Audit data. Each archive is a circular buffer, where each new record replaces the oldest record.

Record Configuration and Retrieval

Users may configure the record structure of Raw Data Archives 1 to 10. Archives 11 and 12 are of fixed format and contain Alarm and Audit data respectively.

For a full discussion of Raw Data Archives, refer to Omni Technical Bulletin 96073.

Each record may contain up to sixteen groups of data points. Each group is defined by its starting index and the number of data points. The addresses used to define the archive records are listed below. The total size of the record must not exceed 250 bytes. The device uses the first 6 bytes for date and time stamp data, leaving 244 bytes for raw data. Each record has its own Boolean trigger. Data is stored when the trigger goes from low to high.

Before a group starting index, number of points in group or trigger for a raw data archive can be changed, archiving must halt. The **Allow Archive Configuration Flag** must be set in the device. Be aware that doing this likely causes the data archive in the device to be reinitialized, including all Raw Data Archives and the Text Archive.

13920 Archive Run-0=stop, 1= start
13921 Reconfigure Archives-0=no configuration changes allowed, 1=configuration changes allowed

This driver may be used to read a Raw Data Archive one record at a time. To read a record, first write the desired record index to the "Requested record" register. Once this value is set, users may read the record with an "RA" tag. Users should ensure that the specified record index does not exceed the maximum number of records allowed for that archive. If the "Last record updated" value is zero, there have been no records saved in the archive since it was last initialized.

Raw Data Archive 1 (address 701)
13500 Group 1-Starting Index
13501 Group 1-Number of Points
to
13530 Group 16-Starting Index
13531 Group 16-Number of points

13900 Trigger Boolean

3701 Maximum number of records
3702 Last record updated
3703 Requested record

Raw Data Archive 2 (address 702)
13540 Group 1-Starting Index
13541 Group 1-Number of Points
to
13570 Group 16-Starting Index
13571 Group 16-Number of points

13901 Trigger Boolean

3704 Maximum number of records
3705 Last record updated
3706 Requested record

Raw Data Archive 3 (address 703)
13580 Group 1-Starting Index
13581 Group 1-Number of Points
to
13610 Group 16-Starting Index
13611 Group 16-Number of points

13902 Trigger Boolean

3707 Maximum number of records
3708 Last record updated
3709 Requested record

Raw Data Archive 4 (address 704)
13620 Group 1-Starting Index
13621 Group 1-Number of Points
to
13650 Group 16-Starting Index
13651 Group 16-Number of points

13903 Trigger Boolean

3710 Maximum number of records
3711 Last record updated
3712 Requested record

Raw Data Archive 5 (address 705)
13660 Group 1-Starting Index
13661 Group 1-Number of Points
to
13690 Group 16-Starting Index
13691 Group 16-Number of points

13904 Trigger Boolean

3713 Maximum number of records
3714 Last record updated
3715 Requested record

Raw Data Archive 6 (address 706)
13700 Group 1-Starting Index
13701 Group 1-Number of Points
to
13730 Group 16-Starting Index
13731 Group 16-Number of points

13905 Trigger Boolean

3716 Maximum number of records
3717 Last record updated
3718 Requested record

Raw Data Archive 7 (address 707)

13740 Group 1-Starting Index
13741 Group 1-Number of Points
to
13770 Group 16-Starting Index
13771 Group 16-Number of points

13906 Trigger Boolean

3719 Maximum number of records
3720 Last record updated
3721 Requested record

Raw Data Archive 8 (address 708)
13780 Group 1-Starting Index
13781 Group 1-Number of Points
to
13810 Group 16-Starting Index
13811 Group 16-Number of points

13907 Trigger Boolean

3722 Maximum number of records
3723 Last record updated
3724 Requested record

Raw Data Archive 9 (address 709)
13820 Group 1-Starting Index
13821 Group 1-Number of Points
to
13850 Group 16-Starting Index
13851 Group 16-Number of points

13908 Trigger Boolean

3725 Maximum number of records
3726 Last record updated
3727 Requested record

Raw Data Archive 10 (address 710)
13860 Group 1-Starting Index
13861 Group 1-Number of Points
to
13890 Group 16-Starting Index
13891 Group 16-Number of points

13909 Trigger Boolean

3728 Maximum number of records
3729 Last record updated
3730 Requested record

Raw Data Archive 11 Alarm (address 711)
Not configurable

3731 Maximum number of records
3732 Last record updated
3733 Requested record

Raw Data Archive 12 Archive (address 712)
Not configurable

3734 Maximum number of records
3735 Last record updated
3736 Requested record

Note: Data is returned from the device as 16-bit registers. Digital I/O must be mapped in blocks of 16 bits.

Raw Data Archive Address Syntax

Tags can be created to access data at a given offset within a Raw Data Archive record. The address syntax is as follows. The default data types are shown in **bold**.

Address	Range	Data Type	Access
RAn_o	n = Archive Number (1-12) o = Word offset (0-125)	Word, **Short**, BCD, DWord, Long, LBCD, Float, String	Read Only
RAn_o.b	n = Archive Number (1-12) o = Word offset (0-125) b = Bit number (0/1-15/16)*	**Boolean**	Read Only

*For more information, refer to Zero-Based Bit Addressing in **Settings**.*

Notes:

1. Only 8 character ASCII string data is supported.
2. If a 16 character ASCII string data address is contained in group configuration, then data can be read as two 8 character ASCII string data items.

Timestamp Format

The first 6 bytes of each record contains the time and date that the record was placed in the archive.

Byte	Description
1	Month (1-12)* Day (1-31)
2	Day (1-31)* Month (1-12)
3	Year (0-99)
4	Hour (0-23)
5	Minute (0-59)
6	Seconds (0-59)

*Date format is set with register 3842 (0=dd/mm/yy, 1= mm/dd/yy).

Alarm/Event Log Record Structure (Address 711)

Field	Data Type	Description
1	3-Byte Date	dd/mm/yy or mm/dd/yy.
2	3-Byte Time	hh/mm/ss.
3	16-bit Integer	Modbus Index # of alarm or event.
4	1 Byte	Alarm Type.
5	1 Byte	0=OK, 1=Alarm.
6	IEEE Float	Value of transducer variable at the time of alarm or event.
7	32-bit Integer	Volume totalizer at the time of the alarm or event.
8	32-bit Integer	Mass totalizer at the time of the alarm or event.

Alarm Types

Type	Description
0	Log event, sound beeper and display in LCD any edge change in bit identified by field #3.
1	Log event, sound beeper and display in LCD rising edge changes in bit identified by field #3.
2	Event Log any edge change in bit identified by field #3. No beeper or LCD display action.
3	Event Log rising edge changes in bit identified by field #3. No beeper or LCD display action.

Audit Event Log Record Structure (Address 712)

Field	Data Type	Description
1	3-Byte Date	dd/mm/yy or mm/dd/yy.
2	3-Byte Time	hh/mm/ss.
3	16-bit Integer	Event number, increments for each event, rolls at 65535.
4	16-bit Integer	Modbus index of variable changed.
5	IEEE Float	Numeric variable value before change-old value.
6	IEEE Float	Numeric variable value after change-new value.
7	16-Char ASCII	String variable value before change-old value.
8	16-Char ASCII	String variable value after change-new value.
9	32-bit Integer	Volume totalizer at time of change.
10	32-bit Integer	Mass totalizer at the time of the change.

Note: Fields 5 and 6 are set to 0.0 when the variable type changed is a string. Fields 7 and 8 contain null characters when the variable type change is not a string. When fields 7 and 8 contain 8 character strings, the remaining 8 characters are padded with nulls.

Omni Text Reports

The Omni Flow computer can generate several different types of text reports. Each of these reports can be read by this driver and sent to the OPC Client as a string value.

Text Report Types

There are a number of report types that can be retrieved from the Omni Flow Computer. They may be read using a "TR" tag. The report types are as follows.

Custom Report Templates

9001 Report Template-Snapshot / Interval
9002 Report Template-Batch
9003 Report Template-Daily
9004 Report Template-Prove

Previous Batch Reports
9101 Batch Report-Last
9102 Batch Report-Second from Last
...
9108 Batch Report-Eighth from last

Previous Prove Reports
9201 Prove Report-Last
9202 Prove Report-Second from last
...
9208 Prove Report-Eighth from last

Previous Daily Reports
9301 Previous Day's Report-Last
9302 Previous Day's Report-Second from last
...
9308 Previous Day's Report-Eighth from last

Last Snapshot Report
9401 Last Local Snapshot / Interval Report

Miscellaneous Report Buffer
9402 Miscellaneous Report Buffer

Preview Monthly Reports
9501 Previous Month's Report - Last
9502 Previous Month's Report - Second from last
...
9508 Previous Month's Report - Eighth from last

Text Report Address Syntax

Address	Range	Data Type	Access
TRn TRn T (triggered read)	n = Report address (9001-9508)	String	Read/Write

Example

To read or write to the Snapshot Report Template (address 9001), create a tag with address "TR9001".

Note: Because it can take several seconds to read a Text Report, the "TR" tags should be kept inactive in the OPC client. Alternatively, triggered reads can be used instead. No other tags on the channel can be read or written to while the driver is reading or writing a Text Report.

Triggered Text Report Reads

As noted above, it is recommended that the Text Report tag be kept inactive, even though it is not always possible. A triggered read capability has been added as an alternative, allowing the Text Report tag to remain active. It also controls when the actual device reads occur with an auxiliary trigger tag.

A triggered read may not begin immediately, depending on when in the Text Report tag's update cycle the trigger is set. After the read attempt has been completed, the driver clears the trigger state. The Text Report tag shows the value and data quality that resulted from the last triggered read attempt.

Text Report Read Trigger Address Syntax

Address	Range	Data Type	Access
TRIG_TRn	n = Report address (9001-9508)	Boolean	Read/Write

Example

To read the Last Batch Report (address 9101) on trigger, create two tags. The first is a Text Report tag with address "TR9101 T", and the second is a Text Report Read Trigger tag with address "TRIG_TR9101".

Note: The Text Report tag address looks like a normal Text Report address followed by a space and the letter "T" for "triggered read". This "T" must be present in the address for triggered reads to work.

To trigger a read, set the trigger tag value to true (non-zero). After the read attempt has been completed, the driver sets the trigger value to false (0). If the read was successful, the Text Report tag's data quality is Good. If the read failed, the Text Report tag's data quality is Bad, and the value is the last value successfully read.

Saving Text Report Data To Disk

The driver has the ability to save Text Report data to disk. This feature is enabled by using Text Report Path tags. These tags are used to write file path strings to the driver's memory. Each report type has its own path buffer. After a successful Text Report read, the driver checks the associated path buffer. If a valid path is stored there, the driver saves the report data as ASCII text in that file. The file is created if needed. The file is overwritten on subsequent Text Report reads.

The path buffers are initialized to empty strings on server start up. The driver does not write Text Report data to file until a valid path is saved in the associated path buffer. Path data is not persistent. The path strings must be rewritten each time the server is restarted. The path values can be changed at any time, allowing users to save data to different files on each read if desired.

Path strings may be up to 255 characters long.

Text Report Path Address Syntax

Address	Range	Data Type	Access
PATH_TRn	n = Report address (9001-9508)	String	Read/Write

Example

To read the Last Batch Report (address 9101) and save the result to disk, create two tags. The first is a Text Report tag with address "TR9101", and the second is a path tag with address "PATH_TR9101".

To save the report data in a file called "LastBatch.txt" (which is to be created in the folder "C:\OmniData\BatchReports") set up the client so that the first thing that it does is write "C:\OmniData\BatchReports\LastBatch.txt" to the path tag. Once this is done, read the Text Report tag. If the path is not set before the first read of the Text Report, the driver is not able to save the data to disk.

Note: To disable this feature, write an empty string to the path tag.

Omni Text Archive

The Omni Flow Computer can also store reports in an archive. This driver can read a range of reports from the archive and send them to the OPC client as a string value.

Reading the Text Archive

Before the text archive can be read, two settings must be made in the device: the archive start date, and the number of days to retrieve. These 32-bit integer values are at addresses 15128 and 15127 respectively. The date format may be specified using the value at address 3842 (0 = dd/mm/yy, 1 = mm/dd/yy). Shortly after the number of days is set, the device begins preparing the data. When the data is ready to be read, the number of days value becomes negative. The Text Archive can be read at any time after the number of days is set. The driver waits for the value to become negative.

Text Archive Address Syntax

Address	Range	Data Type	Access
TA TA T (triggered read)	N/A	String	Read Only

Note: Because it can take several minutes to read a Text Archive, the "TA" tag should be kept inactive in the OPC client. Alternatively, triggered reads can be used instead. This tag should only be read using asynchronous reads, since the maximum synchronous read timeout cannot be increased high enough in the server to read a typical text archive request. No other tags on the channel can be read or written to while the Text Archive is being read.

If a Text Archive read fails midway, users should reset the device's read buffer by writing 999 to the number of days register (15127), and then repeat the normal Text Archive read procedure. Otherwise, the driver may not get the first part of the requested archive range.

Triggered Text Archive Reads

It is recommended that the Text Archive tag be kept inactive even though it is not always possible. A triggered read capability has been added as an alternative, thus allowing the Text Archive tag to remain active. It also controls when the actual device reads occur with an auxiliary trigger tag. The trigger value is stored in the driver's memory and may be read and set using a tag with the address syntax described below.

A triggered read may not begin immediately depending on when in the Text Archive tag's update cycle the trigger is set. After the read attempt has been completed, the driver clears the trigger state. The Text Archive tag shows the value and data quality that resulted from the last triggered read attempt.

Text Archive Read Trigger Address Syntax

Address	Range	Data Type	Access
TRIG_TA	N/A	Boolean	Read/Write

Example

To read the Text Archive on trigger, create two tags. The first is a Text Archive tag with address "TA T", and the second is a Text Archive Read Trigger tag with address "TRIG_TA". Users must create start date and number of days tags.

Note: The Text Archive tag address looks like a normal Text Archive address followed by a space and the letter "T" for "triggered read". This "T" must be present in the address for triggered reads to work.

To trigger a read, set the trigger tag value to true (non-zero). After the read attempt has been completed, the driver sets the trigger value to false (0). If the read was successful, the Text Archive tag's data quality is Good. If the read failed, the Text Archive tag's data quality is Bad and the value is the last value successfully read.

Saving Text Archive Data to Disk

The driver has the ability to save Text Archive data to disk. This feature is enabled using a Text Archive Path tag. This tag is used to write a file path string to the driver's memory. After a successful Text Archive read, the driver checks the associated path buffer. If a valid path is stored there, the driver saves the Text Archive data as ASCII text in that file. The file is created if needed. The file is overwritten on subsequent Text Archive reads.

The path buffer is initialized to an empty string on server start up. The driver does not write Text Archive data to file until a valid path is saved in the associated path buffer. Path data is not persistent. Users must to rewrite the path string each time the server is restarted. The path value can be changed at any time, allowing the data to be saved to different files on each read (if desired).

The path string may be up to 255 characters long. The directory cannot be the root drive (C:\\TextArchive.txt), within the Windows directory, or within the Program Files directory other than the server installation location. The file extension must be either .txt or .log.

Text Archive Path Address Syntax

Address	Range	Data Type	Access
PATH_TA	N/A	String	Read/Write

Example

To read the Text Archive and save the result to disk, create two tags. The first is a Text Archive tag with address "TA", and the second is a path tag with address "PATH_TA". Users must create start date and number of days tags as described above.

To save the Text Archive data in a file called "TextArchive.txt" (which is to be created in the folder "C:\OmniData\ArchiveData") set up the client so that the first thing that it does is write "C:\OmniData\ArchiveData\TextArchive.txt" to the path tag. Once this is done, read the Text Archive tag. If the path is not set before the first read of the Text Archive, the driver is not able to save the data to disk.

Note: To disable this feature, write a empty string to the path tag.

Function Codes Description

Modbus Addressing Model

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

Decimal	Hexadecimal	Description
16	0x10	Preset Multiple Registers
22	0x16	Masked Write Register

Daniels S500 Flow Computer Addressing Model

Decimal	Hexadecimal	Description
01	0x01	Read Coil Status
02	0x02	Read Input Coil
03	0x03	Read Holding Registers
05	0x05	Force Single Coil
16	0x10	Preset Multiple Registers

Channel Properties — Configuration API

The following properties define a channel using the Configuration API service.

General Properties

`common.ALLTYPES_NAME` * Required parameter.

Note: Changing this property causes the API endpoint URL to change.

`common.ALLTYPES_DESCRIPTION`

`servermain.MULTIPLE_TYPES_DEVICE_DRIVER` * Required parameter

`servermain.CHANNEL_DIAGNOSTICS_CAPTURE`

Ethernet Communication Properties

`servermain.CHANNEL_ETHERNET_COMMUNICATIONS_NETWORK_ADAPTER_STRING`

Advanced Properties

`servermain.CHANNEL_NON_NORMALIZED_FLOATING_POINT_HANDLING` * Required parameter

Write Optimizations

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_METHOD`

`servermain.CHANNEL_WRITE_OPTIMIZATIONS_DUTY_CYCLE`

***See Also**: The server help system Configuration API Service section.*

Device Properties — Configuration API

The following properties define a device using the Configuration API service.

General Properties

common.ALLTYPES_NAME

common.ALLTYPES_DESCRIPTION

servermain.DEVICE_CHANNEL_ASSIGNMENT

servermain.MULTIPLE_TYPES_DEVICE_DRIVER

servermain.DEVICE_MODEL

servermain.DEVICE_ID_STRING

servermain.DEVICE_DATA_COLLECTION

servermain.DEVICE_SIMULATED

Scan Mode

servermain.DEVICE_SCAN_MODE * Required parameter

servermain.DEVICE_SCAN_MODE_RATE_MS

servermain.DEVICE_SCAN_MODE_RATE_MS

servermain.DEVICE_SCAN_MODE_PROVIDE_INITIAL_UPDATES_FROM_CACHE

Auto Demotion

servermain.DEVICE_AUTO_DEMOTION_ENABLE_ON_COMMUNICATIONS_FAILURES

servermain.DEVICE_AUTO_DEMOTION_DEMOTE_AFTER_SUCCESSIVE_TIMEOUTS

servermain.DEVICE_AUTO_DEMOTION_PERIOD_MS

servermain.DEVICE_AUTO_DEMOTION_DISCARD_WRITES

Tag Generation

servermain.DEVICE_TAG_GENERATION_ON_STARTUP * Required parameter

servermain.DEVICE_TAG_GENERATION_DUPLICATE_HANDLING * Required parameter

servermain.DEVICE_TAG_GENERATION_GROUP

servermain.DEVICE_TAG_GENERATION_ALLOW_SUB_GROUPS

Tip: To Invoke Automatic Tag Generation, send a PUT with an empty body to the TagGeneration service endpoint on the device.
See Also: *For more information see Services help.*

Timing

servermain.DEVICE_CONNECTION_TIMEOUT_SECONDS

servermain.DEVICE_REQUEST_TIMEOUT_MILLISECONDS

servermain.DEVICE_RETRY_ATTEMPTS

servermain.DEVICE_INTER_REQUEST_DELAY_MILLISECONDS

***See Also**: The server help system Configuration API Service section.*

Event Log Messages

The following information concerns messages posted to the Event Log pane in the main user interface. Consult the 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.

Bad address in block range. | Address range = <start> to <end>.

Error Type:
Error

Possible Cause:

1. An attempt was made to reference a nonexistent location in the specified device.
2. An attempt was made to read more registers than allowed by the protocol.

Possible Solution:

1. Verify the tags assigned to addresses in the specified range on the device and eliminate ones that reference invalid locations.
2. Decrease the register block size value to 125.

See Also:

1. Error Handling
2. Block Sizes

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:

1. Verify the size of the device's memory space and redefine the array length accordingly.
2. Verify the tags assigned to addresses in the specified range on the device and eliminate ones that reference invalid locations.
3. Reduce the array size value to 125.

See Also:

1. Error Handling
2. Block Sizes

Block address responded with exception code. | Address range = <start> to <end>, Exception code = <code>.

Error Type:
Warning

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

Possible Solution:
See Modbus Exception Codes.

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

Error Type:
Warning

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

Possible Solution:
See Modbus Exception Codes.

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

Error Type:
Warning

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

Possible Solution:
See Modbus Exception Codes.

Tag import failed due to low memory resources.

Error Type:
Warning

Possible Cause:
The driver can 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 = <name>.

Error Type:
Warning

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

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

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 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 driver does not support the data type specified in the variable import file.

Possible Solution:
Change the data type specified in the variable import file to one that is supported. If the variable is for a structure, manually edit the file to define each tag required for the structure or configure the required tags manually in the server.

See Also:
Exporting Variables from Concept

Could not read Omni text buffer due to memory allocation problem.

Error Type:
Warning

Possible Cause:
The driver can not allocate memory required for an Omni Text Record or Text Archive read operation.

Possible Solution:
Shut down all unnecessary applications and retry.

No Omni text archive data available in specified date range.

Error Type:
Warning

Possible Cause:
No data is in the text archive for the date range specified by the Start Date register (15128) and the Number of Days register (15127).

Possible Solution:
This is not necessarily an error. Verify there is no data available for specified range.

Write to Omni text report truncated. | Report number = <number>.

Error Type:
Warning

Possible Cause:
An attempt was made to write more than 8192 bytes to a text report. This is a limit imposed by the protocol.

Possible Solution:
Do not write strings greater than the 8192 byte limit. If the string is longer, only the first 8192 characters are written to the device.

Could not read Omni text report due to packet number limit. | Report number = <number>.

Error Type:
Warning

Possible Cause:
Text reports are expected to be 8192 bytes or less. This is a limit imposed by the protocol. The driver read 8192 bytes before encountering the expected end of file character.

Possible Solution:
Verify that the report template used by the device generates reports of 8192 bytes or less.

Write failed. Maximum path length exceeded. | Tag address = '<address>', Maximum length = <number>.

Error Type:
Warning

Possible Cause:
Path length is limited to the indicated number of characters.

Possible Solution:
Use a shorter path.

Error writing Omni text data to file. | Tag address = '<address>', Reason = '<reason>'.

Error Type:
Warning

Possible Cause:
The driver could not write the Omni text data to disk for the indicated reason.

Possible Solution:
Consult the operating system documentation for appropriate corrective measures for the reason indicated.

Omni text output file could not be opened. | Tag address = '<address>', Reason = '<reason>'.

Error Type:
Warning

Possible Cause:
The file specified in an Omni Text Path tag could not be created or opened.

Possible Solution:

Consult the operating system documentation about the reason indicated for appropriate corrective measures. The most likely cause is an invalid path.

See Also:

1. Omni Text Reports
2. Omni Text Archive

Unable to write to address. Unexpected characters in response. | Tag address = '<address>'.

Error Type:

Warning

Unable to read from address. Unexpected characters in response. | Tag address = '<address>'.

Error Type:

Warning

Unable to read block address. Unexpected characters in response. | Address range = <start> to <end>.

Error Type:

Warning

Omni text output file could not be changed. | Tag address = '<address>', Reason = The path specified is not allowed.

Error Type:

Warning

Possible Cause:

The directory specified in an Omni Text Path tag is not allowed.

Possible Solution:

Consult the driver help content and supply a path to a secure location.

See Also:

1. Omni Text Reports
2. Omni Text Archive

Omni text output file could not be changed. | Tag address = '<address>', Reason = The file extension specified must be '.txt' or '.log'.

Error Type:

Warning

Possible Cause:
The extension specified in an Omni Text Path tag is not allowed and must be .txt or .log.

Possible Solution:
Specify a valid extension.

See Also:
1. Omni Text Reports
2. Omni Text Archive

Importing tag database from file. | File name = '<name>'.

Error Type:
Informational

Error Mask Definitions

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

Modbus Exception Codes

The following data is from Modbus Application Protocol Specifications documentation.

Code Dec/Hex	Name	Meaning
01/0x01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type, for example, because it is unconfigured and is being asked to return register values.
02/0x02	ILLEGAL DATA ADDRESS	The data address received in the query is not an allowable address for the server (or slave). More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed. A request with offset 96 and length 5 generates exception 02.
03/0x03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the Modbus protocol is unaware of the significance of any particular value of any particular register.
04/0x04	SLAVE DEVICE FAILURE	An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
05/0x05	ACKNOWLEDGE	The slave has accepted the request and is processing it, but a long duration of time is required to do so. This response is returned to prevent a timeout error from occurring in the master. The master can next issue a Poll Program Complete message to determine if processing is completed.
06/0x06	SLAVE DEVICE BUSY	The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free.
07/0x07	NEGATIVE ACKNOWLEDGE	The slave cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 decimal. The master should request diagnostic or error information from the slave.
08/0x08	MEMORY PARITY ERROR	The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device.
10/0x0A	GATEWAY PATH UNAVAILABLE	Specialized use in conjunction with gateways indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. This usually means that the gateway is misconfigured or overloaded.
11/0x0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Specialized use in conjunction with gateways indicates that no response was obtained from the target device. This usually means that the device is not present on the network.

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

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z