Modbus Plus Driver

Table of Contents

Modbus Plus Driver

Help version 1.061

CONTENTS

Overview

The Modbus Plus Driver provides a reliable way to connect Modbus Plus devices to OPC client applications; including HMI, SCADA, Historian, MES, ERP, and countless custom applications. It is intended for use with a Modicon SA85, or Schneider PCI-85 interface card. This driver does not support configurations where SA85/PCI-85 cards exist in the same computer.

External Dependencies

This driver has external dependencies.

An SA85 or PCI-85 interface adapter and its device driver software (MBPLUS or MBX drivers) are required to communicate to the Modbus Plus network. The interface adapter and device drivers can be purchased from Modicon/Schneider. The server can support up to eight (8) channels per SA85 or PCI-85 card.

Setup

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

Channel and Device Limits

The maximum number of channels supported by this driver is 32. The maximum number of devices supported by this driver is 8192 per channel.
The Modbus Plus Driver supports a maximum of eight channels per SA85 card adapter.

Supported Interface Cards

SA85 Card

Note: Users may also connect to a Modbus Plus network from a Modbus RTU Serial machine via a USB adapter.

For more information on the SA85 card requirements, refer to ***External Dependencies****.*

Supported Communication Modes

The Modbus Plus Driver supports three communication modes, which are used to obtain data from the device. The mode is specified through the device ID format. Options include Solicited, Unsolicited, and Mailbox. Descriptions of the modes are as follows:

- **Solicited**: In this mode, the driver generates read and/or write requests to the device for data. Available addresses include output coils, input coils, internal registers, and holding registers. Output coils and holding register addresses have read / write access, whereas input coils and internal registers have read only access. The device ID format for Solicited Mode is *DM.r1.r2.r3.r4.r5* or *S.r1.r2.r3.r4.r5*.
- **Unsolicited**: In this mode, the driver acts as a virtual PLC on the network. Reads and writes do not originate from the driver. Any client application that reads or writes from an unsolicited device passes data to a local cache that is allocated for the device, not to the physical device. This local cache is located in the PC that is running the driver. Devices on the network read and write to the same cache through unsolicited commands. The device ID format for Unsolicited Mode is *DS.r1.r2.r3.r4.r5*.
- **Mailbox**: In this mode, the driver acts as a storage area for every defined mailbox device. When an unsolicited command is received, the driver detects the routing path from which the message came, and then place the data in the storage area allocated for that device. If the message comes from a device with a routing path that has not been defined as a mailbox device, the message is not processed. Any client application that reads from a mailbox device reads from the storage area contained in the driver instead of the physical device; however, writes are sent to the physical device as well as the local cache. Only holding registers are supported in this mode. The Double data type is not supported. The device ID format for Mailbox Mode is *U.r1.r2.r3.r4.r5*.
  Note: Unsolicited mailbox commands are made possible by the MSTR instruction available in certain Modicon devices. For more information, refer to "Example 2 - Mailbox Mode Single Network" and "Example 3 - Mailbox Mode Bridged Network" in **device ID**.

For information on the communication modes that are supported by the SA85 interface card, refer to the table below.

Mode	SA85 Card
Solicited	x
Unsolicited	x
Mailbox	x

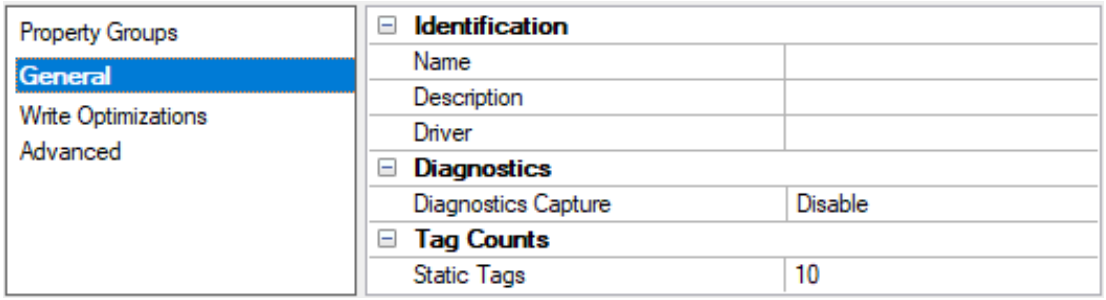
*For more information, refer to **device ID**.*

Polling Multiple Devices

The Modbus Plus Driver can poll multiple devices on a Modbus Plus network and can also act as a single Modbus server device on the Modbus Plus network for other devices to poll. The driver is limited to 8192 devices and supports up to 4 adapters.

Channel Properties — General

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



Identification

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

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

Description: Specify user-defined information about this channel.

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

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

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

Diagnostics

Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead

processing, it is recommended that they be utilized when needed and disabled when not. The default is disabled.

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

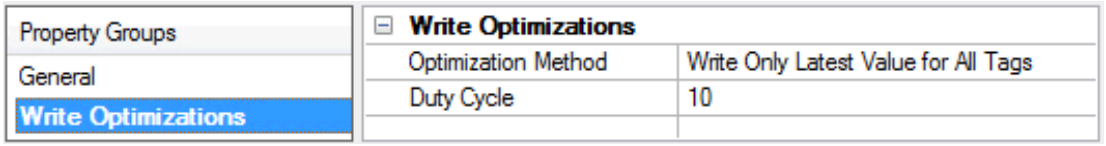
For more information, refer to "Communication Diagnostics" and "Statistics Tags" in the server help.

Tag Counts

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

Channel Properties — Write Optimizations

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



Write Optimizations

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

- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing data to the device as quickly as possible. This mode ensures that everything written from the client applications is sent to the target device. This mode should be selected if the write operation order or the write item's content must uniquely be seen at the target device.
- **Write Only Latest Value for Non-Boolean Tags**: Many consecutive writes to the same value can accumulate in the write queue due to the time required to actually send the data to the device. If the server updates a write value that has already been placed in the write queue, far fewer writes are needed to reach the same final output value. In this way, no extra writes accumulate in the server's queue. When the user stops moving the slide switch, the value in the device is at the correct value at virtually the same time. As the mode states, any value that is not a Boolean value is updated in the server's internal write queue and sent to the device at the next possible opportunity. This can greatly improve the application performance.

  Note: This option does not attempt to optimize writes to Boolean values. It allows users to optimize the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

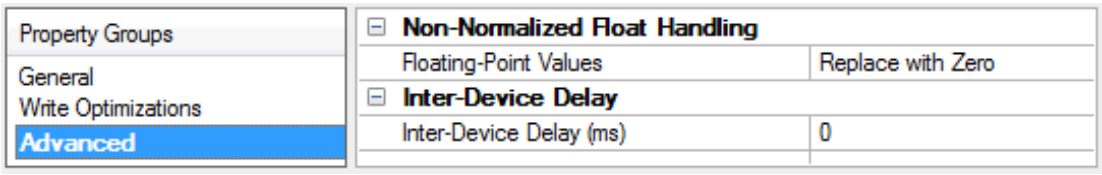
Duty Cycle: is used to control the ratio of write to read operations. The ratio is always based on one read for every one to ten writes. The duty cycle is set to ten by default, meaning that ten writes occur for each read operation. Although the application is performing a large number of continuous writes, it must be ensured that read data is still given time to process. A setting of one results in one read operation for every write

operation. If there are no write operations to perform, reads are processed continuously. This allows optimization for applications with continuous writes versus a more balanced back and forth data flow.

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

Channel Properties — Advanced

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



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

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

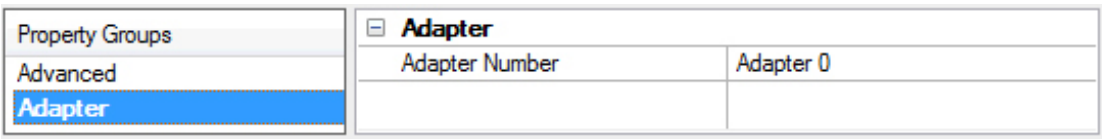
Note: This property is disabled if the driver does not support floating-point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example, EFM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

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

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

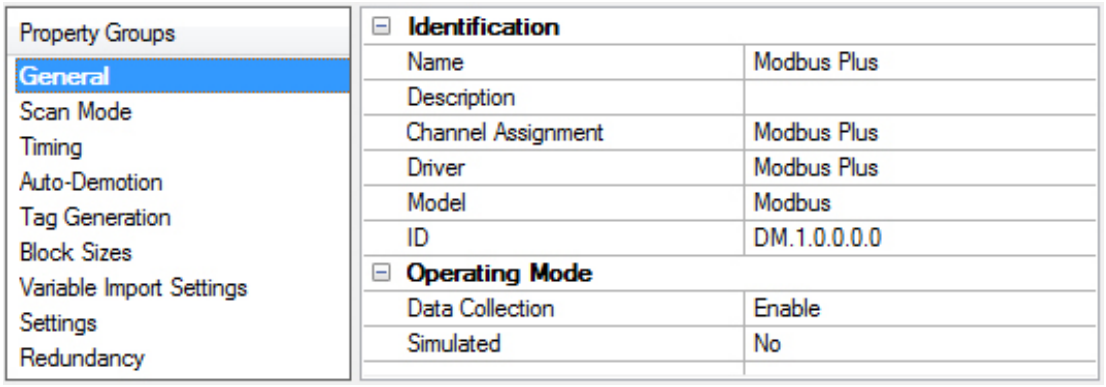
Note: This property is not available for all drivers, models, and dependent settings.

Channel Properties — Adapter



Adapter Number: Specify the number of the adapter to be used by the Modbus Plus card. Valid adapter numbers are 0 to 3. *For card-specific information, refer to* ***Setup***.

Device Properties — General



Identification

Name: User-defined identity of this device.

Description: User-defined information about this device.

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

Driver: Selected protocol driver for this device.

Model: The specific version of the device.

ID: Specify the path to a node on the network. The device ID specifies the path to a node on the network. It consists of five consecutive routing bytes in addition to a mode designator.
For more information, refer to ***Device ID****.*

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 in server help.

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 ID

The device ID specifies the path to a node on the network. It consists of a mode designator and five consecutive routing bytes. The mode may be Data Client (DM), Data Server (DS), or Mailbox. For this driver, the terms Modbus server and unsolicited are used interchangeably.

Solicited Mode (Data Client)

Data client paths start with the prefix DM or S and are used to communicate with another node on the network. The host system acts as the Modbus client in conversations of this type. A DM path can identify a PLC or any other devices that can respond to Modbus Read and Write commands, including another host running the Modbus Plus Driver. The format of a DM path is *DM.r1.r2.r3.r4.r5* or *S.r1.r2.r3.r4.r5*.

Unsolicited Mode (Data Server)

A single data server path can be configured per SA85 card and has the format *DS.1.0.0.0.0*. By defining a server path, users enable the host running the Modbus Plus Driver to simulate a device on the network capable of responding to Read/Write requests from other devices. Other devices can communicate with this simulated device by opening a Data Client path to it.

The simulated device uses Modbus byte ordering: first word is low word of DWord for 32-bit and 64-bit values and first DWord is low DWord for 64-bit values for data encoding. Therefore, the Data Encoding options for the unsolicited device must be set to the following:

- Modbus Byte Order
- First Word Low
- First DWord Low

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

Addresses 1 to 65536 are implemented for output coils, input coils, internal registers and holding registers. The driver responds to any valid request to read or write these values from external devices (Function Codes [decimal] 01, 02, 03, 04, 05, 06, 15, 16). These locations can also be accessed locally by the host PC as tags assigned to the Modbus server device. Write Only access is not allowed for an unsolicited device.

When a Modbus server path is enabled, the Modbus Plus Driver enables eight Modbus server paths on each SA85 card. This allows the remote PLCs and other Modbus Plus devices to access the Modbus server memory of this driver using any of the eight Modbus server paths. The memory accessed is the same in all cases. In terms of an MSTR instruction, users can specify a path of 1 through 8 when defining what path to connect with on the SA85 card serviced by this driver. This can be useful if the application has a large number of remote devices sending data to the PC. If this is the case, users can utilize the eight Modbus server paths to distribute the load from remote nodes. Each Modbus server path in this driver has its own thread of execution to ensure the highest level of performance.

If no Modbus server device is defined in the project, the driver ignores any unsolicited read or write requests it receives.

Mailbox Mode

A Mailbox path starts with the prefix U and provides a path to a physical device. A storage area is provided for this physical device in the Modbus server device defined in the project. Although the physical device sends unsolicited writes to this storage area, they can also be accessed locally by the host PC as tags assigned to the Modbus server device. The format of a mailbox path is *U.r1.r2.r3.r4.r5*.

The driver always opens a Modbus server path when receiving unsolicited mailbox data. The path the driver opens is *DS1.0.0.0.0*. Devices on the same Modbus Plus network communicate with the driver by opening the Data Client path DM.<local node> .1.0.0.0, where the local node is the address set on the SA85 card of the Host Computer. For a description of the path devices on a bridged network use, refer to **Example 3**.

Devices use the Modbus Plus MSTR instruction to provide data to the driver. For the driver to be able to associate the data with a particular device, the device ID path must be embedded in the first five registers of the received data. If the first five registers of data do not match the device ID path of the device in the project, the received data is discarded. Only the Write command is supported for the MSTR instruction.

Notes:

1. The device ID path is embedded in the path from the host PC to the device, not the device path to the host PC.
2. A TIO Module device does not support a Modbus server network address.
3. The device ID should not be changed while OPC clients are connected. If it is, the change does not take effect until all OPC clients are disconnected and then reconnected.

Example 1 - Solicited Mode

A single network consists of four nodes, such that nodes 1 and 4 are host PCs running software that uses the Modbus Plus Driver. Nodes 2 and 3 are PLCs. The following table displays the addressing for the network as from each node.

From	To Node 1	To Node 2	To Node 3	To Node 4
Node 1	----------	DM.2.0.0.0.0	DM.3.0.0.0.0	DM.4.1.0.0.0
Node 2	DM.1.1.0.0.0	----------	DM.3.0.0.0.0	DM.4.1.0.0.0
Node 3	DM.1.1.0.0.0	DM.2.0.0.0.0	----------	DM.4.1.0.0.0
Node 4	DM.1.1.0.0.0	DM.2.0.0.0.0	DM.3.0.0.0.0	----------

Note: To access the simulated device on a host PC, the last non-zero number in the path is always one. This indicates the Modbus server path used by the driver.

Example 2 - Mailbox Mode Single Network

Transferring registers 40020 to 40029 from the device to locations 40001 to 40010 of the host PC. The location of the control block can be different. The host PC address is 7.0.0.0.0. The device address is 3.0.0.0.0.

MSTR Instruction

Control block	40001	-
Data area	40015	Start five registers early
Length	15	Five more than the actual data

Control Block

Register	Contents	Description
40001	1	Write operation
40002	0	Holds error code
40003	15	Number of registers to transfer
40004	1	Starting location in the host PC (Register 40001)
40005	7	Path to host PC
40006	1	Path to host PC
40007	0	Path to host PC
40008	0	Path to host PC
40009	0	Path to host PC

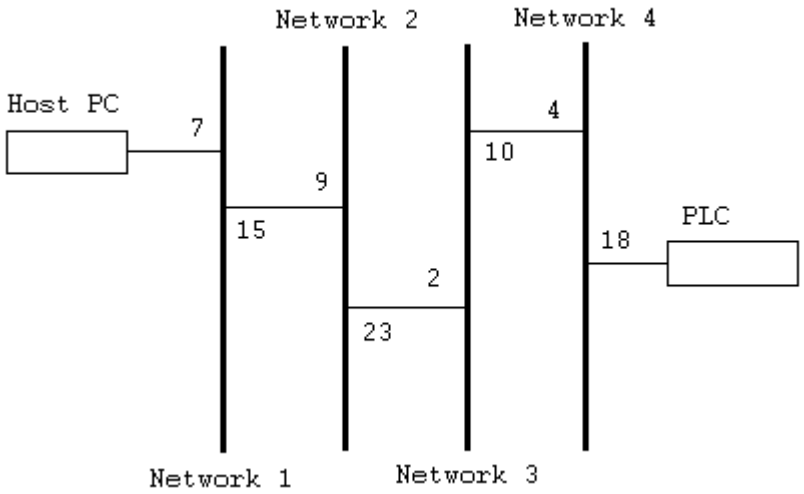
Data Area

Register	Contents	Description
40015	3	Path back to device from host PC, the device ID
40016	0	Path back to device from host PC
40017	0	Path back to device from host PC
40018	0	Path back to device from host PC
40019	0	Path back to device from host PC
40020	-	Actual data start
40029	-	Actual data end

Upon receiving an unsolicited message, the driver does the following:

1. If the message is understood, the driver sends an acknowledgment to the sending device. If messages are received for functions other than **Preset Multiple Registers**, code 0x10, the driver returns a Function Not Implemented response. Preset Multiple Registers is the function code used by devices on the receiving end of an MSTR instruction. The driver returns an exception response if the message is not understood or incomplete.
2. The driver attempts to match the first five registers of data received to the device ID path of a device in the project. If none is found, the data is discarded. If the data is less than six registers, it is discarded immediately.
3. The driver copies n - 5 registers of data starting at the sixth register of the received data to the image map maintained internally for the device (starting at the location indicated in the message). The driver may need to allocate storage for the image map if this is the first data received for these locations.
4. The data is made available to the driver's clients. The data in this example would be referenced as tags with addresses 40001 to 40009 of the device with device ID U.3.0.0.0.0. The client would refer to the device using a logical name assigned when the device was created in the project. The data could also be referenced as an array, such as 40001[10] or 40001[2][5].

Example 3 - Mailbox Mode Bridged Network



The host PC's address from the PLC's perspective is 4.2.9.7.1. The PLC's address from the host PC's perspective is 15.23.10.18.0. This is the device ID path. If the same registers were transferred from the PLC to the same locations in the host PC, the control block and data area would be used in the MSTR instruction according to the tables below. The message would be processed the same.

Note: When using this driver, the host PC can be three networks apart from a device at the most.

MSTR Instruction

Control block	40001	
Data area	40015	Start five registers early.
Length	15	Five more than the actual data.

Control Block

Register	Contents	Description
40001	1	Write operation
40002	0	Holds error code
40003	15	Number of registers to transfer
40004	1	Starting location in the host PC (Register 40001)
40005	4	Path to host PC
40006	2	Path to host PC
40007	9	Path to host PC
40008	7	Path to host PC
40009	1	Path to host PC

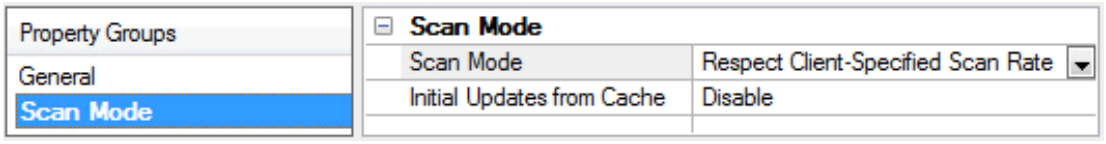
Data Area

Register	Contents	Description
40015	15	Path back to device from host PC, the device ID
40016	23	Path back to device from host PC
40017	10	Path back to device from host PC

Register	Contents	Description
40018	18	Path back to device from host PC
40019	0	Path back to device from host PC
40020	-	Actual data start
40029	-	Actual data end

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.



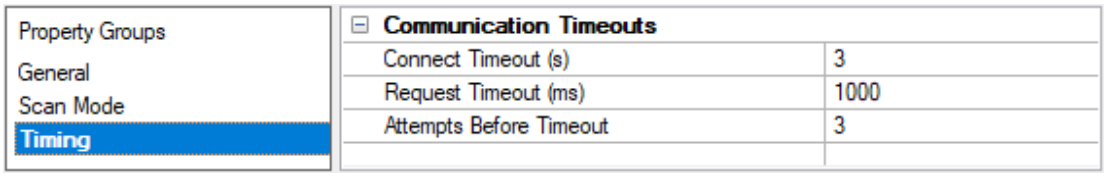
Scan Mode: Specify how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

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

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

Device Properties — Timing

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



Communications Timeouts

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

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

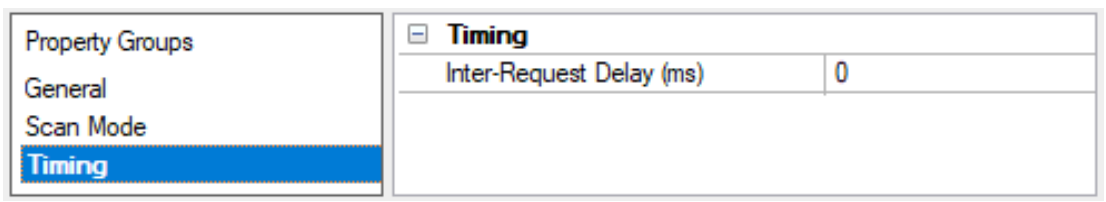
Request Timeout: Specify an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9999 milliseconds (167 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

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

Timing

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

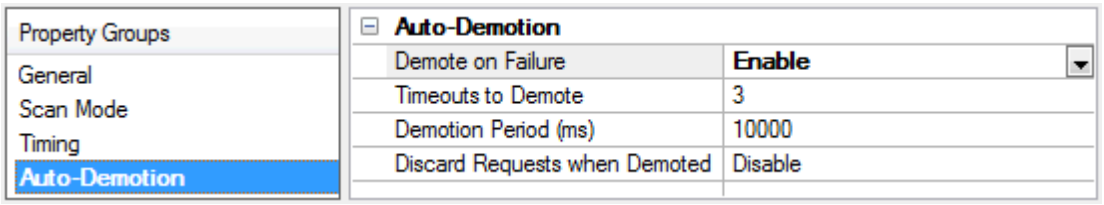
Note: Not all drivers support Inter-Request Delay. This setting does not appear if it is not available.



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

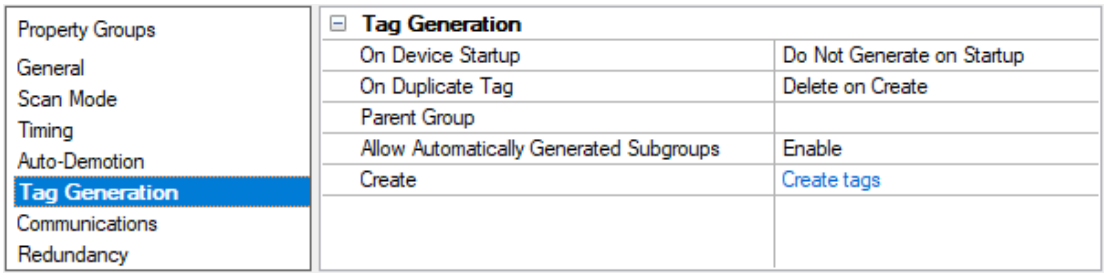
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the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

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

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

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

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

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

Note: Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags.

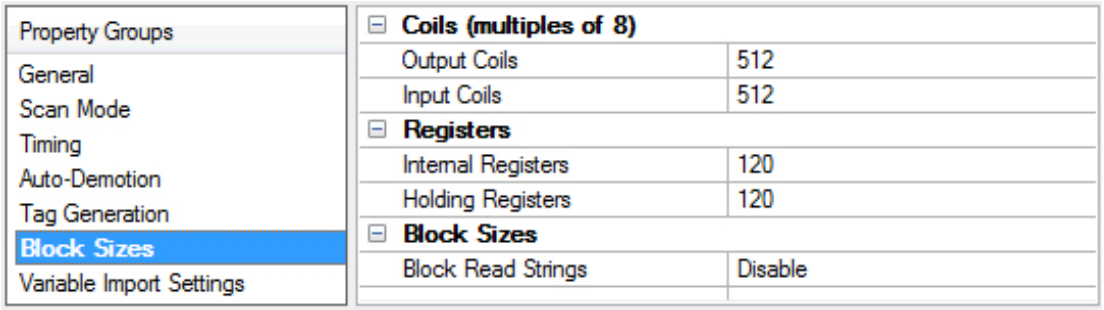
Users should avoid adding tags to the server using names that may match tags that are automatically generated by the driver.

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

Allow Automatically Generated Subgroups: This property controls whether the server automatically creates subgroups for the automatically generated tags. This is the default setting. If disabled, the server generates the device's tags in a flat list without any grouping. In the server project, the resulting tags are named with the address value. For example, the tag names are not retained during the generation process.
Note: If, as the server is generating tags, a tag is assigned the same name as an existing tag, the system automatically increments to the next highest number so that the tag name is not duplicated. For example, if the generation process creates a tag named "AI22" that already exists, it creates the tag as "AI23" instead.

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

Device Properties — 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.

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

Notes:

1. Coil size must be a multiple of 8.
2. For MBX, NETLIB, or NONE; the default is 512 and the maximum is 2000.
3. This property is disabled when an OPC client is connected.

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.

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.

Notes:

1. For MBX, NETLIB, or NONE; the default is 120 and the maximum is 125.
2. This property is disabled when an OPC client is connected.
3. For a TIO Module, use this setting to inform the driver how many bytes are returned when reading data location 400001. For modules that return 2 bytes, set this to 1. For modules that return 3 bytes, set this to 2. The driver uses fixed block lengths (independent from this setting) for all other data locations.
4. The device may not support block read / write operations of the default size. Smaller Modicon PLCs and non-Modicon devices may not support the maximum data transfer lengths supported by the MBPlus network.
5. The device may contain non-contiguous addresses. If this is the case and the driver attempts to read a block of data that encompasses undefined memory, the device probably reject the request.

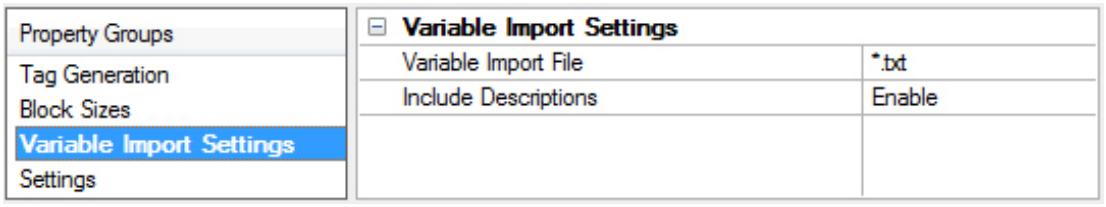
Caution: If the block sizes value is set above 120 and a 32 or 64-bit data type is used for any tag, a "Bad address in block" error can occur. Decrease block size value to 120 to prevent the error from occurring.

Block Sizes

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

Device Properties — Variable Import Settings

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



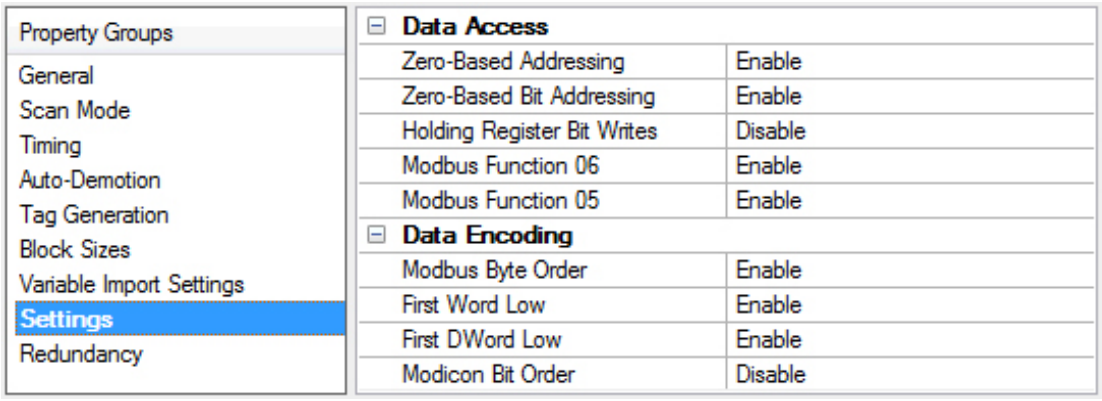
Variable Import File: Specifies the exact location of the variable import file the driver should use when automatic tag database generation is enabled for this device.

Include Descriptions: Enable to use imported tag descriptions (if present in file).

*For more information on how to configure automatic tag database generation and how to create a variable import file, refer to **Automatic Tag Database Generation**.*

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

Device Properties — Settings



Data Access

Zero-Based Addressing: If the address numbering convention for the device starts at one instead of zero, disable. By default, user-entered addresses have one subtracted when frames are constructed to communicate with a Modbus device. If the device doesn't follow this convention, disable **zero-based addressing**. The default behavior follows the convention of the Modicon PLCs.

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

Holding Register Bit Writes: When writing to a bit location within a holding register, the driver should only modify the bit of interest. Some devices support a 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. The default setting is disabled. Enable if the device supports holding register bit access and the driver uses function code 0x16, regardless of the setting for Modbus Function 06. If this setting is disabled, the driver uses either function code 0x06 or 0x10, depending on the Modbus Function 06 property for single register writes.

Notes:

1. When Modbus Byte Order is disabled, the byte order of the masks sent in the command is Intel byte order.

Modbus Function 06: The Modbus Plus Driver has the option of using two Modbus protocol functions to write Holding register data to the target device. In most cases, the driver switches between these two functions based on the number of registers being written. When writing a single 16-bit register, the driver uses the 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 that have implemented the Modbus protocol. Many of these devices support only the use of Modbus function 16 to write to Holding registers regardless of the number of registers to be written.

Modbus Function 06 is used to force the driver to use only Modbus function 16 (if needed). This selection is enabled by default, allowing the driver to switch between 06 and 16 as needed. If a device requires all writes to be done using only Modbus function 16, disable this option.

Note: For bit within word writes, the **Holding Register Bit Writes** property takes precedence over **Modbus Function 06**. If Holding Register Bit Writes is enabled, then function code 0x16 is used no matter what the selection for this property. If it is disabled, this property determines whether function code 0x06 or 0x10 is used for bit within word writes.

Modbus Function 05: The Modbus Plus Driver can use two Modbus protocol functions to write output coil data to the target device. In most cases, the driver switches between these two functions based on the number of coils being written. When writing a single coil, the driver uses the Modbus function 05. When writing an array of coils, the driver uses Modbus function 15. For the standard Modicon PLC, the use of either of these functions is not a problem. There are, however, a large number of third-party devices that have implemented the Modbus protocol. Many of these devices support only the use of Modbus function 15 to write to output coils regardless of the number of coils to be written.

The Modbus Function 05 selection is used to force the driver to use only Modbus function 15 if needed. This property is enabled by default, allowing the driver to switch between 05 and 15 as needed. If a device requires all writes to be done using only Modbus function 15, disable this selection.

Data Encoding

Modbus Byte Order: The byte order used by the Modbus Plus Driver can be changed from the default Modbus byte ordering to Intel byte ordering by using this selection. This selection is enabled by default and is the normal setting for Modbus compatible devices. If the device uses Intel byte ordering, disabling this option allows the driver to properly read Intel formatted data.

First Word Low: Two consecutive registers' addresses in a Modbus device are used for 32-bit data types. Users can specify whether the driver should assume the first word is the low or the high word of the 32-bit value. The default (First Word Low) follows the convention of the Modicon Modsoft programming software.

First DWord Low: Four consecutive registers' addresses in a Modbus device are used for 64-bit data types. Users can specify whether the driver should assume the first DWord is the low or the high DWord of the 64-bit value. The default (First DWord Low) follows the default convention of 32-bit data types.

Modicon Bit Order: When enabled, the driver reversed 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 by default.

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

Data Encoding Options Details

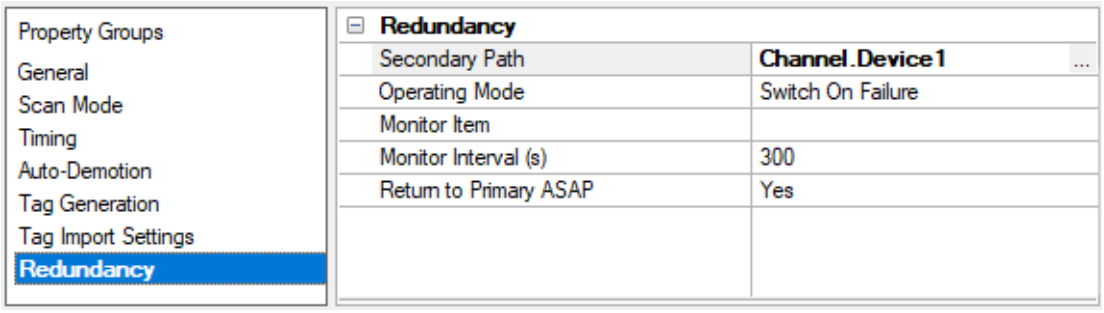
- Modbus Byte Order option sets the data encoding of each register / 16-bit value.
- First Word Low sets the data encoding of each 32-bit value and each double word of a 64-bit value.
- First DWord Low sets the data encoding of each 64-bit value.

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 particular device's documentation to determine the correct settings of the Data Encoding options. The default settings are acceptable for most Modbus devices.

Modbus Byte Order Enabled	High Byte (15..8)	Low Byte (7..0)
Modbus Byte Order Disabled	Low Byte (7..0)	High Byte (15..8)
First Word Low Disabled	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 Enabled	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 Disabled	High Double Word (63..32)	Low Double Word (31..0)

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 Plus Driver utilizes the Automatic Tag Database Generation, which automatically creates tags that access data points used by the device's ladder program. Although it is sometimes possible to query a device for the information needed to build a tag database, this driver must use a Variable Import File instead. Variable import files can be generated using the Concept and ProWORX device programming applications.

Creating the Variable Import File

The import file must be in semicolon-delimited .TXT format, which is the default export file format of the Concept device programming application. The ProWORX programming application can export variable data in this format.

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

Server Configuration

Automatic Tag Database Generation can be customized to fit the application's needs. The primary control options can be set either during the database creation through the wizard or the device properties.

For more information, refer to the server help documentation.

Modbus Plus Driver requires other settings in addition to the basic settings common to all drivers that support automatic tag database generation. Such specialized settings include the requiring the name and location of the variable import file. This information can be specified either under Variable Import Settings in device properties.

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

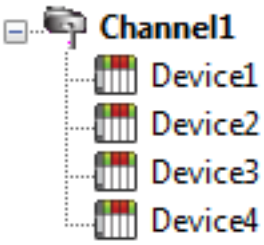
Operation

Depending on the specific configuration, tag generation may start automatically when the server project opens or be initiated manually at some other time. The Event Log shows when the tag generation process started, any errors that occurred while processing the variable import file, and when the process completed.

Optimizing Communications

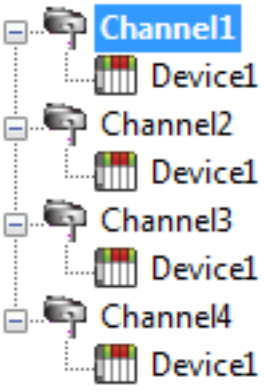
The Modbus Plus Driver has been designed to provide better throughput and take full advantage of the SA85 card. Previously, the Modbus Plus Driver restricted users to configuring a single channel in the server project and required that all Modbus Plus devices that would be accessed be defined under this channel. This meant that the driver had to move between devices one at a time to make requests. Since the OPC Server was already designed to be efficient, the single channel scheme provided enough performance for most application. With the advent of OPC as an enabling technology, however, the size of projects has increased dramatically. To maintain a high level of performance, the Modbus Plus Driver is designed to operate at a high level of efficiency and performance.

Note: Before beginning these changes, back up the server project directory to return to previous settings if needed.



In this project, there is only one channel defined. All devices that need to be accessed are defined under that one channel. The Modbus Plus Driver must move from one device to the next as quickly as possible to gather information at an effective rate. As more devices are added or more information is requested from a single device, the update rate begins to suffer.

The latest version of the Modbus Plus Driver uses multiple channel definitions to boost the application's performance. In this configuration, each channel in the server represents a separate path of execution. By adding additional channels, the application's work load is spread across the new channels. This creates multiple paths of execution that run independently, and results in a significant increase in performance. The image below shows the same application reconfigured to use multiple channels.



Each device can be defined under its own channel. In this configuration, the server can dedicate a single path of execution to the task of gathering data from a single device because each has its own dedicated channel. If the application has fewer devices it can be optimized as displayed.

Even if the application has more devices, there is a gain. While fewer devices may be ideal, the application still benefits from additional channels. Although this means that within a given channel the server must move from device to device, it can now do so with less devices to process on a single path.

Note: The channel limits match the multi-path limitations of the SA85 card as set by the manufacturer.

The application can be redesigned to support multiple channels even if there are a large number of tags defined under each device. For more information, follow the instructions below.

1. In the existing project that is single channel-based, click **Connectivity**| **New Channel** and then name it as desired.
2. Cut **PLC2** from the **ModbusPlus** channel.
3. Paste it under the new channel. The cut and paste functions quickly modify the application to take advantage of the new Modbus Plus Driver.

These examples highlight the most obvious optimizations that are possible with the Modbus Plus Driver. Other possible optimizations include dedicating a single channel to just Global data. To do so, define a new set of device names for each device with global data to be accessed under that new channel. Remember to only access Global data from these newly defined device names.

Data Types Description

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

* The descriptions assume the default 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
TIO Module Addressing

Modbus Addressing

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

5-Digit Addressing vs. 6-Digit Addressing

In Modbus addressing, the first digit of the address specifies the primary table. The remaining digits represent the device's data item. The maximum value is a two-byte unsigned integer (65,535). Six digits are required to represent the entire address table and item. As such, addresses that are specified in the device's manual as 0xxxx, 1xxxx, 3xxxx, or 4xxxx are padded with an extra zero once applied to the Address field of a Modbus tag.

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

Modbus Addressing in Decimal Format

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

Address Type	Range	Data Type	Access	Function Codes
Output Coils	000001-065536	Boolean	Read/Write	01, 05, 15
Input Coils	100001-165536	Boolean	Read Only	02
Internal Registers	300001-365536 300001-365535 300001-365533 3xxxxx.0/1-3xxxxx.15/16** 300001.2H-365536.240H*** 300001.2L-365536.240L***	**Word**, Short, BCD Float, DWord, Long, LBCD Double **Boolean** String String	Read Only* Read Only* Read Only* Read Only* Read Only Read Only	04 04 04 04 04 04
Holding Registers	400001-465536 400001-465535 400001-465533 4xxxxx.0/1-4xxxxx.15/16**	**Word**, Short, BCD Float, DWord, Long, LBCD Double **Boolean**	Read/Write Read/Write Read/Write Read/Write	03, 06, 16 03, 06, 16 03, 06, 16 03, 06, 16, 22

Address Type	Range	Data Type	Access	Function Codes
	400001.2H-465536.240H***	String	Read/Write	03, 16
	400001.2L-465536.240L***	String	Read/Write	03, 16
Global Data	G01-G32	**Word**, Short, BCD	Read/Write	N/A
	G01-G31	Float, DWord, Long, LBCD	Read/Write	N/A
	G01-G29	Double	Read/Write	N/A
	Gxx.0/1-Gxx.15/16**	**Boolean**	Read Only	N/A

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

Modbus Addressing in Hexadecimal Format

Address Type	Decimal Range	Data Type	Access
Output Coils	H000001-H010000	Boolean	Read/Write
Input Coils	H100001-H110000	Boolean	Read Only
Internal Registers	H300001-H310000	**Word**, Short, BCD	Read Only*
	H300001-H30FFFF	Float, DWord, Long, LBCD	Read Only*
	H300001-H30FFFD	Double	Read Only*
	H3yyyyy.0/1-H3yyyyy.F/10	**Boolean**	Read Only*
	H300001.2H-H3FFFF.240H	**String**	Read Only
	H300001.2L-H3FFFF.240L	**String**	Read Only
Holding Registers	H400001-H410000	**Word**, Short, BCD	Read/Write
	H400001-H40FFFF	Float, DWord, Long, LBCD	Read/Write
	H400001-H40FFFD	Double	Read/Write
	H4yyyyy.0/1-H4yyyyy.F/10	**Boolean**	Read/Write
	H400001.2H-H4FFFF.240H	String	Read/Write
	H400001.2L-H4FFFF.240L	String	Read/Write
Global Data	HG01-HG20	**Word**, Short, BCD	Read/Write
	HG01-HG1F	Float, DWord, Long, LBCD	Read/Write
	HG01-HG1D	Double	Read/Write
	HGyy.0/1-HGyy.F/10	**Boolean**	Read Only

* For Modbus server devices, these locations are Read/Write.
** .Bit is string length, range 2 to 240 bytes.

Packed Coils

The Packed Coil address type allows access to multiple consecutive coils as an analog value. This feature is only available when the Modbus model is in Modbus Client Mode. The syntax is as follows:

Output Coils: 0*xxxxx*#*nn*
Input Coils: 1*xxxxx*#*nn*

where

- *xxxxx* is the address of the first coil. Both decimal and hexadecimal values are allowed.
- *nn* is the number of coils packed into an analog value. The valid range is 1-16, and only decimal values are allowed.

Note: The only valid data type is Word. Output Coils have Read/Write access, whereas Input Coils have Read Only access. The bit order is such that the start address is the Least Significant Bit (LSB) of analog value.

Write Only Access

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

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

Mailbox Mode

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

Note: The Double data type is not supported.

String Support

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

Examples

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

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

Array Support

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

4*xxxx* [rows] [cols]
4*xxxx* [cols] with assumed row count equal to one.

For Word, Short and BCD arrays, the base address + (rows * cols) cannot exceed 65536. For Float, DWord, Long and Long BCD arrays, the base address + (rows * cols * 2) cannot exceed 65535. For all arrays, the total number of registers being requested cannot exceed the holding register block size that was specified for this device.

Note: The base address for Global Data cannot exceed 32.

Function Codes Description

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

Configuring the Device for Global Data Communications

Global Data is supported by the SA85 interface card. It is only accessible from a single network. For example, "7.0.0.0.0" can access global data, but "7.1.0.0.0" cannot.

Note: Unsolicited mode does not support Global Data.

Writing Global Data to a Device

The host PC's address from the PLC's perspective is 2.0.0.0.0. The PLC's address from the host PC's perspective is 9.0.0.0.0. This is the device ID path. Users must configure the addresses that the device can read to and write from in the programming software.

Control Block

Register	Contents	Description
Control [1]	5	Function Code for writing Global Data
Control [2]	- 0 = No	The error code. This may not be changed

Register	Contents	Description
	Error	
Control [3]	32	The number of words to write from state RAM to global memory; the maximum is 32 bits
Control [4]	-	Reserved*
Control [5]	2	The Modbus Plus node address to which data is being sent
Control [6]	0	Path to host PC
Control [7]	0	Path to host PC
Control [8]	0	Path to host PC
Control [9]	0	Path to host PC

* This register is application-specific.

Data Area

Register	Contents	Description
DataField [1]-DataField [32]	Data	N/A

Reading Global Data from the Device

The host PC's address from the PLC's perspective is 2.0.0.0.0. The PLC's address from the host PC's perspective is 9.0.0.0.0. This is the device ID path.

Control Block

Register	Contents	Description
Control [1]	6	Function Code for reading Global Data
Control [2]	- 0 = No Error	The error code. This may not be changed
Control [3]	32	The number of words to write from state RAM to global memory; the maximum is 32 bits
Control [4]	-	Reserved*
Control [5]	2	The Modbus Plus node address from which data is read
Control [6]	0	Path to host PC
Control [7]	0	Path to host PC
Control	0	Path to host PC

Register	Contents	Description
[8]		
Control [9]	0	Path to host PC

* This register is application-specific.

Data Area

Register	Contents	Description
DataField [1]-DataField [32]	Data	N/A

TIO Module Addressing

Mailbox Mode is not supported for this model.

TIO Module Addressing in Decimal

Address Type	Range	Data Type	Access
Data I/O*	400001 400001.0/1-400001.15/16**	**Word**, Short **Boolean**	Read/Write Read/Write
Data Input - Latched	400257 400257.0/1-400257.15/16**	**Word**, Short **Boolean**	Read Only Read Only
Module Timeout	461441 461441.0/1-461441.15/16**	**Word**, Short **Boolean**	Read/Write Read/Write
Module Status	463489-463497 4xxxxx.0/1-4xxxxx.15/16**	**Word**, Short **Boolean**	Read Only Read Only
Module ASCII Header	464513	String	Read Only

* The value read from a Data I/O location comes from the module's input register. When writing to this location, the value that is sent modifies the module's output register. Therefore, the value read at this location does not correspond to the value previously written to this location.
** *For more information, refer to "Zero vs. One Based Addressing" in* ***Settings****.*

TIO Module Addressing in Hexadecimal

Address Type	Range	Data Type	Access
Data I/O*	H40001 H40001.0/1-H40001.F/10	**Word**, Short **Boolean**	Read/Write Read/Write
Data Input - Latched	H40101 H40101.0/1-40101.F/10	**Word**, Short **Boolean**	Read Only Read Only
Module Timeout	H4F001 H4F001.0/1-H4F001.F/10	**Word**, Short **Boolean**	Read/Write Read/Write
Module Status	H4F801-H4F809 H4yyyy.0/1-H4yyyy.F/10	**Word**, Short **Boolean**	Read Only Read Only
Module ASCII Header	H4FC01	String	Read Only

* The value read from a Data I/O location comes from the module's input register. When writing to this location, the value that is sent modifies the module's output register. Therefore, the value read at this location does not correspond to the value previously written to this location.

Event Log Messages

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

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

Error Type:
Error

Possible Cause:
An attempt has been made to reference a nonexistent location in the specified device.

Possible Solution:
Verify the addresses of all tags assigned to the device and eliminate ones that reference invalid locations.

Bad address in block. | Block Range = H<start> to H<end>.

Error Type:
Error

Possible Cause:
An attempt has been made to reference a nonexistent location in the specified device.

Possible Solution:
Verify the addresses of all tags assigned to the device and eliminate ones that reference invalid locations.

Unable to start MBPLUS.SYS device.

Error Type:
Error

Possible Cause:
The MBPLUS.SYS driver is not properly configured.

Possible Solution:
Verify that the MBPLUS device can be started and stopped manually using the Control Panel | Devices applet. When the MBPLUS.SYS driver is started manually, the modbus_unsolicited.dll driver can start the driver.

Unable to detect card or start Modbus Plus Services. Verify the card and MBP *.sys drivers are installed properly.

Error Type:
Error

Unable to create system resources required to run this driver.

Error Type:
Error

Unable to initialize channel.

Error Type:
Error

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

Error Type:
Error

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

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

Unable to load channel. Only one channel is allowed per Hilscher adapter. Modify the project so each channel has a unique adapter and reload.

Error Type:
Error

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

Error Type:
Error

Error opening MBPLUS path. | Path = '<path>'.

Error Type:
Warning

Possible Cause:

1. The MBPLUS.SYS driver is not properly configured.
2. The driver cannot open a path on the specified adapter.

Possible Solution:

1. Follow the instructions for installing and configuring the MBPLUS driver.
2. Verify that no more than eight channels are assigned the same adapter number.

Received block length does not match expected length. | Received length = <number> (bytes), Expected length = <number> (bytes).

Error Type:
Warning

Global data not available from device.

Error Type:
Warning

Error reading global data from device.

Error Type:
Warning

Block request on device responded with exception. | Block Range = <start> to <end>, Exception = <code>.

Error Type:
Warning

Possible Cause:
The requested node did not respond.

Possible Solution:
Check the cabling, wiring, and pinning.

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

Error Type:
Warning

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

Possible Solution:
See Modbus Exception Codes.

See Also:
Modbus Exception Codes

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

Error Type:
Warning

Possible Cause:

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

Possible Solution:

See Modbus Exception Codes.

See Also:

Modbus Exception Codes

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

Error Type:

Warning

Possible Cause:

The requested node did not respond.

Possible Solution:

Check the cabling, wiring, and pinning.

Warning: Global Data Disabled, access requires Modicon's 4.0 low-level system drivers.

Error Type:

Warning

Unable to open adapter. | Adapter = <name>.

Error Type:

Warning

Tag import failed due to low memory resources.

Error Type:

Warning

Possible Cause:

The driver cannot allocate memory required to process variable import file.

Possible Solution:

Shut down all unnecessary applications and try again.

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 = <number>.

Error Type:
Warning

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

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

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

Error Type:
Warning

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

Possible Solution:

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

See Also:

Exporting Variables from Concept

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

Error Type:

Warning

Possible Cause:

1. The adapter may not exist.
2. Depends on the error code provided.

Possible Solution:

Verify that the proper adapter number has been chosen in channel properties. Use SyCon to determine adapter ordering.

Note:

Does not apply to the SA85 card. Code -1, -33 for the Hilscher CIF card.

See Also:

SyCon User Manual

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

Error Type:

Warning

Possible Cause:

1. The adapter may not exist.
2. Depends on the error code provided.

Possible Solution:

Verify that the proper adapter number has been chosen in channel properties. Use SyCon to determine adapter ordering.

See Also:

SyCon User Manual

Started MBPLUS.SYS device

Error Type:

Informational

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

Error Type:

Informational

Modbus Exception Codes

The following data is from Modbus Application Protocol Specifications documentation.

Code Dec/Hex	Name	Meaning
01/0x01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the server. This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server is in the wrong state to process a request of this type, for example, because it is unconfigured and is being asked to return register values.
02/0x02	ILLEGAL DATA ADDRESS	The data address received in the query is not an allowable address for the server. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed. A request with offset 96 and length 5 generates exception 02.
03/0x03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the Modbus protocol is unaware of the significance of any particular value of any particular register.
04/0x04	SERVER DEVICE FAILURE	An unrecoverable error occurred while the server was attempting to perform the requested action.
05/0x05	ACKNOWLEDGE	The server has accepted the request and is processing it, but a long duration of time is required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed.
06/0x06	SERVER DEVICE BUSY	The server is engaged in processing a long-duration program command. The client should retransmit the message later when the server is free.
07/0x07	NEGATIVE ACKNOWLEDGE	The server cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 decimal. The client should request diagnostic or error information from the server.
08/0x08	MEMORY PARITY ERROR	The server attempted to read extended memory, but detected a parity error in the memory. The client can retry the request, but service may be required on the server device.
10/0x0A	GATEWAY PATH	Specialized use in conjunction with gateways indicates that the gateway

Code Dec/Hex	Name	Meaning
	UNAVAILABLE	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 server 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