GE SNPX Driver

Table of Contents

GE SNPX Driver

Help version 1.030

CONTENTS

Overview

The GE SNPX Driver provides a reliable way to connect GE SNPX controllers to OPC Client applications, including HMI, SCADA, Historian, MES, ERP and countless custom applications. This driver is intended for use with GE Programmable Logic Controllers.

Setup

Supported Devices

Series GE Micro
Series 90-30 311/313, 331/341, 350,360
Series 90-70 731/732, 771/772, 781/782
GE-OPEN Wide range model support

Communication Protocol

GE SNPX

Supported Communication Parameters

Baud Rate: 300, 600, 1200, 2400, 9600 and 19200
Parity: Odd, None
Data Bits: 8
Stop Bits: 1

Note: Not all devices support the listed configurations.

Device IDs

Series 90-30 PLCs support up to 6-character strings (such as 1 and Ge3).
Series 90-70 PLCs support up to 7-character strings (such as 1, Ge7 and Ge).

Note: For peer-to-peer communications an empty string is a valid Node ID.

Flow Control

When using an RS232/RS485 converter, the type of flow control that is required depends on the needs of the converter. Some converters do not require any flow control whereas others require RTS flow. Consult the converter's documentation to determine its flow requirements. An RS485 converter that provides automatic flow control is recommended.

Note: Peer-to-peer communications requires RTS-Always flow control.

Automatic Tag Database Generation

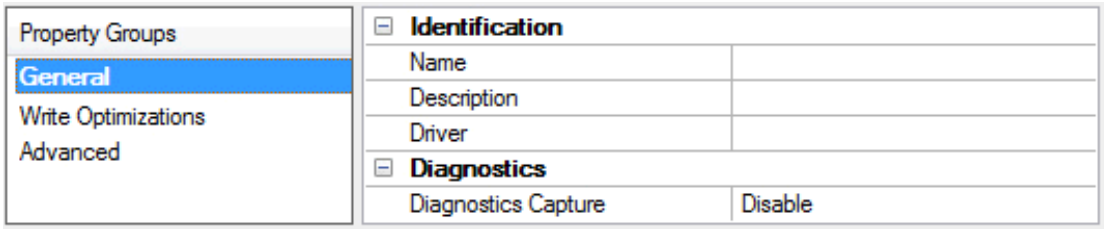
For more information, refer to **Variable Import Settings**.

Cabling

Follow the manufacturer's suggested cabling for the communications port and communications module.

Channel Properties — General

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



Identification

Name: User-defined identity of this channel. In each server project, each channel name must be unique. Although names can be up to 256 characters, some client applications have a limited display window when browsing the OPC server's tag space. The channel name is part of the OPC browser information. The property is required for creating a channel.
For information on reserved characters, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in the server help.

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

Driver: Selected protocol / driver for this channel. This property specifies the device driver that was selected during channel creation. It is a disabled setting in the channel properties. The property is required for creating a channel.
Note: With the server's online full-time operation, these properties can be changed at any time. This includes changing the channel name to prevent clients from registering data with the server. If a client has

already acquired an item from the server before the channel name is changed, the items are unaffected. If, after the channel name has been changed, the client application releases the item and attempts to re-acquire using the old channel name, the item is not accepted. With this in mind, changes to the properties should not be made once a large client application has been developed. Utilize the User Manager to prevent operators from changing properties and restrict access rights to server features.

Diagnostics

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

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

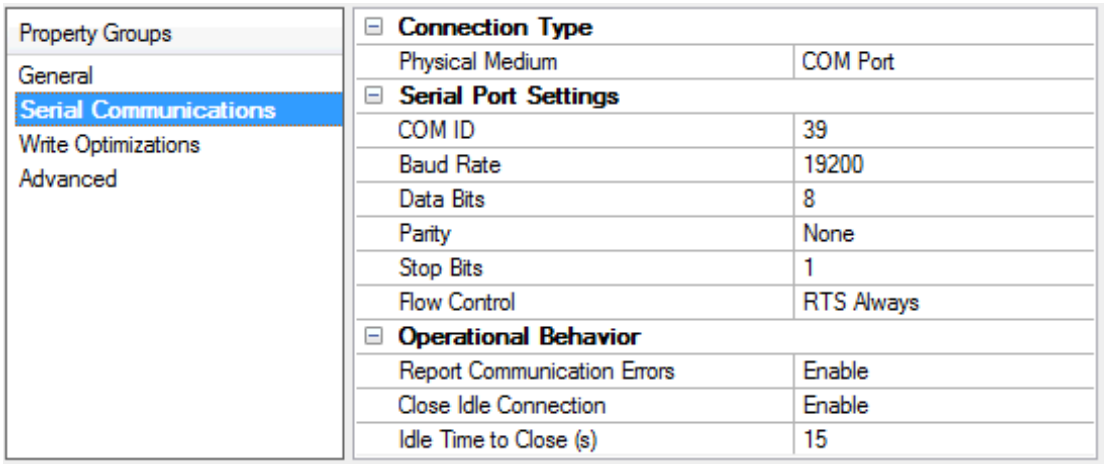
For more information, refer to "Communication Diagnostics" and "Statistics Tags" in the server help.

Channel Properties — Serial Communications

Serial communication properties are available to serial drivers and vary depending on the driver, connection type, and options selected. Below is a superset of the possible properties.

Click to jump to one of the sections: **Connection Type**, **Serial Port Settings** or **Ethernet Settings**, and **Operational Behavior**.

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



Connection Type

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

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

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

Serial Port Settings

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

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

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

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

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

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

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

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

Operational Behavior

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

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

Ethernet Settings

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

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

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

Modem Settings

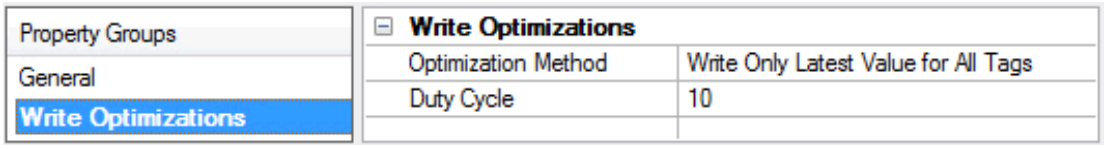
- **Modem**: Specify the installed modem to be used for communications.
- **Connect Timeout**: Specify the amount of time to wait for connections to be established before failing a read or write. The default is 60 seconds.
- **Modem Properties**: Configure the modem hardware. When clicked, it opens vendor-specific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

Operation with no Communications

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

Channel Properties — Write Optimizations

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



Write Optimizations

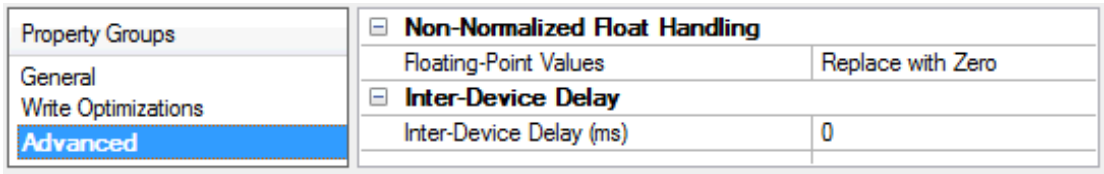
Optimization Method: Controls how write data is passed to the underlying communications driver. The options are:

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

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

Channel Properties — Advanced

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



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

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

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

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

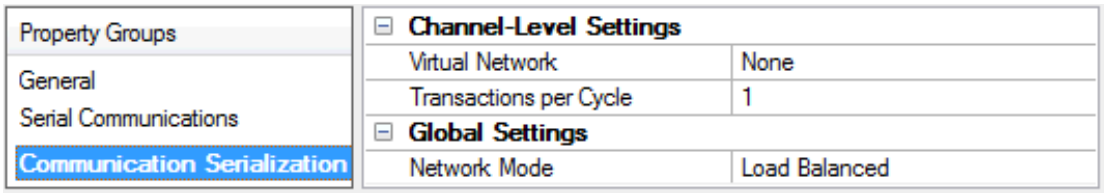
Inter-Device Delay: Specify the amount of time the communications channel waits to send new requests to the next device after data is received from the current device on the same channel. Zero (0) disables the delay.

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

Channel Properties — Communication Serialization

The server's multi-threading architecture allows channels to communicate with devices in parallel. Although this is efficient, communication can be serialized in cases with physical network restrictions (such as Ethernet radios). Communication serialization limits communication to one channel at a time within a virtual network.

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



Channel-Level Settings

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

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

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

Global Settings

- **Network Mode**: This property is used to control how channel communication is delegated. In **Load Balanced** mode, each channel is given the opportunity to communicate in turn, one at a time. In **Priority** mode, channels are given the opportunity to communicate according to the following rules (highest to lowest priority):
  - Channels with pending writes have the highest priority.
  - Channels with pending explicit reads (through internal plug-ins or external client interfaces) are prioritized based on the read's priority.
  - Scanned reads and other periodic events (driver specific).

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

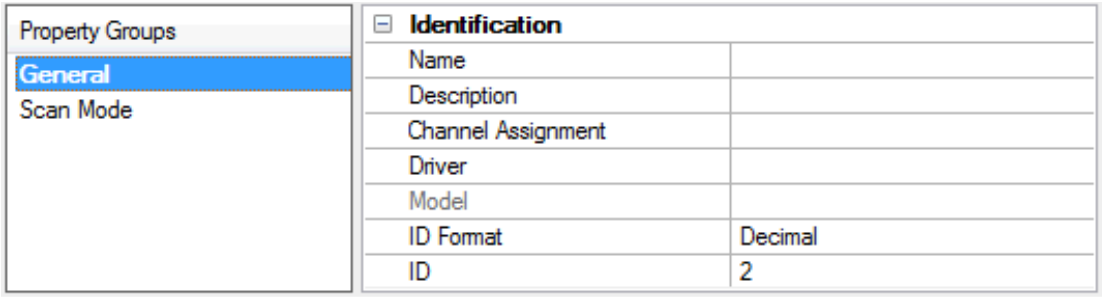
Devices that rely on unsolicited responses should not be placed in a virtual network. In situations where communications must be serialized, it is recommended that Auto-Demotion be enabled.

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

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

Device Properties — General

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



Identification

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

Note: Although descriptive names are generally a good idea, some OPC client applications may have a limited display window when browsing the OPC server's tag space. The device name and channel name become part of the browse tree information as well. Within an OPC client, the combination of channel name and device name would appear as "ChannelName.DeviceName".
For more information, refer to "How To... Properly Name a Channel, Device, Tag, and Tag Group" in server help.

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

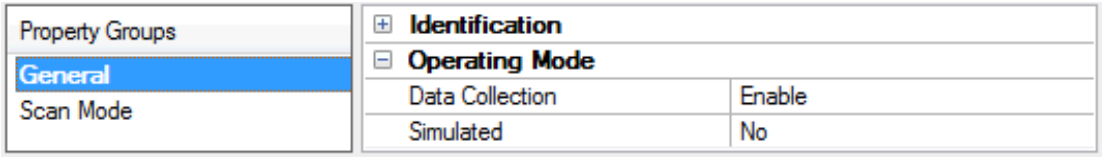
Channel Assignment: User-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

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

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

Operating Mode



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

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

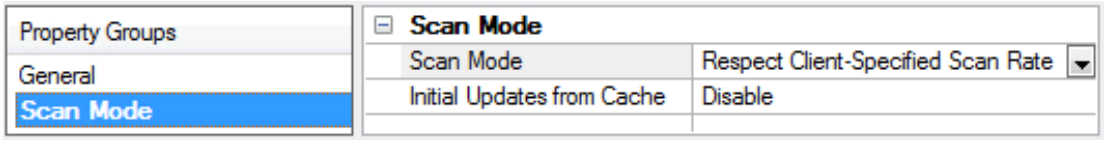
Notes:

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

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

Device Properties — Scan Mode

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



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

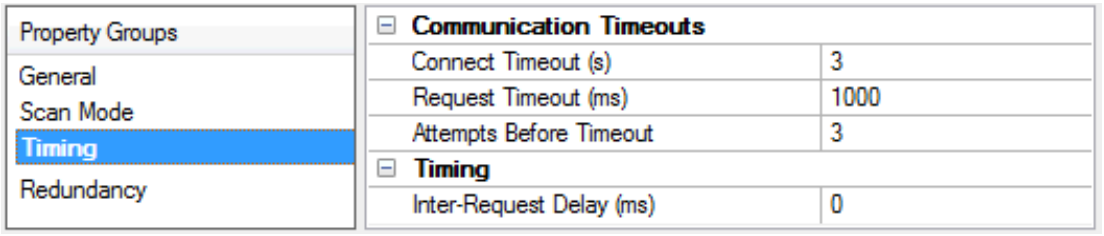
- **Respect Client-Specified Scan Rate**: This mode uses the scan rate requested by the client.
- **Request Data No Faster than Scan Rate**: This mode specifies the value set as the maximum scan rate. The valid range is 10 to 99999990 milliseconds. The default is 1000 milliseconds.
  Note: When the server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

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

device read is used for the initial update for the first client reference only. The default is disabled; any time a client activates a tag reference the server attempts to read the initial value from the device.

Device Properties — 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.

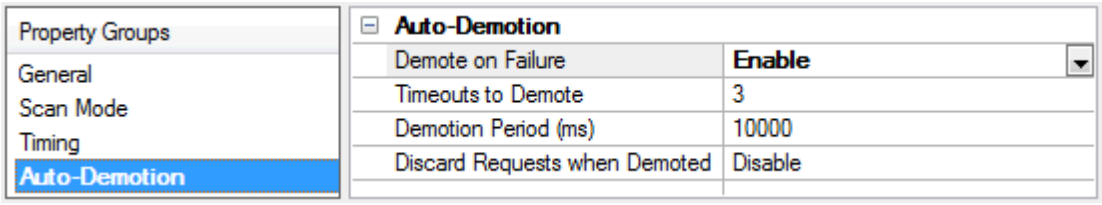
Timing

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.

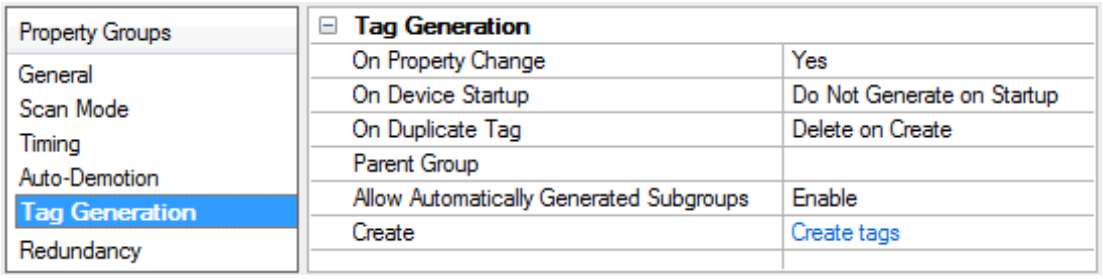
Not all devices and drivers support full automatic tag database generation and not all support the same data types. Consult the data types descriptions or the supported data type lists for each driver for specifics.

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 com-

munications 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.

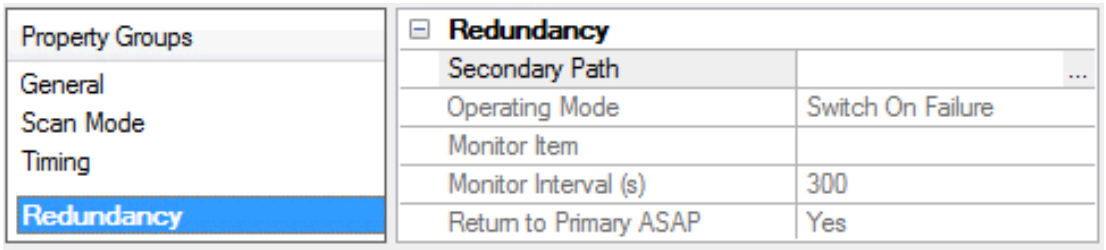
Note: Removing OPC tags affects tags that have been automatically generated by the communications driver as well as any tags that have been added using names that match generated tags. Users should avoid adding tags to the server using names that may match tags that are automatically generated by the 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.

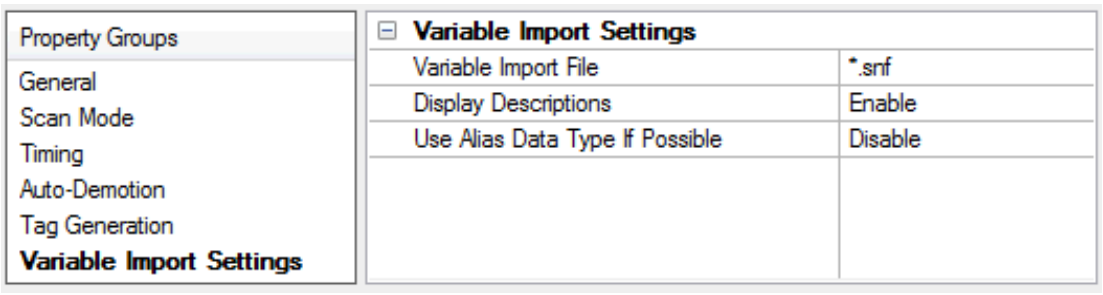
Create: Initiates the creation of automatically generated OPC tags. If the device's configuration has been modified, **Create tags** forces the driver to reevaluate the device for possible tag changes. Its ability to be accessed from the System tags allows a client application to initiate tag database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

Device Properties — Redundancy



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

Device Properties — Variable Import Settings



Variable Import File: This property is used to enter the exact location of the variable import file (*.snf or *.csv file extension) or Logic Developer variable import file (*.txt or other file extension) from which variables will be imported. It is this file that will be used when Automatic Tag Database Generation is instructed to create the tag database. All tags will be imported and expanded according to their respective data types.

Display Descriptions: When enabled, this option will import tag descriptions. If necessary, a description will be given to tags with long names stating the original tag name. Default setting is Enable.

Use Alias Data Type if Possible: When enabled, this option will use the data type assigned to an alias tag in the import file. If the alias data type is incompatible with the source tag data type, the source tag data type will be used instead. Default setting is Disable.

See Also: Automatic Tag Database Generation

Data Types Description

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8-bit value bit 0 is the low bit bit 7 is the high bit
Word	Unsigned 16-bit value bit 0 is the low bit bit 15 is the high bit
Short	Signed 16-bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
DWord	Unsigned 32-bit value bit 0 is the low bit bit 31 is the high bit
Long	Signed 32-bit value bit 0 is the low bit

Data Type	Description
	bit 30 is the high bit bit 31 is the sign bit
BCD	Two byte packed BCD Value range is 0-9999. Behavior is undefined for values beyond this range.
LBCD	Four byte packed BCD Value range is 0-99999999. Behavior is undefined for values beyond this range.
Float	32-bit floating point value The driver interprets two consecutive registers as a floating point value by making the second register the high word and the first register the low word.
String	Null terminated ASCII string Support includes HiLo LoHi byte order selection.
Double	64-bit floating point value

Address Descriptions

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

GE Micro
311
313
331
341
350
360
731
732
771
772
781
782
GE OPEN

Note: Each topic contains tables that display the ranges that are supported by the driver for that model; however, the actual range may vary depending on the configuration of the device.

See Also: Advanced Addressing

GE Micro

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R0001 to R9999 R0001 to R9998 R0001 to R9996	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI1024 AI0001 to AI1023 AI0001 to AI1021	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ256 AQ0001 to AQ255 AQ0001 to AQ253	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Array Support

The following data types support arrays: Byte, Word, Short, DWord, Long and Float. An array is a collection of contiguous elements of a given data type. The maximum array size is 32 DWords (Longs, Floats), 64 Words (Shorts) or 128 Bytes for a total of 1024 bits. For information on the two ways to specify an array, refer to the table below.

Examples

Address	Address Breakdown
G1 [4] includes the following byte addresses:*	G1, G9, G17, G25 1 row implied = 4 bytes 4 x 8 (byte) = 32 total bits
R16 [3][4] includes the following word addresses:	R16, R17, R18, R19 R20, R21, R22, R23 R24, R25, R26, R27

Address	Address Breakdown
	3 rows x 4 columns = 12 words 12 x 16 (word) = 192 total bits
P10 [5] includes the following word addresses:	P10, P11, P12, P13, P14 1 rows x 5 columns = 5 words 5 x 16 (word) = 80 total bits

*G25 indicates the fourth byte beginning at bit 25.

311 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R001 to R512 R001 to R511 R001 to R509	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI01 to AI64 AI01 to AI63 AI01 to AI61	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ001 to AQ032 AQ001 to AQ031 AQ001 to AQ029	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers
Array Support

313 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R001 to R1024 R001 to R1023 R001 to R1021	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI01 to AI64 AI01 to AI63 AI01 to AI61	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ001 to AQ032 AQ001 to AQ031 AQ001 to AQ029	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override
String Access to Registers
Array Support

331 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R0001 to R2048 R0001 to R2047 R0001 to R2045	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI001 to AI128 AI001 to AI127 AI001 to AI125	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ01 to AQ64 AQ01 to AQ63 AQ01 to AQ61	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers

Array Support

341 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write

Device Address	Range	Data Type	Access
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R0001 to R9999 R0001 to R9998 R0001 to R9996	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI1024 AI0001 to AI1023 AI0001 to AI1021	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ256 AQ0001 to AQ255 AQ0001 to AQ253	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers

Array Support

350 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write

Device Address	Range	Data Type	Access
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R0001 to R9999 R0001 to R9998 R0001 to R9996	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI2048 AI0001 to AI2047 AI0001 to AI2045	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ001 to AQ512 AQ001 to AQ511 AQ001 to AQ509	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers

Array Support

360 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S01 to S32 S01 to S25 (every 8th bit) S01 to S17 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only

Device Address	Range	Data Type	Access
Register References	R00001 to R32768 R00001 to R32767 R00001 to R32765	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI2048 AI0001 to AI2047 AI0001 to AI2045	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ001 to AQ512 AQ001 to AQ511 AQ001 to AQ509	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers

Array Support

731 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M2048 M0001 to M2041 (every 8th bit) M0001 to M2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383 R00001 to R16381	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

Device Address	Range	Data Type	Access
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override
String Access to Registers
Array Support

732 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M2048 M0001 to M2041 (every 8th bit) M0001 to M2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383 R00001 to R16381	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override
String Access to Registers
Array Support

771 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383 R00001 to R16381	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override
String Access to Registers
Array Support

772 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383 R00001 to R16381	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers

Array Support

781 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I00001 to I12288 I00001 to I12281 (every 8th bit) I00001 to I12273 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q00001 to Q12288 Q00001 to Q12281 (every 8th bit)	**Boolean*** Byte	Read/Write

Device Address	Range	Data Type	Access
	Q00001 to Q12273 (every 8th bit)	Word, Short, BCD	
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M12288 M00001 to M12281 (every 8th bit) M00001 to M12273 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383 R00001 to R16381	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override
String Access to Registers
Array Support

782 Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I00001 to I12288 I00001 to I12281 (every 8th bit) I00001 to I12273 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q00001 to Q12288 Q00001 to Q12281 (every 8th bit) Q00001 to Q12273 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M12288 M00001 to M12281 (every 8th bit)	**Boolean*** Byte	Read/Write

Device Address	Range	Data Type	Access
	M00001 to M12273 (every 8th bit)	Word, Short, BCD	
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383 R00001 to R16381	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override

String Access to Registers

Array Support

GE OPEN Addressing

The GE OPEN model selection has been provided to supply support for any GE SNPX compatible device that is not currently listed in the standard model selection menu. The data range has been expanded for each data type so that a wide range of PLCs may be addressed. Although the address ranges shown below may exceed the specific PLC's capability, the SNPX driver will respect all messages from the PLC regarding memory range limits. The default data types for dynamic tags are shown in **bold**.

Device Address	Range	Data Type	Access
Discrete Inputs	I0001 to I32768 I0001 to I32761 (every 8th bit) I0001 to I32753 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q00001 to Q32768 Q00001 to Q32761 (every 8th bit) Q00001 to Q32753 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Discrete Globals	G00001 to G32768 G00001 to G32761 (every 8th bit) G00001 to G32753 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M32768 M00001 to M32761 (every 8th bit) M00001 to M32753 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read/Write
Temporary Coils	T00001 to T32768	**Boolean***	Read/Write

Device Address	Range	Data Type	Access
	T00001 to T32761 (every 8th bit) T00001 to T32753 (every 8th bit)	Byte Word, Short, BCD	
Status References (Same for SA, SB, SC)	S00001 to S32768 S00001 to S32761 (every 8th bit) S00001 to S32753 (every 8th bit)	**Boolean*** Byte Word, Short, BCD	Read Only
Register References	R00001 to R32768 R00001 to R32767 R00001 to R32765	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI00001 to AI32768 AI00001 to AI32767 AI00001 to AI32765	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ00001 to AQ32768 AQ00001 to AQ32767 AQ00001 to AQ32765	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*When an array specification is given, the default data type Boolean becomes Byte.

Advanced Addressing

Default Data Type Override
String Access to Registers
Array Support

Advanced Addressing

Default Data Type Override

The default data types for each device type are shown in the table below. To override these defaults, append data type indicators to the device address. The possible data type indicators are as follows:

Indicators	Data Type
F	Float
S	Short
L	Long
M	String
(BCD)	BCD

Examples

Address	Description
R100 F	Access R100 as a floating point value
R300 L	Access R300 as a long
R400–R410 M	Access R400-R410 as a string with a length of 22 bytes. LoHi byte order is assumed.

Note: There must be a space between the register number and the data type indicator.

String Access to Registers

Register space can be accessed as string data by appending the "M" data type indicator. The length of the string is based on how the device address reference is entered. Each register addressed can contain 2 characters. The byte order of characters in registers can be specified by appending an optional "H" for HiLo or "L" for LoHi after the "M" data indicator. If no byte order is specified, LoHi order is assumed.

Examples

Address	Description
R100-R150 M	Access Register R100 as string with a length of 102 bytes. LoHi byte order is assumed.
R400 M	Access Register R400 as a string with a length of 4 bytes. LoHi byte order is assumed.
R405-R405 M	Access Register R405 as a string with a length of 2 bytes. LoHi byte order is assumed.
R100-R150 M H	Access Register R100 as string with a length of 102 bytes. HiLo byte order is explicitly specified.
R100-R150 M L	Access Register R100 as string with a length of 102 bytes. LoHi byte order is explicitly specified.

Note: The maximum string length is 128 bytes. For HiLo byte ordering, the string "AB" would be stored in a register as 0x4142. For LoHi byte ordering, the string "AB" would be stored in a register as 0x4241. There must be a space between the "M" data type indicator and the byte order indicator.

Array Support

An array is a collection of contiguous elements of a given data type. The maximum array size is 16 Doubles, 32 DWords (Longs, Floats), 64 Words (Shorts), or 128 Bytes for a total of 1024 bits. The following data types support arrays: Byte, Word, Short, DWord, Long, Float, and Double. For information on the two ways to specify an array, refer to the table below.

Address	Address Breakdown
G1 [4] includes the following byte addresses:*	G1, G9, G17, G25 1 row implied = 4 bytes 4 x 8 (byte) = 32 total bits
R16 [3][4] includes the following word addresses:	R16, R17, R18, R19 R20, R21, R22, R23 R24, R25, R26, R27 3 rows x 4 columns = 12 words 12 x 16 (word) = 192 total bits
P10 [5] includes the following word addresses:	P10, P11, P12, P13, P14 1 rows x 5 columns = 5 words 5 x 16 (word) = 80 total bits

*G25 indicates the fourth byte beginning at bit 25.

Automatic Tag Database Generation

The GE SNPX Device Driver generates its tags offline and is based on variables imported from a text file. It is offline in the sense that no connection to the device is required to generate tags. The text file (variables to import) can originate from one of the following applications:

1. Cimplicity Machine Edition - Logic Developer
2. VersaPro

There are two parts to Automatic Tag Database Generation: creating a variable import file from the application in use and generating tags based on the variable import file from the OPC server. For information on creating variable import files, refer to **Importing VersaPro Tags** or **Importing LogicDeveloper Tags**. For information on generating tags based on the import file, refer to **Variable Import Settings** and **Tag Generation**. It is recommended that users become familiar with this second part before starting the first part.

Generating Tag Database While Preserving Previously Generated Tag Databases

Under certain circumstances, multiple imports into the server are required to import all tags of interest. This is the case with importing VersaPro System variables and non-System variables into the same OPC server project. In the Tag Generation property group under Device Properties, click the property **On Duplicate Tag**. The options available are "Delete on create," "Overwrite as necessary," "Do not overwrite" and "Do not overwrite, log error. " After the first OPC server import/database creation is done, check that the action is set to "Do not overwrite" or "Do not overwrite, log error" for future imports. This will allow tags to be imported without deleting or overwriting tags that have been imported previously.

Tag Hierarchy

The tags created via automatic tag generation follow a specific hierarchy. The root level groups (or subgroup levels of the group specified in "Add generated tags to the following group") are determined by the variable addresses (such as R, G, M and so forth) referenced. For example, every variable that is of address type "R" will be placed in a root level group called "R". Each array tag is provided in its own subgroup of the parent group. The name of the array subgroup provides a description of the array. For example, an array R10[6] defined in the import file would have a subgroup name "R10_x"; x signifies dimension 1 exists.

Tags in "R10_x" Group

Tag Name	Tag Address	Comment
R10_x	R10[6]	Full array
R10_10	R10	Array element 1
R10_11	R11	Array element 2
R10_12	R12	Array element 3
R10_13	R13	Array element 4
R10_14	R14	Array element 5
R10_15	R15	Array element 6

Import File-to-Server Name Conversions

Leading Underscores, Percents, Pound, and Dollar Signs

Leading underscores (_) in tag names will be replaced with **U**_. This is required since the server does not accept tag/group names beginning with an underscore.

- Leading percents (**%**) in tag names will be replaced with **P_**. This is required since the server does not accept tag/group names beginning with a percent sign.
- Leading pound signs (**#**) in tag names will be replaced with **PD_**. This is required since the server does not accept tag/group names beginning with a pound sign.
- Leading dollar signs (**$**) in tag names will be replaced with **D_**. This is required since the server does not accept tag/group names beginning with a dollar sign.

Invalid Characters In Name

The only characters allowed in the server tag name are A-Z, a-z, 0-9, and underscore (_). As mentioned above, a tag name cannot begin with an underscore. All other invalid characters encountered will be converted to a sequence of characters that are valid. Below is a table showing the invalid character, and the sequence of characters that it is replaced with when encountered in the import file variable name.

Invalid Character	Replaced With
$	D_
%	P_
+	PL_
-	M_
#	PD_
@	A_
<	L_
>	G_
=	E_

Long Names

The GE SNPX Driver is limited to 256 character group and tag names. Therefore, if a tag name exceeds 256 characters, it must be clipped. Names are clipped as follows:

Non-Array

1. Determine a 5-digit Unique ID for this tag.
2. Given a tag name: ThisIsALongTagName...AndProbablyExceeds256
3. Clip tag at 256: ThisIsALongTagName...AndProbablyEx
4. Room is made for the Unique ID: ThisIsALongTagName...AndProba#####
5. Insert this ID: ThisIsALongTagName...AndProba00000

Array

1. Determine a 5-digit Unique ID for this array.
2. Given an array tag name: ThisIsALongTagName...AndProbablyExceeds256_23_45_8
3. Clip tag at 256 while holding on to the element values: ThisIsALongTagName...AndPr_23_45_8
4. Room is made for the Unique ID: ThisIsALongTagName...#####_23_45_8
5. Insert this ID: ThisIsALongTagName...00001_23_45_8

Importing VersaPro Tags

The device driver uses files generated from VersaPro called Shared Name Files (SNF) to generate the tag database. Certain aspects of the Automatic Tag Database Generation process are specific to the application from which variables are imported. The following topics are VersaPro specific. To import tags from an application other than VersaPro, refer to **Automatic Tag Database Generation** to see if the application is supported.

How do I create a VersaPro variable import file (*.SNF)?

See **VersaPro Import Preparation: VersaPro Steps**

How do I configure the OPC Server to use this import file for Automatic Tag Database Generation?

See **VersaPro Import Preparation: OPC Server Steps**

How do I import System Variables since they are not included with All Variables?

See **Generating Tag Database While Preserving Previously Generated Tag Databases**

How do I highlight variables in VersaPro?

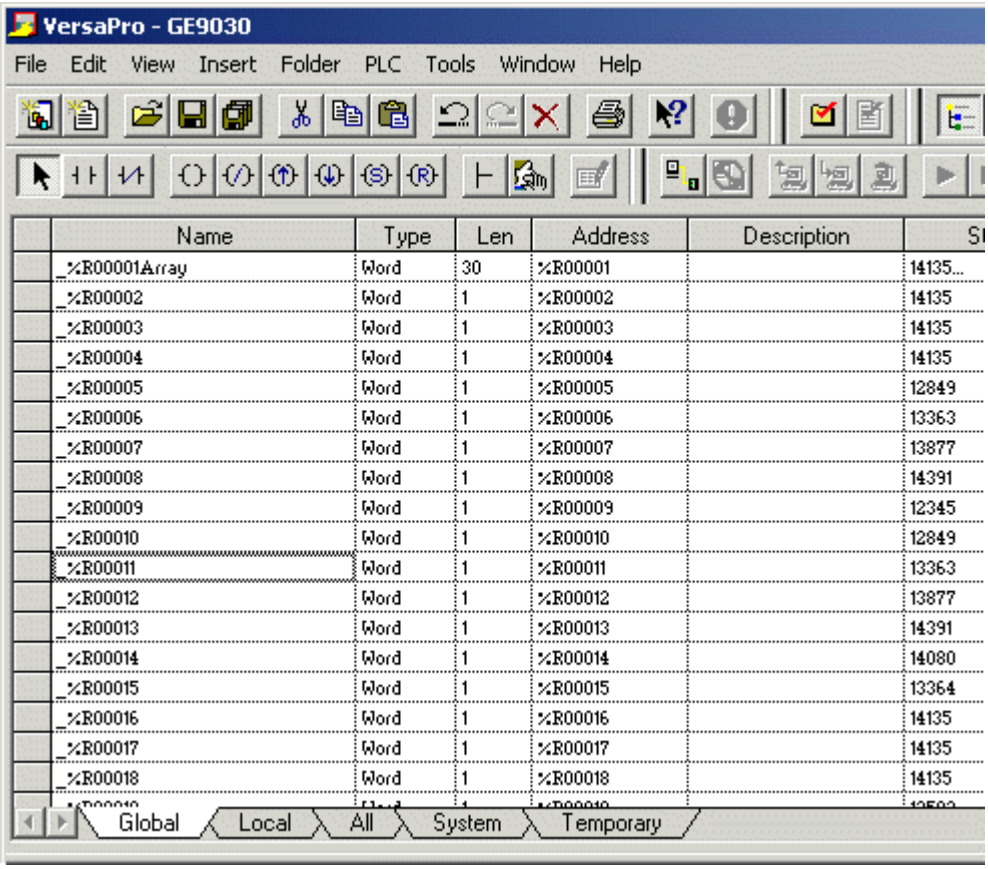
See **Highlighting VersaPro Variables**

How are VersaPro array variables imported?

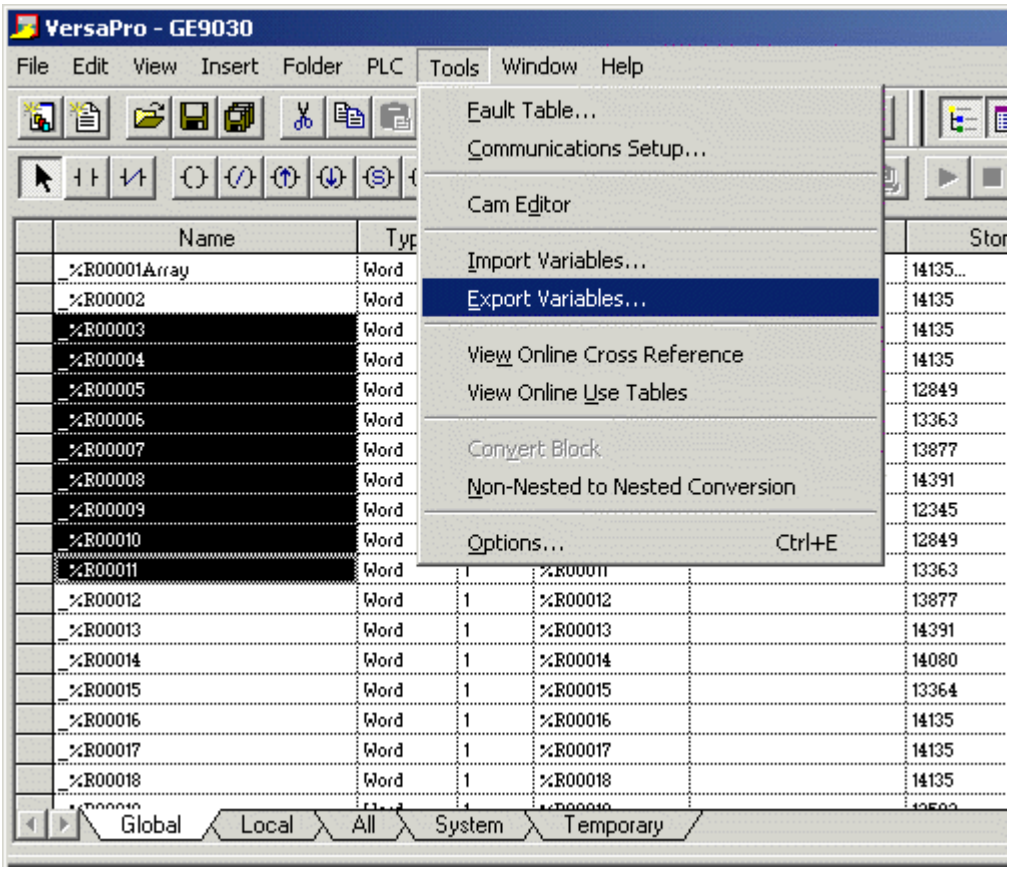
See **VersaPro Array Tag Import**

VersaPro Import Preparation: VersaPro Steps

1. To start, open the VersaPro project containing the tags (variables) that will be ported over to OPC server.
2. Open the **Variable Declaration Table** by clicking **View** | **Variable Declaration Table**.
3. Next, decide the Group to which the tags of interest belong. The default groups available are **Global**, **Local**, **All**, **System**and **Temporary**. The group All does not include the variables from the System group. In order for System and Global/Local/All variables to be imported, multiple imports are required (that is, multiple SNF files must be created).



4. Click on the group's tab to bring its variables to the front, and then highlight all tags of interest. Then click **Tools** | **Export Variables**. For more information, refer to **Highlighting VersaPro Variables**.



5. When prompted, select **Shared Name File** (*.snf) and then specify a name. VersaPro will export the project's contents into this SNF file.

VersaPro Import Preparation: OPC Server Steps

1. Open up the **Device Properties** for the device of interest for which tags will be generated.
2. Select the **Variable Import Settings** property group.
3. Enter or browse for the location of the newly created VersaPro *.snf file.
4. Select the **Tag Generation** property group and utilize as instructed above.
5. The OPC server will state in the Event Log that it is attempting to perform a tag import. When finished, it will state that the tag import has completed. All variable exported out of VersaPro will appear in the OPC server in the layout discussed in **Tag Hierarchy**.

See Also: **Variable Import Settings**

Highlighting VersaPro Variables

Variables can be highlighted in VersaPro using the actions.

Single Variable Selecting

Left-click on the variable of interest while pressing CTRL.

Pick-n-Choose
N/A.

Selecting a Range of Variables
Left-click on the first variable in the range of interest, and then press SHIFT and left-click on the last variable in the range. All variables in the range will be highlighted.

Selecting All Variables
Left-click on a variable within the group of interest in the **Variable Declaration Table**. The variable chosen is irrelevant. Next, click **Edit** | **Select All**. All variables will be highlighted within that target. All variables will be highlighted in that group.

VersaPro Array Tag Import
VersaPro variables have a length specification. Length is the number of elements for the given array variable. In the driver, this element count can be used to create tags in two ways. The first is to create an array tag with data in a row x column format. The second is an expanded group of tags, length in number. The following applies for variables with a length > 1.

Array Tags
Since VersaPro arrays are 1-dimensional, the number of columns is always 1. Thus, an array tag would have the following syntax: <*array variable*>*[#rows = Length]*. This single array tag would retrieve length elements starting at the base address defined in <array variable>. The data will come back formatted in array form for use in HMI's that support arrays.

Individual Elements
Element tags are simply the base address + element number. This has the following form, where n = Length - 1.

<array variable><base address + 0>
<array variable><base address + 1>
<array variable><base address + 2>
. . .
<array variable><base address + n>

These tags are not array tags; they are just the reference tags for the <array variable>. It may be thought of as a listing of all the addresses being referenced in the <array variable>.

Example
Variable Imported:
MyArrayTag, Length = 10, Address = R1

Result as Array Tag:
MyArrayTag [10]

Result as Individual Elements
R1
R2
R3

R4
R5
R6
R7
R8
R9
R10

Note: Variables of type BIT array cannot be accessed as an array tag, only as an expanded group of tags.

Importing LogicDeveloper Tags

The device driver uses user-generated ASCII text files from Logic Developer to generate the tag database. Certain aspects of the Automatic Tag Database Generation process are specific to the application from which variables are imported. The following topics are LogicDeveloper specific. To import tags from an application other than LogicDeveloper, refer to **Automatic Tag Database Generation** to see if the application is supported.

How do I create a LogicDeveloper variable import file (*.TXT)?

See **LogicDeveloper Import Preparation: LogicDeveloper Steps**

How do I configure the OPC server to use this import file for Automatic Tag Database Generation?

See **LogicDeveloper Import Preparation: OPC Server Steps**

How do I highlight variables in LogicDeveloper?

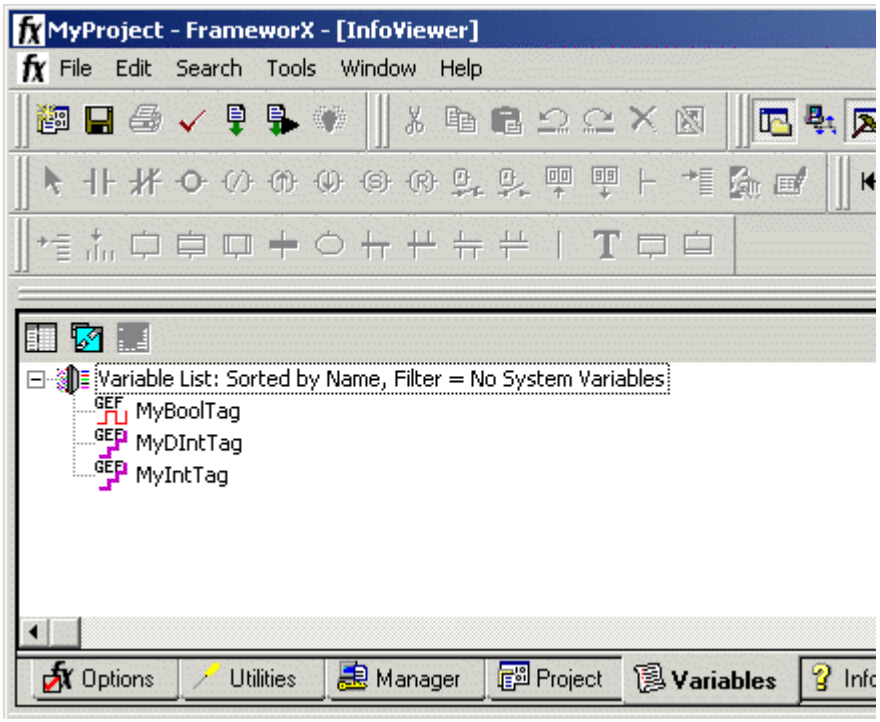
See **Highlighting LogicDeveloper Variables**

How are LogicDeveloper array variables imported?

See **LogicDeveloper Array Tag Import**

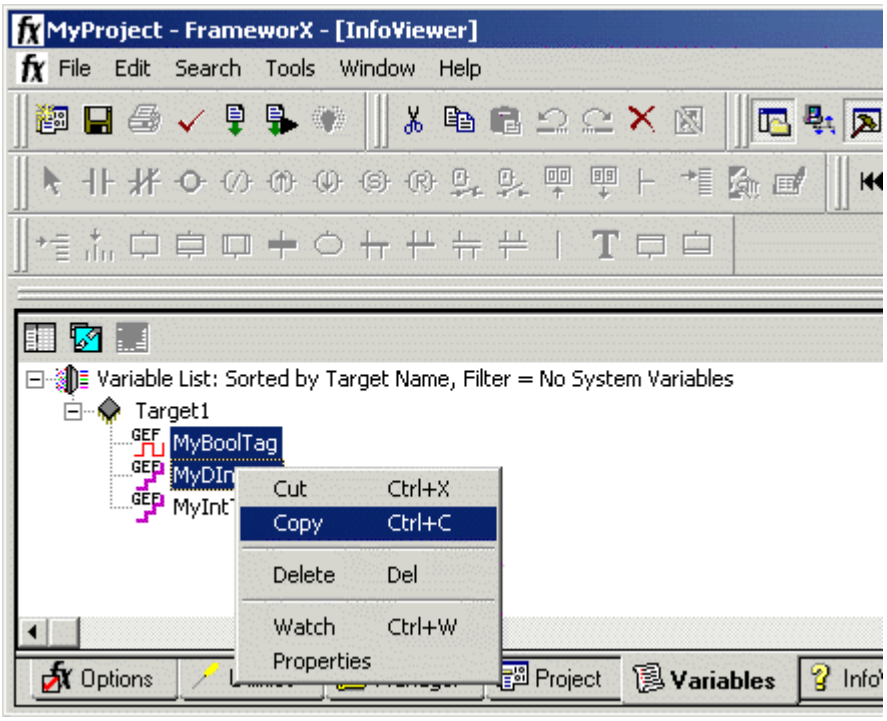
LogicDeveloper Import Preparation: LogicDeveloper Steps

1. To start, open the FrameworX project containing the tags (variables) that will be ported over to OPC server.
2. Open the **Navigator** window (Shift-F4) if it's not already open.
3. Click on **Variables** and select **Variable List View**.

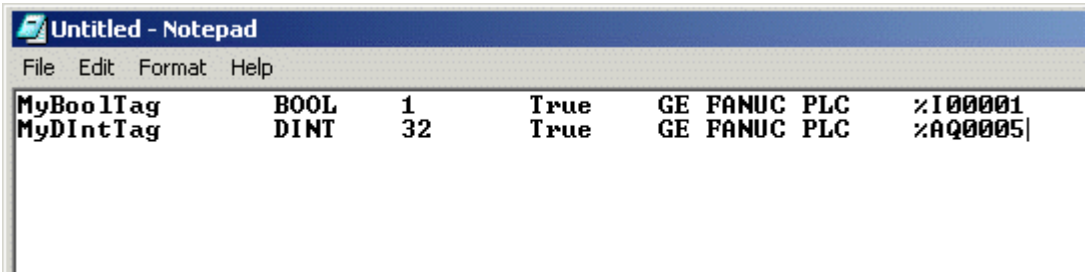


Important: Each FrameworX project contains one or more targets. A target is essentially the device on which the application will run. Variables are created on the target-level, so in order to specify the variables to export, the target of interest must first be decided. In order for the target variables to be imported, the variables must have GE PLC as the Data Source. This can be verified by left-clicking on the variable and looking at its Data Source property in the Inspector window. Internal variables will not be imported.

4. Sort the variables by target. To do so, right-click on the Variable List header and select **Sort** | **Target**.
5. Highlight the tags (variables) of interest in the target of interest. For more information, refer to **Highlighting LogicDeveloper Variables**.
6. Next, click **Edit** | **Copy**.



7. Open a word processing program such as Notepad or Wordpad. This will be the place where the variables will ultimately be saved for importing. Click **Edit** | **Paste**.
8. The variables on the Clipboard will now be pasted to the document, TAB delimited. Do not modify the contents. Modifications may cause the import to fail.



9. Save this text document with the TXT extension (*.txt) in ANSI form.
10. The variables highlighted and copied in Logic Developer are now contained within this text document to be imported into the OPC server.

LogicDeveloper Import Preparation: OPC Server Steps

1. Open up the **Device Properties** for the device of interest for which tags will be generated.
2. Select the **Variable Import Settings** property group.
3. Enter or browse for the location of the newly created Logic Developer *.txt file.
4. Select the **Tag Generation** property group and utilize as instructed above.

5. The OPC server will state in the Event Log that it is attempting to perform a tag import. When finished, it will state that the tag import has completed. All variables exported out of Logic Developer will appear in the OPC server in the layout discussed in **Tag Hierarchy**.

See Also: **Variable Import Settings**.

Highlighting LogicDeveloper Variables

Variables can be highlighted in Logic Developer using the following actions.

Single Variable Selecting

Left-click on the variable of interest.

Pick-n-Choose

Left-click on the first variable of interest, and then press CTRL and left-click on each successive variable of interest. Repeat until all variables of interest are highlighted.

Selecting a Range of Variables

Left-click on the first variable in the range of interest, and then press SHIFT and left-click on the last variable in the range. All variables in the range will be highlighted.

Selecting All Variables

Left-click on a variable within the target of interest in the **Variable List View**. The variable chosen is irrelevant. Next, click **Edit** | **Select All**. All variables will be highlighted within that target.

LogicDeveloper Array Tag Import

Array tags (or individual element breakdowns of array variables) are not supported when importing from Logic Developer.

Importing Proficy Logic Developer Tags

This driver uses import files from Proficy Logic Developer to generate the tag database. Certain aspects of the Automatic Tag Database Generation process are specific to the application from which variables are imported. The following topics are specific to Proficy Logic Developer. To import tags from an application other than Proficy Logic Developer, refer to **Automatic Tag Database Generation** to see if the application is supported.

Note: This driver supports the importing of structured data types.

How do I create a Proficy Logic Developer variable import file (*.snf or *.csv)?

See **Proficy Logic Developer Import Preparation: Logic Developer Steps**

How do I configure the OPC Server to use this import file for Automatic Tag Database Generation?

See **Proficy Logic Developer Import Preparation: OPC Server Steps**

How do I highlight variables in Proficy Logic Developer?

See **Highlighting Proficy Logic Developer Variables**

How are Proficy Logic Developer array variables imported?

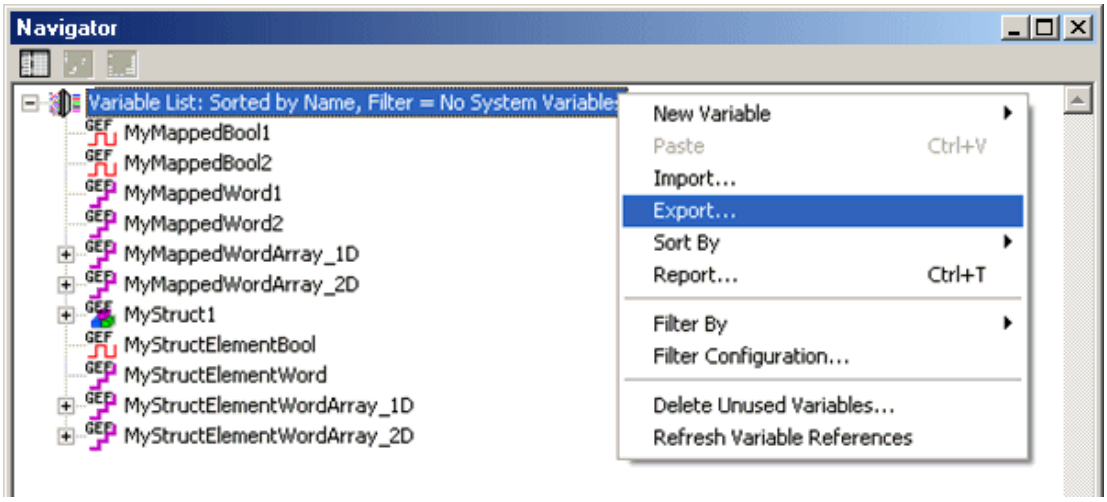
See **Proficy Logic Developer Array Tag Import**

Proficy Logic Developer Import Preparation: Logic Developer Steps

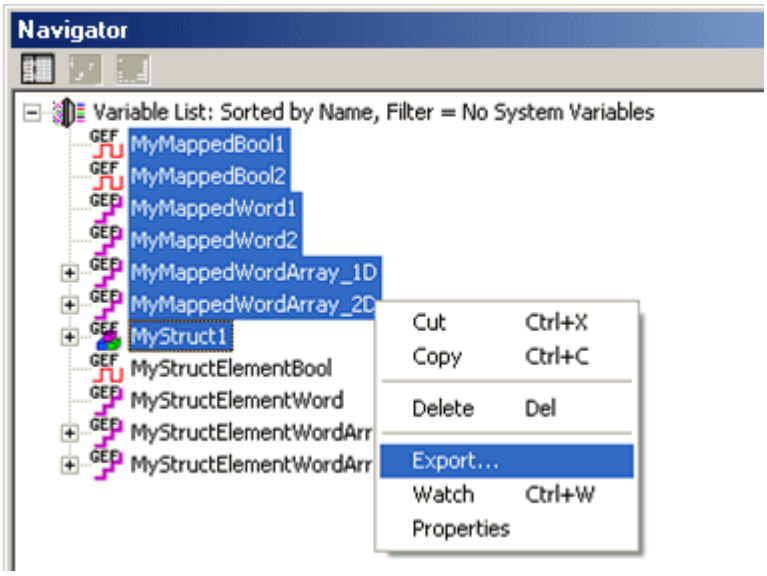
1. To start, open the Proficy Logic Developer project containing the tags (variables) that will be ported over to the OPC server.
2. Open the **Navigator** and select the **Variables** tab.
3. In the **Variable List View**, sort the variables by target. To do so, right-click on the Variable List header and then click **Sort** | **Target**.

   Important: Each Logic Developer project contains one or more targets. A target is essentially the device on which the application will run. Variables are created on the target-level, so in order to specify the variables to export, the target of interest must first be defined. In order for the target variables to be imported, the variables must have GE PLC as the Data Source. This can be verified by left-clicking on the variable and looking at its Data Source properties in the Inspector window. Internal variables will not be imported.

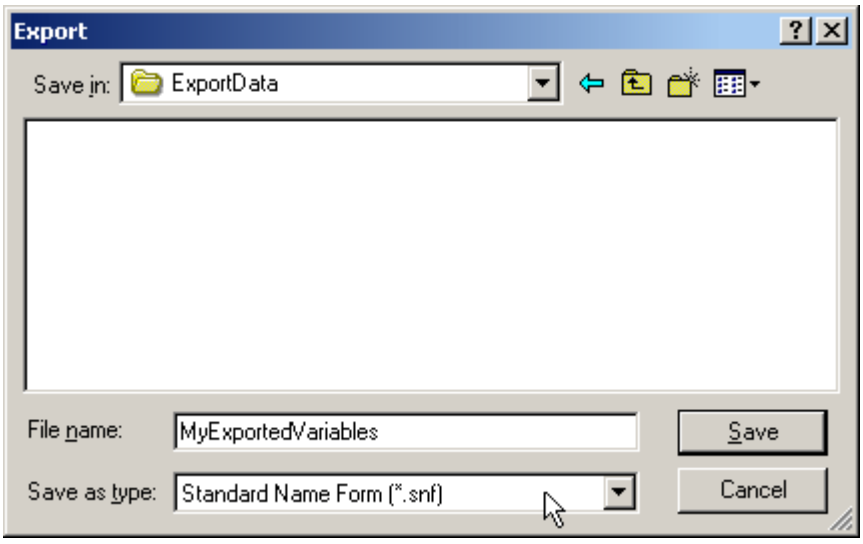
4. To export all of the variables, right-click on the Variable List tree root and click the **Export...** pop-up menu option.



5. To export only selected variables, highlight the tags of interest in the target of interest. Then, right-click on one of the selected variables and click **Export...**.



6. In the **Save as Type** drop-down list, choose between **Standard Name Form** (*.snf) or **Comma Separated Variable** (*.csv) as the export file type.



Proficy Logic Developer Import Preparation: OPC Server Steps

1. Open up the **Device Properties** for the device of interest for which tags will be generated.
2. Select the **Variable Import Settings** property group.
3. Enter or browse for the location of the newly created *.snf or *.csv export file.
4. Select the **Tag Generation** property group and click **Create Tags** in order to import variables now. Other settings on that page may be used to automatically create the database later.
5. The OPC server will state in the Event Log that it is attempting to perform a tag import. When finished, it will state that the tag import has completed. All variables exported out of Logic Developer will appear in the OPC server in the layout discussed in Tag Hierarchy.

See Also: Variable Import Settings

Highlighting Proficy Logic Developer Variables

Variables can be highlighted in Logic Developer using the following actions.

Single Variable Selecting

Left-click on a variable of interest.

Pick-n-Choose

Left-click on the first variable of interest, and then press CTRL and left-click on each successive variable of interest. Repeat until all variables of interest are highlighted.

Selecting a Range of Variables

Left-click on the first variable in the range of interest, and then press SHIFT and left-click on the last variable in the range. All variables in the range will be highlighted.

Selecting All Variables

Left-click on a variable within the target of interest in the **Variable List View**. The variable chosen is irrelevant. Next, click **Edit** | **Select All**. All variables will be highlighted within that target.

Proficy Logic Developer Array Tag Import

Arrays of referenced variables and arrays of symbolic variables will be imported differently.

Referenced Variable Arrays

Arrays of referenced variables will be imported as described in VersaPro Array Tag Import. A group will be created for each array. Each group will contain a single array tag, plus a number of tags addressing the individual array elements.

Symbolic Variable Arrays

A single array tag will be generated for each symbolic variable array in the import file. All symbolic variable array tags will be places in the Symbolic group along with all other symbolic variable tags. The driver will not generate tags for BOOL and STRING symbolic variable arrays.

Error Descriptions

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

Address Validation

Missing address.
Device address '<address>' contains a syntax error.
Address '<address>' is out of range for the specified device or register.
Device address '<address>' is not supported by model '<model name>'.
Data Type '<type>' is not valid for device address '<address>'.
Device address '<address>' is read only.
Array size is out of range for address '<address>'.
Array support is not available for the specified address: '<address>'.

Serial Communications

COMn does not exist.
Error opening COMn.
COMn is in use by another application.
Unable to set comm parameters on COMn.
Communications error on '<channel name>' [<error mask>].

Device Status Messages

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

Device Specific Messages

Invalid tag in block starting at <address> on device <device name>. Block deactivated.

Write Error Messages

Unable to write to tag '<tag address>' on device '<device name>'. The service requested is either not defined or not supported.
Unable to write to tag '<tag address>' on device '<device name>'. The user does not have sufficient privileges to process the request. Minor status error code = '<hexadecimal error code>'.
Unable to write to tag '<tag address>' on device '<device name>'. The CPU has received a message that is out of order.
Unable to write to tag '<tag address>' on device '<device name>'. Service request error. Minor status error code = '<hexadecimal error code>'.
Unable to write to tag '<tag address>' on device '<device name>'. The PLC CPU's service request queue is full: please wait a minimum of 10 ms before sending another service request.
Unable to write to tag '<tag address>' on device '<device name>'. A framing error has occurred.
Unable to write to tag '<tag address>' on device '<device name>'. Device returned major error code <hexadecimal error code> and minor error code <hexadecimal error code>.

Blocked Reads Error Messages

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The service requested is either not defined or not supported.
Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The user does not have sufficient privileges to process the request. Minor status error code = '<hexadecimal error code>'.
Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The CPU has received a message that is out of order.
Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. Service request error. Minor status error code = '<hexadecimal error code>'.

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The PLC CPU's service request queue is full: please wait a minimum of 10 ms before sending another service request.
Unable to read '<byte count>' bytes starting at address '<start tag>' on device '<device name>'. A framing error has occurred.
Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. Device returned major error code <hexadecimal error code> and minor error code <hexadecimal error code>.

Non-Blocked Reads Error Messages

Unable to read tag '<tag address>' on device '<device name>'. The service requested is either not defined or not supported.
Unable to read tag '<tag address>' on device '<device name>'. The user does not have sufficient privileges to process the request. Minor status error code = '<hexadecimal error code>'.
Unable to read tag '<tag address>' on device '<device name>'. The CPU has received a message that is out of order.
Unable to read tag '<tag address>' on device '<device name>'. Service request error. Minor status error code = '<hexadecimal error code>'.
Unable to read tag '<tag address>' on device '<device name>'. The PLC CPU's service request queue is full: please wait a minimum of 10 ms before sending another service request.
Unable to read tag '<tag address>' on device '<device name>'. A framing error has occurred.
Unable to read tag '<tag address>' on device '<device name>'. Device returned major error code <hexadecimal error code> and minor error code <hexadecimal error code>.

Automatic Tag Database Generation Messages

Unable to generate a tag database for device <device name>. Reason: Low memory resources.
Unable to generate a tag database for device <device name>. Reason: Import file is invalid or corrupt.
Database Error: Tag '<orig. tag name>' exceeds 256 characters. Tag renamed to '<new tag name>'.
Database Error: Array tags '<orig. tag name><dimensions>' exceed 256 characters. Tags renamed to '<new tag name><dimensions>'.
Database Error: Data type '<type>' for tag '<tag name>' not found in import file. Setting to Default Type '<type>'.
Database Error: Data type '<type>' for tag '<tag name>' is currently not supported. Tag not created.
Database Error: Logic Developer Variable Arrays are currently not supported. Array Tag (s) '<array tag name>' not created.
Database Error: No Reference Address found for tag '<tag name>' in import file. Tag not created.

Database Error: Only variables with Data Source '<data source name>' are imported. Data Source '<data source name>' is not supported. Tag '<tag name>' not created. Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created.

Missing address

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically has no length.

Solution:
Re-enter the address in the client application.

Device address '<address>' contains a syntax error

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically contains one or more invalid characters.

Solution:
Re-enter the address in the client application.

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

Error Type:
Warning

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

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

Device address '<address>' is not supported by model '<model name>'

Error Type:
Warning

Possible Cause:
A tag address that has been specified statically references a location that is valid for the communications protocol but not supported by the target device.

Solution:
Verify that the address is correct; if it is not, re-enter it in the client application. Also verify that the selected model name for the device is correct.

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

Error Type:
Warning

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

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

Device address '<address>' is Read Only

Error Type:
Warning

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

Solution:
Change the access mode in the client application.

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

Error Type:
Warning

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

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

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

Error Type:
Warning

Possible Cause:

A tag address that has been specified statically contains an array reference for an address type that doesn't support arrays.

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

COMn does not exist

Error Type:
Fatal

Possible Cause:
The specified COM port is not present on the target computer.

Solution:
Verify that the proper COM port has been selected.

Error opening COMn

Error Type:
Fatal

Possible Cause:
The specified COM port could not be opened due an internal hardware or software problem on the target computer.

Solution:
Verify that the COM port is functional and may be accessed by other Windows applications.

COMn is in use by another application

Error Type:
Fatal

Possible Cause:
The serial port assigned to a device is being used by another application.

Solution:
Verify that the correct port has been assigned to the channel.

Unable to set comm parameters on COMn

Error Type:
Fatal

Possible Cause:
The serial parameters for the specified COM port are not valid.

Solution:
Verify the serial parameters and make any necessary changes.

Communications error on '<channel name>' [<error mask>]

Error Type:
Serious

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

Possible Cause:
1. The serial connection between the device and the Host PC is bad.
2. The communications parameters for the serial connection are incorrect.

Solution:
1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications parameters match those of the device.

Device '<device name>' not responding

Error Type:
Serious

Possible Cause:
1. The serial connection between the device and the Host PC is broken.
2. The communications parameters for the serial connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.
4. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

Solution:
1. Verify the cabling between the PC and the PLC device.
2. Verify the specified communications parameters match those of the device.

3. Verify that the Network ID given to the named device matches that of the actual device.
4. Increase the Request Timeout setting so that the entire response can be handled.

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

Error Type:
Serious

Possible Cause:

1. The serial connection between the device and the Host PC is broken.
2. The communications parameters for the serial connection are incorrect.
3. The named device may have been assigned an incorrect Network ID.

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify the specified communications parameters match those of the device.
3. Verify that the Network ID given to the named device matches that of the actual device.

Invalid tag in block starting at <address> on device <device name>. Block deactivated

Error Type:
Serious

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

Solution:
Verify the tags assigned to addresses in the specified range on the device and eliminate ones that reference invalid locations.

Unable to write to tag '<tag address>' on device '<device name>'. The service requested is either not defined or not supported

Error Type:
Warning

Possible Cause:
The requested service is either not defined or not supported.

Solution:

1. Determine whether writes are supported for the tag address.
2. Verify that the device has the latest Firmware revision.

Unable to write to tag '<tag address>' on device '<device name>'. The user does not have sufficient privileges to process the request. Minor status error code = '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
The user does not have sufficient privileges to complete the service request.

Solution:
For the privilege level required to complete the service request, refer to the Minor Status field.

Unable to write to tag '<tag address>' on device '<device name>'. The CPU has received a message that is out of order

Error Type:
Warning

Possible Cause:
The CPU received a malformed message.

Solution:
Resend the write request. This will automatically resend the polled tags, as well.

Unable to write to tag '<tag address>' on device '<device name>'. Service request error. Minor status error code = '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
The CPU cannot process the service request correctly.

Solution:
For the specific error code, refer to the Minor Status field.

Unable to write to tag '<tag address>' on device '<device name>'. The PLC CPU's service request queue is full: please wait a minimum of 10 ms before sending another service request

Error Type:

Warning

Possible Cause:
The PLC CPU's service request queue is full.

Solution:
Resend the write request at a later time. This will automatically resend the polled tags, as well.

Unable to write to tag '<tag address>' on device '<device name>'. A framing error has occurred

Error Type:
Warning

Possible Cause:

1. The packets are misaligned due to the connection between the PC and the device.
2. Bad cabling connects the device and is causing noise.

Solution:

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

Unable to write to tag '<tag address>' on device '<device name>'. Device returned major error code '<hexadecimal error code>' and minor error code '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
An unknown error has occurred.

Solution:
For the meaning of the error code, refer to the manufacturer's documentation.

Unable to read ' <number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The service requested is either not defined or not supported

Error Type:
Warning

Possible Cause:

The requested service is either not defined or not supported.

Solution:

1. Determine whether reads are supported for the tag address.
2. Verify that the device has the latest Firmware revision.

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The user does not have sufficient privileges to process the request. Minor status error code = '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
The user does not have sufficient privileges to complete the service request.

Solution:
For the privilege level required to complete the service request, refer to the Minor Status field.

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The CPU has received a message that is out of order

Error Type:
Warning

Possible Cause:
The CPU received a malformed message.

Solution:
Resend the read request. This will automatically resend the polled tags, as well.

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. Service request error. Minor status error code = '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
The CPU cannot process the service request correctly.

Solution:

For the specific error code, refer to the Minor Status field.

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. The PLC CPU's service request queue is full: please wait a minimum of 10 ms before sending another service request

Error Type:
Warning

Possible Cause:
The PLC CPU's service request queue is full.

Solution:
Resend the read request at a later time. This will automatically resend the polled tags, as well.

Unable to read '<byte count>' bytes starting at address '<start tag>' on device '<device name>'. A framing error has occurred

Error Type:
Warning

Possible Cause:

1. The packets are misaligned due to the connection between the PC and the device.
2. Bad cabling connects the device and is causing noise.

Solution:

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

Unable to read '<number of bytes>' bytes starting at address '<tag address>' on device '<device name>'. Device returned major error code '<hexadecimal error code>' and minor error code '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
An unknown error has occurred.

Solution:
For the meaning of the error code, refer to the manufacturer's documentation.

Unable to read tag '<tag address>' on device '<device name>'. The service requested is either not defined or not supported

Error Type:
Warning

Possible Cause:
The requested service is either not defined or not supported.

Solution:

1. Determine whether reads are supported for the tag address.
2. Verify that the device has the latest Firmware revision.

Unable to read tag '<tag address>' on device '<device name>'. The user does not have sufficient privileges to process the request. Minor status error code = '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
The user does not have sufficient privileges to complete the service request.

Solution:
For the privilege level required to complete the service request, refer to the Minor Status field.

Unable to read tag '<tag address>' on device '<device name>'. The CPU has received a message that is out of order

Error Type:
Warning

Possible Cause:
The CPU received a malformed message.

Solution:
Resend the read request. This will automatically resend the polled tags, as well.

Unable to read tag '<tag address>' on device '<device name>'. Service request error. Minor status error code = '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
The CPU cannot process the service request correctly.

Solution:
For the specific error code, refer to the Minor Status field.

Unable to read tag '<tag address>' on device '<device name>'. The PLC CPU's service request queue is full: please wait a minimum of 10 ms before sending another service request

Error Type:
Warning

Possible Cause:
The PLC CPU's service request queue is full.

Solution:
Resend the read request at a later time. This will automatically resend the polled tags, as well.

Unable to read tag '<tag address>' on device '<device name>'. A framing error has occurred

Error Type:
Warning

Possible Cause:

1. The packets are misaligned due to the connection between the PC and the device.
2. Bad cabling connects the device and is causing noise.

Solution:

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

Unable to read tag '<tag address>' on device '<device name>'. Device returned major error code '<hexadecimal error code>' and minor error code '<hexadecimal error code>'

Error Type:
Warning

Possible Cause:
An unknown error has occurred.

Solution:
For the meaning of the error code, refer to the manufacturer's documentation.

Unable to generate a tag database for device <device name>. Reason: Low memory resources

Error Type:
Warning

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

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

Unable to generate a tag database for device <device name>. Reason: Import file is invalid or corrupt

Error Type:
Warning

Possible Cause:

1. The file specified as the Tag Import File (*.snf or *.csv) is corrupt.
2. The file specified as the Tag Import File is an improperly formatted Logic Developer text file.

Solution:
Select a valid, properly formatted VersaPro/Logic Developer variable import file or retry the tag export process in the respective application to produce a new import file.

See Also:
Automatic Tag Database Generation

Database Error: Tag '<orig. tag name>' exceeds 256 characters. Tag renamed to '<new tag name>'

Error Type:
Warning

Possible Cause:
The name assigned to a tag originates from the variable name in the import file. This name exceeds the 256-character limitation and will be renamed to one that is valid.

Solution:
None.

See Also:
Import File-to-Server Name Conversions

Database Error: Array tags '<orig. tag name><dimensions>' exceed 256 characters. Tags renamed to '<new tag name><dimensions>'

Error Type:
Warning

Possible Cause:
The name assigned to an array tag originates from the variable name in the import file. This name exceeds the 256-character limitation and will be renamed to one that is valid. <Dimensions> define the number of dimensions for the given array tag: XXX for 1 dimension, XXX_YYY for 2. The number of Xs and Ys approximates the number of elements for the respective dimensions. Since such an error will occur for each element, generalizing with XXX and YYY implies all array elements will be affected.

Solution:
None.

See Also:
Import File-to-Server Name Conversions

Database Error: Datatype '<type>' for tag '<tag name>' is currently not supported. Tag not created

Error Type:
Warning

Possible Cause:
The data type <type> as specified in the import file cannot be resolved or isn't natively supported by the GE SNPX Driver. The tag was not automatically generated.

Solution:
For applicable tags, avoid using data type <type> in the VersaPro/Logic Developer projects.

Database Error: Datatype '<type>' for tag '<tag name>' not found in import file. Setting to default

Error Type:
Warning

Possible Cause:
The definition of data type '<type>', for tag <tag name>, could not be found in the import file.

Solution:
This tag will take on the Default type for the given address type as assigned by the GE SNPX Driver.

Database Error: Logic Developer Variable Arrays are currently not supported. Array Tag(s) '<array tag name>' not created

Error Type:
Warning

Possible Cause:
Array tags of 1 or 2 dimensions originating from a Logic Developer import file, are not supported at this time. The array tag(s) were not automatically generated.

Solution:
For applicable tags, avoid using arrays in the Logic Developer projects.

Database Error: No Reference Address found for tag '<tag name>' in import file. Tag not created

Error Type:
Warning

Possible Cause:
Variables without a reference address cannot have a tag created since the reference address determines the tag's address. The tag was not automatically generated.

Solution:
Verify that the <tag name> has a PLC as a data source and that reference address (PLC memory location) has been assigned to it.

Database Error: Only variables with Data Source '<data source name>' are imported. Data Source '<data source name>' is not supported. Tag '<tag name>' not created

Error Type:
Warning

Possible Cause:
In Logic Developer, variables can take on a data value from a number of sources. For use in the OPC server, the source must be a GE SNPX PLC. The tag was not automatically generated.

Solution:
Verify that the <tag name> has a PLC as a data source.

Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created

Error Type:

Warning

Possible Cause:

Boolean or String array tags of 1 or 2 dimensions are not supported at this time.

Solution:

For Boolean array tags, individual array elements of the tag if specified in the import file will be generated. Further, the driver will also automatically create individual elements for the array tag (except for bit within word type Boolean array tags).

Note:

For String array tags, neither the array tag nor the individual elements will be generated. String data type is currently not supported by the driver. Thus, avoid using String data type if possible.

Index

3

7

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W