Omron NJ Ethernet Driver

Table of Contents

Omron NJEthernet Driver

Help version 1.038

CONTENTS

Overview

The Omron NJEthernet Driver provides a reliable way to connect Omron NJEthernet controllers to client applications; including HMI, SCADA, Historian, MES, ERP, and countless custom applications.

Setup

Supported Controllers

Omron NJ301
Omron NJ501

Communication Protocol

Ethernet/IP (CIP over Ethernet) using TCP/IP.

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.

Communications Routing and Timing

Routing provides a way to communicate with a remote NJCPU via a local NJEthernet/IP unit. For more information on routing, refer to Chapter 8 of Omron's NJ-series CPU Unit Built-in Ethernet/IP Port User's Manual W506. Although the material focuses on NJto NJcommunications, the same concepts apply when the driver is communicating with a remote NJCPU.

Routing Timing

When communication with a remote CPU is lost, the driver utilizes different request timeout property than configured in the device's timing settings when performing the following:

- **Identity Requests**: Unconnected messages used to determine the remote CPU's model and Firmware version.
- **Forward Open Requests**: Unconnected messages used to establish a high-level CIP connection with the remote CPU.

In these situations, the local device returns CIP Error 0x01 Ext. Err 0x204, which is defined as an "unconnected request timeout" and indicates that the remote CPU could not be reached. These requests usually occur after a connection has been closed following a read or write request timeout. In this scenario, the device may enter an error state quickly following a read or write request timeout but fail tags slowly thereafter as it waits for the unconnected request timeout to occur. Although the driver waits the entire "unconnected request timeout," it is possible for the local device to respond sooner with this error.

This custom timeout is based on the number of segments in the routing path. Only one attempt is made per request. For more information, refer to the table below.

Number of Segments	Request Timeout (s)	Example
1	15	10.10.110.2\1\0
2	20	10.10.110.2\1\#10\2\172.16.1.3
3	25	10.10.110.2\1\#10\2\172.16.1.3\1\0
4	30	10.10.110.2\1\#10\2\192.168.1.3\1\0\2\172.16.1.4
5	35	10.10.110.2\1\#10\2\192.168.1.3\1\0\2\172.16.1.4\1\0

Note: The driver always utilizes the device's timing settings when performing reads, writes, and Automatic Tag Generation operations regardless of whether routing to a remote CPU or directly to a local CPU.
For more information, refer to ***Device Properties — Timing****.*

Connection Path Specification

The CIP connection path (more commonly known as the routing path) is specified in the Device ID. Communication originates from the Omron NJEthernet Driver on the PC and is directed at the local NJCPU or Ethernet/IP unit. Once at this local unit, the Device ID specifies a way out of the unit and onto the back plane. The routing path then directs the message to the desired remote NJCPU unit.

The routing path itself is a series of Port/Link Address pairs, which are identical to the routing paths described in Chapter 8 of Omron's NJ-series CPU Unit Built-in Ethernet/IP Port User's Manual W506. In that document, "Network type number" is synonymous with Port and "Remote address" is synonymous with Link Address. Within the routing path, both Ports and Link Addresses are delimited by a backslash (no spaces necessary).

Note: A routing path is not necessary when the destination is the local NJCPU unit. The Device ID only needs to contain the IP address of the local CPU unit.

Designator Type	Description	Formats	Range
Port (Network type number)	Specifies a way out of the interface unit in question. Back plane (BP) Port: 1 Ethernet/IP (EIP) Port: 2	Decimal Hex	0-65535 #0000-#FFFF
Link Address (Remote address)	Specifies a destination from the Port. Back plane (BP) Port: Unit Address of the destination unit Ethernet/IP (EIP) Port: IP Address of the remote CPU or Ethernet/IP (EIP) unit	Decimal Hex	Unit: 0-255 IP: Valid dotted-quad IP* #00-#FF

* Host names are not allowed.

Device ID Syntax containing Routing Path

Local CPU (0 Hops)
Local EIP IP \ BP Port \ CPU Unit Address

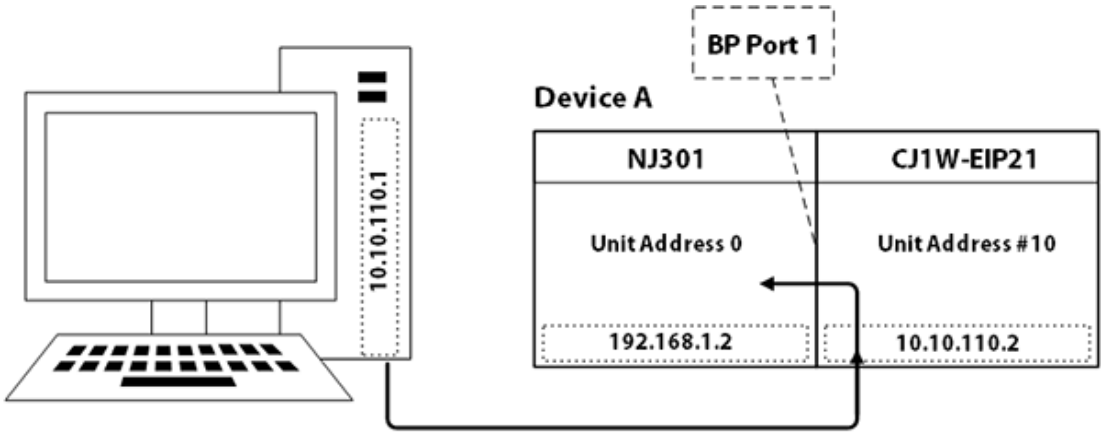
Remote CPU (1 Hop)
Local CPU IP \ BP Port \ EIP Unit Address \ EIP Port \ Remote CPU IP Address or Local CPU IP\ BP Port \ EIP Unit Address \ EIP Port \ Remote EIP IP Address\ BP Port\ Remote CPU Unit Address

Multi-Hop (*N* Hops)
Local CPU IP \ BP Port \ EIP Unit Address \ EIP Port \ Remote CPU IP Address \ BP Port…\ EIP Unit Address \ EIP Port \ Remote CPU IP Address or Local CPU IP \ BP Port \ EIP Unit Address \ EIP Port \ Remote CPU IP Address \ BP Port…\ EIP Unit Address \ EIP Port \ Remote EIP IP Address \ BP Port \ Remote CPU Unit Address

Routing Examples

The routing examples below include the entire device ID.

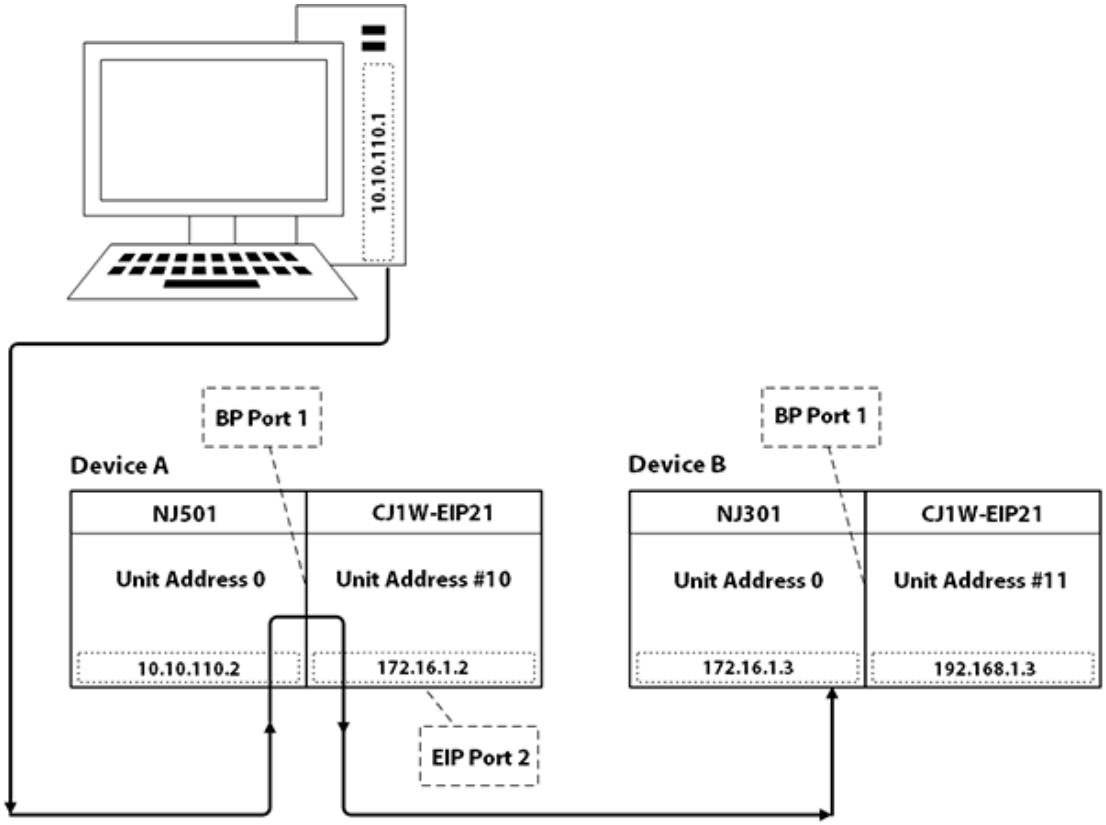
Local Omron NJ(A): 10.10.110.2\1\0



The breakdown of the *10.10.110.2\1\0* path is as follows:

- **10.10.110.2:** IP Address of Device A's EIP unit
- **\1:** Port # of Device A's EIP unit to access Back plane
- **\0:** Unit Address of Device A's CPU unit

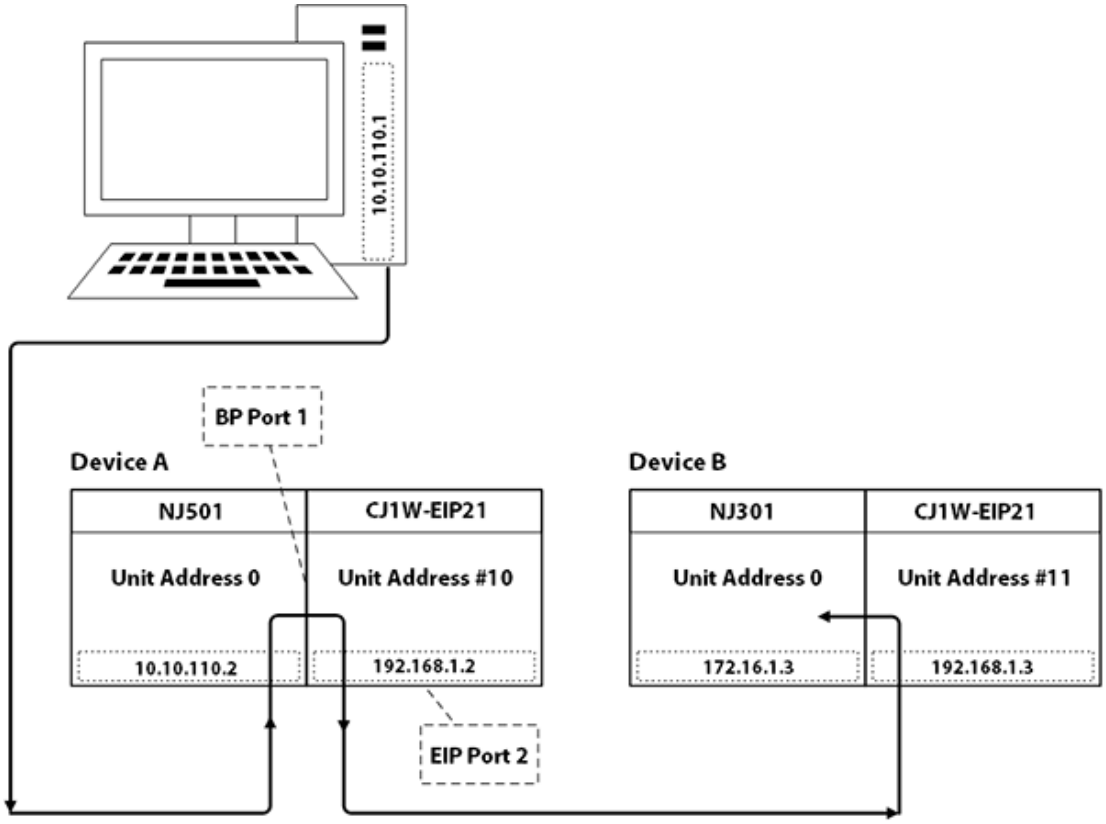
Remote Omron NJ(B) via Omron NJ(A): 10.10.110.2\1\#10\2\172.16.1.3



The breakdown of the *10.10.110.2\1\#10\2\172.16.1.3* path is as follows:

- **10.10.110.2:** IP Address of Device A's CPU unit
- **\1:** Port # of Device A's CPU unit to access Back plane
- **\#10:** Unit Address of Device A's EIP unit (10 hex, 16 dec)
- **\2:** Port # of Device A's EIP unit to access Ethernet/IP
- **\172.16.1.3:** IP Address of Device B's CPU unit

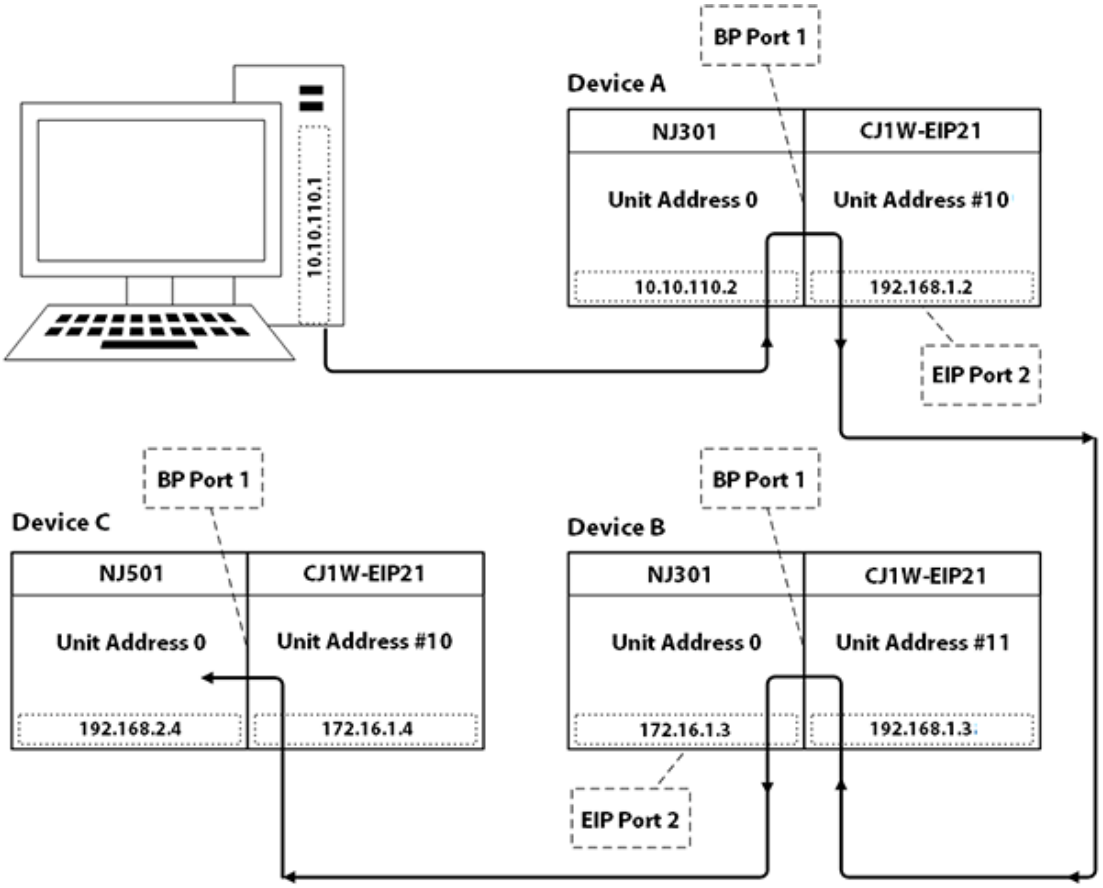
Remote Omron NJ(B) via Omron NJ(A): 10.10.110.2\1\#10\2\192.168.1.3\1\0



The breakdown of the *10.10.110.2\1\#10\2\192.168.1.3\1\0* path is as follows:

- **10.10.110.2:** IP Address of Device A's CPU unit
- **\1:** Port # of Device A's CPU unit to access Back plane
- **\#10:** Unit Address of Device A's EIP unit (10 hex, 16 dec)
- **\2:** Port # of Device A C1J-EIP21 unit to access Ethernet/IP
- **192.168.1.3:** IP Address of Device B's EIP unit
- **\1:** Port # of Device B's EIP unit to access Back plane
- **\0:** Unit Address of Device B's CPU unit

Remote Omron NJ(C) via Omron NJ(A): 10.10.110.2\1\#10\2\192.168.1.3\1\0\2\172.16.1.4\1\0



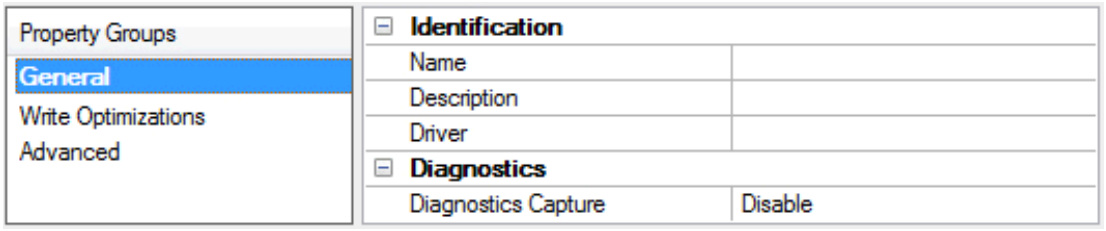
The breakdown of the *10.10.110.2\1\#10\2\192.168.1.3\1\0\2\172.16.1.4\1\0* path is as follows:

- **10.10.110.2:** IP Address of Device A's CPU unit
- **\1:** Port # of Device A's CPU unit to access Back plane
- **\#10:** Unit Address of Device A's EIP unit (10 hex, 16 dec)
- **\2:** Port # of Device A's EIP unit to access Ethernet/IP
- **\192.168.1.3:** IP Address of Device B's EIP unit
- **\1:** Port # of Device B's EIP unit to access Back plane
- **\0:** Unit Address of Device B's CPU unit
- **\2:** Port # of Device B's CPU unit to access Ethernet/IP
- **\172.16.1.4:** IP Address of Device C's EIP unit
- **\1:** Port # of Device C's EIP unit to access Back plane
- **\0:** Unit Address of Device C's CPU unit

*For more information, refer to **Connection Path Specification**. For more information on building a connection/routing path, refer to Chapter 8 of Omron's NJ-series CPU Unit Built-in Ethernet/IP Port User's Manual W506.*

Channel Properties — General

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



Identification

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

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

Description: User-defined information about this channel.

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

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

Note: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-acquire using the old channel name, the item is not accepted. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to server features.

Diagnostics

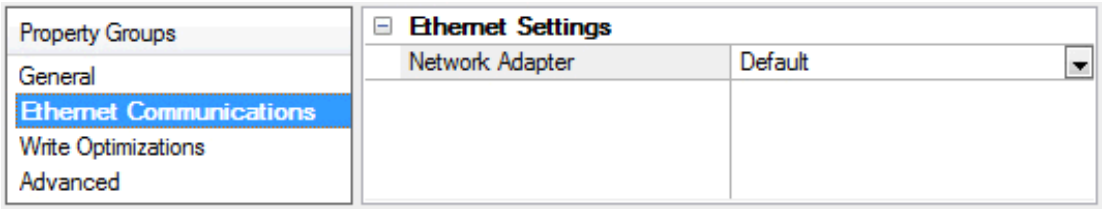
Diagnostics Capture: When enabled, this option makes the channel's diagnostic information available to OPC applications allows the usage of statistics tags that provide feedback to client applications regarding the operation of the channel. Because the server's diagnostic features require a minimal amount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

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

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

Channel Properties — Ethernet Communications

Ethernet Communication can be used to communicate with devices.

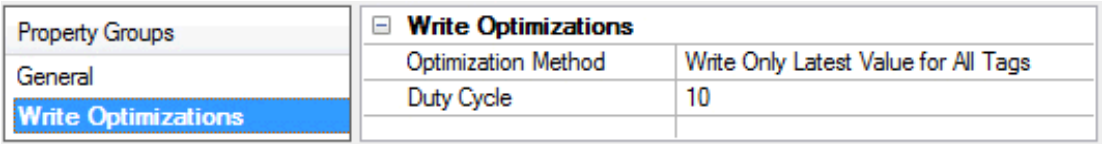


Ethernet Settings

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

Channel Properties — Write Optimizations

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



Write Optimizations

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

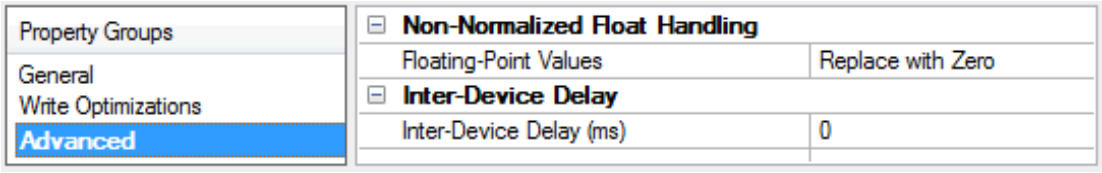
- **Write All Values for All Tags**: This option forces the server to attempt to write every value to the controller. In this mode, the server continues to gather write requests and add them to the server's internal write queue. The server processes the write queue and attempts to empty it by writing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

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

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

Channel Properties — Advanced

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



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

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

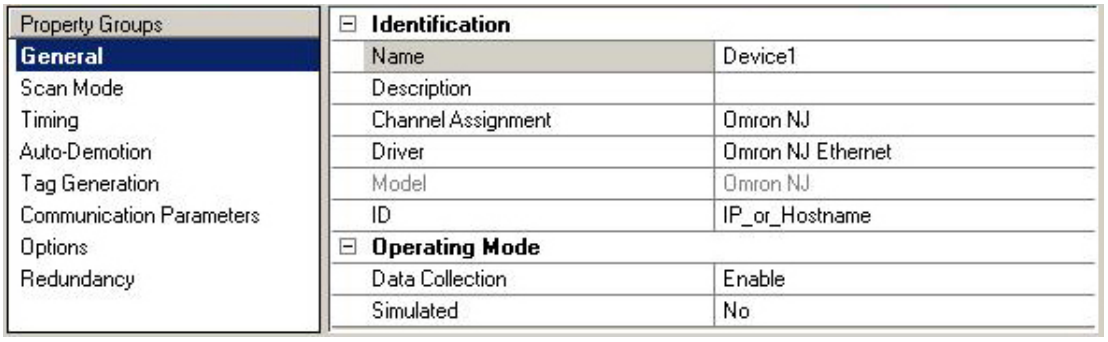
Note: This property is not available if the driver does not support floating point values or if it only supports the option that is displayed. According to the channel's float normalization setting, only real-time driver tags (such as values and arrays) are subject to float normalization. For example,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

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

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

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

Device Properties — General



Identification

Name: User-defined identity of this device.

Description: User-defined information about this device.

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

Driver: Selected protocol driver for this device.

Model: The specific version of the device. *For a list of models that support the FINS Communications Service, refer to the manufacturer's website.*

ID: The ID specifies the path to the destination NJCPU unit.

- The device ID for a local NJCPU unit is specified as the IP or host name of the local CPU unit. It must be a valid dotted-quad IP Address or host name. For example, "192.168.1.100" or "NJ001".
- The device ID for a remote NJCPU unit is specified as the IP or host name of the local CPU or Ethernet/IP unit plus a CIP Connection Path (also known as Routing Path) to the remote CPU unit. For example, "192.168.1.100\1\#10\2\10.10.110.2".

For information on connection path syntax, refer to Connection Path Specification.

Operating Mode

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

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

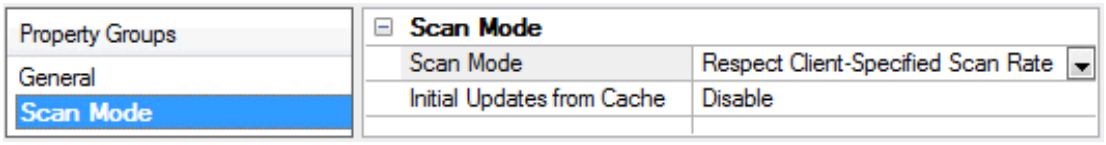
Notes:

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

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

Device Properties — Scan Mode

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



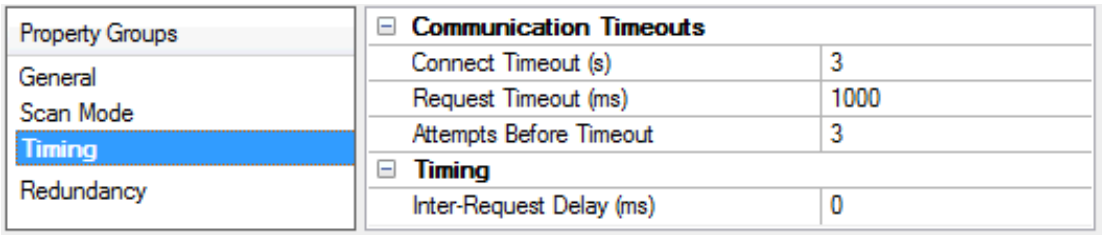
Scan Mode: Specifies how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

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

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

Device Properties — Timing

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



Communications Timeouts

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

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

Request Timeout: This property specifies an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9,999,999 milliseconds (167.6667 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

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

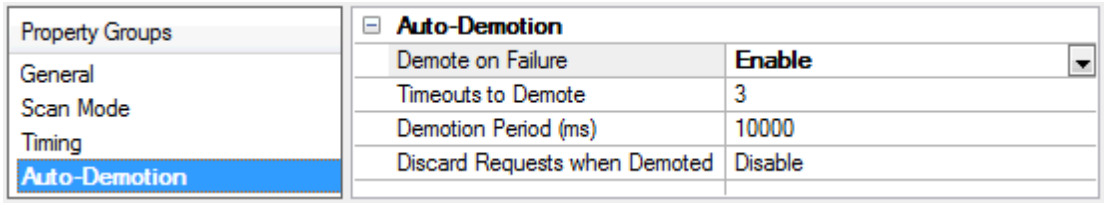
Timing

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

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

Device Properties — Auto-Demotion

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



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

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

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

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

Device Properties — Tag Generation

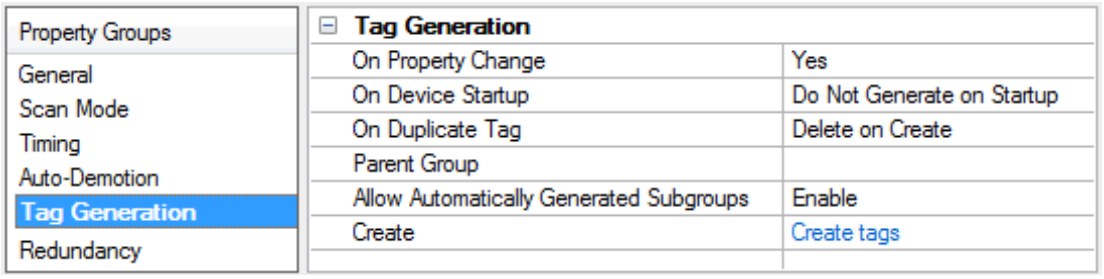
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the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

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

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

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

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



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

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

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

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

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

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

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

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

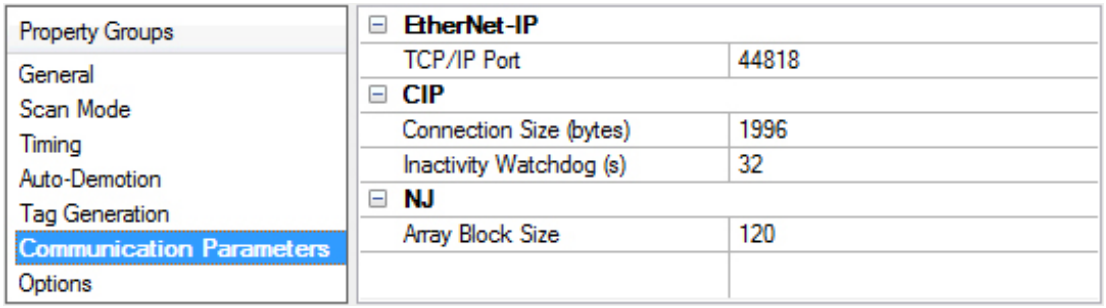
Users should avoid adding tags to the server using names that may match tags that are automatically generated by the driver.

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

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

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

Device Properties — Communications Parameters



EtherNet-IP

TCP/IP Port: Specify the TCP/IP port number that the device is configured to use. The default setting is 44818.

CIP

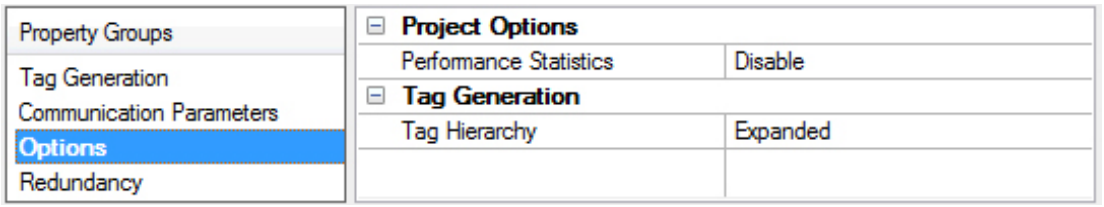
Connection Size: Specify the number of bytes available on the CIP connection for data requests and responses. The valid range is 500 to 1996 bytes. The default setting is 1996 bytes.
Tip: The Connection Size value may be requested through the System Tag _CIPConnectionSizeRequested.

Inactivity Watchdog: Specify the amount of time a connection can remain idle (without Read/Write transactions) before being closed by the controller. In general, the larger the watchdog value, the more time it takes for connection resources to be released by the controller. The default setting is 32 seconds.

NJ

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

Device Properties — Options



Project Options

Performance Statistics: Choose to gather communication statistics to analyze the driver's performance. When enabled, the driver tracks the number and types of client-server tag updates. On restart of the server application, the results are displayed in the Event Log. The default setting is disabled for normal operation.

Notes:

1. Once a project is configured for optimal performance, disable Performance Statistics for best performance.
2. Statistics are sent to the Event Log on shutdown, so the server must be re-launched to view the results.

For more information about performance and diagnostics, see Performance Statistics and Tuning.

Tag Generation

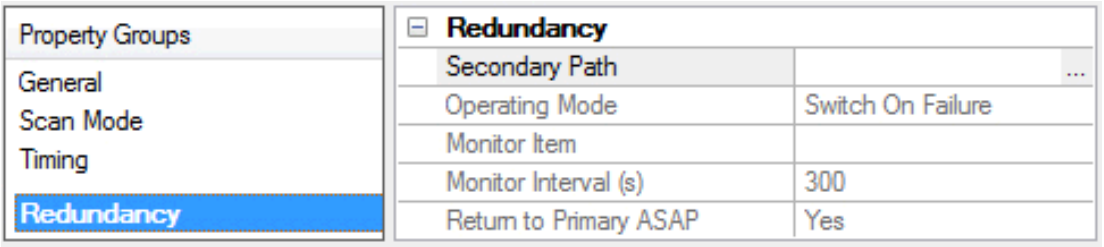
Tag Hierarchy: Select how the tag hierarchy appears. The default setting is Expanded.

- **Condensed** In this mode, the server tags created by automatic tag generation follow a group/tag hierarchy consistent with the tag's address. Groups are created for every segment preceding the period.
- **Expanded** In Expanded Mode, tag groups are created for every segment preceding the period (as in Condensed Mode), but groups are also created for array tags. This is the default setting.

For more information on how groups are created, refer to Tag Hierarchy.

Tip: To use this functionality, enable **Allow Sub Groups**.

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.

Optimizing Communications

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

Connection Size

Increasing the Connection Size allows more Read/Write requests per data packet, which provides greater throughput. Although it also increases the CPU load and response turnaround time, it significantly improves performance. For more information, refer to **Communications Parameters**.

Multi-Request Packets

The Omron NJEthernet Driver has been designed to optimize reads and writes by including multiple requests in a single transaction. This provides drastic improvement in performance over single tag transactions. The only limitation is the number of data bytes that can fit in a single transaction.

- Because read and write requests specify variables' addresses in ASCII format, users should keep the size of the variables' names to a minimum. The smaller the variable name, the more tags that fit in a single transaction, and the fewer transactions needed to process all tags.

Blocking Array Elements

To optimize the reading of basic 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 of 3840 elements means a maximum of 3840 array elements are read in one request.

As discussed in **Communication Parameters**, the block size is adjustable and should be chosen based on the project at hand. For example, if array elements 0 to 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 to 26 would be serviced in one request and element 3839 would be serviced on the next.

Strings

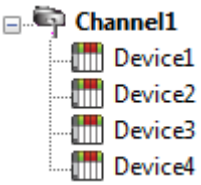
It is recommended that string variables be defined with the smallest string length necessary to serve their purpose. In Sysmac Studio, string variables are defined with a length of 256 by default. Reading these string variables with large string lengths requires extra device communications and may affect performance.

Optimizing the Application

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

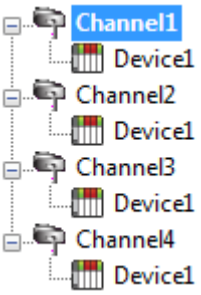
The server refers to communications protocols like Omron NJEthernet as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must then be defined under that channel. Each of these devices represents a single Omron NJCPU 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 Omron NJEthernet 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 Omron NJEthernet Driver could only define one channel, then all devices needed for the project would have to be created beneath it; 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 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 benefits from additional channels. Although spreading the device load across all channels causes the server to move from device to device again, it can now do so with far less devices to process on a single channel.

Performance Statistics and Tuning

The Performance Statistics feature provides benchmarks and statistics about the Omron NJEthernet application's performance. It can affect the server's performance because it is an additional layer of processing. As such, it is disabled by default. To use the Performance Statistics feature, access the device properties and expand the **Options** group. For **Performance Statistics**, select **Enable**.

Types of Performance Statistics

Performance Statistics provide meaningful numerical results across three scopes: device, channel, and driver. Descriptions of the types are as follows:

- **Device:** These statistics provide the data access performance on a particular device.
- **Channel:** These statistics provide the average data access performance for all the devices under a given channel with Performance Statistics enabled.
- **Driver:** These statistics provide the average data access performance for all devices using the Omron NJEthernet Driver with Performance Statistics enabled.

Choosing a Statistic Type

The type of statistics needed depends on the application. In general, driver statistics provide a true measure of the application's performance, whereas channel and device statistics are most relevant while tuning the application. For example, moving 10 certain tags from Device A to Device B may increase the performance of Device A. Moving Device A from Channel 1 to Channel 2 may increase the performance of Channel 1. These are good examples of situations when device and channel statistics should be used.

Locating Statistics

Server statistics are outputted to the server's Event Log upon shutdown. To view the results, shut down the server and then restart it.

Differences between Server Statistics and Performance Statistics

Performance Statistics provide the makeup of the types of reads performed (such as device reads vs. cache reads) whereas server statistics provide a general read count value.

Tuning the Application for Increased Performance

To increase device and channel statistic results, keep variable names to a minimum length and use Variable Arrays as often as possible. F*or more information, refer to* ***Optimizing Communications****.*

For information on increasing driver statistic results, refer to the instructions below. *For more information, refer to* ***Optimizing the Application****.*

1. Devices should be spread across channels. More than one device should not be put on a channel unless necessary.
2. Load should be spread evenly across devices. A single device should not be overloaded unless necessary.
3. The same Variable Tag should not be referenced across different devices.

Data Type Descriptions

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8-bit value
Char	Signed 8-bit value
Word	Unsigned 16-bit value bit 0 is the low bit bit 15 is the high bit
Short	Signed 16-bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
Long	Signed 32-bit value
DWord	Unsigned 32-bit value
Float	32-bit IEEE floating point
Double	64-bit IEEE floating point
Date	64-bit date and time value
String	Null-terminated Unicode string
Default	*

* If the data type is specified as "Default" when creating a Static Tag, the driver queries the controller for the tag's data type and sets the canonical data type for items referencing that Static Tag to the query result. If a data type is not specified when creating a Dynamic Tag, the driver queries the controller for the tag's data type and sets the canonical data type for that Dynamic Tag to the query result.

Address Descriptions

The Omron NJEthernet Driver uses a tag or symbol-based addressing structure referred to as variables. These tags differ from conventional PLC data items in that the tag name itself is the address, not a file or register number. Users can access the controller's basic data types. Although some of the system-defined types are structures, they are ultimately based on these basic data types. Thus, all basic members of a structure are accessible.

Omron Data Type	Description	Data Type	Range
BOOL	Single bit value	Boolean	0, 1
SINT	Signed 8-bit value	Char	-128 to 127
USINT	Unsigned 8-bit value	Byte	0 to 255
BYTE	Bit string (8 bits)	Byte	0 to 255
INT	Signed 16-bit value	Short	-32768 to 32767
UINT	Unsigned 16-bit value	Word	0 to 65535
WORD	Bit string (16 bits)	Word	0 to 65535
DINT	Signed 32-bit value	Long	-2147483648 to 2147483647
UDINT	Unsigned 32-bit value	DWord	0 to 4294967295
DWORD	Bit string (32 bits)	DWord	0 to 4294967295
LINT	Signed 64-bit value	Double	-9223372036854775808 to 9223372036854775807
ULINT	Unsigned 64-bit value	Double	0 to 18446744073709551615
REAL	32-bit IEEE floating point	Float	-3.402823e+38 to -1.175495e-38 0 1.175495e-38 to 3.402823e+38
LREAL	64-bit IEEE floating point	Double	-1.79769313486231e+308 to -2.22507385850721e-308 0 2.22507385850721e-308 to 1.79769313486231e+308
DATE AND TIME	Unsigned 64-bit value	Date	The date/time variable format is: YYYY-MM-DDTHH:MM:SS.MS. The supported range is 1970-01-01T00:00:00.000 to 2106-02-06T23:59:59.999.
STRING	Character string	String	String lengths range from 1 to 1985 characters. This equates to variables defined in Sysmac Studio as STRING[2] and STRING[1986] respectively. The extra character accounts for the null terminator.

Omron Data Type	Description	Data Type	Range
Enumeration	Signed 32-bit value	Long	-2147483648 to 2147483647*

* The valid values for an enumeration are actually a subset of the values in the specified range. The subset of values is determined by the configuration of the enumeration in the Omron NJdevice.

Client/Server Tag Address Rules

Variable names correspond to Client/Server Tag addresses. Both variable names and Client/Server Tag addresses follow the IEC 61131-3 identifier rules. Descriptions of the rules are as follows:

- Can only contain alphanumeric characters and underscores
- Can have as many as 127 characters per segment
- Characters are not case sensitive
- Spaces are ignored

Client/Server Tag Name Rules

Tag name assignment in the server differs from address assignment in that names cannot begin with an underscore. For syntax and examples, refer to **Address Formats**.

Important: If a tag address is large enough that it exceeds the protocol limit of 511 bytes, it will fail validation with an "Address out of range" error. If this occurs, reduce the number of characters in the tag address until it passes validation.

Data Type Coercion

The Omron NJEthernet Driver can coerce some Omron data types in the controller to more than one server data type. For example, the tag for a SINT variable in the controller can be created with a Byte server data type. For a list of the supported data type coercions for all Omron data types that are supported by the driver, refer to the table below.

Omron Data Type	Data Type
BOOL	Boolean
SINT, USINT, OR BYTE	Char or Byte
INT, UINT, OR WORD	Short or Word
DINT, UDINT, DWORD, OR ENUM	Long or DWord
LINT OR ULINT	Double
REAL	Float
LREAL	Double
DATE AND TIME	Date
STRING	String

Address Formats

A Variable 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. Descriptions of the variable types are as follows:

- **Array Element**: A variable may be defined in the controller using the following syntax: *ARRAY[x1..x2, y1..y2, z1..z2] OF TYPE*, where *TYPE* is one of the Omron data types listed in **Address Descriptions**. To access individual elements, specify the *x*, *y*, and *z* offsets. The driver blocks read requests on the last dimension. For example, with a variable like "MyArray[1,0]" and "MyArray[1,4]," the driver performs a single request for five elements starting at "MyArray[1,0]." For more information, refer to **Communication Parameters** and **Optimizing Communications**.
- **Array**: A variable may be defined in the controller using the following syntax: *ARRAY[x1..x2, y1..y2, z1..z2] OF TYPE*, where *TYPE* is one of the Omron data types listed in **Address Descriptions**. To access multiple elements in a single client item, use the array type syntax. Like Array Elements, the driver performs a single request to read and write multiple array elements. The difference with arrays is that all items in the array is provided to the client in an atomic operation. String Arrays are not supported.

  Note: Not all clients support array types. For support information, refer to the client application.
- **Basic**: A variable defined with a basic type and no array syntax.
- **String**: A variable defined with the string basic type.

Tip: All Symbolic Variable Tag names in Sysmac Studio can be copied and pasted into the server's tag address field and be valid.

Array Element

At least one dimension (but no more than three) must be specified.

Syntax	Example
<Variable Tag name> [dim1]	tag_1 [5]
<Variable Tag name> [dim 1, dim2]	tag_1 [2, 3]
<Variable Tag name> [dim 1, dim2, dim 3]	tag_1 [2, 58, 547]

Examples
MyBooleanArray[31]
MyBooleanArray3D[2,2,7]
MySintArray[1]
MyLrealArray[65535]
MySintArray2D[1,2]
MyLrealArray2D[2,500]
MySintArray3D[2,3,9]
MyLrealArray3D[2,10,10]

Array

With this format, multiple elements of a Variable Array are read and written in a single transaction. The client must support array types (such as "VT_ARRAY"). Client data is organized in a row by column format to facilitate one dimension (1 row, *y* columns) or two dimensions (*x* rows, *y* columns). This format is supported for one-dimensional, two-dimensional, and three-dimensional Variable Arrays only. Like Array Elements, at least one dimension (but no more than three) must be specified.

Note: All Omron data types that are supported by this driver support the array format except string and date and time.

Important: Spanning an array across multiple Variable Array dimensions is not supported. If an array is created on a two-dimensional or three-dimensional Variable Array, the size of that array (which is rows by

columns) must not exceed the bounds of the last dimension. For example, given a Variable Array "MySintArray3D" defined as ARRAY[0..2,0..3,0..9] OF SINT, the Array Tag MySintArray3D[0,0,0]{10} is valid because it references elements [0,0,0..9] that all lie within the last dimension. MySintArray3D[0,0,0]{11} is invalid, however, because it is attempting to reference elements [0,0,0..9] and [0,1,0] that exceed the bounds of the last dimension by one element.

Syntax	Example
<Variable Tag name> [dim 1 offset] {# of columns}	tag_1 [5]{8}
<Variable Tag name> [dim 1 offset, dim 2 offset] {# of columns}	tag_2 [0, 5]{8}
<Variable Tag name> [dim 1 offset, dim 2 offset, dim 3 offset] {# of columns}	tag_3 [1,0, 5]{8}
<Variable Tag name> [dim 1 offset] {# of rows}{# of columns}	tag_4 [5]{2}{4}
<Variable Tag name> [dim 1 offset, dim 2 offset] {# of rows}{# of columns}	tag_1 [0,5]{2}{4}
<Variable Tag name> [dim 1 offset, dim 2 offset, dim 3 offset] {# of rows}{# of columns}	tag_1 [1,0,5]{2}{4}

Note: The number of elements to read and/or write equals the number of rows multiplied by the number of columns. If no rows are specified, the number of rows defaults to 1. At least one element of the array must be addressed. Rows x Columns must be between 1 and 65535.

Examples
MyBooleanArray[0]{32}
MyBooleanArray3D[2,2,7]{1}
MySintArray[1]{5}
MyLrealArray[65535]{1}
MySintArray2D[1,2]{10}
MyLrealArray2D[2,500]{14}{20}
MySintArray3D[2,3,9]{10}
MyLrealArray3D[2,10,10]{14}{20}

Basic

Syntax	Example
<Variable Tag name>	tag_1

Examples
MyBool
MyByte
MyInt
MyWord
MyReal

String

The number of characters to read and/or write equals the string length, which must be at least 2. Although Sysmac Studio allows a variable to be defined as STRING [1], reads and writes are not possible because one character is reserved for a null terminator. As such, it is recommended that string variables be defined with a length of 2 to 1986. To account for this null terminator, the valid range for string lengths in the driver is 1 to 1985. Strings support any character encoded in UTF-8. One UTF-8 character can equal 1 to 4 bytes.

Note: A 256 byte string containing characters that require multiple bytes when encoded in UTF-8 represents fewer than 256 characters.

Syntax	Example
<Variable Tag name> / <string length>	tag_1 / 255

Examples
MyString256/255
MyString1986/1985
MyString1986/100
MyStruct[23].Banners[4].Output/10
MyStringArray3D[2,10,10]/255

Tag Scope

Note: Local variables can only be read and written in a POU (program, function, or function block) in which it is defined.

Global Tags

Global Tags are Variable 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 Variable Data Type and the address format being used.

Structure Tag Addressing

Structure Tags are tags with one or more member tags, which can be basic or structured in nature.

<structure name> . <basic-type tag>

This implies that a substructure would be addressed as:

<structure name> . <substructure name> .<basic-type tag>

Arrays of structures would be addressed as follows:

<structure array name> [dim1, dim2, dim3] . <basic-type tag>

Again, this implies that an array of substructures would be addressed as:

<structure name> . <substructure array name> [dim1, dim2, dim3] . <basic-type tag>

Note: The examples given above are only a few of the many addressing possibilities involving structures. They are displayed to provide an introduction to structure addressing.
For more information, refer to Omron NJdocumentation.

Predefined Term Tags

The tags displayed in the table below can be used to obtain general processor information from a PLC.

Tag Name	Data Type	Description
#DEVICETYPE	Word	An integer value that corresponds to the "ProdType" attribute specified in the PLC's EDS file.
#REVISION	String	Firmware revision displayed as *<major>.<minor>*.

Tag Name	Data Type	Description
#PRODUCTNAME	String	The processor name that corresponds to the "ProdName" attribute specified in the PLC's EDS file.
#PRODUCTCODE	Word	An integer value that corresponds to the "ProdCode" attribute specified in the PLC's EDS file.
#VENDORID	Word	An integer value that corresponds to the "VendCode" attribute specified in the PLC's EDS file.

Automatic Tag Database Generation

The Omron NJEthernet Driver can be configured to automatically generate a list of tags within the server that correspond to the Global Variables used in the Omron SYSMAC NJSeries controller program and that are published to the network as inputs, outputs, or publish-only variables.

To generate tags from the device:

1. In the Configuration, select the device for which tags should be generated.
2. Right-click and select **Properties...** to open the device properties.
3. Locate and expand the **Tag Generation** section.
4. Locate the Create property. The Value is blue text that reads Create tags.
5. Click the Create tags text to initiate tag database creation.
6. Click the **Close** button to exit the dialog box.
7. Check the Event Log for messages confirming successful generation.

For more information about custom settings, see the server help file.

Notes:

1. It is recommended that all communications to the Omron NJdevice cease during tag database creation process.
2. Variable tags generated for enumerations are data type Long.

See Also: *Address Formats and Address Descriptions.*

Tag Hierarchy

The tags created by automatic tag generation can follow one of two hierarchies: Expanded or Condensed. To enable this functionality, ensure that Allow Sub Groups is enabled in device properties. The default setting is Expanded.

Expanded Mode

In Expanded Mode, tag groups are created for every segment preceding the period (as in Condensed Mode), but are also created in logical groupings. Groups created include the following:

- Structures and substructures
- Unions

- Arrays

Basic Global tags (or non-structure, non-union, and non-array tags) are placed at the device level. Each structure, union, and array tag is provided in its own subgroup of the parent group.

The name of the structure, union, or array subgroup also provides a description of the structure, union, or array. For example, an array tag1[1,6] defined in the controller would have a subgroup name of "tag1[*x*,*y*]" where *x* signifies dimension 1 exists and *y* signifies dimension 2 exists. The tags within an array subgroup are all the elements of that array. The tags within a structure subgroup are the structure members themselves. If a structure contains an array, then an array subgroup of the structure group is created as well. The tags within a union subgroup are the union members themselves. If a union contains an array, then an array subgroup of the union group is created as well.

Array Tag Groups
A group is created for each array that contains array elements. Group names have the notation: <*array name*>[*x*,*y*,*z*] where:

- [*x*,*y*,*z*] is a 3 dimensional array
- [*x*,*y*] is a 2 dimensional array
- [*x*] is a 1 dimensional array

Array Tags have the notation: <*tag element*>[*XXXXX*,*YYYYY*,*ZZZZZ*]. For example, element tag1[12,2,987] would have the tag name "tag1[12,2,987]".

Condensed Mode

In Condensed Mode, the server tags created by automatic tag generation follow a group/tag hierarchy consistent with the tag's address. Groups are created for every segment preceding the period. Groups created include the following:

- Structures and substructures
- Unions

Note: Groups are not created for arrays.

Event Log Messages

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

Internal error occurred while attempting to write tag. Unexpected data type. | Tag address = '<address>', Data type = '<type>', DTRV = <code>.

Error Type:
Error

The following errors occurred uploading controller project from device. Resorting to symbolic protocol.

Error Type:
Error

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during an automatic tag generation request.

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

See Also:
CIP Error Codes

Unknown error occurred.

Error Type:
Error

Low memory resources.

Error Type:
Error

Invalid or corrupt controller project detected while synchronizing. Synchronization will be retried shortly.

Error Type:
Error

Project download detected while synchronizing. Synchronization will be retried shortly.

Error Type:

Error

Encapsulation error occurred while uploading project information. | Encapsulation error = <code>.

Error Type:
Error

Possible Cause:
The device returned an error within the encapsulation portion of the Ethernet/IP packet during an automatic tag generation request.

Possible Solution:
The driver attempts to recover from this error.

Note:
This excludes error 0x02, which is device-related and not driver-related.

Error occurred while uploading project information. | CIP error = <code>, Extended error = <code>.

Error Type:
Error

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during an automatic tag generation request.

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

See Also:
CIP Error Codes

Framing error occurred while uploading project information.

Error Type:
Error

Possible Cause:

1. The packets are misaligned due to the connection or disconnection between the PC and device.
2. There is bad cabling connecting the device causing noise.

Possible Solution:

1. Place the device on less noisy network.
2. Increase the Request Timeout and/or Request Attempts.

CIP connection timed-out while uploading project information.

Error Type:
Error

Database error. CIP connection timed-out while uploading project information.

Error Type:
Error

Database error. Error occurred while uploading project information. | CIP error = <code>, Extended error = <code>.

Error Type:
Error

Possible Cause:
The device returned an error within the CIP portion of the Ethernet/IP packet during an automatic tag generation request.

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

See Also:
CIP Error Codes

Database error. Encapsulation error occurred during register session request. | Encapsulation error = <code>.

Error Type:
Error

Database error. Framing error occurred during register session request.

Error Type:
Error

Possible Cause:
The device returned an error within the encapsulation portion of the Ethernet/IP packet during an automatic tag generation request.

Possible Solution:
The driver attempts to recover from this error.

Note:
This excludes error 0x02, which is device-related and not driver-related.

See Also:
Encapsulation Error Codes

Database error. Internal error occurred.

Error Type:
Error

Database error. Encapsulation error occurred during fwd. open request. | Encapsulation error = <code>.

Error Type:
Error

Possible Cause:
The device returned an error within the encapsulation portion of the Ethernet/IP packet during an automatic tag generation request.

Possible Solution:
The driver attempts to recover from this error.

Note:
This excludes error 0x02, which is device-related and not driver-related.

See Also:
Encapsulation Error Codes

Database error. No more connections available for fwd. open request.

Error Type:
Error

Possible Cause:
Omron devices support a finite number of connections. The connection limit has been exceeded.

Possible Solution:
Reduce the number of connections from the servers to the device and try again.

Database error. Error occurred during fwd. open request. | CIP error = <code>, Extended error = <code>.

Error Type:
Error

Possible Cause:
Omron devices support a finite number of connections. The connection limit has been exceeded.

Possible Solution:
Reduce the number of connections from the servers to the device and try again.

Database error. Framing error occurred during fwd. open request.

Error Type:

Error

Possible Cause:

1. The packets are misaligned due to the connection and/or disconnection between the PC and device.
2. There is bad cabling connecting the device causing noise.

Possible Solution:

1. Place the device on less noisy network.
2. Increase the Request Timeout and/or Request Attempts.

Database error. Encapsulation error occurred while uploading project information. | Encapsulation error = <code>.

Error Type:
Error

Possible Cause:
The device returned an error within the encapsulation portion of the Ethernet/IP packet during an automatic tag generation request.

Possible Solution:
The driver attempts to recover from this error.

Note:
This excludes error 0x02, which is device-related and not driver-related.

See Also:
Encapsulation Error Codes

Database error. Framing error occurred while uploading project information.

Error Type:
Error

Possible Cause:

1. The packets are misaligned due to the connection and/or disconnection between the PC and device.
2. There is bad cabling connecting the device causing noise.

Possible Solution:

1. Place the device on less noisy network.
2. Increase the Request Timeout and/or Request Attempts.

Frame received from device contains errors.

Error Type:
Warning

Possible Cause:

1. The packets are misaligned due to connection and/or disconnection between the PC and device.
2. There is bad cabling connecting the device causing noise.

Possible Solution:

1. Place the device on less noisy network.
2. Increase the Request Timeout and/or Attempts.

Error occurred during a request to device. | CIP error = <code>, Extended error = <code>.

Error Type:
Warning

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

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

See Also:
CIP Error Codes

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 more or fewer bytes than expected were received.

Possible Solution:
If this error occurs frequently, there may be an issue with the cabling or device. Increasing the retry attempts allows 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. Incorrect request service code.
2. Received more or fewer bytes than expected.
3. There may be an issue with the cabling or device.

Possible Solution:
Increasing the Retry Attempts gives the driver more opportunities to recover from this error.

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

Error Type:
Warning

Possible Cause:

1. The specified tag address and count failed due to an incorrect request service code.
2. The specified tag address and count failed because more or fewer bytes than expected were received.

Possible Solution:
If this error occurs frequently, there may be an issue with the cabling or the device itself. Increasing the request attempts allows more opportunities to recover from this error. In response to this error, the elements of the block are deactivated and it is not processed again.

Block read request failed due to a framing error. | Block Size = <number> (bytes), Tag Name = '<tag>'.

Error Type:
Warning

Possible Cause:

1. The specified tag address and count failed due to an incorrect request service code.
2. The specified tag address and count failed because more or fewer bytes than expected were received.

Possible Solution:
If this error occurs frequently, there may be an issue with the cabling or the device itself. Increasing the request attempts allows more opportunities to recover from this error. In response to this error, the elements of the block are deactivated and it is not processed again.

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

Error Type:
Warning

Possible Cause:
1. The Ethernet connection between the device and the host PC is broken.
2. The communication parameters for the Ethernet connection are incorrect.
3. The named device may have been assigned an incorrect IP 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 IP address given to the named device matches that of the actual device.

Unable to read tag. | 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 Ethernet/IP packet during a read request for the specified tag.

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

See Also:
CIP Error Codes

Unable to read block on device. | Block Size = <number> (elements), Starting 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 Ethernet/IP packet during a read request for the specified tag.

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

See Also:
CIP Error Codes

Unable to read block on device. | Block size = <number> (bytes), Tag name = '<tag>', CIP error = <code>, Extended error = <code>.

Error Type:
Warning

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

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

See Also:
CIP Error Codes

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

Error Type:
Warning

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

Possible Solution:
Remove references to this variable. In response to this error, the tag is deactivated and not processed again.

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

Error Type:
Warning

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

Possible Solution:
Remove references to this variable. In response to this error, the tag is deactivated and not processed again.

Unable to read block on device. Controller tag data type unknown. Tag deactivated. | Block Size = <number> (elements), Starting tag address = '<address>', Data type = <type>.

Error Type:
Warning

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

Possible Solution:
Remove references to this variable. In response to this error, the block is deactivated and not processed again.

Unable to write to tag. Data type not supported. | Tag address = '<address>', 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 and try again.

Unable to read tag. Data type not supported. Tag deactivated. | Tag address = '<address>', 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 will not be processed again.

Unable to read block on device. Data type not supported. Block deactivated. | Block size = <number> (elements), Starting tag address = '<address>', Data type = '<type>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tags to the specified tag address and count failed because the client tag data type is not supported.

Possible Solution:
Change the data type for tags within this block to supported types. In response to this error, elements of the block are deactivated and will not be processed again.

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

Error Type:
Warning

Possible Cause:
A write request for the specified tag failed because the client tag data type is illegal for the given variable.

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

See Also:
Data Type Coercion

Unable to read tag. Data type is illegal for this tag. | Tag address = '<address>', Data type = '<type>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client's tag data type is illegal for the given variable.

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

See Also:
Data Type Coercion

Unable to read block on device. Data type is illegal for this block. | Block size = <number> (elements), Starting tag address = '<address>', Data type = '<type>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client tag data type is illegal for the given variable.

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

See Also:
Data Type Coercion

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

Error Type:
Warning

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

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

Unable to read tag. 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 variable.

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

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

Error Type:
Warning

Possible Cause:
A read request for the specified tags to the specified tag address and count failed because the driver does not support multi-element array access to the given variable.

Possible Solution:
Change the data type or address for tags within this block to one that is supported. In response to this error, elements of the block are deactivated and will not be processed again.

Unable to write to tag. Native tag size mismatch. | Tag address = '<address>'.

Error Type:
Warning

Unable to read tag. Native tag size mismatch. | Tag address = '<address>'.

Error Type:
Warning

Unable to read block on device. Native tag size mismatch. | Block size = <number> (elements), Starting tag address = '<address>'.

Error Type:
Warning

Unable to read block on device. Native tag size mismatch. | Block size = <number> (bytes), Tag name = '<tag>'.

Error Type:
Warning

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

Error Type:
Warning

Possible Cause:

1. The Ethernet connection between the device and the host PC is broken.
2. The communication parameters for the Ethernet connection are incorrect.
3. The named device may have been assigned an incorrect IP 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 IP address given to the named device matches that of the actual device.

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

Error Type:
Warning

Unable to read block. Block deactivated. | Block size = <number> (elements), Starting tag address = '<address>'.

Error Type:
Warning

Unable to read block on device. Block deactivated. | Block Size = <number> (bytes), Tag Name = '<tag>'.

Error Type:

Warning

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

Error Type:
Warning

Unable to read tag. Data type is illegal for this tag. | Tag address = '<address>', Data type = '<type>'.

Error Type:
Warning

Possible Cause:
A read request for the specified tag failed because the client tag data type is illegal for the given variable.

Possible Solution:
Change the tag's data type to one that is supported.

See Also:
Data Type Coercion

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

Error Type:
Warning

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

Error Type:
Warning

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

Error Type:
Warning

Unable to read block on device. Block deactivated. | Block Size = <number> (elements), Starting Tag address = '<address>', CIP error = <code>, Extended error = <code>.

Error Type:
Warning

Device returned more data than expected while reading tag. Verify the address includes an element offset and all dimensions in that offset. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The tag address references an array variable, but not all dimensions were specified in the element offset or no element offset was specified. For example, the array variable "MyArray" is defined as ARRAY[0..2,0..9] OF INT. Creating a tag address like MyArray[0]{2}@Word Array may cause this error to occur because the second dimension is not specified. In this example, a read of the tag would result in 10 INTs returned versus the 2 INTs that are expected. Likewise, creating a tag address like MyArray@Word may cause this error to occur because neither the first nor second dimension was specified. In this example, a read of the tag would result in 30 INTs returned versus the 1 INT that is expected.

Possible Solution:
Add the fully-qualified element offset to the tag address. In the examples above, the correct tag address could be MyArray[0,0]{2} and MyArray[0,0] respectively.

Note:
This message is only a warning. Reads succeed, but are less efficient because of the extra overhead in the read response. It is recommended that users fix tag addresses that exhibit this behavior.

Device returned more data than expected while reading block. Verify the address includes an element offset and all dimensions in that offset. | Block Size = <number> (elements), Starting Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The tag address references an array variable, but not all dimensions were specified in the element offset or no element offset was specified. For example, the array variable "MyArray" is defined as ARRAY[0..2,0..9] OF INT. Creating a tag address like MyArray[0]{2}@Word Array may cause this error to occur because the second dimension is not specified. In this example, a read of the tag would result in 10 INTs returned versus the 2 INTs that are expected. Likewise, creating a tag address like MyArray@Word may cause this error to occur because neither the first nor second dimension was specified. In this example, a read of the tag would result in 30 INTs returned versus the 1 INT that is expected.

Possible Solution:
Add the fully-qualified element offset to the tag address. In the examples above, the correct tag address could be MyArray[0,0]{2} and MyArray[0,0] respectively.

Note:
This message is only a warning. Reads succeed, but are less efficient because of the extra overhead in the read response. It is recommended that users fix tag addresses that exhibit this behavior.

Unable to write to tag. Address exceeds current CIP connection size. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The tag address size resulted in a request frame that exceeds the protocol limits.

Possible Solution:
Increase the CIP Connection Size to a value that accommodates the tag request frame.

Unable to read block on device. Address exceeds current CIP connection size. | Block size = <number> (elements), Starting tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The tag address size resulted in a request frame that exceeds the protocol limits.

Possible Solution:
Increase the CIP Connection Size to a value that accommodates the tag request frame.

Unable to read tag. Address exceeds current CIP connection size. | Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The tag address size resulted in a request frame that exceeds the protocol limits.

Possible Solution:
Increase the CIP Connection Size to a value that accommodates the tag request frame.

Requested CIP connection size is not supported by this device. Automatically falling back to maximum size. | Requested size = <number> (bytes), Maximum size = <number> (bytes).

Error Type:
Warning

Current value not supported for an XML element on this model. Automatically setting to new value. | Current value = '<value>', XML element = '{<namespace>}<element>', Model = '<model>', New value = '<value>'.

Error Type:

Warning

Database error. Data type of complex type is not supported. A tag for this member will not be added to the database. | Data type = <type>, Complex Type = '<type>', Member = '<name>'.

Error Type:
Warning

Possible Cause:
The device configuration of the complex type member uses an unsupported data type.

Possible Solution:
Modify the device configuration of the complex type member to use a supported data type.

See Also:
Address Descriptions

Database error. Unable to resolve CIP data type for tag. Tag is not added to the database. | Data type = <type>, Tag name = '<tag>'.

Error Type:
Warning

Possible Cause:

1. A communications error occurred during automatic tag database generation that was caused by Ethernet encapsulation, the device, or framing.
2. The CIP data type is not supported by the driver.

Possible Solution:

1. Correct the communications error and retry automatic tag database generation.
2. Change the variable data type to one that is supported.

Database error. Address validation failed for tag. Tag is not added to the database. | Tag name = '<tag>', Tag address = '<address>'.

Error Type:
Warning

Possible Cause:
The tag address size exceeds the protocol limits.

Possible Solution:
Reduce the number of characters in the tag name. If the error persists, reduce the number of characters in the structure or union (including any nested structures or unions) under which the tag is located.

Unable to retrieve the identity for device. | Encapsulation error = <code>.

Error Type:
Warning

Possible Cause:
The identity was not retrieved because the device returned an error within the encapsulation portion of the Ethernet/IP packet during a request.

Possible Solution:
The driver attempts to recover from such an error.

Note:
This excludes error 0x02, which is device-related, not driver-related.

See Also:
Encapsulation Error Codes

Unable to retrieve the identity for device. | CIP error = <code>, Extended error = <code>.

Error Type:
Warning

Possible Cause:
The identity was not retrieved because the device returned an error within the CIP portion of the Ethernet/IP packet during a request.

Possible Solution:
The solution depends on the error code returned.

See Also:
CIP Error Codes

Unable to retrieve the identity for device. Frame received contains errors.

Error Type:
Warning

Possible Cause:

1. The packets are misaligned due to connection and/or disconnection between the PC and device.
2. There is bad cabling connecting the devices causing noise.
3. The wrong frame size was received.
4. There is a TNS mismatch.
5. An invalid response command was returned from the device.
6. The device is not Ethernet/IP enabled.

Possible Solution:

1. The driver recovers from this error without intervention. There may be an issue with the cabling, the network, or the device itself.
2. Verify that the device being communicated with is an Omron Ethernet/IP-enabled device.

Encapsulation error occurred during a request. | Encapsulation error = <code>.

Error Type:
Informational

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

Possible Solution:
The driver attempts to recover from such an error.

Note:
This excludes error 0x02, which is device-related, not driver-related.

See Also:
Encapsulation Error Codes

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

Error Type:
Informational

Possible Cause:
Memory required for generation could not be allocated. The process is cancelled.

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

Database error. Tag renamed because it exceeds maximum character length. | Tag name = '<tag>', Maximum length = <number>, New tag name = '<tag>'.

Error Type:
Informational

Database error. Array tags renamed because it exceeds max character length. | Array tag name = '<name>', Maximum length = <number>, New array tag name = '<name>'.

Error Type:
Informational

Database status: Tags imported. | Data types = <type>, Tags imported = <number>.

Error Type:
Informational

Database status: Generating OPC tags.

Error Type:
Informational

Database status: Building tag projects, please wait. | Tag project count = <number>.

Error Type:
Informational

Database status: Retrieving controller project.

Error Type:
Informational

Elapsed Time = <number> (seconds)

Error Type:
Informational

Symbolic Device Reads = <number>

Error Type:
Informational

Symbolic, Array Block Device Reads = <number>

Error Type:
Informational

Symbolic, Array Block Cache Reads = <number>

Error Type:
Informational

Symbol Instance Non-Block Device Reads = <number>

Error Type:
Informational

Symbol Instance Non-Block, Array Block Device Reads = <number>

Error Type:
Informational

Symbol Instance Non-Block, Array Block Cache Reads = <number>

Error Type:
Informational

Symbol Instance Block Device Reads = <number>

Error Type:
Informational

Symbol Instance Block Cache Reads = <number>

Error Type:
Informational

Tags Read = <number>

Error Type:
Informational

Packets Sent = <number>

Error Type:
Informational

Packets Received = <number>

Error Type:
Informational

Initialization Transactions = <number>

Error Type:
Informational

Read/Write Transactions = <number>

Error Type:
Informational

Avg. Packets Sent/Second = <number>

Error Type:
Informational

Avg. Packets Received/Second = <number>

Error Type:
Informational

Avg. Tag Reads/Second = <number>

Error Type:

Informational

Avg. Tags/Transaction = <number>

Error Type:
Informational

Error Type:
Informational

%s | DEVICE STATISTICS

Error Type:
Informational

Average Device Turn-Around Time = <number> (milliseconds).

Error Type:
Informational

%s | CHANNEL STATISTICS

Error Type:
Informational

DRIVER STATISTICS

Error Type:
Informational

Details. | IP = '<address>', Vendor ID = <vendor>, Device type = <type>, Product code = <code>, Revision = <version>, Product name = '<name>', Product SN = <number>.

Error Type:
Informational

Errors occurred retrieving controller project.

Error Type:
Informational

Internal driver error occurred.

Error Type:
Informational

Invalid or corrupt controller project detected while synchronizing. Try again later.

Error Type:

Informational

Project download detected while synchronizing. Try again later.

Error Type:
Informational

Low memory resources.

Error Type:
Informational

Unknown error occurred.

Error Type:
Informational

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 Error Codes
CIP Error Codes

Encapsulation Error Codes

The following error codes are in hexadecimal.

Status (Hex)	Description
0001	Sender issued an invalid or unsupported encapsulation command.
0002	Insufficient memory resources available in the receiver to handle the command.
0003	Poorly formed or incomplete information in the data portion of the encapsulation message.
0004 - 0063	Reserved
0064	Originator used an invalid session handle when sending an encapsulation message to the target.
0065	Invalid length in header.
0066 - 0068	Reserved
0069	Requested protocol version is not supported.
006A - FFFF	Reserved

CIP Error Codes

The following error codes are in hexadecimal.

Status (Hex)	Description
01	Connection-related service failed along the connection path. *See Also:* ***0x01 Extended Error***

Status (Hex)	Description
	Codes
02	Resources needed for the object to perform the requested service were unavailable.
03	Invalid parameter value.
04	Path segment error. Tag does not exist in the device.
05	Path destination unknown. Structure member does not exist or array element is out of range.
06	Partial transfer; only part of the expected data was transferred.
07	Loss of connection.
08	Service not supported. The requested service was not implemented or was not defined for this class or instance.
09	Invalid attribute value.
0A	Attribute list error.
0B	Object is already in the mode/state requested by the service.
0C	Object cannot perform the requested service in its current mode/state. Project change may be in progress. *See Also: 0x0C Extended Error Codes*
0D	Requested instance of object to be created already exists.
0E	A request to modify a non-editable attribute was received.
0F	A permission / privilege check failed.
10	The device's current mode/state prohibits the execution of the requested service.
11	Reply data too large. The data to be transmitted in the response buffer is larger than the allocated response buffer.
12	The service specified an operation that would fragment a primitive data value.
13	Not enough data. The service did not supply enough data to perform the specified operation.
14	Attribute not supported.
15	Too much data. The service supplied more data than expected.
16	The object specified does not exist in the device.
17	The fragmentation sequence for this service is not currently active for this data.
18	The attribute data of this object was not saved prior to the requested service.
19	The attribute data of this object was not saved due to a failure during the attempt.
1A	Routing failure; request packet too large.
1B	Routing failure; response packet too large.
1C	Missing attribute in list entry data.
1D	Invalid attribute value list.
1E	Embedded service error. One or more services returned an error within a multiple-service packet service.
1F	Vendor-specific error. Consult vendor documentation. *See Also: 0x1F Extended Error Codes*
20	Invalid parameter. Parameter does not meet the requirements of the CIP specification or Omron specification. *See Also: 0x20 Extended Error Codes*
21	An attempt was made to write to a write-once medium that has already been written.
22	Invalid reply received. Reply service code does not match the request service code or reply message is shorter than the minimum expected reply size.

Status (Hex)	Description
23	The message received is larger than the receiving buffer can handle.
24	The format of the received message is not supported by the server.
25	The key segment included as the first segment in the path does not match the destination module.
26	The size of the path sent with the service request is not large enough to allow the request to be routed to an object or too much routing data was included.
27	Unexpected attribute in list.
28	The member ID specified in the request does not exist in the specified class, instance, or attribute.
29	A request to modify a non-modifiable member was received.
2A	DeviceNet-specific error.
2B	A CIP to Modbus translator received an unknown Modbus exception code.
2C	A request to read a non-readable attribute was received.
2D	A requested object instance cannot be deleted.
2E	The object supports the service, but not for the designated application path (for example, attribute).
2F - CF	Reserved by CIP.
D0 - FF	Object class specific errors.

0x01 Extended Error Codes

The following error codes are in hexadecimal.

Extended Status (Hex)	Description
0100	Connection in use or duplicate Forward Open request. Originator is trying to make a connection to a target with an established connection.
0101 - 0102	Reserved by CIP.
0103	Transport class and trigger combination specified is not supported by the target application.
0104	Reserved by CIP.
0105	See CIP Safety Specification.
0106	Ownership conflict. The connection cannot be established because another connection has exclusively allocated some of the resources required for this connection.
0107	Target connection not found. Typically returned in response to Forward Close request when the connection to be closed is not found at the target node.
0108	Invalid network connection parameter. Connection type, priority, or fixed/variable is not supported by the device.
0109	Invalid connection size. Target does not support the specified connection size.
010A -	Reserved by CIP.

Extended Status (Hex)	Description
010F	
0110	Target for connection not configured.
0111	RPI not supported. Returned if device cannot support the request (O->T or T->O RPI) or the connection timeout multiplier produces a timeout value that is not supported by the device.
0112	RPI value(s) not acceptable. Returned if the RPI value(s) in the Forward Open request are outside the range required by the application in the target device or the target is producing at a different interval.
0113	Out of connections. The maximum number of connections supported by the Connection Manager has been reached.
0114	Vendor ID or product code specified in the electronic key logical segment does not match the product code or vendor ID of the device.
0115	Device type specified in the electronic key logical segment does not match the device type of the device.
0116	The major and minor revision specified in the electronic key logical segment do not match the revision of the device.
0117	The produced or consumed application path specified in the connection path does not correspond to a valid produced or consumed application path within the target application.
0118	An application path specified for the configuration data does not correspond to a configuration application or is inconsistent with the consumed or produced application path.
0119	No non-listen only-connection types currently open. Returned when an attempt is made to establish a listen-only type to a target which has no non-listen-only connection already established.
011A	The maximum number of connections supported by this instance of the target object has been exceeded.
011B	The Production Inhibit Time is greater than the T->O RPI.
011C	Transport class requested in the transport type/trigger parameter is not supported.
011D	Production trigger requested in the transport type/trigger parameter is not supported.
011E	Direction requested in the transport type/trigger parameter is not supported.
011F	O->T fixed/variable flag is not supported.
0120	T->O fixed/variable flag is not supported.
0121	O->T Priority code is not supported.
0122	T->O Priority code is not supported.
0123	O->T Connection type is not supported.
0124	T->O Connection type is not supported.
0125	O->T Redundant owner flag is not supported.
0126	Data segment in the connection path parameter does not contain an acceptable number of 16 bit words for the configuration application path requested.
0127	Size of the consuming object declared in the Forward Open request and available on the target does not match the connection size declared in the O->T network connection parameter.
0128	Size of the producing object declared in the Forward Open request and available on the tar-

Extended Status (Hex)	Description
	get does not match the connection size declared in the T->O network connection parameter.
0129	Configuration application path specified in the connection path does not correspond to a valid configuration application path within the target application.
012A	Consumed application path specified in the connection path does not correspond to a valid consumed application path within the target application.
012B	Produced application path specified in the connection path does not correspond to a valid produced application path within the target application.
012C	Originator attempted to connect to a configuration tag name not in the list of tags defined in the target.
012D	Originator attempted to connect to a consuming tag name not in the list of tags defined in the target.
012E	Originator attempted to connect to a producing tag name not in the list of tags defined in the target.
012F	Combination of configuration, consume, and/or produce application paths specified in the connection path are inconsistent with each other.
0130	Information in the data segment is not consistent with the format of the consumed data.
0131	Information in the data segment is not consistent with the format of the produced data.
0132	Null Forward Open request is not supported by target.
0133	Connection timeout multiplier (inactivity watchdog) specified is reserved or produces a timeout value too large for the device.
0134 - 0202	Reserved by CIP.
0203	Connection timed out.
0204	Unconnected request timeout. This may be the result of congestion at the destination node or a node not being powered up or present.
0205	Connection tick time and connection timeout combination in unconnected request is not supported by an intermediate node.
0206	Message too large for unconnected_send service.
0207	Unconnected message acknowledge was received, but a data response message was not received.
0208 - 0300	Reserved by CIP.
0301	Insufficient connection buffer memory is available.
0302	Producer along path cannot allocate sufficient bandwidth for the connection on its link.
0303	No consumed connection ID filter available.
0304	Device unable to send scheduled priority data.
0305	Connection scheduling information in the originator device is not consistent with the connection scheduling information on the target network.
0306	Connection scheduling information in the originator device cannot be validated on the target network.

Extended Status (Hex)	Description
0307 - 0310	Reserved by CIP.
0311	A port specified in a port segment is not available or does not exist.
0312	Link address specified in a port segment is not valid for the target network type.
0313 - 0314	Reserved by CIP.
0315	Invalid segment type or segment value in the connection path.
0316	The connection path in the Forward Close service does not match the connection path in the connection being closed.
0317	Scheduled network segment is not present or the value within the segment is invalid.
0318	Link address to self (loopback) is invalid.
0319	Secondary system in dual-chassis redundant system is unable to duplicate connection request made to primary system.
031A	Request for a module connection has been refused because part of the corresponding data is already included in a rack connection.
031B	Request for a rack connection has been refused because part of the corresponding data is already included in a module connection.
031C	Miscellaneous connection-related error occurred.
031D	Redundant connection mismatch.
031E	The configured number of consumers for a producing application is already reached.
031F	No consumers configured for a producing application to use.
0320 - 07FF	Vendor-specific error.
0800	Network link in path to module is offline.
0801 - 080F	See CIP Safety Specification.
0810	Target application does not have valid data to produce for the requested connection.
0811	Originator application does not have valid data to produce for the requested connection.
0812	Node address has changed since the network was scheduled.
0813	A multicast connection has been requested between a producer and a consumer that are on different subnets and the producer is not configured for off-subnet multicast.
0814	Information in the data segment indicates that the format of the produced and/or consumed data is not valid.
0815 - FCFF	Reserved by CIP.

0x0C Extended Error Codes

The following error codes are in hexadecimal.

Extended Status (Hex)	Description
8010	A download is in progress.
8011	There is an error in tag memory.

For unlisted error codes, refer to the Omron documentation.

0x1F Extended Error Codes

The following error codes are in hexadecimal.

Extended Status (Hex)	Description
0101	Could be one of the following errors: • The combined size of the variable type and read address is incorrect. • The variable type specification is incorrect. • The read start address exceeds the range of the variable area. • The read end address exceeds the range of the variable area. • There are too many elements.
0102	Could be one of the following errors: • The number of elements does not match the size of the write data. • The variable type specification is not correct. • A read only area is included in the write area.
0104	A variable type is out of range.
8001	An internal error occurred.
800D	There is an error in the registered tag information.
8014	An internal error occurred.
8016	A variable is not correctly registered.

For unlisted error codes, refer to the Omron documentation.

0x20 Extended Error Codes

The following error code is in hexadecimal.

Extended Status (Hex)	Description
8017	More than one element was specified for a variable that does not have elements.
8018	Zero elements or data that exceeded the range of the array was specified for an array.
8022	The data type specified in the request service data does not agree with the tag information. The AddInfo Length in the request service data is not 0.
8028	Value is out of range.

For unlisted error codes, refer to the Omron documentation.

Index

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

W