Yaskawa Memobus Plus Driver

Table of Contents

Yaskawa Memobus Plus Driver

Help version 1.016

CONTENTS

Overview

The Yaskawa Memobus Plus Driver provides a reliable way to connect Yaskawa Memobus Plus (SA85) devices to OPC client applications; including HMI, SCADA, Historian, MES, ERP, and countless custom applications. It is intended for use with a Modicon SA85 Network card.

The driver can poll multiple devices (PLCs) on a Memobus Plus network. It can also act as a single slave device on the Memobus Plus network for other devices to poll. The driver is currently limited to eight channels and 2048 devices. It is only designed to use Adapter 0.

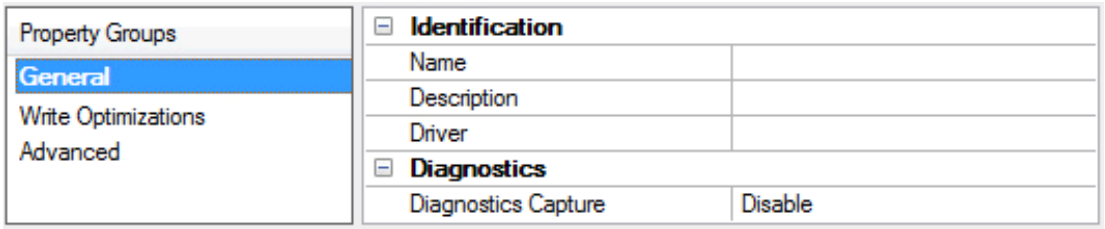
The Yaskawa Memobus Plus Driver now supports the use of additional channel definitions for improved performance. These additional channels allow users to configure applications to take full advantage of the SA85 card. For more information on using multiple channels, refer to **Optimizing Communications**.

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 OPC server can support up to 8 Modbus Plus channels per SA85 or PCI-85 card.

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.
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.
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

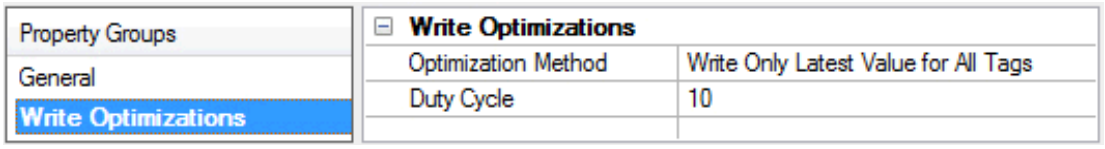
Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications. Because the server's diagnostic features require a minimal amount of overhead processing, it is recommended that they be utilized when needed and disabled when not. The default is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

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

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

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

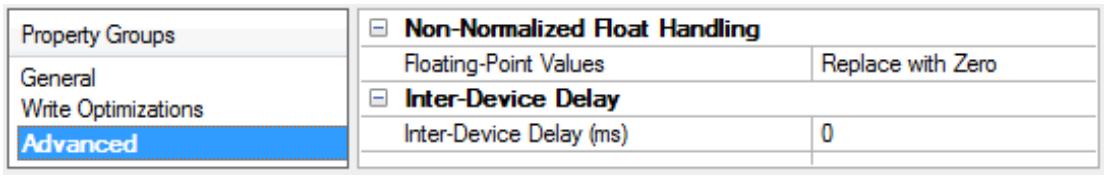
value to the device. This mode optimizes all writes by updating the tags currently in the write queue before they are sent. This is the default mode.

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 — 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.

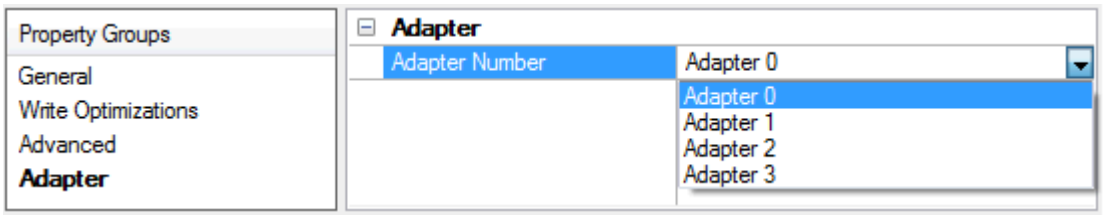
Note: This property is not available if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example,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

Channel Properties — Adapter

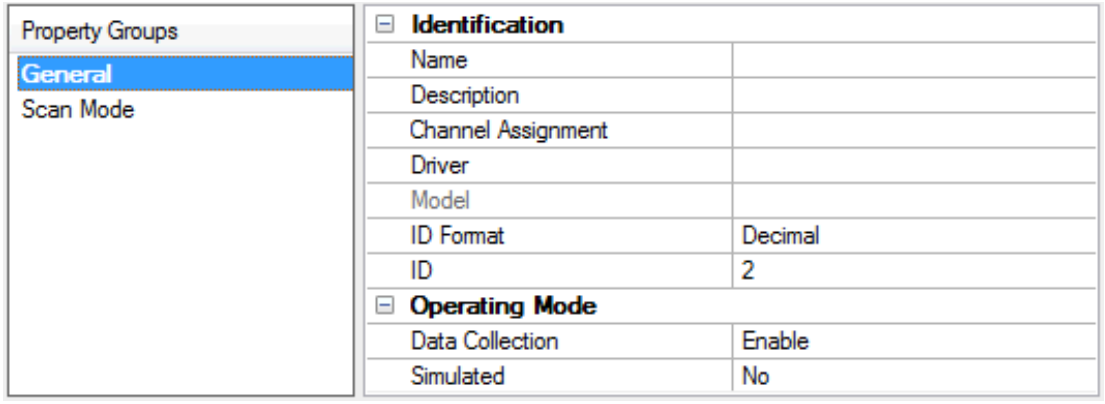


Adapter

Adapter Number: Specify the adapter number to be used by the Memobus Plus card. Up to eight channels may be assigned the same adapter number. Valid adapter numbers are 0 to 3.

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. This property specifies the driver selected during channel creation. It is disabled in the channel properties.

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 station / node / identity / address. The type of ID entered depends on the communications driver being used. For many 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 ID format can be Decimal, Octal, and Hexadecimal. 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.

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 ID (PLC Network Address)

The Device ID specifies the path to a node on the network, and consists of a path type designator (either slave or master) and five consecutive routing bytes. The **Data Master** address paths start with the prefix DM and are used to communicate with another node on the network. The host PC acts as the master in conversations of this type. A DM path can identify a PLC or any other devices that can respond to Memobus read and write commands, including another host PC running the Memobus Plus driver. The format of a DM path is **DM.r1.r2.r3.r4.r5**.

One **Data Slave** path is allowed per project and has the format **DS.1.0.0.0.0**. A slave path will enable the host PC running the Memobus Plus driver to simulate a PLC 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 Master path to it. Addresses 1 to 65536 are implemented for output coils, input coils, internal registers and holding registers. The driver will respond to any valid request to read or write these values from external devices. These locations can also be accessed locally by the host PC as tags assigned to the slave device.

If a slave device is not defined in the project, the driver will ignore any unsolicited read or write requests it receives.

Note: A TIO Module device does not support a slave network address.

Addressing Example

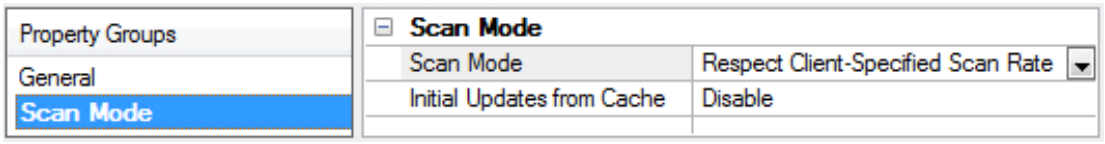
Suppose that the single network consists of four nodes such that nodes 1 and 4 are host PCs running software that uses the Memobus Plus driver and nodes 2 and 3 are PLCs. The following table identifies the addressing for the network as seen from each node.

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 must always be one. This indicates the slave path the driver uses.

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:

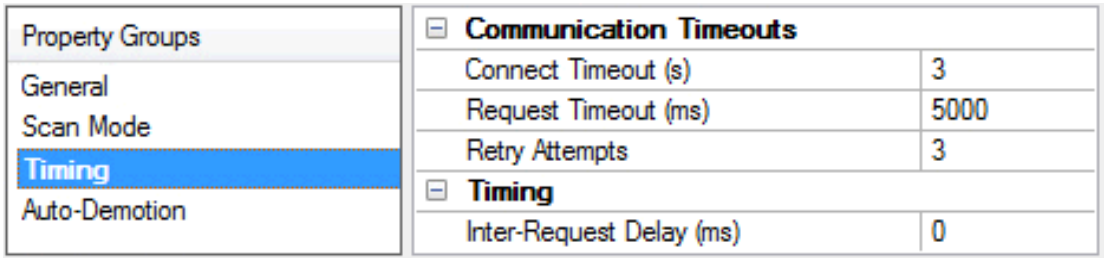
- **Respect Client-Specified Scan Rate**: This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate**: This mode specifies the maximum scan rate to be used. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
  Note: When the server 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 — 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: 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.

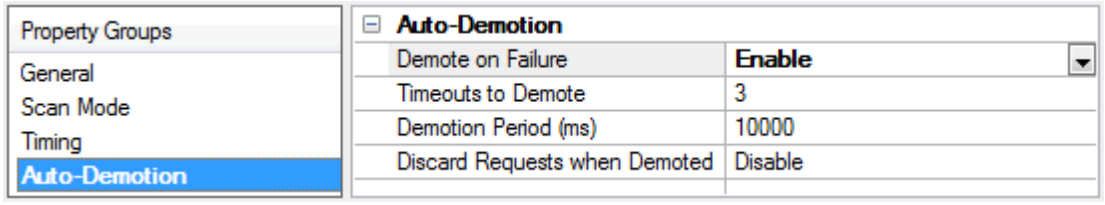
Timing

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.

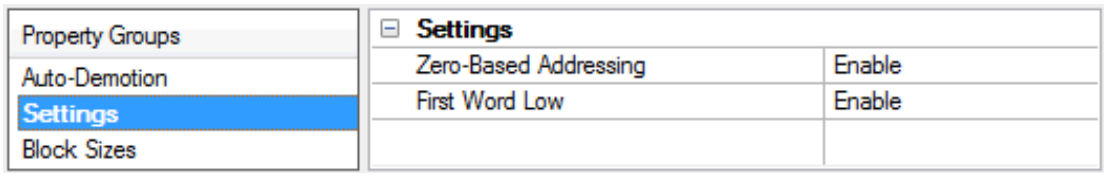
Tip: Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

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 — Settings



Zero-Based Addressing: If the address numbering convention for the device starts at one as opposed to zero, this property can be specified. By default, addresses have one subtracted when frames are constructed to communicate with a Yaskawa Memobus device. This means that if holding register 40001 is requested, the driver actually ask for the zero's holding register in the device. If the device doesn't follow this convention, disable **Zero-Based Addressing** in Device Properties. The default behavior follows the convention of the Yaskawa PLCs.

First Word Low: Two consecutive registers addresses in a Yaskawa Memobus device are used for 32-bit data types. 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 Yaskawa Memosoft programming software.

Device Properties — Block Sizes

Coils

Coils: Coils can be read from 8 to 800 points (bits) at a time. The default is 512 coils.

Registers

Registers: Registers can be read from 1 to 120 locations (words) at a time. The default is 120 registers.

Reasons to Change the Default Block Sizes

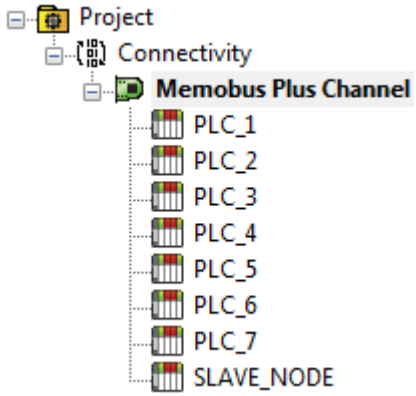
1. The device may not support block read/write operations of the default size. Smaller Yaskawa PLCs and non-Yaskawa devices may not support the maximum data transfer lengths supported by the Memobus Plus network.
2. The device may contain non-contiguous addresses. If this is the case and the driver attempts to read a block of data that encompasses undefined memory, the device will probably reject the request.

Optimizing Communications

The Yaskawa Memobus Plus Driver has been redesigned to provide better throughput and take full advantage of the SA85 card. In the past, the Yaskawa Memobus Plus Driver restricted users to configuring a single channel in the OPC Server project. All Memobus Plus devices were defined under this one channel. In simple terms, this meant that the driver had to move between devices one at a time making requests. The OPC Server was already designed to be efficient and for most applications the single channel scheme provides more than enough performance. With the advent of OPC as an enabling technology, the size of projects have increased dramatically and have also caused a decrease in performance. To remedy this, the Yaskawa Memobus Plus Driver operates at a new level of efficiency and performance.

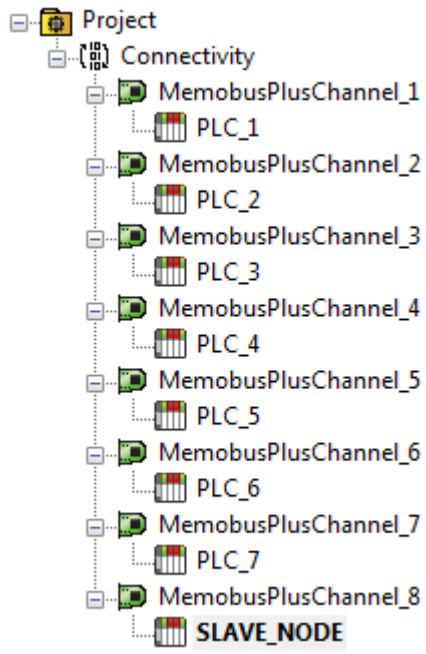
Note: Before implementing any changes, it is suggested that users make a backup of their existing OPC Server project directory in order to return to the previous current settings quickly, if needed.

An example of how the application may appear when configured using a single channel is shown below.



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

If the Yaskawa Memobus Plus Driver could only define one single channel, then the example shown above would be the only option available; however, the Yaskawa Memobus Plus Driver can define up to 8 channels. Using multiple channels distributes the data collection workload by simultaneously issuing multiple requests to the network. An example of how the same application may appear when configured using multiple channels to improve performance is shown below.



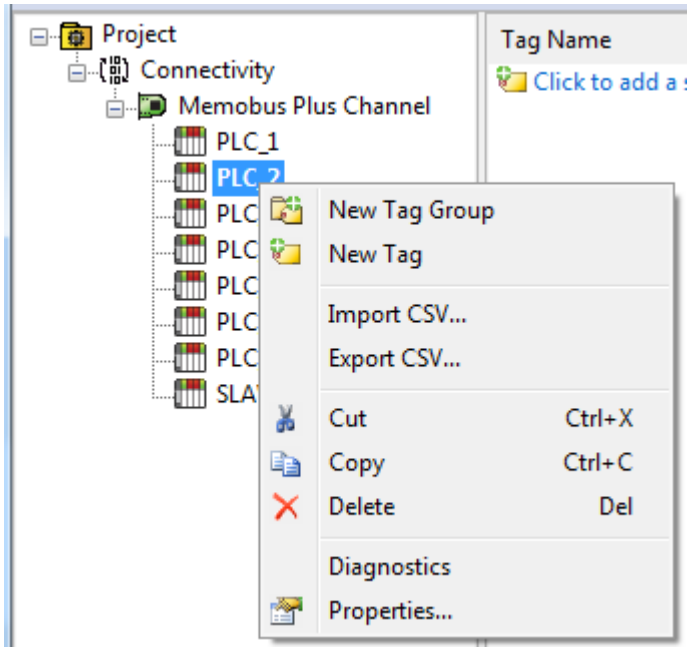
Each device has now been defined under its own channel. In this new configuration, a single path of execution is dedicated to the task of gathering data from each device. If the application has 8 or fewer devices, it can be optimized exactly how it is shown here.

The performance will improve even if the application has more than 8 devices. While 8 or fewer devices may be ideal, the application will still benefit from additional channels. Although by spreading the device load across all channels will cause the server to move from device to device again, it can now do so with far less devices to process on a single channel.

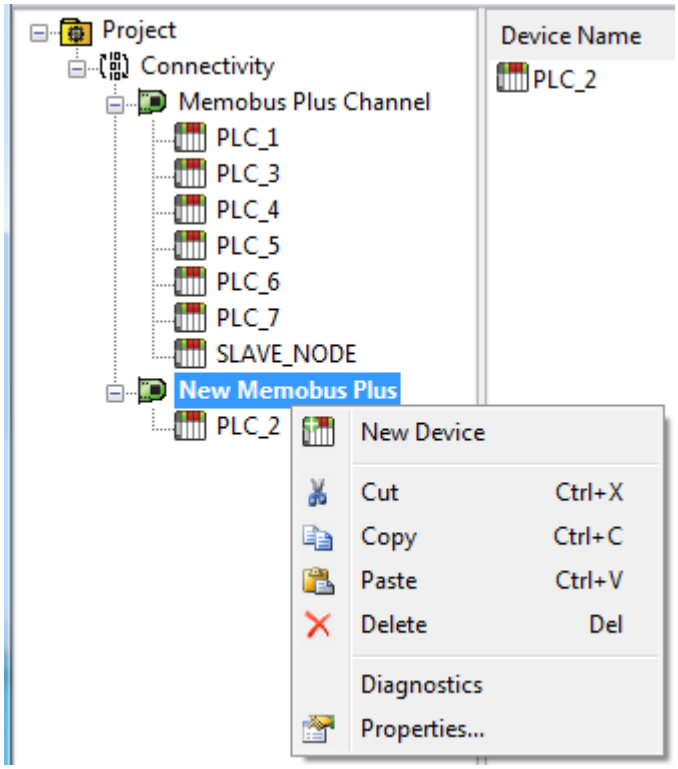
The 8-channel limit matches the multi-path limitations of the SA85 card as set by Modicon.

This application can support multiple channels even if it contains a large number of tags defined under each device. To do so, open the existing OPC Server project while it is still configured using a single channel. Next, click Channels | Add Channel in ordet to add a new Memobus Plus channel. Name the channel as desired. In this example, NewMemobusPlus is used.

Next, use the cut and paste options available by right-clicking on a device or by using the Edit menu. In this example, PLC_2 is cut from the MemobusPlus channel and then pasted under the NewMemobusPlus channel.



PLC_2 being cut from the Memobus Plus Channel.



PLC_2 being pasted beneath the New Memobus Plus channel.

Other possible Memobus Plus Driver optimizations include dedicating a single channel to just Global Data in order to receive fast Global Data updates. To do so, define a new set of device names for each device whose Global Data will be accessed under that new channel. The important thing to remember in this case is to only access Global Data from these newly defined device names. There are many possible configurations that may benefit the application; the new Memobus Plus driver gives users the opportunity to investigate them.

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.
Float*	32 bit floating point value. The driver interprets two consecutive registers as a floating point value by making the second 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 word, and bit 15 of register 40002 would be bit 31 of the 32 bit word.
String	Null terminated ASCII string. Includes Hi-Lo Lo-Hi byte order selection.

*The descriptions of the bit significance assume default, 'first word low' setting for 32 bit data types.

Yaskawa Memobus Plus Driver 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.

Output Coils

	Decimal Addressing
Type	Boolean
Format	0xxxxx
Security	Read/Write
Range	1-65536

Array Support

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

0xxxxx[cols] with assumed row count of 1
0xxxxx[rows][cols]

The number of coils requested cannot exceed the coil block size set for the device.

Examples

1. The 255th output coil would be addressed as '0255' using decimal addressing.
2. The 35005th output coil would be addressed as '035005' using decimal addressing.

Input Coils

	Decimal Addressing
Type	Boolean

	Decimal Addressing
Format	1xxxxx
Security	Read
Range	1-65536

*These locations are Read/Write for Slave Devices.

Array Support

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

1xxxxx[cols] with assumed row count of 1
1xxxxx[rows][cols]

The number of coils requested cannot exceed the coil block size set for the device.

Examples

1. The 127th input coil would be addressed as '10127' using decimal addressing.
2. The 35005th output coil would be addressed as '135005' using decimal addressing.

Internal Registers

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

	Decimal Addressing
Type	**Word**, Short, BCD
Format	3xxxxx
Security	Read Only*
Range	1-65536
Type	**Boolean**
Format	3xxxxx.bb
Security	Read Only*
Range	xxxxx.0-xxxxx.15
Type	Float, DWord, Long, LBCD
Format	3xxxxx
Security	Read Only*
Range	1-65535

*These locations are Read/Write for Slave Devices.

Array Support

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

3xxxxx[cols] with assumed row count of 1

3xxxxx[rows][cols]

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

Examples

1. To access Internal Register 1000, enter 301000.
2. To access Internal Register 11000, enter 311000.
3. To access Internal Register 300 bit 11, enter 300300.11.

Holding Registers

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

	Decimal Addressing
Type	**Word**, Short, BCD
Format	4xxxxx
Security	Read/Write
Range	1-65536
Type	**Boolean**
Format	4xxxxx.bb
Security	Read/Write
Range	xxxxx.0-xxxxx.15
Type	Float, DWord, Long, LBCD
Format	4xxxxx
Security	Read/Write
Range	1-65535
Type	**String**
String with HiLo byte order	400001.2H-465536.240H .Bit is string length, range 2 to 240 bytes.
String with LoHi byte order	400001.2L-465536.240L .Bit is string length, range 2 to 240 bytes.
Security	Read/Write

String Support

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

Examples

1. To access a string starting at 40200 with a length of 100 bytes and Hi-Lo byte order, enter: 40200.100H.
2. To access a string starting at 40500 with a length of 78 bytes and Lo-Hi byte order, enter: 40500.78L.

Array Support

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

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

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

Examples

1. To access Holding Register 5000, enter 405000.
2. To access Holding Register 14000, enter 414000.
3. To access Holding Register 500 bit 11, enter 400500.11.

Global Data

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

Note: Global Data is not supported for the slave device.

	Decimal Addressing
Type	**Word**, Short, BCD
Format	Gxx
Security	Read/Write
Range	1-32
Type	Float, DWord, Long, LBCD
Format	Gxx
Security	Read/Write
Range	1-31

Array Support

Arrays are also supported for Global Data. The syntax for declaring an array (shown using decimal addressing) is as follows:

Gxx[cols] with assumed row count of 1
Gxx[rows][cols]

For Word, Short and BCD arrays, the base address + (rows * cols) cannot exceed 32.
For Float, DWord, Long and Long BCD arrays, the base address + (rows * cols * 2) cannot exceed 32.

Examples

1. To access Global Register 11, enter G11.
2. To access Global Register 30, enter G30.

Constant Registers

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

	Decimal Addressing
Type	**Word**, Short, BCD
Format	7xxxxx
Security	Read/Write
Range	1-65536
Type	**Boolean**
Format	7xxxxx.bb
Security	Read/Write
Range	xxxxx.0-xxxxx.15
Type	Float, DWord, Long, LBCD
Format	7xxxxx
Security	Read/Write
Range	1-65535
Type	**String**
String with HiLo byte order	700001.2H-765536.240H .Bit is string length, range 2 to 240 bytes.
String with LoHi byte order	700001.2L-765536.240L .Bit is string length, range 2 to 240 bytes.
Security	Read/Write

String Support

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

Examples

1. To access a string starting at 70200 with a length of 100 bytes and Hi-Lo byte order, enter: 70200.100H.
2. To access a string starting at 70500 with a length of 78 bytes and Lo-Hi byte order, enter: 70500.78L.

Array Support

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

7xxxxx[cols] with assumed row count of 1
7xxxxx[rows][cols]

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

Examples

1. To access Constant Register 300, enter 703000.
2. To access Constant Register 1400, enter 714000.
3. To access Constant Register 500 bit 11, enter 700500.11.

Link Coils

	Decimal Addressing
Type	Boolean
Format	D1xxxx or D2xxxx
Security	Read/Write
Range	1-9999

Array Support

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

D1xxxx[cols] with assumed row count of 1
D1xxxx[rows][cols]

The number of coils requested cannot exceed the coil block size set for the device.

Examples

1. The 10255th Link coil would be addressed as 'D10255' using decimal addressing.
2. The 20005th Link coil would be addressed as 'D20005' using decimal addressing.

Link Registers

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

	Decimal Addressing
Type	**Word**, Short, BCD
Format	R1xxxx or R2xxxx
Security	Read/Write
Range	1-9999
Type	**Boolean**
Format	R1xxxx.bb or R2xxxx.bb
Security	Read/Write
Range	xxxx.0-xxxx.15
Type	Float, DWord, Long, LBCD
Format	R1xxxx or R2xxxx
Security	Read/Write
Range	1-9998
Type	**String**
String with HiLo byte order	R10001.2H-R19999.240H R20001.2H-R29999.240H .Bit is string length, range 2 to 240 bytes.
String with LoHi byte order	R10001.2L-R19999.240L R20001.2L-R29999.240L .Bit is string length, range 2 to 240 bytes.
Security	Read/Write

String Support

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

Examples

1. To access a string starting at R10200 with a length of 100 bytes and Hi-Lo byte order, enter: R10200.100H.
2. To access a string starting at R10500 with a length of 78 bytes and Lo-Hi byte order, enter: R10500.78L.

Array Support

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

R1xxxx[cols] with assumed row count of 1
R1xxxx[rows][cols]

For Word, Short and BCD arrays, the base address + (rows * cols) cannot exceed 9999.
For Float, DWord, Long and Long BCD arrays, the base address + (rows * cols * 2) cannot exceed 9998.
For all arrays, the total number of registers being requested cannot exceed the register block size that was specified for this device.

Examples

1. To access Link Register 10500, enter R10500.
2. To access Link Register 21400, enter R21400.
3. To access Link Register 10500 bit 11, enter R10500.11

MC Coils

	Decimal Addressing
Type	Boolean
Format	Y1xxxx or Y2xxxx
Security	Read/Write
Range	1-9999

Array Support

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

Y1xxxx[cols] with assumed row count of 1
Y1xxxx[rows][cols]

The number of coils requested cannot exceed the coil block size set for the device.

Examples

1. The 10255th MC coil would be addressed as 'Y10255' using decimal addressing.
2. The 20005th MC coil would be addressed as 'Y20005' using decimal addressing.

MC Control Coils

	Decimal Addressing
Type	Boolean
Format	Q1xxxx or Q2xxxx
Security	Read/Write
Range	1-9999

Array Support

Arrays are also supported for the MC Control coil addresses. The syntax for declaring an array (shown using decimal addressing) is as follows:

Q1xxxx[cols] with assumed row count of 1
Q1xxxx[rows][cols]

The number of coils requested cannot exceed the coil block size set for the device.

Examples

1. The 10255th MC Control coil would be addressed as 'Q10255' using decimal addressing.
2. The 20005th MC Control coil would be addressed as 'Q20005' using decimal addressing.

MC Relays

	Decimal Addressing
Type	Boolean
Format	X1xxxx or X2xxxx
Security	Read Only
Range	1-9999

Array Support

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

X1xxxx[cols] with assumed row count of 1
X1xxxx[rows][cols

The number of relays requested cannot exceed the coil block size set for the device.

Examples

1. The 10255th MC Relay would be addressed as 'X10255' using decimal addressing.
2. The 20005th MC Relay coil would be addressed as 'X20005' using decimal addressing.

MC Control Relays

	Decimal Addressing
Type	Boolean
Format	P1xxxx or P2xxxx
Security	Read Only
Range	1-9999

Array Support

Arrays are also supported for the MC Control Relay addresses. The syntax for declaring an array (shown using decimal addressing) is as follows:

P1xxxx[cols] with assumed row count of 1
P1xxxx[rows][cols]

The number of relays requested cannot exceed the coil block size set for the device.

Examples

1. The 10255th MC Control Relay would be addressed as 'P10255' using decimal addressing.
2. The 20005th MC Control Relay coil would be addressed as 'P20005' using decimal addressing.

MC Code Relays

	Decimal Addressing
Type	Boolean
Format	M1xxxx or M2xxxx
Security	Read Only
Range	1-9999

Array Support

Arrays are also supported for the MC Code Relay addresses. The syntax for declaring an array (shown using decimal addressing) is as follows:

M1xxxx[cols] with assumed row count of 1
M1xxxx[rows][cols]

The number of relays requested cannot exceed the coil block size set for the device.

Examples

1. The 10255th MC Code Relay would be addressed as 'M10255' using decimal addressing.
2. The 20005th MC Code Relay coil would be addressed as 'M20005' using decimal addressing.

Error Descriptions

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

Address Validation

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

Device Status Messages

Started MBPLUS.SYS device
Device '<device name>' is not responding
Unable to write to '<address>' on device '<device name>'

Device Specific Messages

Unable to start MBPLUS.SYS device
Unable to communicate with MBPLUS.VXD
Unable to open MBPLUS slave path
Error opening MBPLUS path: <ID>
Bad address in block [<start address> to <end address>] on device '<device name>'

Missing address

Error Type:

Warning

Possible Cause:

A tag address that has been specified dynamically has no length.

Solution:

Re-enter the address in the client application.

Device address '<address>' contains a syntax error

Error Type:

Warning

Possible Cause:

A tag address that has been specified dynamically contains one or more invalid characters.

Solution:

Re-enter the address in the client application.

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

Error Type:
Warning

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

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

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

Error Type:
Warning

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

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

Device address '<address>' is Read Only

Error Type:
Warning

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

Solution:
Change the access mode in the client application.

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

Error Type:
Warning

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

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

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

Error Type:

Warning

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

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

Started MBPLUS.SYS device

Error Type:
Information

Possible Cause:
Posted by the driver when it successfully starts the MBPLUS.SYS device driver. This is a Windows NT only message and will not be seen if the MBPLUS.SYS driver is already running when this driver starts.

Solution:
N/A

Device '<device name>' is not responding

Error Type:
Serious

Possible Cause:

1. The PLC network card may not be correctly installed in the host PC.
2. The named device may not be connected to the PLC network.
3. The named device may have been assigned an incorrect Network ID.
4. The driver cannot open a path on the specified adapter.
5. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device property.

Solution:

1. Verify the network card installation using the supplied utility software.
2. Check the PLC network connections.
3. Verify that the Network ID given to the named device matches that of the actual device.
4. Verify that no more than eight channels are assigned the same adapter number.
5. Increase the Request Timeout property so that the entire response can be handled.

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

Error Type:

Serious

Possible Cause:

1. The PLC network card may not be correctly installed in the host PC.
2. The named device may not be connected to the PLC network.
3. The named device may have been assigned an incorrect Network ID.

Solution:

1. Verify the network card installation using the supplied utility software.
2. Check the PLC network connections.
3. Verify that the Network ID given to the named device matches that of the actual device.

Unable to start MBPLUS.SYS device

Error Type:

Fatal

Possible Cause:

Probable cause is that the MBPLUS.SYS driver was not properly configured.

Solution:

Verify that the MBPLUS device can be started and stopped manually using the Control Panel|Devices applet. Once the MBPLUS.SYS driver can be started manually, the Yaskawa_Memobus_Plus.dll driver will also be able to start the driver.

Unable to communicate with MBPLUS.VXD

Error Type:

Fatal

Possible Cause:

Probable cause is that the MBPLUS.VXD driver was not properly configured or not installed.

Solution:

Install or properly setup the MBPLUS.VXD before running this driver. Proper configuration can be checked using the test programs that come with the VXD driver.

Unable to open MBPLUS slave path

Error Type:

Fatal

Probable Cause:

The driver was unable to open a slave path with the MBPLUS.SYS driver on Windows NT or the MBPLUS.VXD driver on Windows 95. Probable cause is that the MBPLUS driver is not properly installed.

Solution:
Install or properly setup the MBPLUS.VXD before running this driver. Proper configuration can be checked using the test programs that come with the VXD driver.

Error opening MBPLUS path: <ID>

Error Type:
Serious

Possible Cause:
1. The MBPLUS.SYS driver for Windows NT or the MBPLUS.VXD driver for Windows 95 has not been properly configured.
2. The driver cannot open a path on the specified adapter.

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.

Bad address in block [<start address> to <end address>] on device '<device name>'

Error Type:
Serious

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

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

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

R

S

T

U

W

Y