Allen-Bradley Micro800 Serial Driver

Table of Contents

Allen-Bradley Micro800 Serial Driver

Help version 1.039

CONTENTS

Overview

The Allen-Bradley Micro800 Serial Driver provides a reliable way to connect Allen-Bradley Micro800 controllers over Serial to OPC client applications; including HMI, SCADA, Historian, MES, ERP, and countless custom applications.

Setup

Supported Devices

Micro830
Micro850

Note: The connection is made over an embedded Serial port or plug-in serial module.

Communication Protocol

Rockwell Automation Fragmentation Protocol (CIP over DF1).

DH-485 and DH+ Support

An Allen Bradley KF3 or compatible device is needed to connect the driver to the DH-485 network. There are four options for communicating to a device on DH+ using the Allen-Bradley Micro800 Serial Driver.

- Allen Bradley KF2 or compatible device.
- 1784-U2DHP USB converter. This converter appears as a new serial port to the system.
- DataLink DL Interface Cards (PCI/ISA/PC104). These cards add virtual serial ports for seamless configuration.
- DataLink DL4500 Ethernet-to-DH+ Converter. Configure the device for Ethernet Encapsulation. NIC is required.

Ethernet Encapsulation

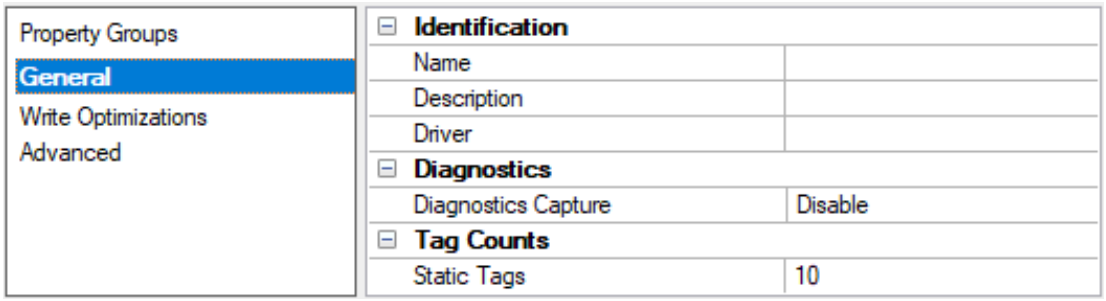
This driver supports **Ethernet Encapsulation**, which allows the driver to communicate with serial devices attached to an Ethernet network using a terminal server. Ethernet Encapsulation mode may be invoked through **Physical Medium** in Channel Properties.

Channel and Device Limits

The maximum number of channels supported by this driver is 256. The maximum number of devices supported by this driver is 1024 per channel.

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 in the server help.

Tag Counts

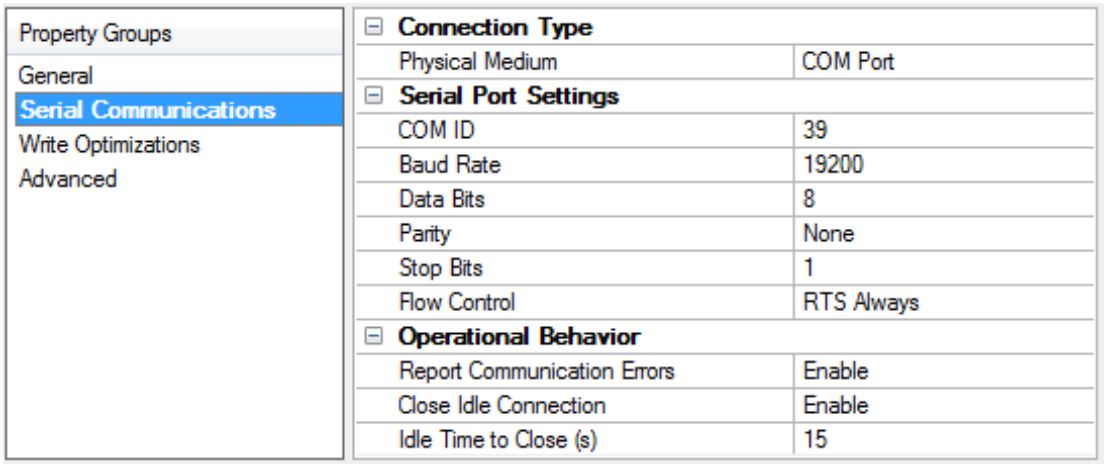
Static Tags: Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

Channel Properties — Serial Communications

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

Notes:

- With the server's online full-time operation, these properties can be changed at any time. Utilize proper user role and privilege management to prevent operators from changing properties or accessing server features.
- Users must define the specific communication parameters to be used. Depending on the driver, channels may or may not be able to share identical communication parameters. Only one shared serial connection can be configured for a Virtual Network *(see **Channel Properties — Serial Communications**)*.



Connection Type

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

1. **None**: Select None to indicate there is no physical connection, which displays the **Operation with no Communications** section.
2. **COM Port**: Select Com Port to display and configure the **Serial Port Settings** section.
3. **Modem**: Select Modem if phone lines are used for communications, which are configured in the **Modem Settings** section.
4. **Shared**: Verify the connection is correctly identified as sharing the current configuration with another channel. This is a read-only property.

Serial Port Settings

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

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

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

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

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

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

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

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

Operational Behavior

- **Report Communication Errors**: Enable or disable reporting of low-level communications errors. When enabled, low-level errors are posted to the Event Log as they occur. When disabled, these same errors are not posted even though normal request failures are. The default is Enable.
- **Close Idle Connection**: Choose to close the connection when there are no longer any tags being referenced by a client on the channel. The default is Enable.
- **Idle Time to Close**: Specify the amount of time that the server waits once all tags have been removed before closing the COM port. The default is 15 seconds.

Modem Settings

- **Modem**: Specify the installed modem to be used for communications.
- **Connect Timeout**: Specify the amount of time to wait for connections to be established before failing a read or write. The default is 60 seconds.
- **Modem Properties**: Configure the modem hardware. When clicked, it opens vendor-specific modem properties.

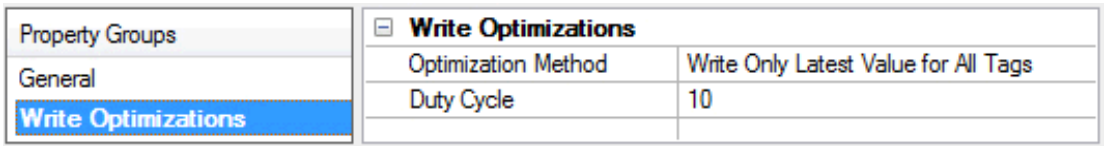
- **Auto-Dial**: Enables the automatic dialing of entries in the Phonebook. The default is Disable. *For more information, refer to "Modem Auto-Dial" in the server help.*
- **Report Communication Errors**: Enable or disable reporting of low-level communications errors. When enabled, low-level errors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

Operation with no Communications

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

Channel Properties — 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:

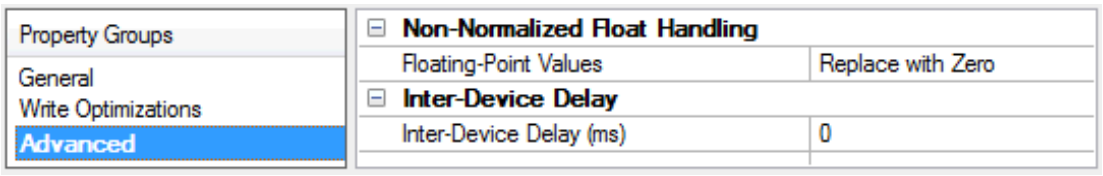
- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing 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.

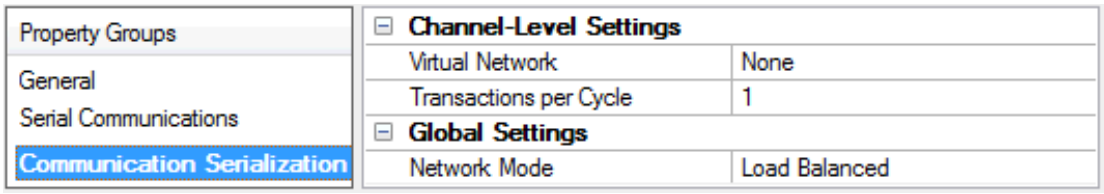
Note: This property is not available for all drivers, models, and dependent settings.

Channel Properties — Communication Serialization

The server's multi-threading architecture allows channels to communicate with devices in parallel. Although this is efficient, communication can be serialized in cases with physical network restrictions (such as

Ethernet radios). Communication serialization limits communication to one channel at a time within a virtual network.

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



Channel-Level Settings

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

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

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

Global Settings

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

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

The default is Load Balanced and affects *all* virtual networks and channels.
Devices that rely on unsolicited responses should not be placed in a virtual network. In situations where communications must be serialized, it is recommended that Auto-Demotion be enabled.

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

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

Channel Properties — Ethernet Encapsulation

Ethernet Encapsulation can be used over wireless network connections (such as 802.11b and CDPD packet networks) and has also been developed to support a wide range of serial devices. With a terminal server device, users can place RS-232 and RS-485 devices throughout the plant while still allowing a single localized PC to access the remotely mounted devices. Ethernet Encapsulation also allows an individual network IP address to be assigned to devices as needed. Multiple terminal servers provide users access to hundreds of serial devices from a single PC. One channel can be defined to use the local PC serial port while another channel can be defined to use Ethernet Encapsulation.

Note: These properties are only available to serial drivers. The properties displayed depend on the selected communications driver and supported functionality.

Network Adapter: Specify the network adapter.

Device Address: Specify the four-field IP address of the terminal server to which this device is attached. IPs are specified as *YYY.YYY.YYY.YYY*. The *YYY* designates the IP address: each *YYY* byte should be in the range of 0 to 255. Each channel has its own IP address.

Port: Configure the Ethernet port that used when connecting to a remote terminal server. The valid range is 1 to 65535, with some numbers reserved. The default is 2101.

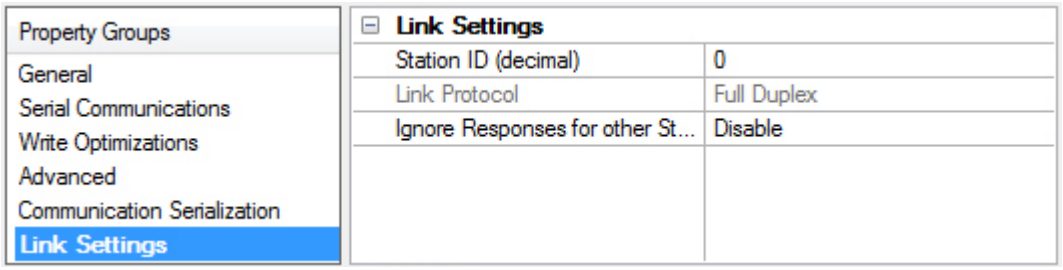
Protocol: Specify TCP/IP or UDP communication, which depends on the nature of the terminal server being used. The default is TCP/IP. *For more information on the protocol available, refer to the terminal server's help documentation.*

Important: The Ethernet Encapsulation mode is completely transparent to the actual serial communications driver. Users must configure the remaining device properties as if they were connecting to the device directly on the local PC serial port.

Connect Timeout: Specify the amount of time that is required to establish a socket connection for a remote device to be adjusted. In many cases, the connection time to a device can take longer than a normal communications request to that same device. The valid range is 1 to 999 seconds. The default is 3 seconds.

Note: With the server's online full-time operation, these properties can be changed at any time. Utilize proper user role and privilege management to prevent operators from changing properties or accessing server features.

Channel Properties — Link Settings



Link Settings

Station ID: This property specifies a unique Network ID for the local machine. It should be set based on the device with which it is communicating (excluding radio modems). The format is in decimal. The default is 0.

Link Protocol: The Allen-Bradley Micro800 Serial Driver supports Full-Duplex, which is used over a point-to-point link, allowing for high performance two-way communications between peers.

Ignore Responses for other Stations: When enabled, this property limits the acceptance of responses to those that are destined for the station as indicated by the Station ID. This property only applies to Full Duplex. The default is disabled.

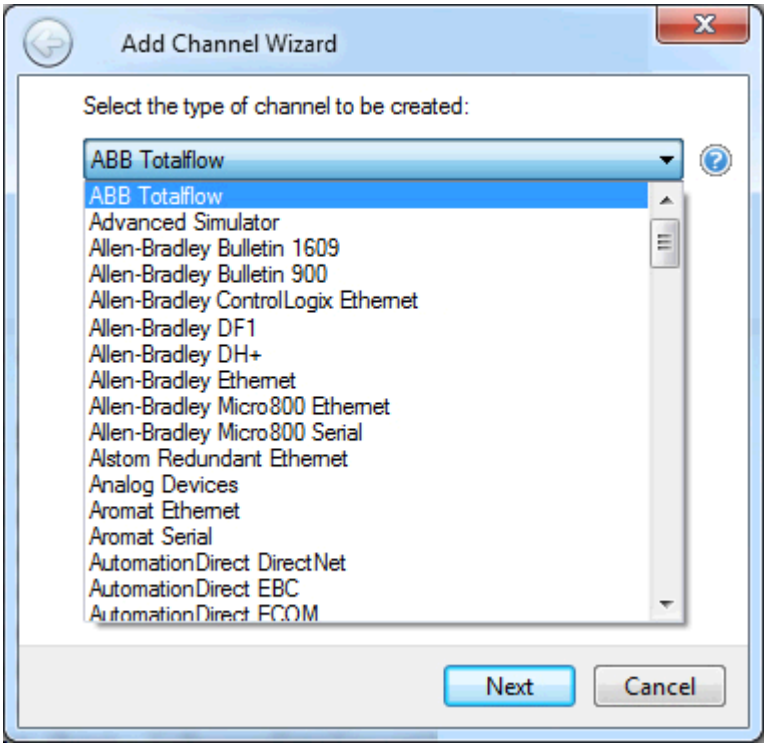
Note: If the destination device is on a DH+ or DH-485 network, communication must go through a Serial-to-DH+/DH-485 converter (that is, a KF2/KF3 module). In this case, the device being communicated with is the converter and not the destination device itself. For this configuration, the Station ID should be set to the converter's node address. The range for DH-4850 is 1 to 63. If the destination device is not on a DH+ or DH-485 network, the device being communicated with is a Micro800. The Station ID for this configuration can be set to an arbitrary unique address. The range is 0 to 255.

Channel Creation Wizard

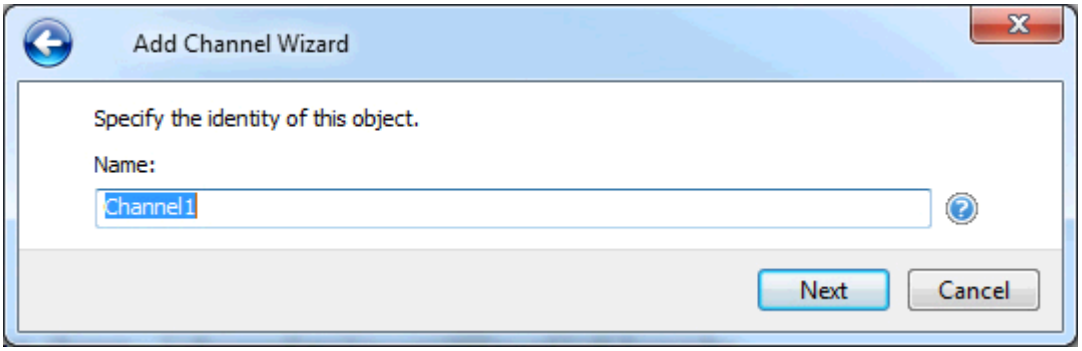
The Channel Creation Wizard steps through the process of configuring a channel (defined by the protocol being used). Once a channel is defined, its properties and settings are used by all devices assigned to that channel. The specific properties are dependent on the protocol or driver selected.

1. In the tree view, right-click on the **Connectivity** node and select **New Channel** (or choose **Edit | Connectivity | New Channel**).

2. Select type of channel to be created from the drop-down list of available drivers.



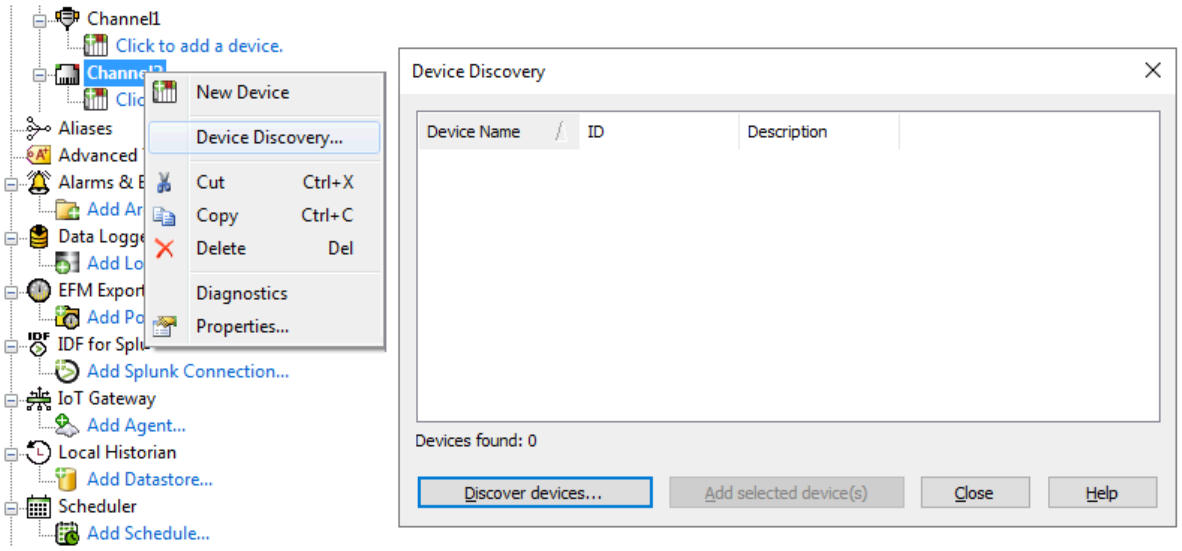
3. Click **Next**.
4. Enter a name for the channel to help identify it (used in tag paths, event log messages, and aliasing).



5. Click **Next**.
6. Configure the **channel properties** according to the options and environment.
7. Review the summary for the new channel and choose **Back** to make changes or **Finish** to close.

Device Discovery Procedure

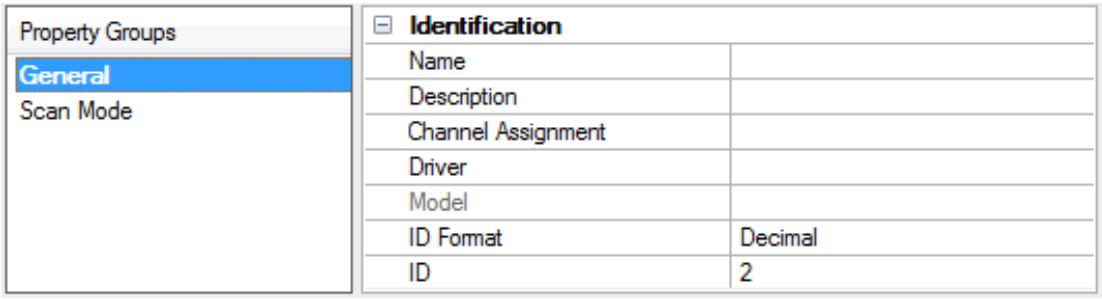
Device Discovery is available for drivers that support locating devices on the network. Once devices are found, they may be added to a channel. The maximum number of devices that can be discovered at once is 65535.



1. Select the channel in which devices should be discovered and added.
2. Right click on the channel node and select **Device Discovery...**
3. Specify the discovery properties, which are driver-specific, such as address range, timeout, discovery scope.
4. Click **OK**.
5. Devices discovered populate the dialog with the following information / headings **Device Name**, **ID**, **Description**.
6. If any discovered device is of interest, select the desired device(s) and click **Add selected device (s)...**.
7. Click **Close**.

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

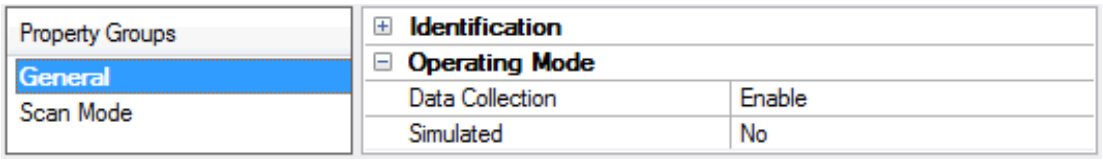
Channel Assignment: Specify the user-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

Model: Specify the type of device that is associated with this ID. The contents of the drop-down menu depend 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 documentation.*

ID: Specify the device's driver-specific station or node. The type of ID entered depends on the communications driver being used. For many communication drivers, the ID is a numeric value. Drivers that support a Numeric ID provide users with the option to enter a numeric value whose format can be changed to suit the needs of the application or the characteristics of the selected communications driver. The format is set by the driver by default. Options include Decimal, Octal, and Hexadecimal.

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.

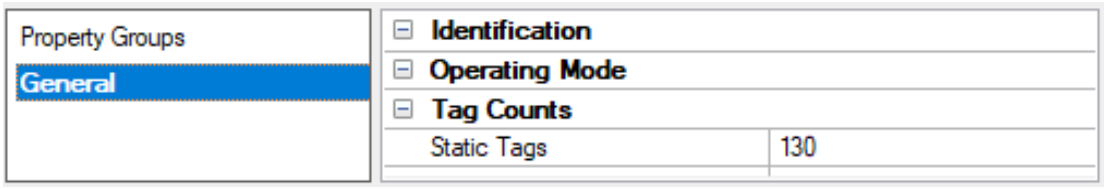
Simulated: Place the device into or out of Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client 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 data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

Notes:

1. Updates are not applied until clients disconnect and reconnect.
2. The 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.
3. 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.
4. When a device is simulated, updates may not appear faster than one (1) second in the client.

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

Tag Counts



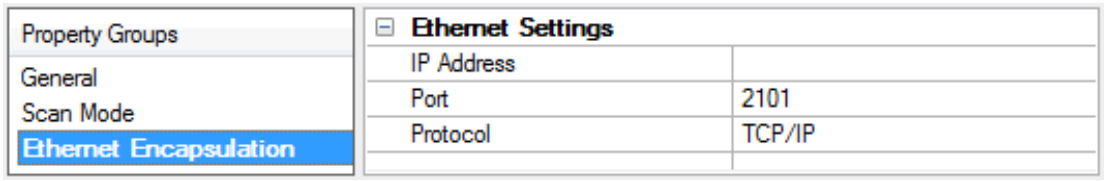
Static Tags: Provides the total number of defined static tags at this level (device or channel). This information can be helpful in troubleshooting and load balancing.

Device Properties — Ethernet Encapsulation

Ethernet Encapsulation is designed to provide communication with serial devices connected to terminal servers on the Ethernet network. A terminal server is essentially a virtual serial port. The terminal server converts TCP/IP messages on the Ethernet network to serial data. Once the message has been converted to a serial form, users can connect standard devices that support serial communications to the terminal server.

For more information, refer to "How to... Use Ethernet Encapsulation" in server help.

Ethernet Encapsulation is transparent to the driver; configure the remaining properties as if connecting to the device directly on a local serial port.



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

Port: Configure the Ethernet port to be used when connecting to a remote terminal server.

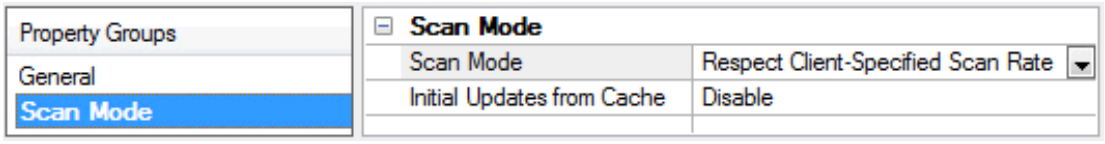
Protocol: Set TCP/IP or UDP communications. The selection depends on the nature of the terminal server being used. The default protocol selection is TCP/IP. For more information on available protocols, refer to the terminal server's help documentation.

Notes

1. With the server's online full-time operation, these properties can be changed at any time. Utilize proper user role and privilege management to prevent operators from changing properties or accessing server features.
2. The valid IP Address range is greater than (>) 0.0.0.0 to less than (<) 255.255.255.255.

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.



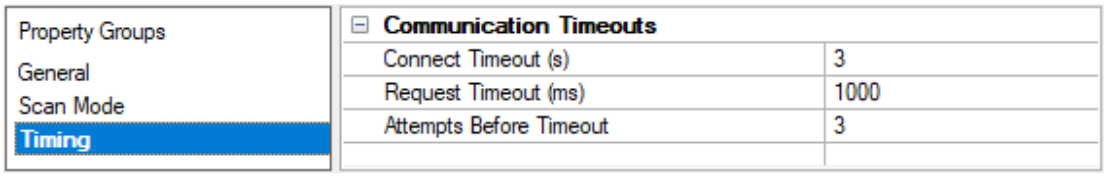
Scan Mode: Specify how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

- **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.

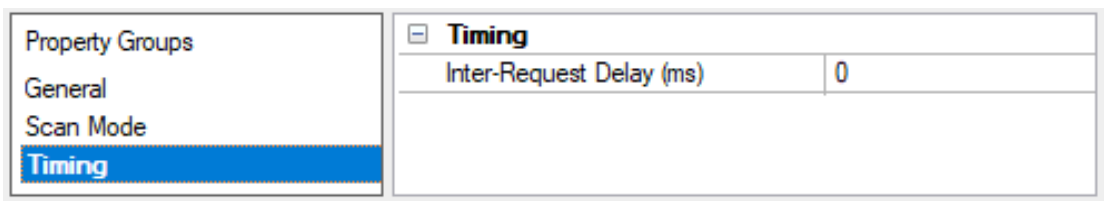
Request Timeout: Specify an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9999999 milliseconds (167 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout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

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

Timing

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

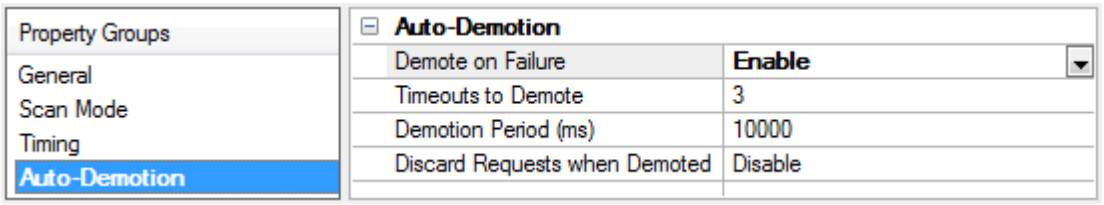
Note: Not all drivers support Inter-Request Delay. This setting does not appear if it is not available.



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.



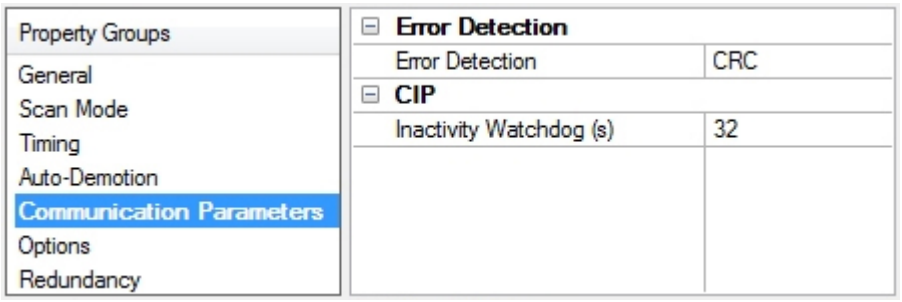
Demote on Failure: When enabled, the device is automatically taken off-scan until it is responding again.
Tip: Determine when a device is off-scan by monitoring its demoted state using the _AutoDemoted system tag.

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 — Communications Parameters



Error Detection

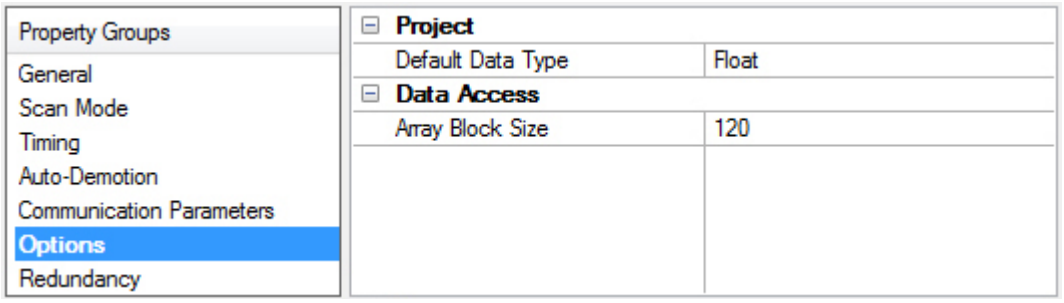
Error Detection: Select the method of error detection / checksum method expected by the device from Block Check Character (BCC) or Cyclic Redundancy Check (CRC). The default is CRC.

CIP

Inactivity Watchdog: Specify the amount of time, in seconds, a connection can remain idle (without read/write transactions) before being closed by the controller. In general, the larger the value, the more

time it takes for connection resources to be released by the controller (and vice versa). The default is 32 seconds.

Device Properties — Options



Project

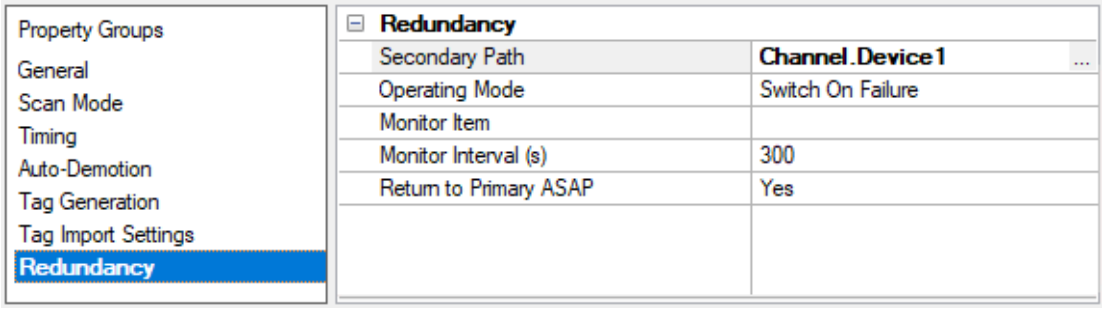
Default Data Type: Select the data type assigned to a client / server tag when the default type is selected during tag addition / modification / import. The default is Float. Tags are assigned the default data type when a dynamic tag is created in the client with Native as its assigned data type and when a static tag is created in the server with Default as its assigned data type.

Data Access

Array Block Size: Specify the maximum number of atomic array elements to read in a single transaction. The range is from 30 to 3840 elements. The default is 120 elements.

For Boolean arrays, a single element is considered a 32-element bit array. Setting the block size to 30 elements translates to 960 bit elements, whereas 3840 elements translate to 122880 bit elements.

Device Properties — Redundancy



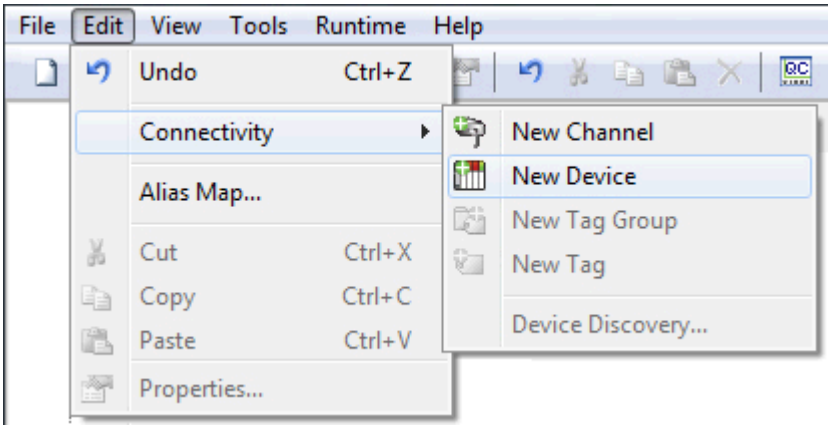
Redundancy is available with the Media-Level Redundancy Plug-In.

*Consult the website, a sales representative, or the **user manual** for more information.*

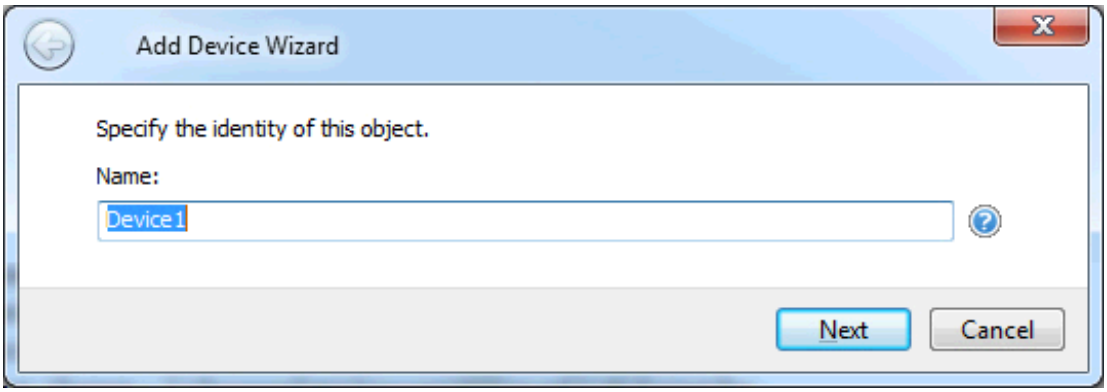
Device Creation Wizard

The Device Creation Wizard steps through the process of configuring a device for communication and data collection. The specific properties are dependent on the protocol or driver selected.

1. In the tree view, locate and select the channel to which device(s) are being added.
2. Right-click and select **New Device** or choose **Edit | Connectivity | New Device**).



3. Enter a name for the device to help identify it (used in tag paths, event log messages, and aliasing).



4. Click **Next**.
5. Configure the **device properties** according to the options and environment.
6. Review the summary for the new device and choose **Back** to make changes or **Finish** to close.

Performance Optimizations

Several guidelines may be applied to the Allen-Bradley Micro800 Serial Driver to gain maximum performance. For more information on optimization at the communication and application levels, select a link from the list below.

Optimizing Communications
Optimizing Application

Optimizing Communications

As with any programmable controller, there are a variety of ways to enhance the overall performance and system communications.

Keep Native Tag Names Short

Native Tags read from and write to the device by specifying its symbolic name in the communications request. As such, the longer the tag name is, the larger the request.

Array Elements Blocked

To optimize the reading of atomic array elements, read a block of the array in a single request instead of individually. The more elements read in a block, the greater the performance. Since transaction overhead and processing consumes the most time, do as few transactions as possible while scanning as many desired tags as possible. This is the essence of array element blocking.

Block sizes are specified as an element count. A block size of 120 elements means that a maximum of 120 array elements are read in one request. The maximum block size is 3840 elements. Boolean arrays are treated differently: in protocol, a Boolean array is a 32-bit array. Thus, requesting element 0 is requesting bits 0 through 31. To maintain consistency in discussion, a Boolean array element is considered a single bit. In summary, the maximum number of array elements (based on block size of 3840) that can be requested is as follows: 122880 BOOL, 3840 SINT, 3840 INT, 3840 DINT, 3840 LINT, and 3840 REAL.

The block size is adjustable, and should be chosen based on the project at hand. For example, if array elements 0-26 and element 3839 are tags to be read, then using a block size of 3840 is overly large and detrimental to the driver's performance. This is because all elements between 0 and 3839 are read on each request, even though only 28 of those elements are of importance. In this case, a block size of 30 is more appropriate. Elements 0-26 would be serviced in one request and element 3839 would be serviced on the next.

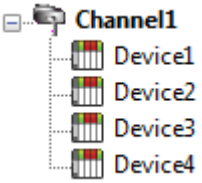
See Also: *Options*

Optimizing Applications

The Allen-Bradley Micro800 Serial Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the driver is fast, there are a couple of guidelines that can be used to gain maximum performance.

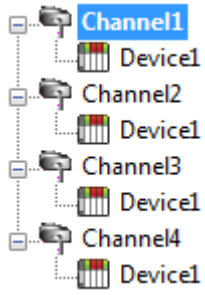
The server refers to communications protocols like Allen-Bradley Micro800 Serial as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must then be defined under that channel. Each of these devices represents a single Micro800 CPU from which data is collected. While this approach to defining the application provides a high level of performance, it doesn't take full advantage of the Allen-Bradley Micro800 Serial Driver or the

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



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

If the Allen-Bradley Micro800 Serial Driver could only define one single channel, then the example shown above would be the only option available; however, the driver can define up to 256 channels. Using multiple channels distributes the data collection workload by simultaneously issuing multiple requests to the network. An example of how the same application may appear when configured using multiple channels to improve performance is shown below.



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

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

Data Types Description

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8-bit value
Char	Signed 8-bit value
Word	Unsigned 16-bit value
Short	Signed 16-bit value
DWord	Unsigned 32-bit value
Long	Signed 32-bit value
BCD	Two byte packed BCD, four decimal digits
LBCD	Four byte packed BCD, eight decimal digits
Float	32-bit IEEE Floating point
Double	64-bit IEEE Floating point
Date	64-bit Date/Time
String	Null terminated character array

Address Descriptions

Micro800 uses a tag or symbol-based addressing structure referred to as Native Tags. These tags differ from conventional PLC data items in that the tag name itself is the address, not a file or register number.

The Allen-Bradley Micro800 Serial Driver allows users to access the controller's atomic data types: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, and SHORT_STRING. Although some of the pre-defined types are structures, they are ultimately based on these atomic data types. Thus, all non-structure (atomic) members of a structure are accessible. For example, a TIMER cannot be assigned to a server tag but an atomic member of the TIMER can be assigned to the tag (for example, TIMER.EN, TIMER.ACC, and so forth). If a structure member is a structure itself, both structures must be expanded to access an atomic member of the substructure. This is more common with user-defined and module-defined types, and is not found in any of the pre-defined types.

Atomic Data Type	Description	Client Type	Range
BOOL	Single bit value	VT_BOOL	0, 1
SINT	Signed 8-bit value	VT_U1	-128 to 127
USINT	Unsigned 8-bit value	VT_UI1	0 to 255
BYTE	Bit string (8 bits)	VT_UI1	0 to 255
INT	Signed 16-bit value	VT_I2	-32,768 to 32,767
UINT	Unsigned 16-bit value	VT_UI2	0 to 65535
WORD	Bit string (16 bits)	VT_UI2	0 to 65535
DINT	Signed 32-bit value	VT_I4	-2,147,483,648 to 2,147,483,647
UDINT	Unsigned 32-bit value	VR_UI4	0 to 4294967296
DWORD	Bit string (32 bits)	VR_UI4	0 to 4294967296
LINT	Signed 64-bit value	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
ULINT	Unsigned 64-bit value	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
LWORD	Bit string (64 bits)	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
REAL	32-bit IEEE Floating point	VT_R4	1.1755 E-38 to 3.403E38, 0, -3.403E-38 to -1.1755
LREAL	64-bit IEEE Floating point	VT_R8	-1.798E+308 to -2.225E-308, 0, 2.225E-308 to 1.798E+308
SHORT_STRING	Character string. The maximum number of characters allowed is 80.	VT_BSTR	

See Also: *Advanced Use Cases*

Client/Server Tag Address Rules

Native Tag names correspond to Client/Server Tag addresses. Both Native Tag names (entered via the Connected Components Workbench) and Client/Server Tag addresses follow the IEC 1131-3 identifier rules. Descriptions of the rules are as follows:

- Must begin with an alphabetic character or an underscore
- Can only contain alphanumeric characters and underscores
- Can have as many as 40 characters
- Cannot have consecutive underscores
- Characters are not case sensitive

For optimum performance, keep Native Tag names to a minimum in size. The smaller the name, the more requests that can fit in a single transaction.

Client/Server Tag Name Rules
Tag name assignment in the server differs from address assignment in that names cannot begin with an underscore.

See Also: *Performance Optimizations*

Address Formats

A Native Tag may be addressed statically in the server or dynamically from a client in several ways. The tag's format depends on its type and intended usage. For example, the bit format would be used when accessing a bit within a SINT-type tag. For information on address format and syntax, refer to the table below.

Note: Every format is native to Connected Components Workbench (CCW) except for the Array formats. Therefore, when referencing an atomic data type, a CCW tag name could be copied and pasted into the server's tag address field and be valid.

See Also: *Advanced Use Cases*

Format	Syntax
Array Element	<Native Tag name> [dim1, dim2, dim3]
Array w/ Offset*	<Native Tag name> {# columns} <Native Tag name> {# rows}{# columns}
Array w/o Offset*	<Native Tag name> {# columns} <Native Tag name> {# rows}{# columns}
Bit	<Native Tag name>.bit <Native Tag name>.[bit]
Standard	<Native Tag name>
String	<Native Tag name>

*Since these formats may request more than one element, the order in which array data is passed depends on the dimension of the array tag. For example, if rows times cols = 4 and the Native Tag is a 3X3 element array, then the elements that are being referenced are array_tag [0,0], array_tag [0,1], array_tag [0,2], and array_tag [1,0] in that exact order. The results would be different if the Native Tag were a 2X10 element array. For more information, refer to **Ordering of Array Data**.

Expanded Address Formats

Array Element
At least 1 dimension (but no more then 3) must be specified.

Syntax	Example	Notes
<Native Tag Name> [dim1]	tag_1 [5]	N/A
<Native Tag name> [dim1, dim2]	tag_1 [2, 3]	N/A
<Native Tag name> [dim1, dim2, dim3]	tag_1 [2, 58, 547]	N/A

Array With Offset

Since this class may request more than one element, the order in which array data is passed depends on the dimension of the Array Tag.

Syntax	Example	Notes
<Native Tag name> [offset] {# of columns}	tag_1 [5]{8}	The number of elements to Read/Write equals the number of rows multiplied by the number of columns. If no rows are specified, the number of rows default to 1. At least 1 element of the array must be addressed. The array begins at a zero offset (array index equals 0 for all dimensions).
<Native Tag name> [offset] {# of rows}{# of columns}	tag_1 [5]{2}{4}	

Note: If rows*cols = 4 and the Native Tag is a 3X3 element array, then the elements that are being referenced are array_tag [0,0], array_tag [0,1], array_tag [0,2] and array_tag [1,0] in that exact order. The results would be different if the Native Tag were a 2X10 element array.

Array Without Offset

Since this class may request more than one element, the order in which array data is passed depends on the dimension of the Array Tag.

Syntax	Example	Notes
<Native Tag name> {# of columns}	tag_1 {8}	The number of elements to Read/Write equals the number of rows multiplied by the number of columns. If no rows are specified, the number of rows default to 1. At least 1 element of the array must be addressed. The array begins at a zero offset (array index equals 0 for all dimensions).
<Native Tag name> {# of rows}{# of columns}	tag_1 {2}{4}	

Note: For example, if rows*cols = 4 and the Native Tag is a 3X3 element array, then the elements that are being referenced are array_tag [0,0], array_tag [0,1], array_tag [0,2] and array_tag [1,0] in that exact order. The results would be different if the Native Tag were a 2X10 element array.

Bit

Syntax	Example	Notes
<Native Tag name> . bit	tag_1 . 0	N/A
<Native Tag name> . [bit]	tag_1 . [0]	N/A

Standard

Syntax	Example	Notes
<Native Tag name>	tag_1	N/A

String

Syntax	Example	Notes
<Native Tag name>	tag_1	The number of characters to Read/Write equals the string length and must be at least 1.

For more information on how elements are referenced for 1, 2 and 3 dimensional arrays, refer to ***Ordering of Array Data****.*

Tag Scope

The scope of variables can be local to a program or global to a controller.

- Local variables are assigned to a specific program in the project; they are available only to that program.
- Global variables belong to the controller in the project; they are available to any program in the project.

Local Variables

Local variables (program-scoped tags) cannot be accessed directly through the communications port of the controller, so are not directly supported within the driver. If access is required, cut and paste the tags from the Local variable table to the Global variable table.

Global Variables

Global Variables (controller-scoped tags) are Native Tags that have global scope in the controller. Any program or task can access Global Tags; however, the number of ways a Global Tag can be referenced depends on both its Native Data Type and the address format being used.

User-Defined Data Types

Users may create unique data types, e.g. STRING with 12 characters rather than 80. These user-defined data types may be used as local or global variables.

Structured Variables

There are no structured variables in Micro800 controllers. Users may build unique Data Types, but each member must have a unique name.

Addressing Atomic Data Types

The table below contains suggested usage and addressing possibilities for each Native Data Type given the available address formats. For each data type's advanced addressing possibilities, click **Advanced**.

Note: Empty cells do not necessarily indicate a lack of support.

BOOL

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type	Boolean	Boolean	Boolean Array		

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Advanced		(BOOL 1 dimensional array)	(BOOL 1 dimensional array)		
Example	BOOLTAG	BOOLARR[0]	BOOLARR[0]{32}		

SINT, USINT, and BYTE

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Byte, Char	Byte, Char	Byte Array, Char Array (SINT 1/2/3 dimensional array)	Boolean (Bit w/i SINT)	
Example	SINTTAG	SINTARR[0]	SINTARR[0]{4}	SINTTAG.0	j

INT, UINT, and WORD

Tag	Standard	Array Element	Array w/o Offset	Bit	String
Data Type Advanced	Word, Short	Word, Short	Word Array, Short Array (INT 1/2/3 dimensional array)	Boolean (Bit w/i INT)	
Example	INTTAG	INTARR[0]	INTARR[0]{4}	INTTAG.0	

DINT, UDINT, and DWORD

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	DWord, Long	DWord, Long	DWord Array, Long Array	Boolean (Bit w/i DINT)	Advanced
Example	DINTTAG	DINTARR[0]	DINTARR[0]{4}	DINTTAG.0	

LINT, ULINT, and LWORD

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Double, Date	Double, Date	Double Array		
Example	LINTTAG	LINTARR[0]	LINTARR[0]{4}		

REAL

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type	Float	Float	Float Array		

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Advanced					
Example	REALTAG	REALARR[0]	REALARR[0]{4}		

LREAL

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	Double	Double	Double Array		
Example	LREALTAG	LREALARR[0]	LREALARR[0]{4}		

SHORT_STRING

Tag	Standard	Array Element	Array w/wo Offset	Bit	String
Data Type Advanced	String	String			
Example	STRINGTAG	STRINGARR[0]			

See Also: Address Formats

Addressing Structured Data Types

Structures cannot be referenced at the structure level: only the atomic structure members can be addressed. For more information, refer to the examples below.

Native Tag
MyTimer @ TIMER

Valid Client/Server Tag
Address = MyTimer.ACC
Data type = DWord

Invalid Client/Server Tag
Address = MyTimer
Data type = ??

Addressing STRING Data Type

STRING is a pre-defined Native Data Type whose structure contains two members: DATA and LEN. DATA is an array of SINTs and stores the characters of the string. LEN is a DINT and represents the number of characters in DATA to display to a client.

Because LEN and Data are atomic members, they must be referenced independently from the client/server.

Description	Syntax	Example
STRING Value	DATA/<Maximum STRING Length>	MYSTRING.DATA/82

Description	Syntax	Example
Actual STRING Length	LEN	MYSTRING.LEN

Reads

The string read from DATA is terminated by the following:

a. The first null terminator encountered.

b. The value in LEN if a) doesn't occur first.

c. The <Maximum STRING Length> if either a) or b) doesn't occur first.

Example
MYSTRING.DATA contains "Hello World" in the PLC, but LEN is manually set to 5. A read of MYSTRING.DATA/82 displays "Hello". If LEN is set to 20, MYSTRING.DATA/82 displays "Hello World".

Writes

When a STRING value is written to DATA, the driver also writes to LEN with the length of DATA written. If the write to LEN fails for any reason, the write operation to DATA is considered failed as well (despite the fact that the DATA write to the controller succeeded).

Note: This behavior was designed specifically for Native Tags of type STRING (or a custom derivative of STRING). The following precautions apply to users who wish to implement their own string in UDTs.

- If a UDT exists that has a DATA member referenced as a string and a LEN member referenced as a DINT, the write to LEN succeeds regardless of the intentions of LEN for the given UDT. Care must be taken when designing UDTs to avoid this possibility if LEN is not intended to be the length of DATA.
- If a UDT exists that has a DATA member referenced as a string but does not have a LEN member, the write to LEN fails silently without consequence to DATA.

Example
MYSTRING.DATA/82 holds the value "Hello World." MYSTRING.LEN holds 11. If the value "Alarm Triggered" is written to MYSTRING.DATA/82, 15 is written to MYSTRING.LEN. If the write to MYSTRING.LEN fails, MYSTRING.LEN holds its previous value of 11 while MYSTRING.DATA/82 displays the first 11 characters ("Alarm Trigg"). If the write to MYSTRING.DATA/82 fails, neither tag is affected.

Ordering of Array Data

One-Dimensional Arrays - array [dim1]

1-dimensional array data is passed to and from the controller in ascending order.

for (dim1 = 0; dim1 < dim1_max; dim1++)

Example: 3 element array
array [0]
array [1]
array [2]

Two-Dimensional Arrays - array [dim1, dim2]

2-dimensional array data is passed to and from the controller in ascending order.

for (dim1 = 0; dim1 < dim1_max; dim1++)
for (dim2 = 0; dim2 < dim2_max; dim2++)

Example: 3X3 element array
array [0, 0]
array [0, 1]
array [0, 2]
array [1, 0]
array [1, 1]
array [1, 2]
array [2, 0]
array [2, 1]
array [2, 2]

Three-Dimensional Arrays - array [dim1, dim2, dim3]

3-dimensional array data is passed to and from the controller in ascending order.

for (dim1 = 0; dim1 < dim1_max; dim1++)
for (dim2 = 0; dim2 < dim2_max; dim2++)
for (dim3 = 0; dim3 < dim3_max; dim3++)

Example: 3X3x3 element array
array [0, 0, 0]
array [0, 0, 1]
array [0, 0, 2]
array [0, 1, 0]
array [0, 1, 1]
array [0, 1, 2]
array [0, 2, 0]
array [0, 2, 1]
array [0, 2, 2]
array [1, 0, 0]
array [1, 0, 1]
array [1, 0, 2]
array [1, 1, 0]
array [1, 1, 1]
array [1, 1, 2]
array [1, 2, 0]
array [1, 2, 1]
array [1, 2, 2]
array [2, 0, 0]
array [2, 0, 1]
array [2, 0, 2]
array [2, 1, 0]
array [2, 1, 1]
array [2, 1, 2]
array [2, 2, 0]
array [2, 2, 1]
array [2, 2, 2]

Advanced Use Cases

For more information on the advanced use cases for a specific atomic data type, select a link from the list below.

BOOL

BOOL

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	Boolean	The Native Tag must be a 1 dimensional array.
Array w/ Offset	Boolean Array	1. The Native Tag must be a 1 dimensional array. 2. The offset must lay on a 32-bit boundary. 3. The number of elements must be a factor of 32.
Array w/o Offset	Boolean Array	1. The Native Tag must be a 1 dimensional array. 2. The number of elements must be a factor of 32.
Bit	Boolean	1. The Native Tag must be a 1 dimensional array. 2. The range is limited from 0 to 31.
Standard	Boolean, Byte, Char, Word, Short, BCD, DWord, Long, LBCD, Float*	None
String	Not supported.	

***The Float value equals the face value of the Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted signify common use cases.

BOOL Atomic Tag - booltag = True

Server Tag Address	Format	Data Type	Notes
booltag	Standard	Boolean	Value = True
booltag	Standard	Byte	Value = 1
booltag	Standard	Word	Value = 1
booltag	Standard	DWord	Value = 1

Server Tag Address	Format	Data Type	Notes
booltag	Standard	Float	Value = 1.0
booltag [3]	Array Element	Boolean	Invalid: Tag is not an array.
booltag [3]	Array Element	Word	Invalid: Tag is not an array.
booltag {1}	Array w/o Offset	Word	Invalid: Not supported.
booltag {1}	Array w/o Offset	Boolean	Invalid: Not supported.
booltag [3] {32}	Array w/ Offset	Boolean	Invalid: Tag is not an array.
booltag . 3	Bit	Boolean	Invalid: Tag is not an array.
booltag / 1	String	String	Invalid: Not supported.
booltag / 4	String	String	Invalid: Not supported.

BOOL Array Tag - bitarraytag = [0,1,0,1]

Server Tag Address	Format	Data Type	Notes
bitarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
bitarraytag	Standard	Byte	Invalid: Tag cannot be an array.
bitarraytag	Standard	Word	Invalid: Tag cannot be an array.
bitarraytag	Standard	DWord	Invalid: Tag cannot be an array.
bitarraytag	Standard	Float	Invalid: Tag cannot be an array.
bitarraytag [3]	Array Element	Boolean	Value = True
bitarraytag [3]	Array Element	Word	Invalid: Bad data type.
bitarraytag {3}	Array w/o Offset	Word	Invalid: Tag cannot be an array.
bitarraytag {1}	Array w/o Offset	Word	Invalid: Tag cannot be an array.
bitarraytag {1}	Array w/o Offset	Boolean	Invalid: Array size must be a factor of 32.
bitarraytag {32}	Array w/o Offset	Boolean	Value = [0,1,0,1,...]
bitarraytag [3] {32}	Array w/ Offset	Boolean	Offset must begin on 32-bit boundary.
bitarraytag[0]{32}	Array w/ Offset	Boolean	Value = [0,1,0,1,...]
bitarraytag[32]{64}	Array w/ Offset	Boolean	Syntax valid. Element is out of range.
bitarraytag . 3	Bit	Boolean	Value = True
bitarraytag / 1	String	String	Invalid: Not supported.
bitarraytag / 4	String	String	Invalid: Not supported.

SINT, USINT, and BYTE

For more information on the format, refer to ***Address Formats***.

Format	Supported Data Types	Notes
Array Element	Byte, Char, Word, Short, BCD, DWord, Long, LBCD, Float***	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array, Word Array, Short Array, BCD Array**, DWord Array, Long Array, LBCD Array**, Float Array**,***	The Native Tag must be an array.

Format	Supported Data Types	Notes
Array w/o Off-set	Boolean Array	1. Use this case to have the bits within a SINT in array form. This is not an array of SINTs in Boolean notation. 2. Applies to bit-within-SINT only. Example: tag_1.0{8}. 3. The .bit plus the array size cannot exceed 8 bits. Example: tag_1.1{8} exceeds a SINT, tag_1.0{8} does not. If accessing more than a single element, the Native Tag must be an array.
	Byte Array, Char Array, Word Array, Short Array, BCD Array**, DWord Array, Long Array, LBCD Array**, Float Array**,***	
Bit	Boolean	1. The range is limited from 0 to 7. 2. If the Native Tag is an array, the bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0.
Standard	Boolean*, Byte, Char, Word, Short, BCD, DWord, Long, LBCD, Float***	None
String	String	1. If accessing a single element, the Native Tag does not need to be an array. **Note:** The value of the string is the ASCII equivalent of the SINT value. Example: SINT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string is the null-terminated ASCII equivalent of all the SINTs in the string. 1 character in string = 1 SINT.

*Non-zero values are clamped to True.
**Each element of the array corresponds to an element in the SINT array. Arrays are not packed.
***Float value equals the face value of Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted signify common use cases for SINT, USINT, and BYTE.

SINT, USINT, and BYTE Atomic Tag - sinttag = 122 (decimal)

Server Tag Address	Format	Data Type	Notes
sinttag	Standard	Boolean	Value = True
sinttag	Standard	Byte	Value = 122
sinttag	Standard	Word	Value = 122
sinttag	Standard	DWord	Value = 122
sinttag	Standard	Float	Value = 122.0
sinttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
sinttag [3]	Array Element	Byte	Invalid: Tag is not an array.
sinttag {3}	Array w/o Off-set	Byte	Invalid: Tag is not an array.
sinttag {1}	Array w/o Off-set	Byte	Value = [122]
sinttag {1}	Array w/o Off-set	Boolean	Invalid: Bad data type.
sinttag [3] {1}	Array w/ Offset	Byte	Invalid: Tag is not an array.
sinttag . 3	Bit	Boolean	Value = True
sinttag . 0 {8}	Array w/o Off-set	Boolean	Value = [0,1,0,1,1,1,1,0] Bit value of 122
sinttag / 1	String	String	Invalid: Syntax / data type not supported.
sinttag / 4	String	String	Invalid: Syntax / data type not supported.

SINT, USINT, and BYTE Array Tag - sintarraytag [4,4] = [[83,73,78,84],[5,6,7,8],[9,10,11,12],[13,14,15,16]]

Server Tag Address	Format	Data Type	Notes
sintarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
sintarraytag	Standard	Byte	Invalid: Tag cannot be an array.
sintarraytag	Standard	Word	Invalid: Tag cannot be an array.
sintarraytag	Standard	DWord	Invalid: Tag cannot be an array.
sintarraytag	Standard	Float	Invalid: Tag cannot be an array.
sintarraytag [3]	Array Element	Byte	Invalid: Server tag missing dimension 2 address.
sintarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
sintarraytag [1,3]	Array Element	Byte	Value = 8
sintarraytag {10}	Array w/o Off-set	Byte	Value = [83,73,78,84,5,6,7,8,9,10]
sintarraytag {2} {5}	Array w/o Off-set	Word	Value = [83,73,78,84,5] [6,7,8,9,10]
sintarraytag {1}	Array w/o Off-set	Byte	Value = 83
sintarraytag {1}	Array w/o Off-set	Boolean	Invalid: Bad data type.

Server Tag Address	Format	Data Type	Notes
sintarraytag [1,3] {4}	Array w/ Offset	Byte	Value = [8,9,10,11]
sintarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
sintarraytag [1,3] . 3	Bit	Boolean	Value = 1
sintarraytag [1,3] . 0 {8}	Array w/o Offset	Boolean	Value = [0,0,0,1,0,0,0,0]
sintarraytag / 1	String	String	Invalid: Syntax / data type not supported.
sintarraytag / 4	String	String	Invalid: Syntax / data type not supported.

INT, UINT, and WORD

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	Byte, Char**, Word, Short, BCD, DWord, Long, LBCD, Float****	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array**, Word Array, Short Array, BCD Array, DWord Array, Long Array, LBCD Array***, Float Array ***,****	The Native Tag must be an array.
Array w/o Offset	Boolean Array Byte Array, Char Array**, Word Array, Short Array, BCD Array, DWord Array, Long Array, LBCD Array ***, Float Array ***,****	1. Use this case to have the bits within an INT in array form. This is not an array of INTs in Boolean notation. 2. Applies to bit-within-INT only. Example: tag_1.0{16}. 3. The .bit plus the array size cannot exceed 16 bits. Example: tag_1.1{16} exceeds an INT, tag_1.0{16} does not. If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	1. The range is limited from 0 to 15. 2. If the Native Tag is an array, the bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0.
Standard	Boolean*, Byte, Char**, Word, Short, BCD, DWord, Long, LBCD, Float****	None
String	String	1. If accessing a single element, the controller tag does not need to be an array. **Note:** The value of the string is the ASCII equivalent of the INT value (clamped to 255). Example: INT = 65dec = "A".

Format	Supported Data Types	Notes
		2. If accessing more than a single element, the Native Tag must be an array. The value of the string is the null-terminated ASCII equivalent of all the INTs (clamped to 255) in the string. 1 character in string = 1 INT, clamped to 255. INT strings are not packed. For greater efficiency, use SINT strings or the STRING structure instead.

*Non-zero values are clamped to True.
**Values exceeding 255 are clamped to 255.
*** Each element of the array corresponds to an element in the INT array. Arrays are not packed.
****Float value equals the face value of Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted signify common use cases for INT, UINT, and WORD.

INT, UINT, and WORD Atomic Tag - inttag = 65534 (decimal)

Server Tag Address	Class	Data Type	Notes
inttag	Standard	Boolean	Value = True
inttag	Standard	Byte	Value = 255
inttag	Standard	Word	Value = 65534
inttag	Standard	DWord	Value = 65534
inttag	Standard	Float	Value = 65534.0
inttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
inttag [3]	Array Element	Word	Invalid: Tag is not an array.
inttag {3}	Array w/o Offset	Word	Invalid: Tag is not an array.
inttag {1}	Array w/o Offset	Word	Value = [65534]
inttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
inttag [3] {1}	Array w/ Offset	Word	Invalid: Tag is not an array.
inttag . 3	Bit	Boolean	Value = True
inttag . 0 {16}	Array w/o Offset	Boolean	Value = [0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] Bit value of 65534
inttag / 1	String	String	Invalid: Syntax / data type not supported.
inttag / 4	String	String	Invalid: Syntax / data type not supported.

INT, UINT, and WORD Array Tag - intarraytag [4,4] = [[73,78,84,255],[256,257,258,259],[9,10,11,12],[13,14,15,16]]

Server Tag Address	Class	Data Type	Notes
intarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
intarraytag	Standard	Byte	Invalid: Tag cannot be an array.
intarraytag	Standard	Word	Invalid: Tag cannot be an array.
intarraytag	Standard	DWord	Invalid: Tag cannot be an array.
intarraytag	Standard	Float	Invalid: Tag cannot be an array.
intarraytag [3]	Array Element	Word	Invalid: Server tag is missing dimension 2 address.
intarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
intarraytag [1,3]	Array Element	Word	Value = 259
intarraytag {10}	Array w/o Offset	Byte	Value = [73,78,84,255,255,255,255,255,9,10]
intarraytag {2} {5}	Array w/o Offset	Word	Value = [73,78,84,255,256] [257,258,259,9,10]
intarraytag {1}	Array w/o Offset	Word	Value = 73
intarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
intarraytag [1,3] {4}	Array w/ Offset	Word	Value = [259,9,10,11]
intarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
intarraytag [1,3] . 3	Bit	Boolean	Value = 0
intarraytag [1,3] . 0 {16}	Array w/o Offset	Boolean	Value = [1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0] Bit value for 259
intarraytag / 1	String	String	Invalid: Syntax / data type not supported.
intarraytag / 3	String	String	Invalid: Syntax / data type not supported.

DINT, UDINT, and DWORD

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	Byte, Char**, Word, Short, BCD***, DWord, Long, LBCD, Float****	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array**, Word Array, Short Array, BCD Array***, DWord Array, Long Array, LBCD Array, Float Array****	The Native Tag must be an array.
Array w/o Offset	Boolean Array	1. Use this case to have the bits within an DINT in array form. This is not an array of DINTs in Boolean notation. 2. Applies to bit-within-DINT only. Example: tag_1.0{32}. 3. The .bit plus the array size cannot exceed 32 bits. Example: tag_1.1{32} exceeds an DINT,

Format	Supported Data Types	Notes
	Byte Array, Char Array**, Word Array, Short Array, BCD Array***, DWord Array, Long Array, LBCD Array, Float Array****	tag_1.0{32} does not. If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	1. The range is limited from 0 to 31. 2. If Native Tag is an array, bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0.
Standard	Boolean* Byte, Char** Word, Short, BCD*** DWord, Long, LBCD Float****	None
String	String	1. If accessing a single element, the controller tag does not need to be an array. **Note:** The value of the string is the ASCII equivalent of the DINT value (clamped to 255). Example: SINT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string is the null-terminated ASCII equivalent of all the DINTs (clamped to 255) in the string. 1 character in string = 1 DINT, clamped to 255. DINT strings are not packed. For greater efficiency, use SINT strings or the STRING structure instead.

*Non-zero values are clamped to True.
**Values exceeding 255 are clamped to 255.
***Values exceeding 65535 are clamped to 65535.
****Float value equals face value of Native Tag in Float form (non-IEEE Floating point number).

Examples

Examples highlighted

DINT, UDINT, and DWORD Atomic Tag - dinttag = 70000 (decimal)

Server Tag Address	Format	Data Type	Notes
dinttag	Standard	Boolean	Value = True
dinttag	Standard	Byte	Value = 255
dinttag	Standard	Word	Value = 65535
dinttag	Standard	DWord	Value = 70000
dinttag	Standard	Float	Value = 70000.0
dinttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
dinttag [3]	Array Element	DWord	Invalid: Tag is not an array.
dinttag {3}	Array w/o Offset	DWord	Invalid: Tag is not an array.
dinttag {1}	Array w/o Offset	DWord	Value = [70000]
dinttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type
dintag [3] {1}	Array w/ Offset	DWord	Invalid: Tag is not an array.
dinttag . 3	Bit	Boolean	Value = False
dinttag . 0 {32}	Array w/o Offset	Boolean	Value = [0,0,0,0,1,1,1,0,1,0,0,0,1,0,0,0,1,0,...0] Bit value for 70000
dinttag	String	String	Invalid: Syntax / data type not supported.
dinttag	String	String	Invalid: Syntax / data type not supported.

DINT, UDINT, and DWORD Array Tag - dintarraytag [4,4] = [[68,73,78,84],[256,257,258,259], [9,10,11,12],[13,14,15,16]]

Server Tag Address	Format	Data Type	Notes
dintarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
dintarraytag	Standard	Byte	Invalid: Tag cannot be an array.
dintarraytag	Standard	Word	Invalid: Tag cannot be an array.
dintarraytag	Standard	DWord	Invalid: Tag cannot be an array.
dintarraytag	Standard	Float	Invalid: Tag cannot be an array.
dintarraytag [3]	Array Element	DWord	Invalid: Server tag missing dimension 2 address.
dintarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
dintarraytag [1,3]	Array Element	DWord	Value = 259
dintarraytag {10}	Array w/o Offset	Byte	Value = [68,73,78,84,255,255,255,255,9,10]
dintarraytag {2}{5}	Array w/o Offset	DWord	Value = [68,73,78,84,256] [257,258,259,9,10]
dintarraytag {1}	Array w/o Offset	DWord	Value = 68
dintarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.

Server Tag Address	Format	Data Type	Notes
dintarraytag [1,3]{4}	Array w/ Offset	DWord	Value = [259,9,10,11]
dintarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
dintarraytag [1,3] . 3	Bit	Boolean	Value = 0
dintarraytag [1,3] .0 {32}	Array w/o Offset	Boolean	Value = [1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0] Bit value for 259
dintarraytag	String	String	Invalid: Syntax / data type not supported.
dintarraytag	String	String	Invalid: Syntax / data type not supported.

LINT, ULINT, and LWORD

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	Double* Date**	The Native Tag must be an array.
Array w/ Offset	Double Array*	The Native Tag must be an array.
Array w/o Offset	Double Array*	If accessing more than a single element, the controller tag must be an array.
Bit	Not supported.	Not supported.
Standard	Double* Date**	None
String	Not supported.	Not supported.

*Double value equals the face value of controller tag in Float form (non-IEEE Floating point number).
**Date values are in universal time (UTC), not localized time.

Examples

Examples highlighted signify common use cases for LINT, ULINT, and LWORD.

LINT, ULINT, and LWORD Atomic Tag - linttag = 2007-01-01T16:46:40.000 (date) == 1.16767E+15 (decimal)

Server Tag Address	Format	Data Type	Notes
linttag	Standard	Boolean	Invalid: Boolean is not supported.
linttag	Standard	Byte	Invalid: Byte is not supported.
linttag	Standard	Word	Invalid: Word is not supported.
linttag	Standard	Double	Value = 1.16767E+15
linttag	Standard	Date	Value = 2007-01-01T16:46:40.000*
linttag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
linttag [3]	Array Element	Double	Invalid: Tag is not an array.

Server Tag Address	Format	Data Type	Notes
linttag {3}	Array w/o Offset	Double	Invalid: Tag is not an array.
linttag {1}	Array w/o Offset	Double	Value = [1.16767E+15]
linttag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lintag [3] {1}	Array w/ Offset	Double	Invalid: Tag is not an array.
linttag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
linttag / 1	String	String	Invalid: Syntax/data type not supported.

*Date values are in universal time (UTC), not localized time.

LINT, ULINT, and LWORD Array Tag -
dintarraytag [2,2] = [0, 1.16767E+15],[9.4666E+14, 9.46746E+14] where:
1.16767E+15 == 2007-01-01T16:46:40.000 (date)
9.4666E+14 == 1999-12-31T17:06:40.000
9.46746E+14 == 2000-01-1T17:00:00.000
0 == 1970-01-01T00:00:00.000

Server Tag Address	Format	Data Type	Notes
lintarraytag	Standard	Boolean	Invalid: Boolean not supported.
lintarraytag	Standard	Byte	Invalid: Byte not supported.
lintarraytag	Standard	Word	Invalid: Word not supported.
lintarraytag	Standard	Double	Invalid: Tag cannot be an array.
lintarraytag	Standard	Date	Invalid: Tag cannot be an array.
lintarraytag [1]	Array Element	Double	Invalid: Server tag missing dimension 2 address.
lintarraytag [1,1]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
lintarraytag [1,1]	Array Element	Double	Value = 9.46746E+14
lintarraytag [1,1]	Array Element	Date	Value = 2000-01-01T17:00:00.000*
lintarraytag {4}	Array w/o Offset	Double	Value = [0, 1.16767E+15, 9.4666E+14, 9.46746E+14]
lintarraytag {2} {2}	Array w/o Offset	Double	Value = [0, 1.16767E+15][9.4666E+14, 9.46746E+14]
lintarraytag {4}	Array w/o Offset	Date	Invalid: Date array not supported.
lintarraytag {1}	Array w/o Offset	Double	Value = 0
lintarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lintarraytag [0,1] {2}	Array w/ Offset	Double	Value = [1.16767E+15, 9.4666E+14]
lintarraytag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.

Server Tag Address	Format	Data Type	Notes
lintarraytag / 1	String	String	Invalid: Syntax/data type not supported.

*Date values are in universal time (UTC), not localized time.

REAL

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	Byte, Char**, Word, Short, BCD***, DWord, Long, LBCD, Float****	The Native Tag must be an array.
Array w/ Offset	Byte Array, Char Array**, Word Array, Short Array, BCD Array***, DWord Array, Long Array, LBCD Array, Float Array****	The Native Tag must be an array.
Array w/o Offset	Boolean Array Byte Array, Char Array**, Word Array, Short Array, BCD Array***, DWord Array, Long Array, LBCD Array, Float Array****	1. Use this case to have the bits within a REAL in array form. This is not an array of REALs in Boolean notation. 2. Applies to bit-within-REAL only. Example: tag_1.0{32}. 3. The .bit plus the array size cannot exceed 32 bits. Example: tag_1.1{32} exceeds a REAL, tag_1.0{32} does not. If accessing more than a single element, the Native Tag must be an array.
Bit	Boolean	1. The range is limited from 0 to 31. 2. If the Native Tag is an array, the bit class reference must be prefixed by an array element class reference. Example: tag_1 [2,2,3].0. **Note:** Float is cast to a DWord to allow referencing of bits.
Standard	Boolean* Byte, Char** Word, Short, BCD*** DWord, Long, LBCD Float****	None
String	String	1. If accessing a single element, the controller tag does not need to be an array. **Note:** The value of the string is the ASCII

Format	Supported Data Types	Notes
		equivalent of the REAL value (clamped to 255). Example: SINT = 65dec = "A". 2. If accessing more than a single element, the Native Tag must be an array. The value of the string is the null-terminated ASCII equivalent of all the REALs (clamped to 255) in the string. 1 character in string = 1 REAL, clamped to 255. REAL strings are not packed. For greater efficiency, use SINT strings or the STRING structure instead.

*Non-zero values are clamped to True.
**Values exceeding 255 are clamped to 255.
***Values exceeding 65535 are clamped to 65535.
****Float value is a valid IEEE single precision Floating point number.

Examples

Examples highlighted signify common use cases.

REAL Atomic Tag - realtag = 512.5 (decimal)

Server Tag Address	Format	Data Type	Notes
realtag	Standard	Boolean	Value = True
realtag	Standard	Byte	Value = 255
realtag	Standard	Word	Value = 512
realtag	Standard	DWord	Value = 512
realtag	Standard	Float	Value = 512.5
realtag [3]	Array Element	Boolean	Invalid: Tag is not an array. Also, Boolean is invalid.
realtag [3]	Array Element	DWord	Invalid: Tag is not an array.
realtag {3}	Array w/o Offset	DWord	Invalid: Tag is not an array.
realtag {1}	Array w/o Offset	Float	Value = [512.5]
realtag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
realtag [3] {1}	Array w/ Offset	Float	Invalid: Tag is not an array.
realtag . 3	Bit	Boolean	Value = True
realtag . 0 {32}	Array w/o Offset	Boolean	Value = [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,...0] Bit value for 512
realtag	String	String	Invalid: Syntax / data type not supported.
realtag	String	String	Invalid: Syntax / data type not supported.

REAL Array Tag - realarraytag [4,4] = [[82.1,69.2,65.3,76.4],[256.5,257.6,258.7,259.8],[9.0,10.0,11.0,12.0],[13.0,14.0,15.0,16.0]]

Server Tag Address	Format	Data Type	Notes
realarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
realarraytag	Standard	Byte	Invalid: Tag cannot be an array.
realarraytag	Standard	Word	Invalid: Tag cannot be an array.
realarraytag	Standard	DWord	Invalid: Tag cannot be an array.
realarraytag	Standard	Float	Invalid: Tag cannot be an array.
realarraytag [3]	Array Element	Float	Invalid: Server tag missing dimension 2 address.
realarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
realarraytag [1,3]	Array Element	Float	Value = 259.8
realarraytag {10}	Array w/o Offset	Byte	Value = [82,69,65,76,255,255,255,255,9,10]
realarraytag {2} {5}	Array w/o Offset	Float	Value = [82.1,69.2,65.3,76.4,256.5] [257.6,258.7,259.8,9,10]
realarraytag {1}	Array w/o Offset	Float	Value = 82.1
realarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
realarraytag [1,3] {4}	Array w/ Offset	Float	Value = [259.8,9.0,10.0,11.0]
realarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
realarraytag [1,3] . 3	Bit	Boolean	Value = 0
realarraytag [1,3] . 0 {32}	Array w/o Offset	Boolean	Value = [1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0] Bit value for 259
realarraytag	String	String	Invalid: Syntax / data type not supported.
realarraytag	String	String	Invalid: Syntax / data type not supported.

LREAL

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	Double*	The Native Tag must be an array.
Array w/ Offset	Double Array	The Native Tag must be an array.
Array w/o Offset	Double Array	If accessing more than a single element, the Native Tag must be an array.

Format	Supported Data Types	Notes
Bit	Boolean	Invalid: Syntax/Data type not supported.
Standard	Double*	None
String	String	Invalid: Syntax/Data type not supported.

*Double value is a valid IEEE double precision Floating point number.

Examples

Examples highlighted signify common use cases.

LREAL Atomic Tag – lrealtag = 512.5 (decimal)

Server Tag Address	Format	Data Type	Notes
lrealtag	Standard	Boolean	Invalid: Data type not supported.
lrealtag	Standard	Byte	Invalid: Data type not supported.
lrealtag	Standard	Word	Invalid: Data type not supported.
lrealtag	Standard	DWord	Invalid: Data type not supported.
lrealtag	Standard	Double	Value = 512.5
lrealtag [3]	Array Element	Boolean	Invalid: Tag is not an array, and Boolean is invalid.
lrealtag [3]	Array Element	DWord	Invalid: Tag is not an array.
lrealtag {3}	Array w/o Offset	DWord	Invalid: Tag is not an array.
lrealtag {1}	Array w/o Offset	Double	Value = [512.5]
lrealtag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lrealtag [3] {1}	Array w/ Offset	Float	Invalid: Tag is not an array.
lrealtag . 3	Bit	Boolean	Invalid: Data type not supported.
lrealtag . 0 {32}	Array w/o Offset	Boolean	Invalid: Data type not supported.
lrealtag	String	String	Invalid: Syntax / data type not supported.
lrealtag	String	String	Invalid: Syntax / data type not supported.

LREAL Array Tag - realarraytag [4,4] = [[82.1,69.2,65.3,76.4],[256.5,257.6,258.7,259.8],[9.0,10.0,11.0,12.0],[13.0,14.0,15.0,16.0]]

Server Tag Address	Format	Data Type	Notes
lrealarraytag	Standard	Boolean	Invalid: Tag cannot be an array.
lrealarraytag	Standard	Byte	Invalid: Tag cannot be an array.
lrealarraytag	Standard	Word	Invalid: Tag cannot be an array.
lrealarraytag	Standard	DWord	Invalid: Tag cannot be an array.
lrealarraytag	Standard	Double	Invalid: Tag cannot be an array.

Server Tag Address	Format	Data Type	Notes
lrealarraytag [3]	Array Element	Double	Invalid: Server tag missing dimension 2 address.
lrealarraytag [1,3]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
lrealarraytag [1,3]	Array Element	Double	Value = 259.8
lrealarraytag {10}	Array w/o Offset	Byte	Invalid: Data type not supported.
lrealarraytag {2} {5}	Array w/o Offset	Double	Value = [82.1,69.2,65.3,76.4,256.5] [257.6,258.7,259.8,9,10]
lrealarraytag {1}	Array w/o Offset	Double	Value = 82.1
lrealarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lrealarraytag [1,3] {4}	Array w/ Offset	Double	Value = [259.8,9.0,10.0,11.0]
lrealarraytag . 3	Bit	Boolean	Invalid: Tag must reference atomic location.
lrealarraytag [1,3] . 3	Bit	Boolean	Value = 0
lrealarraytag [1,3] . 0 {32}	Array w/o Offset	Boolean	Invalid: Syntax/Data type not supported.
lrealarraytag	String	String	Invalid: Syntax / data type not supported.
lrealarraytag	String	String	Invalid: Syntax / data type not supported.

SHORT_STRING

*For more information on the format, refer to **Address Formats**.*

Format	Supported Data Types	Notes
Array Element	String	The Native Tag must be an array.
Array w/ Offset	N/A	N/A
Array w/o Offset	N/A	N/A
Bit	N/A	N/A
Standard	String	The length of the string is based on the length encoding contained within the Native Tag. If the string contains non-printable characters, these are included in the string.
String	N/A	The length of the string needs to be specified in the tag address.

Examples

Examples highlighted signify common use cases.

SHORT_STRING Atomic Tag – stringtag = "mystring"

Server Tag Address	Format	Data Type	Notes
stringtag	Standard	String	Value = mystring.
stringtag	Standard	Byte	Invalid: Byte is not supported.
stringtag	Standard	Word	Invalid: Word is not supported.
stringtag [3]	Array Element	Boolean	Invalid: Tag is not an array, and Boolean is invalid.
stringtag [3]	Array Element	Double	Invalid: Tag is not an array.
stringtag {3}	Array w/o Offset	Double	Invalid: Tag is not an array.
stringtag {1}	Array w/o Offset	Double	Value = [1.16767E+15].
stringtag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
lintag [3] {1}	Array w/ Offset	Double	Invalid: Tag is not an array.
stringtag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
stringtag / 1	String	String	Invalid: Syntax/data type not supported.

SHORT_STRING Array Tag – stringarraytag[2,2] = [one,two].[three,four]

Server Tag Address	Format	Data Type	Notes
stringarraytag	Standard	Boolean	Invalid: Boolean not supported.
stringarraytag	Standard	Byte	Invalid: Byte not supported.
stringarraytag	Standard	Word	Invalid: Word not supported.
stringarraytag	Standard	Double	Invalid: Tag cannot be an array.
stringarraytag	Standard	Date	Invalid: Tag cannot be an array.
stringarraytag [1]	Array Element	Double	Invalid: Server tag missing dimension 2 address.
stringarraytag [1,1]	Array Element	Boolean	Invalid: Boolean not allowed for array elements.
stringarraytag [1,1]	Array Element	String	Value: "four"
stringarraytag {4}	Array w/o Offset	String	Invalid: String array not supported.
stringarraytag {2} {2}	Array w/o Offset	String	Invalid: String array not supported.
stringarraytag {1}	Array w/o Offset	Boolean	Invalid: Bad data type.
stringarraytag [0, 1] {2}	Array w/ Offset	String	Value: "three"
stringarraytag . 3	Bit	Boolean	Invalid: Syntax/data type not supported.
stringarraytag / 1	String	String	Invalid: Syntax not supported.

Error Codes

The following sections define error codes that may be encountered in the server's Event Log. For more information on a specific error code type, select a link from the list below.

Encapsulation Protocol Error Codes
CIP Error Codes

Encapsulation Protocol Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0001	Command not handled.
0002	Memory not available for command.
0003	Poorly formed or incomplete data.
0064	Invalid Session ID.
0065	Invalid length in header.
0069	Requested protocol version not supported.
0070	Invalid Target ID.

CIP Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0001	Connection Failure*
0002	Insufficient resources
0003	Value invalid
0004	IOI could not be deciphered or tag does not exist
0005	Unknown destination
0006	Data requested would not fit in response packet
0007	Loss of connection
0008	Unsupported service
0009	Error in data segment or invalid attribute value
000A	Attribute list error
000B	State already exists
000C	Object Model conflict
000D	Object already exists
000E	Attribute not configurable
000F	Permission denied
0010	Device state conflict
0011	Reply will not fit
0012	Fragment primitive

Error Code	Description
0013	Insufficient command data/parameters specified to execute service
0014	Attribute not supported
0015	Too much data specified
001A	Bridge request too large
001B	Bridge response too large
001C	Attribute list shortage
001D	Invalid attribute list
001E	Embedded service error
001F	Failure during connection**
0022	Invalid reply received
0025	Key segment error
0026	Number of IOI words specified does not match IOI word count
0027	Unexpected attribute in list

- ***See Also:** ***0x0001 Extended Error Codes***
- ****See Also:** ***0x001F Extended Error Codes***

Allen-Bradley Specific Error Codes

Error Code (hex)	Description
00FF	General Error*

- ***See Also:** ***0x00FF Extended Error Codes***
- *For unlisted error codes, refer to the Rockwell Automation documentation.*

0x0001 Extended Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0100	Connection in use.
0103	Transport not supported.
0106	Ownership conflict.
0107	Connection not found.
0108	Invalid connection type.
0109	Invalid connection size.
0110	Module not configured.
0111	EPR not supported.
0114	Wrong module.
0115	Wrong device type.
0116	Wrong revision.
0118	Invalid configuration format.
011A	Application out of connections.

Error Code	Description
0203	Connection timeout.
0204	Unconnected message timeout.
0205	Unconnected send parameter error.
0206	Message too large.
0301	No buffer memory.
0302	Bandwidth not available.
0303	No screeners available.
0305	Signature match.
0311	Port not available.
0312	Link address not available.
0315	Invalid segment type.
0317	Connection not scheduled.
0318	Link address to self is invalid.

- *For unlisted error codes, refer to the Rockwell Automation documentation.*

0x001F Extended Error Codes

The following error codes are in hexadecimal.

Error Code	Description
0203	Connection timed out.

- *For unlisted error codes, refer to the Rockwell Automation documentation.*

0x00FF Extended Error Codes

The following error codes are in hexadecimal.

Error Code	Description
2104	Address out of range.
2105	Attempt to access beyond end of data object.
2106	Data in use.
2107	Data type is invalid or not supported.

- *For unlisted error codes, refer to the Rockwell Automation documentation.*

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.

Controller not supported. | Vendor ID = <vendor>, Product type = <type>, Product code = <code>, Product name = '<product>'.

Error Type:
Warning

Frame received from device contains errors.

Error Type:
Warning

Possible Cause:

1. The packets are misaligned (due to connection/disconnection between the PC and device).
2. There is bad cabling connecting the device causing noise.
3. An incorrect frame size was received.
4. There is a TNS mismatch.
5. An invalid response command was returned from the device.

Possible Solution:
While the driver can recover from this error without intervention, there may be an issues with the cabling or the device itself that should be corrected.

Write request for tag failed due to a framing error. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:

1. A write request for the specified tag failed after so many retries due to an incorrect request service code.
2. A write request for the specified tag failed after so many retries because the number of bytes received was more or fewer than expected.

Possible Solution:

1. There may be an issue with the cabling or the device itself.
2. Increase the retry attempts to give the driver more opportunities to recover from this error.

Read request for tag failed due to a framing error. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:

1. A read request for the specified tag failed after so many retries due to an incorrect request service code.
2. A read request for the specified tag failed after so many retries because the number of bytes received was more or fewer than expected.

Possible Solution:

1. There may be an issue with the cabling or the device itself.
2. Increase the retry attempts to give the driver more opportunities to recover from this error.

Block read request failed due to a framing error. | Block start = '<address>', Block size = <number> (elements).

Error Type:
Warning

Possible Cause:

1. A read request for the specified tag failed after so many retries due to an incorrect request service code.
2. A read request for the specified tag failed after so many retries because the number of bytes received was more or fewer than expected.

Possible Solution:

1. There may be an issue with the cabling or the device itself.
2. Increase the retry attempts to give the driver more opportunities to recover from this error.

Unable to write to tag on device. | Tag address = '<address>', CIP error = <code>, Extended error = <code>.

Error Type:
Warning

Possible Cause:

The device returned an error within the CIP portion of the packet during a write request for the specified tag.

Possible Solution:
The solution depends on the error codes returned. Consult the CIP and Extended code definitions.

See Also:
CIP Error Codes

Unable to read tag from device. | Tag address = '<address>', CIP error = <code>, Extended error = <code>.

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the packet during a read request for the specified tag.

Possible Solution:
The solution depends on the error codes returned. Consult the CIP and Extended code definitions.

See Also:
CIP Error Codes

Unable to read block from device. | Block start = '<address>', Block size = <number>, CIP error = <code>, Extended error = <code>.

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the packet during a block read request for the specified tag.

Possible Solution:
The solution depends on the error codes returned. Consult the CIP and Extended code definitions.

See Also:
CIP Error Codes

Unable to write to tag on device. Controller tag data type unknown. | Tag address = '<address>', Unknown data type = <type>.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the Native Tag data type is not currently supported.

Possible Solution:
Contact Technical Support to request that support may be added for this type.

Unable to read tag from device. Controller tag data type unknown. Tag deactivated. | Tag address = '<address>', Unknown data type = <type>.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the Native Tag data type is not currently supported.

Possible Solution:
Contact Technical Support to request that support may be added for this type.

Unable to read block from device. Controller tag data type unknown. Block deactivated. | Block start = '<address>', Block size = <number>, Unknown data type = <type>.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the Native Tag data type is not currently supported.

Possible Solution:
Contact Technical Support to request that support may be added for this type.

Unable to write to tag on device. Data type not supported. | Tag address = '<address>', Unsupported data type = '<type>'.

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the client tag data type is not supported.

Possible Solution:
Change the tag data type to one that is supported. In response to this error, the tag is deactivated and not processed again.

See Also:
Addressing Atomic Data Types

Unable to read tag from device. Data type not supported. Tag deactivated. | Tag address = '<address>', Unsupported data type = '<type>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client tag data type is not supported.

Possible Solution:
Change the tag data type to one that is supported. In response to this error, the tag is deactivated and not processed again.

See Also:
Addressing Atomic Data Types

Unable to read block from device. Data type not supported. Block deactivated. | Block start = '<address>', Block size = <number> (elements), Unsupported data type = '<type>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client tag data type is not supported.

Possible Solution:
Change the tag data type to one that is supported. In response to this error, the tag is deactivated and not processed again.

See Also:
Addressing Atomic Data Types

Unable to write to tag. Data type is illegal for tag. | Tag address = '<address>', Illegal data type = '<type>'.

Error Type:
Warning

Possible Cause:
A request for the specified tag failed because the tag data type is not supported.

Possible Solution:
Change the tag data type to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean remedies the problem.

See Also:
Addressing Atomic Data Types

Unable to read tag from device. Data type is illegal for this tag. Tag deactivated. | Tag address = '<address>', Illegal data type = '<type>'.

Error Type:
Warning

Possible Cause:
A request for the specified tag failed because the tag data type is not supported.

Possible Solution:
Change the tag data type to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean remedies the problem.

See Also:
Addressing Atomic Data Types

Unable to read block from device. Data type is illegal for this block. Block deactivated. | Block start = '<address>', Block size = <number> (elements), Illegal data type = '<type>'.

Error Type:
Warning

Possible Cause:
A request for the specified tag failed because the tag data type is not supported.

Possible Solution:
Change the tag data type to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean remedies the problem.

See Also:
Addressing Atomic Data Types

Unable to write to tag on device. Tag does not support multi-element arrays. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the driver does not support multi-element array access to the given Native Tag.

Possible Solution:
Change the tag data type or address to one that is supported.

See Also:
Addressing Atomic Data Types

Unable to read tag from device. Tag does not support multi-element arrays. Tag deactivated. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the driver does not support multi-element array access to the given Native Tag.

Possible Solution:
Change the tag data type or address to one that is supported. In response to this error, the tag is deactivated and not processed again.

See Also:
Addressing Atomic Data Types

Unable to read block from device. Block does not support multi-element arrays. Block deactivated. | Block start = '<address>', Block size = <number> (elements).

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the driver does not support multi-element array access to the given Native Tag.

Possible Solution:
Change the tag data type or address to one that is supported. In response to this error, the block is deactivated and not processed again.

See Also:
Addressing Atomic Data Types

Unable to write to tag on device. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
1. The connection between the device and the host PC is broken.
2. The communication parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect address.

Possible Solution:
1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.
3. Verify that the address given to the named device matches that of the actual device.

Note:
In response to this error, the tag is deactivated and not processed again.

Unable to read tag from device. Tag deactivated. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communication parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect address.

Possible Solution:

1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.
3. Verify that the address given to the named device matches that of the actual device.

Note:
In response to this error, the tag is deactivated and not processed again.

Unable to read block from device. Block deactivated. | Block start = '<address>', Block size = <number>.

Error Type:
Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communication parameters for the connection are incorrect.
3. The named device may have been assigned an incorrect address.

Possible Solution:

1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.
3. Verify that the address given to the named device matches that of the actual device.

Note:
In response to this error, the block is deactivated and not processed again.

Device responded with CIP error. | Status code = <code>, Extended status code = <code>.

Error Type:
Warning

Possible Cause:
The device returned an error within the CIP portion of the packet during a request. All reads and writes within the request failed.

Possible Solution:
The solution depends on the error codes returned. Consult the CIP codes.

Memory could not be allocated for tag. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The resources needed to build a tag could not be allocated. The tag is not added to the project.

Possible Solution:
Close any unused applications and/or increase the amount of virtual memory and try again.

Device responded with DF1 error.

Error Type:
Warning

Possible Cause:
The server sent an invalid response.

Possible Solution:

1. The driver attempts to recover from this error.
2. The solution depends on the error codes returned by the device.

See Also:
Error Matrix

Unable to read tag from device. Internal memory is invalid. | Tag address = '<address>'.

Error Type:
Warning

Unable to read tag from device. Data type is illegal for tag. | Tag address = '<address>', Illegal data type = '<type>'.

Error Type:

Warning

Possible Cause:

A request for the specified tag failed because the tag data type is not supported.

Possible Solution:

1. Verify or correct the data type requested.
2. Change the tag data type to one that is supported. For example, data type Short is illegal for a BOOL array Native Tag. Changing the data type to Boolean remedies the problem.

See Also:

Addressing Atomic Data Types

Unable to read tag from device. Internal memory is invalid. Tag deactivated. | Tag address = '<address>'.

Error Type:

Warning

Unable to read block from device. Internal memory is invalid. Block deactivated. | Block start = '<address>', Block size = <number> (elements).

Error Type:

Warning

Unable to write to address on device. Internal memory is invalid. | Tag address = '<address>'.

Error Type:

Warning

Unable to read block from device. Block deactivated. | Block start = '<address>', Block size = <number>, CIP error = <code>, Extended error = <code>.

Error Type:

Warning

Possible Cause:

1. The connection between the device and the host PC is broken.
2. The communication parameters for the connection are incorrect.

Possible Solution:

1. Verify the cabling between the PC and the device.
2. Verify that the correct port has been specified for the named device.

3. Verify that the address given to the named device matches that of the actual device.

Note:
In response to this error, elements of the block are deactivated and not processed again.

See Also:
CIP Error Codes

Device identity details. | ID = <ID>, Vendor ID = <vendor>, Product Type = <type>, Product Code = <code>, Revision = '<revision>', Product Name = '<product>', Product S/N = <number>.

Error Type:
Informational

Device does not support Fragmented Read/Write Services. Automatically falling back to Non-Fragmented Services.

Error Type:
Informational

Glossary

Native Tag-Based Addressing

Term	Definition
Array Element	Element within a native Array Tag. For client/server access, the element must be an atomic. For example, ARRAYTAG [0].
Array with Offset	Client/Server array tag whose address has a native Array Element specified. For example, ARRAYTAG [0] {5}.
Array w/o Offset	Client/Server array tag whose address has no native Array Element specified. For example, ARRAYTAG {5}.
Atomic Data Type	A pre-defined, non-structured Native data type. For example, SINT, DINT.
Atomic Tag	A Native Tag defined with an Atomic Data Type.
Client	An HMI/SCADA or data bridging software package utilizing OPC, DDE, or proprietary client/server protocol to interface with the server.
Client/Server Data Type	Data type for tags defined statically in the server or dynamically in a client. The data types supported in the client depends on the client in use.*
Client/Server Tag	Tag defined statically in the server or dynamically in a client. These tags are different entities than Native Tags. A Native Tag name becomes a Client/Server Tag address when referencing such Native Tag.
Client/Server Array	Row x column data presentation format supported by the server and by some clients. Not all clients support arrays.
CCW	Connected Components Workbench.
Native Data Type	A data type defined in CCW for Micro800 controllers.
Native Tag	A tag defined in CCW for Micro800 controllers.
Native Array Data Type	A multi-dimensional array (1, 2 or 3 dimensions possible) supported in CCW for Micro800 controllers. All atomic data types support Native Arrays. Not all structured data types support Native Arrays.
Array Tag	A Native Tag defined with a native Array Data Type.
Pre-Defined Data Type	A Native data type supported and pre-defined by CCW for Micro800 controllers.*
User-Defined Data Type	A Native data type supported by CCW and defined by the user for Micro800 controllers.*
Server	The OPC/DDE/proprietary server utilizing this d river.
Structured Data Type	A pre-defined or user-defined data type, consisting of members whose data types are atomic or structure in nature.
Structure Tag	A Native Tag defined with a Structured Data Type.

*The data types supported in the server are listed in **Data Types Description**.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W