GE Ethernet Driver

Table of Contents

GE Ethernet Driver

Help version 1.074

CONTENTS

Overview

The GE Ethernet Driver provides a reliable way to connect GE Ethernet controllers to OPC Client applications; including HMI, SCADA, Historian, MES, ERP and countless custom applications. It is intended for use with GE Programmable Logic Controllers that may be accessed via an Ethernet module.

Setup

Supported Devices

Series 90-30 311/313, 331/341, 350, 360
Series 90-70 731/732, 771/772, 781/782
GE OPEN (Wide Range Model Support)
Horner OCS (Horner's Operator Control Stations)
PACSystems RX3i and RX7i
VersaMax Family

Communication Protocol

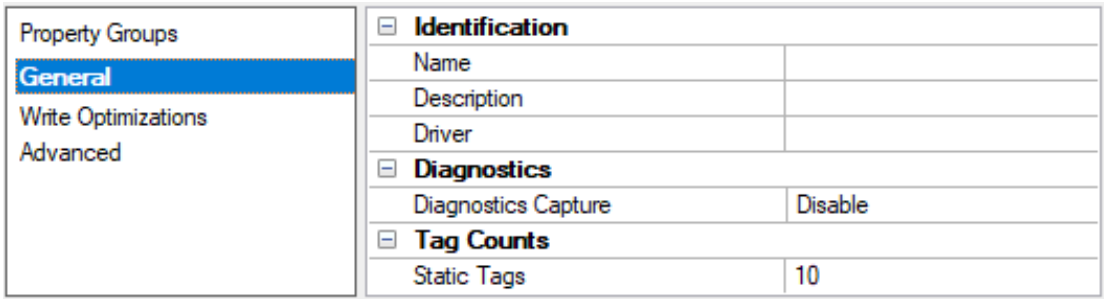
Ethernet, using Winsock V1.1 or higher

Channel and Device Limits

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

Channel Properties — General

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



Identification

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

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

Description: Specify user-defined information about this channel.

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

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

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

Diagnostics

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

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

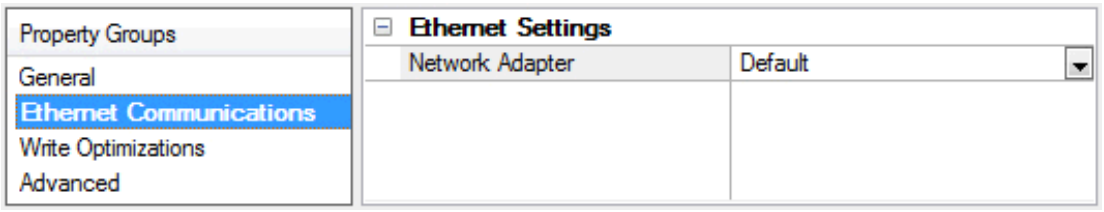
For more information, refer to "Communication Diagnostics" and "Statistics Tags" in the server help.

Tag Counts

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

Channel Properties — Ethernet Communications

Ethernet Communication can be used to communicate with devices.

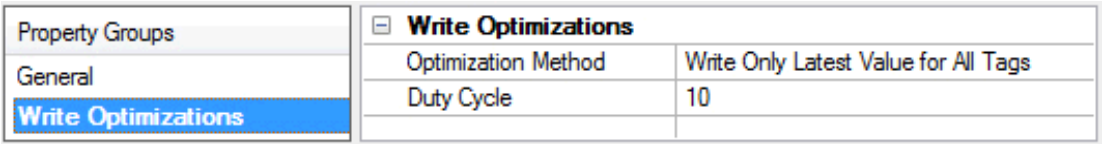


Ethernet Settings

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

Channel Properties — Write Optimizations

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



Write Optimizations

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

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

the operation of HMI data without causing problems with Boolean operations, such as a momentary push button.

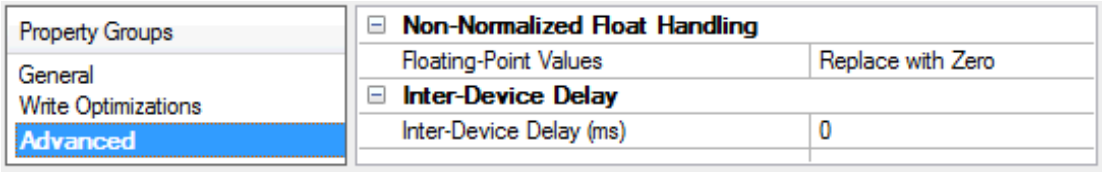
- **Write Only Latest Value for All Tags**: This option takes the theory behind the second optimization mode and applies it to all tags. It is especially useful if the application only needs to send the latest value to the device. This mode optimizes all writes by updating the tags currently in the write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

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

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

Channel Properties — Advanced

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



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

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

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

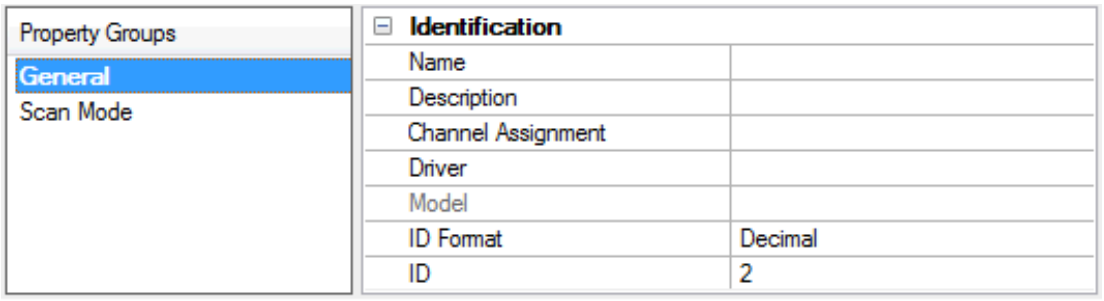
For more information on the floating-point values, refer to "How To ... Work with Non-Normalized Floating-Point Values" in the server help.

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

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

Device Properties — General

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



Identification

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

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

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

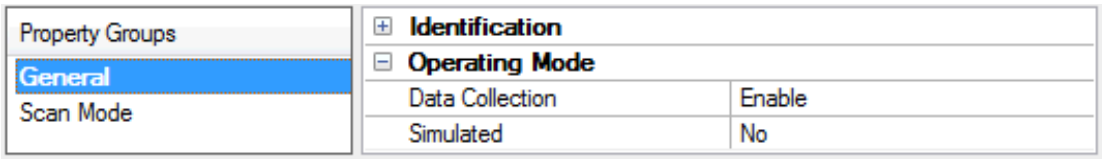
Channel Assignment: Specify the user-defined name of the channel to which this device currently belongs.

Driver: Selected protocol driver for this device.

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

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

Operating Mode



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

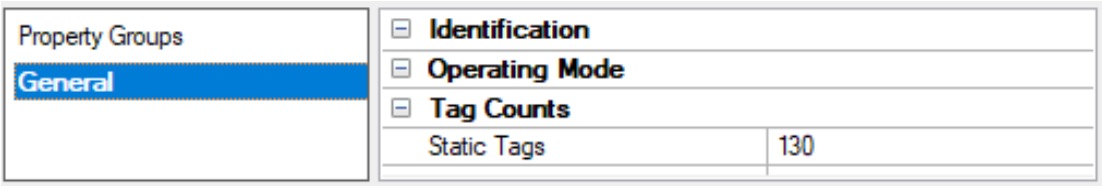
Simulated: Place the device into or out of Simulation Mode. In this mode, the driver does not attempt to communicate with the physical device, but the server continues to return valid OPC data. Simulated stops physical communications with the device, but allows OPC data to be returned to the OPC client as valid data. While in Simulation Mode, the server treats all device data as reflective: whatever is written to the simulated device is read back and each OPC item is treated individually. The data is not saved if the server removes the item (such as when the server is reinitialized). The default is No.

Notes:

1. 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. When a device is simulated, updates may not appear faster than one (1) second in the client.

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

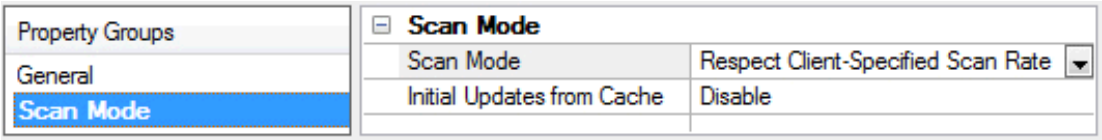
Tag Counts



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

Device Properties — Scan Mode

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



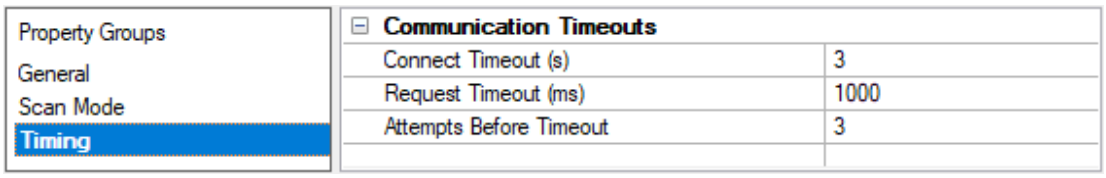
Scan Mode: Specify how tags in the device are scanned for updates sent to subscribing clients. Descriptions of the options are:

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

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

Device Properties — Timing

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



Communications Timeouts

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

Request Timeout: Specify an interval used by all drivers to determine how long the driver waits for a response from the target device to complete. The valid range is 50 to 9999999 milliseconds (167 minutes). The default is usually 1000 milliseconds, but can vary depending on the driver. The default timeout for most

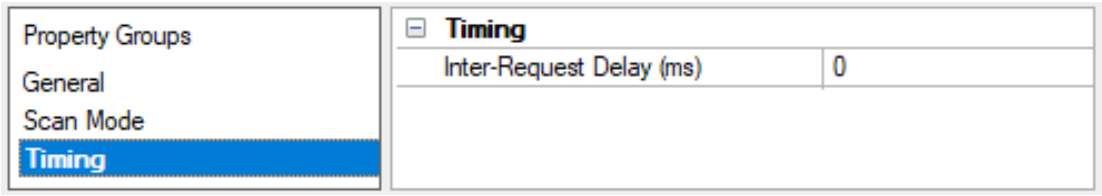
serial drivers is based on a baud rate of 9600 baud or better. When using a driver at lower baud rates, increase the timeout to compensate for the increased time required to acquire data.

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

Timing

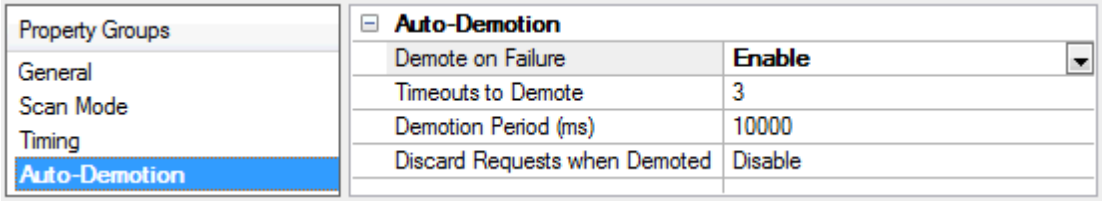
Inter-Request Delay: Specify how long the driver waits before sending the next request to the target device. It overrides the normal polling frequency of tags associated with the device, as well as one-time reads and writes. This delay can be useful when dealing with devices with slow turnaround times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

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



Device Properties — Auto-Demotion

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



Demote on Failure: When enabled, the device is automatically taken off-scan until it is responding again.

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

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

Demotion Period: Indicate how long the device should be placed off-scan when the timeouts value is reached. During this period, no read requests are sent to the device and all data associated with the read

requests are set to bad quality. When this period expires, the driver places the device on-scan and allows for another attempt at communications. The valid range is 100 to 3600000 milliseconds. The default is 10000 milliseconds.

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

Device Properties — Tag Generation

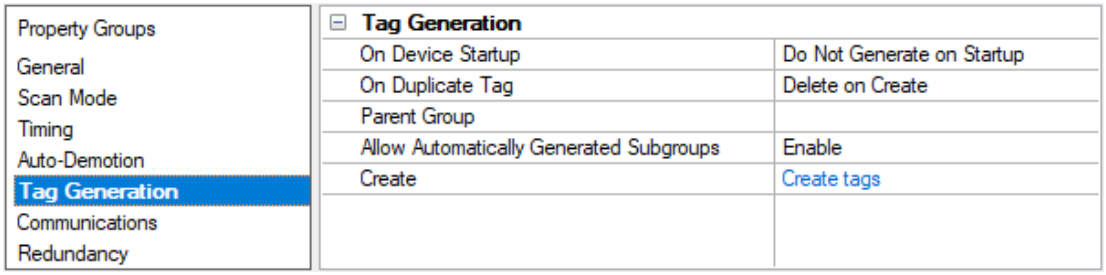
The automatic tag database generation features make setting up an application a plug-and-play operation. Select communications drivers can be configured to automatically build a list of tags that correspond to device-specific data. These automatically generated tags (which depend on the nature of the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

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

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

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

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



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

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

- **Do Not Generate on Startup**: This option prevents the driver from adding any OPC tags to the tag space of the server. This is the default setting.

- **Always Generate on Startup**: This option causes the driver to evaluate the device for tag information. It also adds tags to the tag space of the server every time the server is launched.
- **Generate on First Startup**: This option causes the driver to evaluate the target device for tag information the first time the project is run. It also adds any OPC tags to the server tag space as needed.

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

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

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

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

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

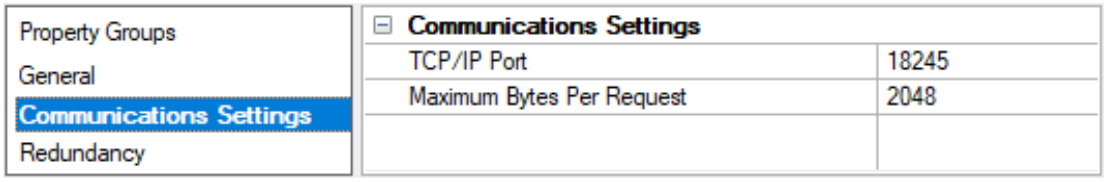
Parent Group: This property keeps automatically generated tags from mixing with tags that have been entered manually by specifying a group to be used for automatically generated tags. The name of the group can be up to 256 characters. This parent group provides a root branch to which all automatically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

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

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

Device Properties — Communications Settings

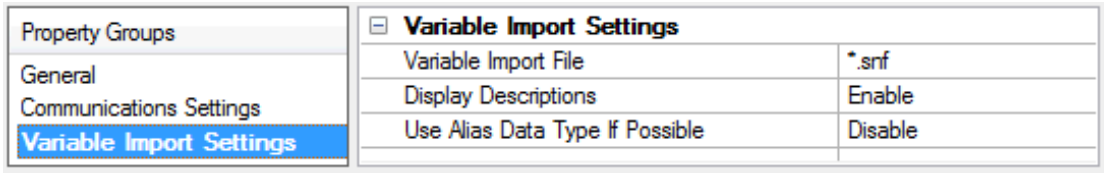
Define the device's port number and the maximum number of bytes of data that can be received in a single request.



TCP/IP Port: Specify the TCP/IP port number that the remote device is configured to use. The default setting is 18245.

Maximum Bytes Per Request: Specify the number of bytes that may be requested from a device at one time. To refine this driver's performance, configure the request size to one of the following settings: 32, 64, 128, 256, 512, 1024, or 2048 bytes. The default value is 2048 bytes.

Device Properties — Variable Import Settings



Variable Import File: This property specifies 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 imports tag descriptions. If necessary, a description is given to tags with long names that state the original tag name. Default setting is Enable.

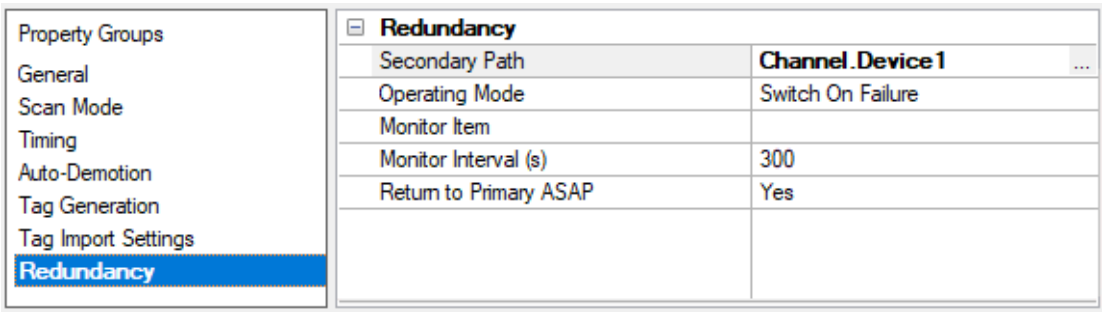
Use Alias Data Type If Possible: Enable to use the data type assigned to an alias tag in the import file. If the alias data type is incompatible with the source tag data type, the source tag data type is used instead. Default setting is Disable.

See Also: *Automatic Tag Database Generation*

Device Properties — PLC Settings

CPU Slot Location: Specify the physical location of the device on the rack. Depending on the model, the CPU may be fixed and therefore unmodifiable. For CPUs that take up two slots, the leftmost slot covered is the CPU slot. The valid range is 0 to 15. The default setting is slot 1.

Device Properties — Redundancy



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

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

Automatic Tag Database Generation

The GE Ethernet Device Driver generates its tags offline based on variables imported from a text file. It is off-line in that a connection to the device is not required to generate tags. The text file (variables to import) can originate from one of the following applications:

1. Proficy Machine Edition - Logic Developer
2. Cimplicity Machine Edition - Logic Developer
3. 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.

Resources:

*For information on creating variable import files, refer to **Importing VersaPro Tags**, **Importing Proficy Logic Developer Tags** or **Importing Cimplicity Logic Developer Tags**.*

*For information on generating tags based on the import file, refer to **Variable Import Settings** and **Tag Generation**.*

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 on the property **On Duplicate Tag**. After the first OPC server import/database creation completes, set the action to **Do not overwrite** or **Do not overwrite, log error** for future imports. This allows tags to be imported without deleting or overwriting tags that were previously imported.

Tag Hierarchy

The tags created through Automatic Tag Generation follow a specific hierarchy. The root level groups (or subgroup levels of the group specified in the "Add generated tags to the following group" property) are determined by the variable addresses referenced (such as R, G, M, and so forth). 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 that 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

Symbolic Variable Arrays

Symbolic variable tags (in PACSystems only) are placed in a group called Symbolic. Symbolic variable arrays are not automatically broken out into individual element tags, and will not be placed in a separate group. A single array tag will be generated for each symbolic variable array in the Symbolic group. For example, if a 2x3 array of symbolic variables named "MySymbolicArray" is defined in the import file, then a single tag with name "MySymbolicArray" and address "!MySymbolicArray"[2][3] would be generated in the Symbolic group.

Although symbolic variable arrays are not automatically broken out into individual array element tags, an array element tag will be placed in the Symbolic group if an individual array element symbolic variable is listed in the import file. For example, an import file that contains an entry for array element (0,1) of the "MySymbolicArray" would appear as "MySymbolicArray[0,1]". A tag with address "!MySymbolicArray{0}{1}" would be generated in the Symbolic group.

See Also: Symbolic Variables

Import File-to-Server Name Conversions

Leading Characters

- The server does not accept group and tag names beginning with an underscore. Leading underscores (_) in tag names will be replaced with **U**_.
- The server does not accept group and tag names beginning with a percent sign. Leading percents (**%**) in tag names will be replaced with **P**_.
- The server does not accept group and tag names beginning with a pound sign. Leading pound signs (**#**) in tag names will be replaced with **PD**_.
- The server does not accept group and tag names beginning with a dollar sign. Leading dollar signs (**$**) in tag names will be replaced with **D**_.

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. The table below displays the invalid character and sequence of characters that it is replaced with when encountered in the import file variable name.

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

Group and Tag Name Length Limitations

The GE Ethernet Driver limits group and tag names to 256 characters. Names that exceed 256 characters will be truncated as follows:

Non-Array

1. Determine a 5-digit Unique ID for this tag.
2. Given a tag name: ThisIsALongTagName...AndProbablyExceeds256
3. Truncate 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
3. Truncate tag at 256 while holding on to the element values: ThisIsALongTagNameAndPr_23
4. Room is made for the Unique ID: ThisIsALongTagName...#####_23
5. Insert this ID: ThisIsALongTagName...00001_23

Importing VersaPro Tags

This driver uses Shared Name Files (SNF) that are generated from VersaPro to generate the tag database. Certain aspects of the Automatic Tag Database Generation process depend on the application from which variables are imported. For more information on a specific aspect of VersaPro tag import, select a link from the list below.

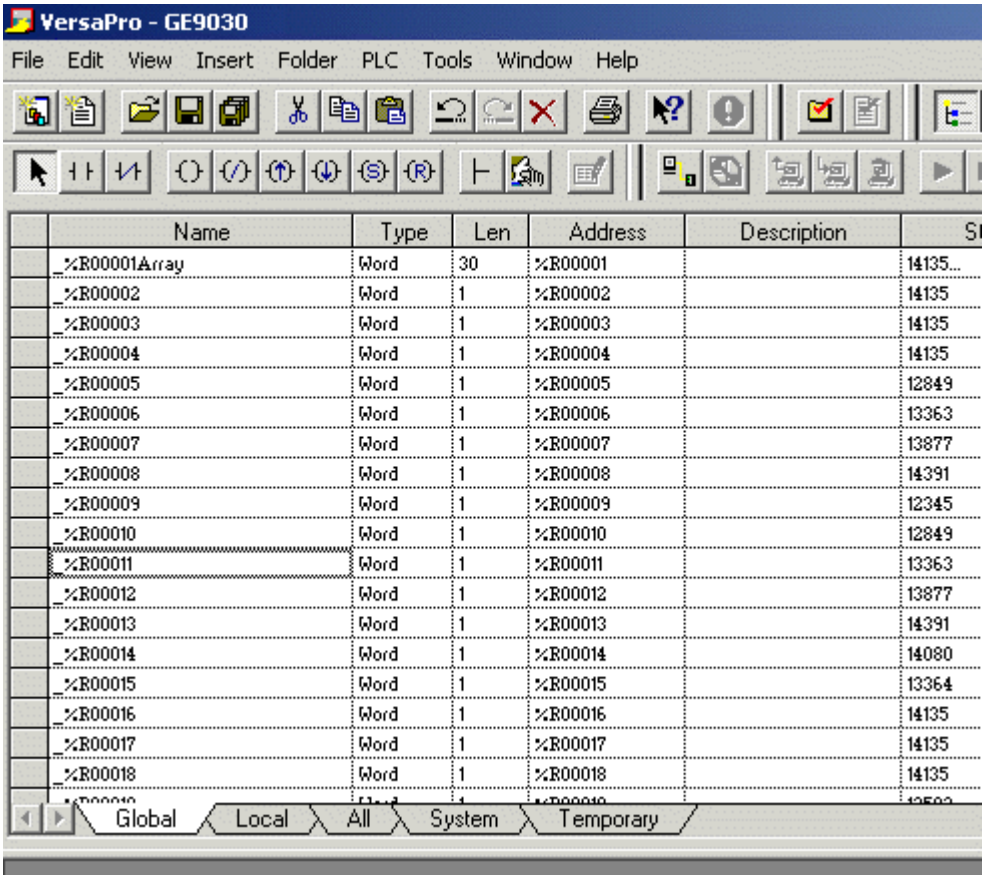
VersaPro Import Preparation: VersaPro Steps
VersaPro Import Preparation: OPC Server Steps
Highlighting VersaPro Variables
VersaPro Array Tag Import

Note: To import tags using a different application, refer to **Automatic Tag Database Generation** to verify that the application is supported.

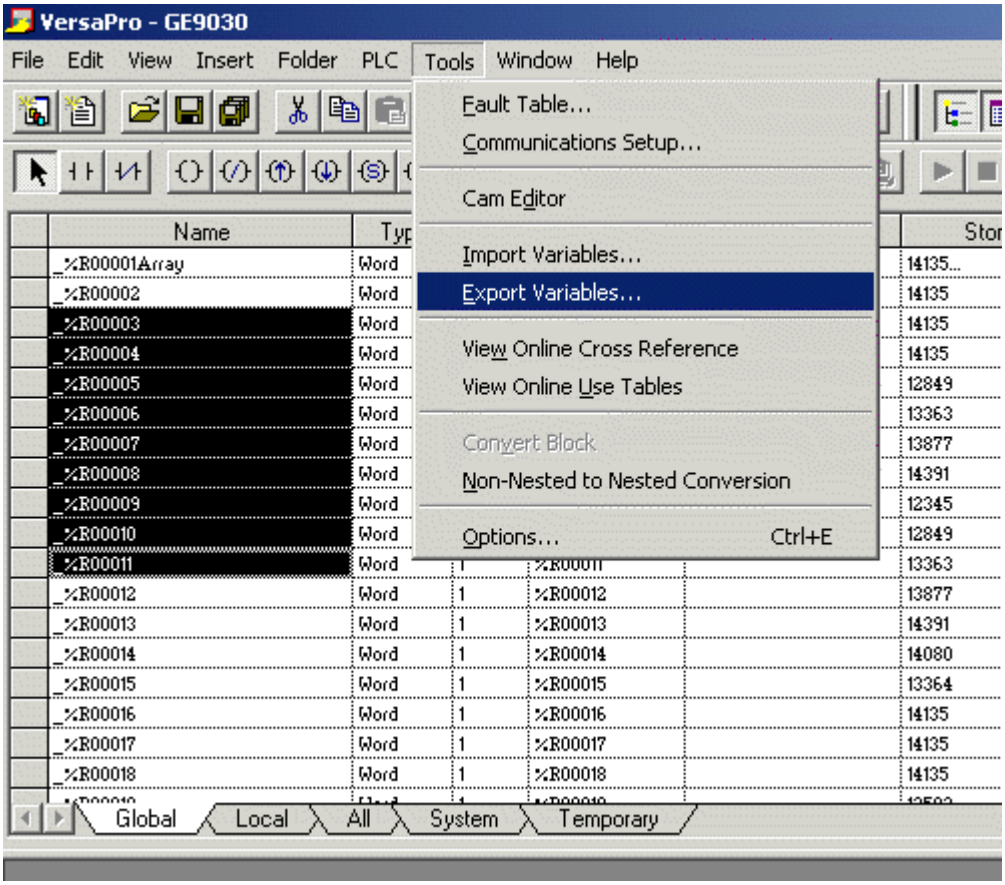
VersaPro Import Preparation: VersaPro Steps

1. To start, open the VersaPro project containing the tags (variables) that will be ported to the OPC server.
2. If the **Variable Declaration Table** is not already open, click **View** | **Variable Declaration Table**.
3. Next, specify the group to which the tags of interest belong. The default groups are **Global**, **Local**, **All**, **System**, and **Temporary**.

   Note: The All group does not include the variables from the System group. Multiple imports (or multiple SNF files) are required to import System, Global, Local, and All variables.



4. Next, click on the group's tab to bring its variables to the front. Then, highlight the tags of interest and click **Tools** | **Export Variables**.



5. When prompted, select **Shared Name File (*.snf)** and specify a name.

Note: VersaPro will export the project's contents into this SNF file.

See Also: Highlighting VersaPro Variables

VersaPro Import Preparation: OPC Server Steps

1. To start, right-click on the device for which tags will be generated and select **Device Properties**. Then, open the **Variable Import Settings** property group.
2. In **Variable Import File**, enter or browse for the location of the VersaPro *.snf file newly created. Then, click **Apply**.
3. Next, select the **Tag Generation** property group. Then, click **Create Tags**.
4. 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 VersaPro will appear in the OPC server in the layout discussed in Tag Hierarchy.
5. Once finished, click **OK**.

See Also: Variable Import Settings

Highlighting VersaPro Variables

Variables can be highlighted in VersaPro in the following ways:

- **Single Variable Selecting:** To do so, left-click on a variable of interest while pressing CTRL.
- **Selecting a Range of Variables:** To do so, left-click on the first variable in the range of interest. Then, press SHIFT while left-clicking on the last variable in the range. All variables in the range will be highlighted.
- **Selecting All Variables:** To do so, left-click on a variable within the group of interest in the Variable Declaration Table. The variable chosen is irrelevant. Then, click **Edit** | **Select All**. All variables will be highlighted with that group.

VersaPro Array Tag Import

Variables in VersaPro 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 the following 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 Tag

Since VersaPro arrays are 1-dimensional, the number of columns is always 1. As such, an array tag would have the following syntax:

<*array variable*>[#*rows* = *Length*]

Note: 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 to be used in HMIs that support arrays.

Individual Elements

Element tags are 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 that are 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 can only be accessed as an expanded group of tags (not as an array tag).

Importing Cimplicity Logic Developer Tags

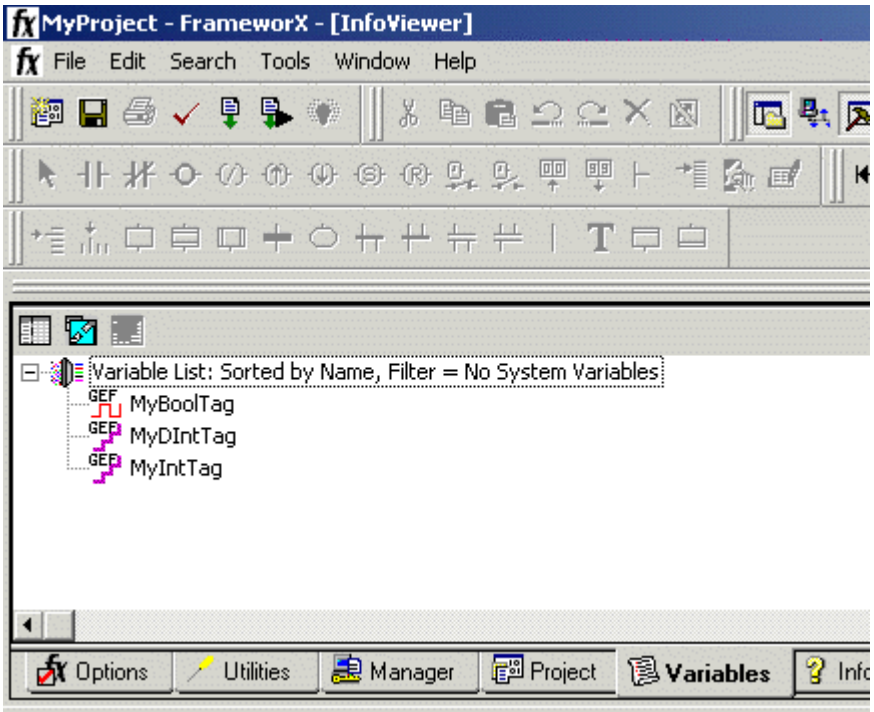
This driver uses user-generated ASCII text files from Logic Developer to generate the tag database. Certain aspects of the Automatic Tag Database Generation process depend on the application from which variables are imported. For more information on a specific aspect of Logic Developer tag import, select a link from the list below.

Cimplicity Logic Developer Import Preparation: Logic Developer Steps
Cimplicity Logic Developer Import Preparation: OPC Server Steps
Highlighting Cimplicity Logic Developer Variables
Cimplicity Logic Developer Array Tag Import

Note: To import tags using a different application, refer to Automatic Tag Database Generation to verify that the application is supported.

Cimplicity Logic Developer Import Preparation: Logic Developer Steps

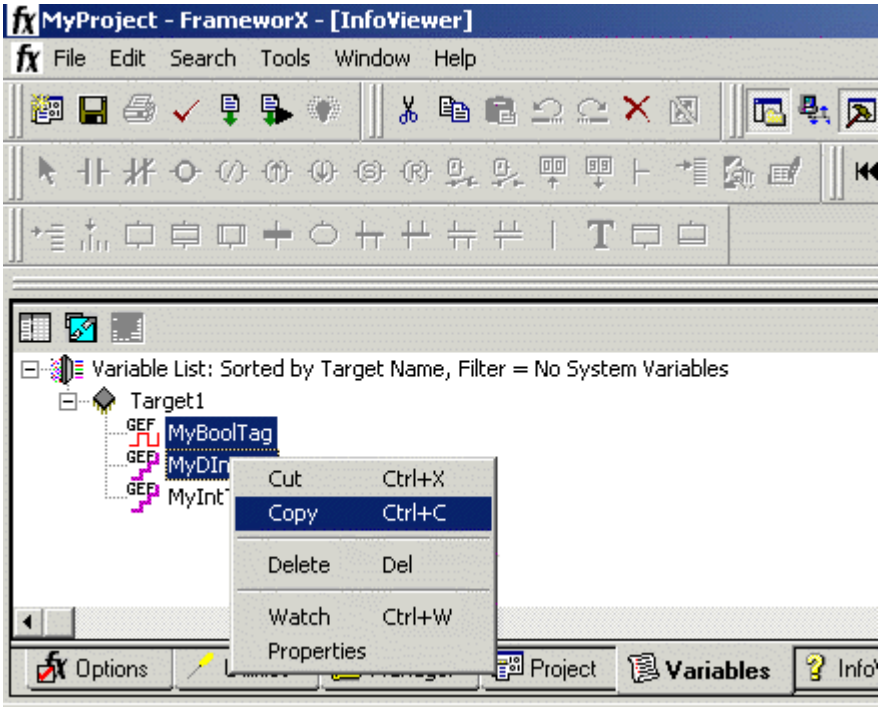
1. To start, open the FrameworX project containing the tags (variables) that will be ported to the OPC server. Then, open the **Navigator** window by pressing **Shift+F4**.
2. Next, click on the **Variables** tab and select **Variable List View**.



Note: Each FrameworX project contains one or more targets, which is essentially the device on which the application will run. Because variables are created on the target level, a target of interest

must be specified before the variables that will be exported may be. In order for the target variables to be imported, the variables must have **GE PLC** as the **Data Source**. To verify this, left-click on the variable and then look at the **Data Source** properties in the **Inspector** window. Internal variables will not be imported.

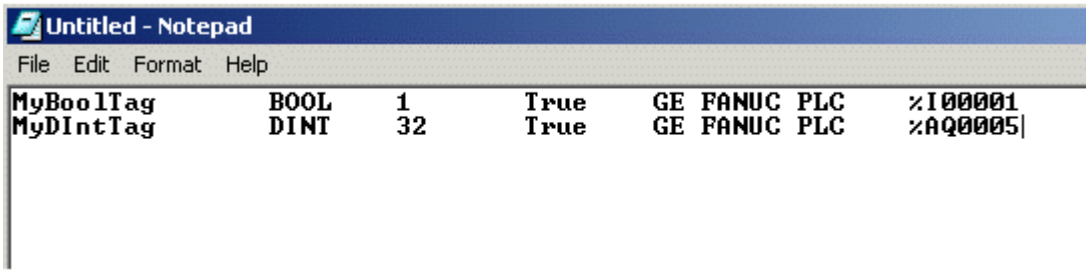
3. Next, sort the variables by target by right-clicking on the **Variable List** and selecting **Sort** | **Target**.
4. Then, highlight the tags of interest in the target of interest and click **Edit** | **Copy**.



Note: The highlighted variables should now be copied to the Clipboard.

5. Next, open a word processing program like Notepad or Wordpad. Then, click **Edit** | **Paste**.

Note: 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.



6. Save the text document with the text extension (.txt) in ANSI form.
7. The variables are now contained within the text document and can be imported into the OPC server.

Cimplicity Logic Developer Import Preparation: OPC Server Steps

1. To start, right-click on the device for which tags will be generated and select **Device Properties**. Then, open the **Variable Import Settings** property group.
2. In **Variable Import File**, enter or browse for the location of the Logic Developer *.txt file newly created. Then, click **Apply**.
3. Next, select the **Tag Generation** property group. Then, click **Create Tags**.
4. 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**.
5. Once finished, click **OK**.

See Also: **Variable Import Settings**

Highlighting Cimplicity Logic Developer Variables

Variables can be highlighted in Logic Developer in the following ways:

- **Single Variable Selecting:** To do so, left-click on a variable of interest.
- **Pick-n-Choose:** To do so, left-click on the first variable of interest. Then, press CTRL while left-clicking on each successive variable of interest. Repeat until all variables of interest are highlighted.
- **Selecting a Range of Variables:** To do so, left-click on the first variable in the range of interest. Then, press SHIFT while left-clicking on the last variable in the range. All variables in the range will be highlighted.
- **Selecting All Variables:** To do so, left-click on a variable within the target of interest in the Variable List View. The variable chosen is irrelevant. Then, click **Edit** | **Select All**. All variables will be highlighted within that target.

Cimplicity Logic Developer Array Tag Import

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

Importing Proficy Logic Developer Tags

This driver uses user-generated ASCII text files from Proficy Logic Developer to generate the tag database. Certain aspects of the Automatic Tag Database Generation process depend on the application from which variables are imported. For more information on a specific aspect of Proficy Logic Developer tag import, select a link from the list below.

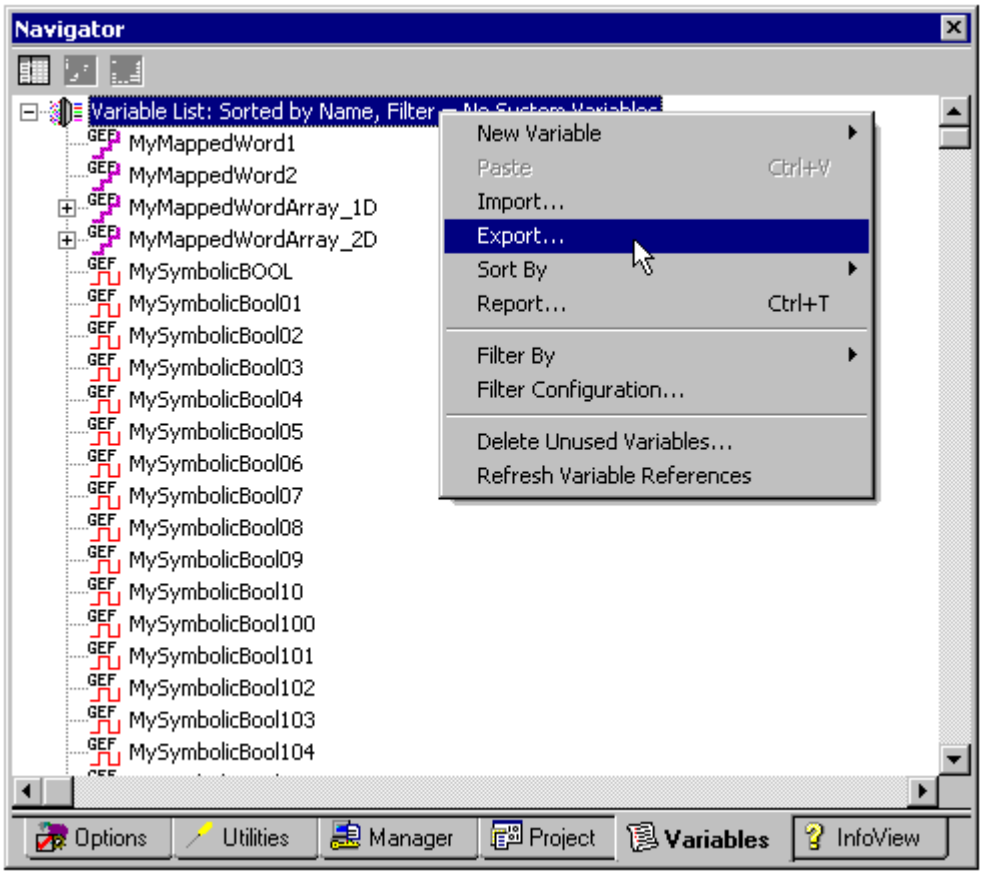
Proficy Logic Developer Import Preparation: Logic Developer Steps
Proficy Logic Developer Import Preparation: OPC Server Steps
Highlighting Proficy Logic Developer Variables
Proficy Logic Developer Array Tag Import

Note: To import tags using a different application, refer to **Automatic Tag Database Generation** to verify that the application is supported.

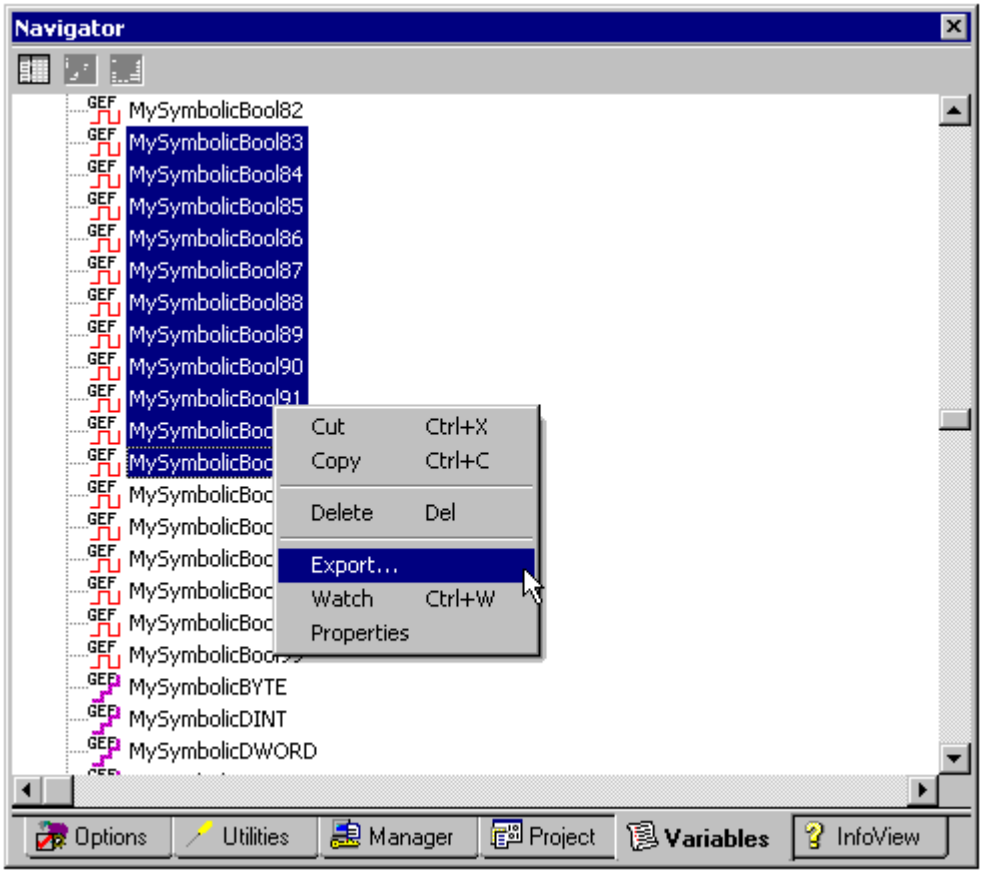
Proficy Logic Developer Import Preparation: Logic Developer Steps

1. To start, open the Proficy Logic Developer project containing the tags (variables) that will be ported to the OPC server. Then, open the **Navigator** window by pressing **Shift-F4**.
2. Next, click on the **Variables** tab to bring the project's variables to the front. Then, click **Variable List View**.

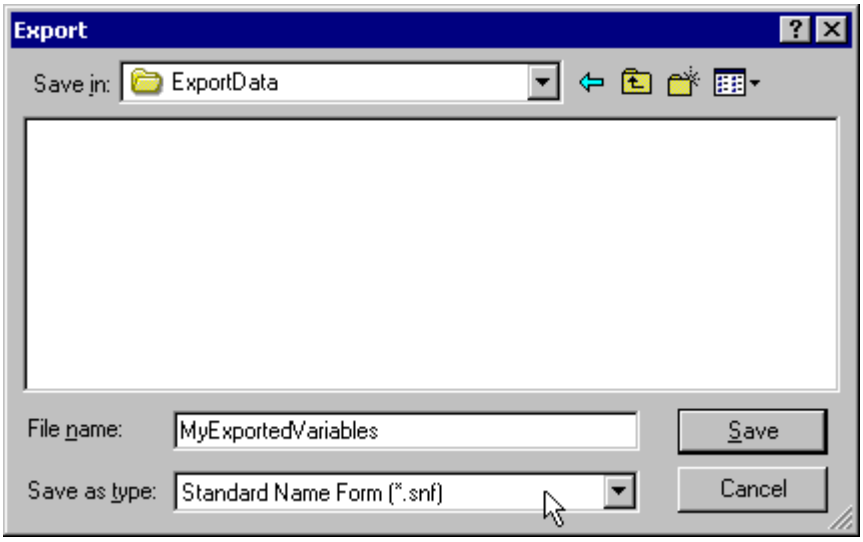
   Note: Each Logic Developer project contains one or more targets, which is essentially the device on which the application will run. Because variables are created on the target level, the target of interest must be specified before the variables that will be exported may be. For the target variables to be imported, the variables must have **GE PLC** chosen as the **Data Source**. To verify this, left-click on the variable and then look at the **Data Source** properties in the **Inspector** window. Internal variables will not be imported.
3. Next, sort the variables by target by right-clicking on the **Variable List** header and selecting **Sort** | **Target**.
4. To export all of the variables, right-click on the **Variable List** and click **Export**.

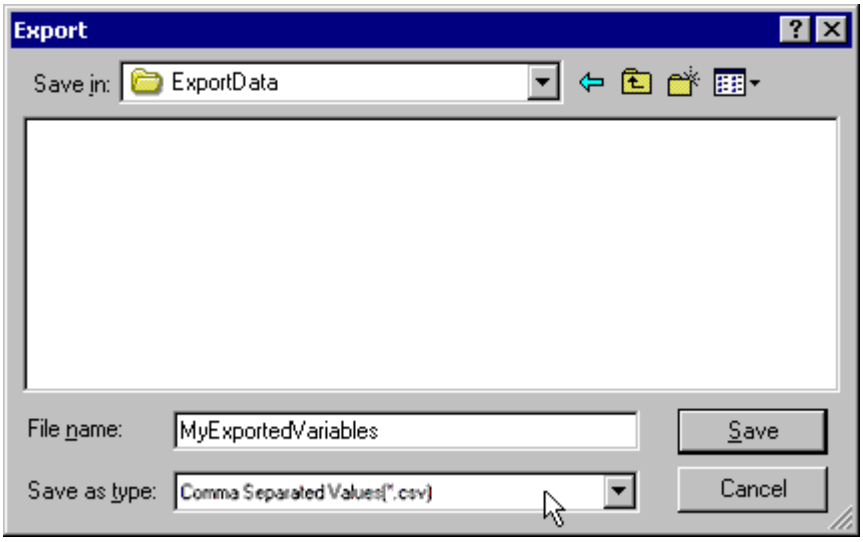


5. To export selected variables, highlight the tags of interest in the target of interest. Then, right-click on one of the selected variables and click **Export**.



6. In the **Save as Type** drop-down list, select **Standard Name Form (*.snf)** or **Comma Separated Variable (*.csv)** as the export file type. The dialogs should appear as shown below.





7. Once finished, click **Save**.

Proficy Logic Developer Import Preparation: OPC Server Steps

1. To start, right-click on the device for which tags will be generated and select **Device Properties**. Then, open the **Variable Import Settings** property group.
2. In **Variable Import File**, enter or browse for the location of the export file newly created *.snf or *.csv file. Then, click **Apply**.
3. Next, select the **Tag Generation** property group, and then click **Create Tags** to import variables. Alternatively, use the other settings in that property group to automatically create the database later.
4. 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**.
5. Once finished, click **OK**.

See Also: **Variable Import Settings**

Highlighting Proficy Logic Developer Variables

Variables can be highlighted in Logic Developer in the following ways:

- **Single Variable Selecting:** To do so, left-click on a variable of interest.
- **Pick-n-Choose:** To do so, left-click on the first variable of interest. Then, press CTRL while left-clicking on each successive variable of interest. Repeat until all variables of interest are highlighted.
- **Selecting a Range of Variables:** To do so, left-click on the first variable in the range of interest. Then, press SHIFT while left-clicking on the last variable in the range. All variables in the range will be highlighted.
- **Selecting All Variables:** To do so, left-click on a variable within the target of interest in the Variable List View. The variable chosen is irrelevant. Then, 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 placed in the Symbolic group along with all other symbolic variable tags. The driver will not generate tags for BOOL and STRING symbolic variable arrays.

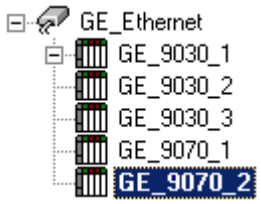
Importing Array Elements (Index)
When importing array elements (such as "MyArrayTag[1,2]") into the driver, the '[' and ']' characters will be replaced with '{' and '}' respectively for internal reasons. The '[' and ']' characters are reserved by this driver for array notation. While communicating with the PLC at Runtime, however, the replacement will be reversed by the driver to comply with the standard GE syntax.

See Also: **Symbolic Variables**

Optimizing Communications

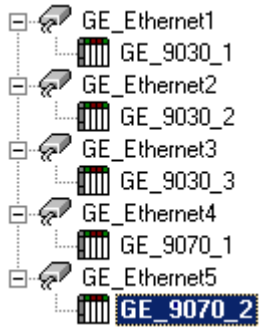
The GE Ethernet Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the GE Ethernet Driver is fast, there are a couple of guidelines that can be used to control and optimize the application and gain maximum performance.

Our server refers to communications protocols like GE Ethernet as a channel. Each channel defined in the application represents a separate path of execution in the server. Once a channel has been defined, a series of devices must then be defined under that channel. Each of these devices represents a single GE controller from which data will be collected. While this approach to defining the application will provide a high level of performance, it won't take full advantage of the GE Ethernet Driver or the network. An example of how the application may appear when configured using a single channel is shown below.



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

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



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

The performance will improve even if the application has more than 256 devices. While fewer devices may be ideal, the application will still benefit from additional channels. Although by spreading the device load across all channels will cause the server to move from device to device again, it can now do so with far less devices to process on a single channel.

Tip:
The GE Ethernet Driver performance can also be affected by Maximum Bytes Per Request (i.e. block size), which refers to the number of bytes that may be requested from a device at one time. It is available on each device. To refine the performance of this driver when reading symbolic variables, configure Maximum Bytes Per Request to one of the following settings: 32, 64, 128, 256, 512, 1024, or 2048 bytes. Depending on the specific GE device, the block size setting can have a dramatic effect on the application.

- **GE Ethernet Driver for server versions 5.10 and earlier:** The default setting of 256 bytes is recommended; however, users whose applications contain large requests for consecutively ordered data should increase the block size.
- **GE Ethernet Driver for server versions 5.11 and later:** The default setting of 2048 bytes is recommended. These versions have been enhanced to utilize multiple block reads in a single request. By requesting multiple smaller blocks of data, users can potentially reduce the number of unnecessary bytes in the response (especially when the requested data is not contiguous in memory). The Block Size property limits the number of bytes returned for all the blocks.

See Also: Communications Settings

Note: For PACSystem Models It is recommended that referenced (mapped) variables and symbolic variables be placed on separate devices. For more information, refer to **Symbolic Variables**.

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 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.
Double	64-bit floating point value The driver interprets four consecutive registers as a floating point value.
String	Null-terminated ASCII string Support includes HiLo LoHi byte order selection.

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.

PACSystems Addressing

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

Device Address	Range	Data Type	Access
Discrete Inputs	I00001 to I32768 I00001 to I32761 (every 8th bit) I00001 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 G7680 G00001 to G7673 (every 8th bit) G00001 to G7665 (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	T0001 to T1024 T0001 to T1017 (every 8th bit) T0001 to T1009 (every 8th bit)	**Boolean** Byte Word, Short, BCD	Read/Write
Status References	S001 to S128 S001 to S121 (every 8th bit)	**Boolean** Byte	Read Only

Device Address	Range	Data Type	Access
(Same for SA, SB, SC)	S001 to S113 (every 8th bit)	Word, Short, BCD	
Register References	R1.0 to R32640.15 R00001 to R32640 R00001 to R32639 R00001 to R32637	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI32640.15 AI00001 to AI32640 AI00001 to AI32639 AI00001 to AI32637	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ32640.15 AQ00001 to AQ32640 AQ00001 to AQ32639 AQ00001 to AQ32637	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers*	P1.0 to P8192.15 P00001 to P8192 P00001 to P8191 P00001 to P8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L8192<SUBPRGM>.15 L00001<SUBPRGM> to L8192<SUBPRGM> L00001<SUBPRGM> to L8191<SUBPRGM> L00001<SUBPRGM> to L8189<SUBPRGM>	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Bulk Memory	W1.0 to W33554432.15 W00001 to W33554432 W00001 to W33554431 W00001 to W33554429	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*Program specified in **PLC Settings**.

**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under "Block Properties."

*For more information, refer to "Importing Array Elements (Index)" in **Proficy Logic Developer Array Tag Import**.*

See Also: **Symbolic Variables**

Advanced Addressing

Symbolic Variables

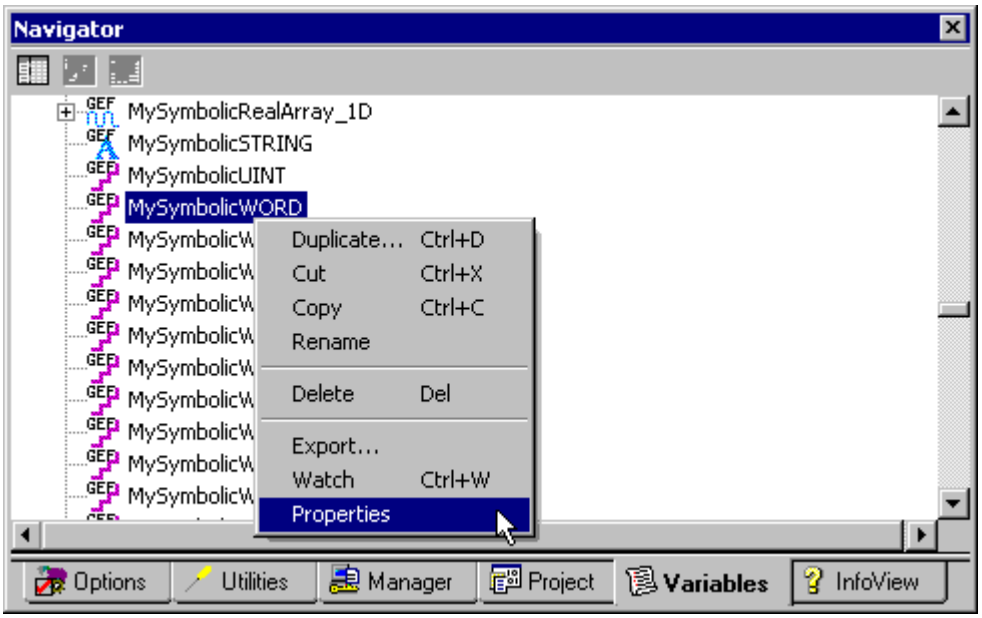
A variable is a named storage space for data in the PLC. There are two types of variables: Reference and Symbolic. Descriptions of the variables are as follows:

- **Reference Variable:** This variable is mapped to a specific I/O or internal register location in the PLC (such as I00001, R00100, and so forth).
- **Symbolic Variable:** This variable is not mapped to a specific register. The PLC will allocate memory for symbolic variables as needed from its managed memory area. This driver has the ability to read and write to symbolic variables defined in the PLC. Symbolic variables are only available in PAC Systems PLCs.

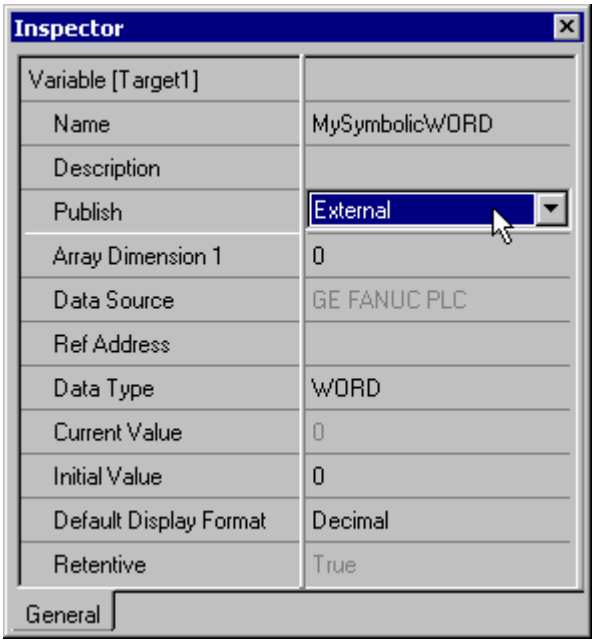
Note: Writing a non-zero value to the _ForceSymbolRefresh system tag causes the driver to re-download symbolic information from the PLC.

Creating a Symbolic Variable

1. To start, load the PLC's current configuration project into the Proficy Logic Developer. Then, open the **Navigator View** by pressing **Shift-F4**.
2. Next, click on the **Variables** tab.
3. To create a new variable, right-click on the **Variables View** and select **New Variable**. To edit an existing variable, right-click on it and then select **Properties**.



4. In both cases, the variable's **Properties Inspector** dialog will appear as shown below.



Note: The variable displayed above is symbolic because the **Ref Address** parameter is left blank. The variable is named "MySymbolicWORD" and has a data type of Word. An array of variables can be created by setting the **Array Dimension 1** parameter to a value greater than zero. The **Array Dimension 2** parameter will appear in the variable inspector if Array Dimension 1 is set. A single dimension array is defined when Array Dimension 2 is left at zero. The total number of array elements is limited by the **Maximum Bytes Per Request** property in Device Properties. For more information, refer to **Device Setup**.

For a symbolic variable to be visible to this driver, **Publish** must be set to **External**.

5. Next, download the modified project to the PLC.
6. To access a symbolic variable with the OPC server, create a tag whose address references the variable by name. It must be preceded by the "!" character.

Examples

The syntax for addressing array elements uses curly brackets. These characters are specific to this server: other servers or products may access array elements using square brackets instead. The appropriate conversion will be done internally by the driver at Runtime while communicating with the PLC. For addressing purposes, users must utilize curly brackets: square brackets are reserved by this driver for array notation.

Important: The tag must be assigned a data type that is compatible with the data type of the symbolic variable; however, if the tag address references a bit of the symbolic variable, then the tag data type must be Boolean. Furthermore, the dimensions of the array tag must be the same as the variable. The tag properties can only be validated during Runtime.

1. !MySymbolicArray1D [4] addresses the symbolic variable named "MySymbolicArray1D," which is assumed to have its Array Dimension 1 property set to 4.
2. !MySymbolicArray1D {3} addresses the index (3) of the above 1D symbolic array variable "MySymbolicArray1D."

3. !MySymbolicArray2D [3][4] addresses the symbolic variable named "MySymbolicArray2D," which is assumed to have its Array Dimension 1 property set to 3, and its "Array Dimension 2" property set to 4.
4. !MySymbolicArray2D {2,3} addresses the index (2,3) of the above 2D symbolic array variable "MySymbolicArray2D."
5. !MySymbolicVariable.5 addresses the bit index 5 of the symbolic variable named "MySymbolicVariable."
6. !MySymbolicVariable.30 addresses the bit index 30 of the symbolic variable named "MySymbolicVariable," which is assumed to have a native data type of Dint or DWord.
7. !MySymbolicArray{3}.8 addresses the bit index 8 of the element at index 3 of the symbolic array named "MySymbolicArray," which is assumed to have a native data type of Int, Uint, Word, Dint, or DWord.
8. !MySymbolicArray{3,2}.8 addresses the bit index 8 of the element at index 2 of the fourth array of the symbolic array named "MySymbolicArray," which is assumed to have a native data type of Int, Uint, Word, Dint, or DWord.
9. !MySymbolicUDTArray{5}.MyUDTMember.7 addresses the bit index 7 of the User Defined Type (UDT) member variable named "MyUDTMember" of the element at index 5 of the symbolic array variable named "MySymbolicUDTArray."
10. !MyUDT.MyUDTMember.16 addresses the bit index 16 of the symbolic variable named "MyUDTMember" that is part of a UDT named "MyUDT." The MyUDTMember variable is assumed to have a native data type of Dint or DWord.

Compatible Data Types

The driver assigns a default type of Word to tags that use Symbolic Addressing; however, Boolean is the default data type assigned to the tags that have a bit index appended to the symbolic address.

Native Type (Device)	Compatible Types (Server)
Bool	Bool (no arrays)
Byte	Byte, Char
Int	Word, Short
Uint	Word, Short
Word	Word, Short
Dint	DWord, Long
DWord	DWord, Long
Real	Float, Double
LReal	Double
String	String (no arrays)

Note: Users should be sure to specify an appropriate data type override for each dynamic tag. Dynamic Tags are only defined in the client application and are created in the server on demand by the client. Their data type can be specified by appending one of the following strings to the item name: @Boolean, @Byte, @Char, @Short, @Word, @Long, @DWord, @Float or @String. For more information, refer to the OPC server help file.

Example
ItemName = "Channel1.Device1.!MySymbolicWordVariable @DWord"

Performance Considerations

Block Size
This driver attempts to optimize performance by reading blocks of symbolic variable memory. For the GE Ethernet Driver released in server versions 5.10 and earlier, it was generally faster to read a single large block of data (such as where only the first and last few bytes were needed to update two tags) than it was to perform two separate reads for the few bytes needed for each tag. The amount of symbolic variable memory read per request was limited by the **Maximum Bytes Per Request** property located in Device Properties. As such, it was typically advantageous to maximize this setting.

In some cases, however, performance would increase by reducing that setting. An example of this is when the variables being read most frequently were widely scattered in the controller's managed memory area. Users cannot control the mapping of symbolic variable data in the controller's memory: mapping largely depends on when the variables were added to the configuration project. As such, if performance was a primary concern for the GE Ethernet Driver in those server versions, it was recommended that users experiment with the Maximum Bytes Per Request property.

The GE Ethernet Driver released in server version 5.11 has been enhanced to improve performance by allowing multiple small blocks in a single request. Only the desired bytes will be requested when the data is not contiguous, and a single request may include multiple small block requests. The Maximum Bytes Per Request property will limit the total number of bytes in the response, but not the number of bytes in the individual blocks. As such, using this value's maximum setting will likely improve performance.

Multi-Item Writes
This driver will also attempt to optimize performance by performing multi-item writes. The number of tags included in a request is limited by the Maximum Bytes Per Request property. Unlike block reads, the variables' location in the controller's memory is of little consequence. Writes can further be optimized with the **Write Optimizations** properties located in Channel Properties.

Note: If the driver finds that the variable named in a tag's address does not exist in the controller (or if it does not match the data type or array dimension), its OPC quality will be set to Bad. The driver will continue to check the controller for the named variable should a new configuration be downloaded. It is recommended that users do not utilize those tags in the client applications because they will remain invalid.

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

Device Address	Range	Data Type*	Access
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	R1.0 to R512.15 R001 to R512 R001 to R511 R001 to R509	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI64.15 AI01 to AI64 AI01 to AI63 AI01 to AI61	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ032.15 AQ001 to AQ032 AQ001 to AQ031 AQ001 to AQ029	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
** ***See Also:*** ***Special Items***

Advanced Addressing

Bit Access to Registers
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)	**Boolean** Byte	Read/Write

Device Address	Range	Data Type*	Access
	G0001 to G1265 (every 8th bit)	Word, Short, BCD	
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	R1.0 to R1024.15 R001 to R1024 R001 to R1023 R001 to R1021	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI64.15 AI01 to AI64 AI01 to AI63 AI01 to AI61	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ032.15 AQ001 to AQ032 AQ001 to AQ031 AQ001 to AQ029	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
** ***See Also***: *Special Items*

Advanced Addressing

Bit Access to Registers
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	**Boolean**	Read/Write

Device Address	Range	Data Type*	Access
	G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	Byte Word, Short, BCD	
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	R1.0 to R2048.15 R0001 to R2048 R0001 to R2047 R0001 to R2045	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI128.15 AI001 to AI128 AI001 to AI127 AI001 to AI125	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ64.15 AQ01 to AQ64 AQ01 to AQ63 AQ01 to AQ61	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
** **See Also**: Special Items.

Advanced Addressing

Bit Access to Registers
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	R1.0 to R9999.15 R0001 to R9999 R0001 to R9998 R0001 to R9996	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI1024.15 AI0001 to AI1024 AI0001 to AI1023 AI0001 to AI1021	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ256.15 AQ0001 to AQ256 AQ0001 to AQ255 AQ0001 to AQ253	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
** **See Also**: Special Items.

Advanced Addressing

Bit Access to Registers
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)	**Boolean** Byte	Read/Write

Device Address	Range	Data Type*	Access
	Q0001 to Q2033 (every 8th bit)	Word, Short, BCD	
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
Register References	R1.0 to R9999.15 R0001 to R9999 R0001 to R9998 R0001 to R9996	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI2048.15 AI0001 to AI2048 AI0001 to AI2047 AI0001 to AI2045	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ512.15 AQ001 to AQ512 AQ001 to AQ511 AQ001 to AQ509	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
** **See Also**: Special Items.

Advanced Addressing

Bit Access to Registers
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	**Boolean**	Read/Write

Device Address	Range	Data Type*	Access
	Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	Byte Word, Short, BCD	
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
Register References	R1.0 to R32768.15 R00001 to R32768 R00001 to R32767 R00001 to R32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI2048.15 AI0001 to AI2048 AI0001 to AI2047 AI0001 to AI2045	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ512.15 AQ001 to AQ512 AQ001 to AQ511 AQ001 to AQ509	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
** **See Also**: Special Items.

Advanced Addressing

Bit Access to Registers
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

Device Address	Range	Data Type*	Access
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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P1.0 to P16384.15 P00001 to P16384 P00001 to P16383 P00001 to P16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15 L00001<SUBPRGM> to L16384<SUBPRGM> L00001<SUBPRGM> to L16383<SUBPRGM> L00001<SUBPRGM> to L16381<SUBPRGM>	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items****			

Device Address	Range	Data Type*	Access
(PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**.
**** **See Also**: **Special Items**.

Advanced Addressing

Bit Access to Registers
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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P1.0 to P16384.15 P00001 to P16384 P00001 to P16383 P00001 to P16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float	Read/Write

Device Address	Range	Data Type*	Access
		Double	
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15 L00001<SUBPRGM> to L16384<SUBPRGM> L00001<SUBPRGM> to L16383<SUBPRGM> L00001<SUBPRGM> to L16381<SUBPRGM>	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items**** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**.
**** **See Also**: **Special Items**.

Advanced Addressing

Bit Access to Registers
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)	**Boolean** Byte	Read/Write

Device Address	Range	Data Type*	Access
	Q0001 to Q2033 (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	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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P1.0 to P16384.15 P00001 to P16384 P00001 to P16383 P00001 to P16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15 L00001<SUBPRGM> to L16384<SUBPRGM> L00001<SUBPRGM> to L16383<SUBPRGM> L00001<SUBPRGM> to L16381<SUBPRGM>	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items**** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**.
**** **See Also**: **Special Items**.

Advanced Addressing

Bit Access to Registers
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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P1.0 to P16384.15 P00001 to P16384 P00001 to P16383 P00001 to P16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15	**Boolean**	Read/Write

Device Address	Range	Data Type*	Access
	L00001<SUBPRGM> to L16384<SUBPRGM> L00001<SUBPRGM> to L16383<SUBPRGM> L00001<SUBPRGM> to L16381<SUBPRGM>	**Word**, Short, BCD DWord, Long, LBCD, Float Double	
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items**** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**.
**** **See Also**: **Special Items**.

Advanced Addressing

Bit Access to Registers
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) 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

Device Address	Range	Data Type*	Access
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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P1.0 to P16384.15 P00001 to P16384 P00001 to P16383 P00001 to P16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15 L00001<SUBPRGM> to L16384<SUBPRGM> L00001<SUBPRGM> to L16383<SUBPRGM> L00001<SUBPRGM> to L16381<SUBPRGM>	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items**** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**.

**** **See Also**: Special Items.

Advanced Addressing

Bit Access to Registers
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) 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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P1.0 to P16384.15 P00001 to P16384 P00001 to P16383 P00001 to P16381	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15 L00001<SUBPRGM> to L16384<SUBPRGM> L00001<SUBPRGM> to	**Boolean** **Word**, Short, BCD DWord, Long, LBCD,	Read/Write

Device Address	Range	Data Type*	Access
	L16383<SUBPRGM> L00001<SUBPRGM> to L16381<SUBPRGM>	Float Double	
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192 AI0001 to AI8191 AI0001 to AI8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items**** (PLC Status, Time, etc)			

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**
**** **See Also**: **Special Items**

Advanced Addressing

Bit Access to Registers
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 SRTP compatible device that is not currently listed in the standard model selection menu. The ranges of data for each data type have been expanded to allow a wide range of GE PLCs to be addressed. Although the address ranges shown here may exceed the specific PLC's capability, the driver will respect all messages from the PLC regarding memory range limits.

Note: 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	**Boolean**	Read/Write

Device Address	Range	Data Type*	Access
	G00001 to G32761 (every 8th bit) G00001 to G32753 (every 8th bit)	Byte Word, Short, BCD	
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 T00001 to T32761 (every 8th bit) T00001 to T32753 (every 8th bit)	**Boolean** Byte Word, Short, BCD	Read/Write
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	R1.0 to R32768.15 R00001 to R32768 R00001 to R32767 R00001 to R32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P00001 to P32768 P00001 to P32767 P00001 to P32765	**Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Registers**	L00001<SUBPRGM>.0 to L32768<SUBPRGM>.15 L00001<SUBPRGM> to L32768<SUBPRGM> L00001<SUBPRGM> to L32767<SUBPRGM> L00001<SUBPRGM> to L32765<SUBPRGM>	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI32768.15 AI00001 to AI32768 AI00001 to AI32767 AI00001 to AI32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ32768.15 AQ00001 to AQ32768 AQ00001 to AQ32767 AQ00001 to AQ32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items**** (PLC Status, Time, etc)			
Bulk Memory	W1.0 to W65536.15 W00001 to W65536 W00001 to W65535 W00001 to W65533	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float	Read/Write

Device Address	Range	Data Type*	Access
		Double	

*Default data type of Boolean becomes Byte when an array specification is given.
**The SubProgram name/Block name <SUBPRGM> must match the name assigned in the PLC. This name can be found in VersaPro under Block Properties.
***Program specified in **PLC Settings**
**** **See Also:** **Special Items**

Advanced Addressing

Bit Access to Registers
Default Data Type Override
String Access to Registers
Array Support

Horner OCS Addressing

The Horner OCS model selection has been provided to supply support for Horner Operator Control Stations. The ranges of data for each data type have been expanded to allow a wide range of Horner OCS devices to be addressed. Although the address ranges shown here may exceed the specific device's capability, the driver will respect all messages from the OCS regarding memory range limits.

Note: The default data types for dynamic tags are **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	Q00001 to Q20488 Q00001 to Q2041 (every 8th bit) Q00001 to Q2033 (every 8th bit)	**Boolean** Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M2048 M00001 to M2041 (every 8th bit) M00001 to M2033 (every 8th bit)	**Boolean** Byte Word, Short, BCD	Read/Write
Temporary Coils	T00001 to T2048 T00001 to T2041 (every 8th bit) T00001 to T2033 (every 8th bit)	**Boolean** Byte Word, Short, BCD	Read/Write
Discrete Globals	G1 to G32768 G1 to G32761 G1 to G32753	**Boolean** **Byte** **Word**, Short, BCD	Read/Write
Status References (Same for SA, SB, and SC)	S1 to S32768 S1 to S32761 S1 to S32753	**Boolean** **Byte** **Word**, Short, BCD	Read/Write
Register References	R1.0 to R32768.15 R00001 to R32768 R00001 to R32767 R00001 to R32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

Device Address	Range	Data Type*	Access
Analog Inputs	AI1.0 to AI32768.15 AI00001 to AI32768 AI00001 to AI32767 AI00001 to AI32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ32768.15 AQ00001 to AQ32768 AQ00001 to AQ32767 AQ00001 to AQ32765	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write

*Default data type of Boolean becomes Byte when an array specification is given.

Advanced Addressing

Bit Access to Registers
Default Data Type Override
String Access to Registers
Array Support

VersaMax Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I001 to I2048 I001 to I2041 (every 8th bit) I001 to I2033 (every 8th bit)	**Boolean** Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q2048 Q001 to Q2041 (every 8th bit) Q001 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 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	R1.0 to R2048.15 R001 to R2048 R001 to R2047 R001 to R2045	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Inputs	AI1.0 to AI128.15 AI01 to AI128	**Boolean** **Word**, Short, BCD	Read/Write

Device Address	Range	Data Type*	Access
	AI01 to AI127 AI01 to AI125	DWord, Long, LBCD, Float Double	
Analog Outputs	AQ1.0 to AQ128.15 AQ001 to AQ128 AQ001 to AQ127 AQ001 to AQ125	**Boolean** **Word**, Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Special Items** (PLC Status, Time, etc)			

*The default data type, Boolean, changes to Byte when an array specification is given.
** **See Also**: Special Items

Advanced Addressing
Bit Access to Registers
Default Data Type Override
String Access to Registers
Array Support

Advanced Addressing

Bit Access to Registers
Register based device types such as R, P, L, AI and AQ can be accessed at the bit level by appending a bit number to the register address. The valid bit number range is 0 to 15. The bit number must be preceded by either a period (".") or a colon (":").

Examples

1. R100.12 allows Read/Write access to bit 12 of Register 100.
2. L50SUB1.3 allows Read/Write access to bit 3 of local register L50 in subprogram SUB1.

Note: All device addresses can be prefixed with a % sign if needed, such as %R100. This can aid in converting from other OPC servers or communications drivers.

Default Data Type Override
The default data types for each device type are shown in PACSystems Addressing. The defaults can be overridden by appending 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.)
P100 S	Access P100 as a short
L200SUB1-L210SUB1 M	Access L200-L210 in subprogram SUB1 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 indicator. The length of the string is based on how the device address reference is entered. Each register addressed can contain two 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-R200 M	Access Register R100 as string with a length of 202 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.)
L100SUB1-L100SUB1 M	Access local register L100 in subprogram SUB1 as a string with a length of 2 bytes. (LoHi byte order is assumed.)
R100-R200 M H	Access Register R100 as string with a length of 202 bytes. HiLo byte order is explicitly specified.
R100-R200 M L	Access Register R100 as string with a length of 202 bytes. LoHi byte order is explicitly specified.

Notes: The maximum string length is 255 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. Arrays are supported by the following data types: byte, word, short, DWord, long, float and char. The maximum array size is determined by the "Maximum Bytes Per Request" property in Device Properties. If it is set at its maximum value, arrays can be specified up to 1024 elements for 16 bit data types and 512 elements for 32 bit data types. For more information, refer to **Maximum Bytes Per Request**.

Maximum Array Size for Referenced or Mapped Variables	512 DWords (longs, floats), 1024 words (shorts) or 2048 bytes and chars for a total of 16384 bits
Maximum Array Size for Symbolic Vari-	512 DWords (longs, floats), 1024 words (shorts) or 1024 bytes

ables	and chars for a total of 16384 bits

Note: The number of usable bytes is directly reflective of the block size.

Examples

Address	Address Breakdown
G1 [4] includes the following byte addresses **Note:** G25 indicates the fourth byte beginning at bit 25.	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

Special Items

Though not visible in the server configuration, dynamic tags with these addresses are automatically created and can be browsed by the OPC client. They will be found under the <Channel Name>.<Device Name>._ InternalTags group. If the OPC client does not support browsing or a non-OPC client is being used, users can create their own static and dynamic tags using these addresses. The default data types for dynamic tags are shown in **bold**.

Device Address	Range	Data Type	Access
_OVERSWEEP	Oversweep flag. This is meaningful only when constant sweep mode is active. 1 = Constant Sweep value exceeded 0 = No oversweep condition exists	**Boolean**	Read Only
_CONSWEEP	Constant sweep mode. 1 = Constant Sweep Mode active 0 = Constant Sweep Mode not active	**Boolean**	Read Only
_NEWFT	PLC Fault entry since last read. 1 = PLC Fault Table has changed since last read by the server 0 = PLC Fault Table is unchanged since last read by the server	**Boolean**	Read Only
_NEWIOFT	IO Fault entry since last read. 1 = IO Fault Table has changed since last read by the server 0 = IO Fault Table is unchanged since last read by the server	**Boolean**	Read Only
_FTSTATUS	PLC Fault entry present 1 = One or more fault entries in PLC Fault Table	**Boolean**	Read Only

Device Address	Range	Data Type	Access
	0 = PLC Fault table is empty		
_IOSTATUS	IO Fault entry present 1 = One or more fault entries in IO Fault Table 0 = IO Fault Table is empty	**Boolean**	Read Only
_PROGATTACH	Programmer attachment in system flag. 1 = Programmer attachment found in system 0 = No programmer attachment found in system	**Boolean**	Read Only
_ENSWITCH	Front panel ENABLED/DISABLED switch setting. 1 = Outputs disabled 0 = Outputs enabled	**Boolean**	Read Only
_RUN	Front panel RUN/STOP switch setting. 1 = Run 0 = Stop		
_OEMPROT	OEM protected bit. 1 = OEM protection in effect 0 = No OEM protection	**Boolean**	Read Only
_SPRGCHG	Program changed flag. 1 = Program changed 0 = No program change (90-70 release 2.x and later)	**Boolean**	Read Only
_STATE	Current PLC State 0 = Run I/O enabled 1 = Run I/O disabled 2 = Stop I/O disabled 3 = CPU stop faulted 4 = CPU halted 5 = CPU suspended 6 = Stop I/O enabled	**Byte**, Char	Read Only
_PROGNUM	Control program number SRTP Master is currently logged into. -1 = SRTP Master is not logged into a task 0 = SRTP Master is logged into task 0	Byte, **Char**	Read Only
_PRIVLEVEL	Current privilege level of server for accessing memory in PLC CPU (0-4)	**Byte**, Char	Read Only
_SWEEPTIME	Time taken by the last complete sweep for the main control program task. The value returned is a whole number representing the number of	**Word**, Short	Read Only

Device Address	Range	Data Type	Access
	tenths of a millisecond. The Sweep Time in milliseconds is 1/10 of this value. For example, if _SWEEPTIME is 123, the actual Sweep Time is 12.3 milliseconds.		
_SNPID	CPU Controller ID.	**String**	Read Only
_PROGNAME	Name of control program task in the PLC CPU.	**String**	Read Only
SNUM PROGS	Number of control program tasks defined for the PLC CPU.	**Byte**, Char	Read Only
SPROG FLAGS	Bit flags indicating which control programs tasks have programmers attached. Bit 0 = control program 1.	**Byte**, Char	Read Only
_TIME	Internal time and date of the PLC CPU. Format: SSMMHHDDMoYYDw where SS = Seconds (0-59) MM == Minutes (0-59) HH = Hours (0-23) DD = Day (1- 31) Mo = Month (1-12) YY = Year (0-99) Wd = Day of week (1-Sunday, 2-Monday, etc)	**String**	Read Only
_SECOND	PLC Time: Seconds (0-59)	**Byte**, Char	Read Only
_MINUTE	PLC Time: Minutes (0-59)	**Byte**, Char	Read Only
_HOUR	PLC Time: Hours (0-23)	**Byte**, Char	Read Only
_DAY	PLC Date: Day (1- 31)	**Byte**, Char	Read Only
_MONTH	PLC Date: Month (1-12)	**Byte**, Char	Read Only
_YEAR	PLC Date: Year (0-99)	**Byte**, Char	Read Only
_DOW	PLC Date: Day of week (1-Sunday, 2-Monday, etc)	**Byte**, Char	Read Only

Error Descriptions

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

Address Validation

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

Device Status Messages

Attempting to reacquire symbolic variable mapping information for device '<device name>'. Cannot read or write symbolic variables until complete
Device '<device name>' is not responding
Unable to write to '<address>' on device '<device name>'

Driver Error Messages

Bit access is invalid for the device data type '<native data type>' at address '<address>' in device '<device>'
Bit index '<bit index>' is not valid for the device data type '<native data type>' at address '<address>' in device '<device>'
Device '<device name>' returned error code <error num> reading n byte(s) starting at <address>
The address length of the symbolic variable addressed by '<tag name>' on device '<device>' is bigger than the configured block size for this device. Deactivating tag
The array size of address '<address>' on device '<device name>' must be the same as symbolic variable's array size of [rows][cols]
The data type of the symbolic variable addressed by '<address>' on device '<device name>' is not compatible with tag
The symbolic discrete array size of '<size in bytes>' bytes at address '<address>' on device '<device>' exceeds the configured '<block size>' maximum request bytes
The symbolic variable name in tag address '<address>' does not exist in current configuration of device '<device name>'
Winsock initialization failed (OS Error = n)
Winsock V1.1 or higher must be installed to use the GE Ethernet device driver

Automatic Tag Database Generation Messages

Database Error: Array tags '<orig. tag name><dimensions>' exceed 256 characters. Tags renamed to '<new tag name><dimensions>'
Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created

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

Error Type:
Warning

Possible Cause:

1. A tag address that has been specified statically via DDE references a location that is beyond the range of supported locations for the device.
2. A tag address has been assigned a bit index that is out of range (greater than or equal to 32).

Solution:

1. Verify the address is correct; if it is not, re-enter it in the client application.
2. Remove the bit index syntax from the tag address or change it to one that is in range for the data type.

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

Error Type:
Warning

Possible Cause:

1. The number of elements in an array definition exceeds the range of elements supported by the address type.
2. The number of elements in an array definition exceeds the maximum number of data values that may be obtained from the device in a single request.

Solution:
Reduce the number of referenced elements in the array definition.

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.

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>' 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.

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.

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.

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.

Attempting to reacquire symbolic variable mapping information for device '<device name>'. Cannot read or write symbolic variables until complete

Error Type:
Warning

Possible Cause:
A new configuration is being downloaded to the device or the configuration has been lost (possibly due to power failure).

Solution:
The driver should automatically recover once a new device configuration has been restored.

Device '<device name>' not responding

Error Type:
Serious

Possible Cause:

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

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the IP address given to the named device matches that of the actual device.

Bit access is invalid for the device data type '<native data type>' at address '<address>' in device '<device>'

Error Type:
Warning

Possible Cause:
A tag address has been assigned a bit index that is invalid for its native data type.

Solution:
Remove the bit index syntax from the tag address or modify the symbolic variable in the controller to one of the following native data types: Byte, Int, Uint, Word, Dint, or DWord.

Bit index '<bit index>' is not valid for the device data type '<native data type>' at address '<address>' in device '<device>'

Error Type:
Warning

Possible Cause:
A tag address has been assigned a bit index that is out of range for its native data type.

Solution:
Correct the tag address's bit index to be within the native data type's range.

The native data type ranges are as follows:

- **Byte:** 0-7.
- **Int, Uint, Word:** 0-15.
- **Dint and DWord:** 0-31.

Device '<device name>' returned error code <error num> reading n byte(s) starting at <address>

Error Type:
Error

Possible Cause:
An attempt has been made to read a location that does not exist.

Solution:
Review the address map for the device in question and make necessary adjustments in the client application.

The address length of the symbolic variable addressed by '<tag name>' on device '<device>' is bigger than the configured block size for this device. Deactivating tag

Error Type:
Error

Possible Cause:
The requested tag is larger than the configured block size for this device. This can be caused by the name being too big or requested packet being too big.

Solution:
Increase the block size or change name size.

The array size of address '<address>' on device '<device name>' must be the same as symbolic variable's array size of [rows][cols]

Error Type:
Warning

Possible Cause:
The array dimensions given in the tag address, if any, are not the same as the array size of the referenced symbolic variable.

Solution:
Determine the array dimensions of the variable as currently defined in the device. Adjust the tag address to match.

The data type of the symbolic variable addressed by '<address>' on device '<device name>' is not compatible with tag

Error Type:
Warning

Possible Cause:
The native data type of the symbolic variable, as currently defined in the device, is not compatible with the data type of the tag.

Solution:
Adjust the data type of the tag to match the variable's native data type.

See Also:
Symbolic Variables

The symbolic discrete array size of '<size in bytes>' bytes at address '<address>' on device '<device>' exceeds the configured '<block size>' maximum request bytes.

Error Type:
Warning

Possible Cause:
The number of bytes required to obtain a symbolic discrete array in a single request exceeds the maximum configured block size.

Solution:
1. Increase the Maximum Bytes Per Request property (located in **Device Properties** | **Communications Settings**) to accommodate the actual size of the array.
2. If the symbolic discrete array is larger than 2048 bytes, reconfigure the controller to assign the symbolic array to non-discrete memory. For example, a response containing a symbolic non-discrete DWord array of 64 elements contains 256 bytes (64 elements * 4 bytes per DWord).
3. Separate the array into smaller arrays to reduce the number of referenced elements in each array definition.
4. Read the individual array elements in separate requests (instead of reading the array itself).

Notes:
1. Symbolic variables assigned to discrete memory consume one byte per bit. For example, a byte variable actually requires eight bytes to obtain the data for each bit. Therefore, a response containing a symbolic discrete DWord array of 64 elements contains 2048 bytes (64 elements * 4 bytes per DWord * 8 bytes per byte).
2. For more information, refer to "Importing Array Elements (Index)" in Proficy Logic Developer Array Tag Import.

See Also:
Communications Settings

The symbolic variable name in tag address '<address>' does not exist in current configuration of device '<device name>'.

Error Type:
Warning

Possible Cause:
The named variable does not exist in the current device configuration.

Solution:
1. Verify that variable is defined in current device configuration.
2. Check the spelling of the variable name.

Note: To optimize driver performance, it is recommended that all tags with invalid symbolic variable addresses be removed from the server configuration or not be used by a client application.

Winsock initialization failed (OS Error = n)

Error Type:
Fatal

OS Error	Indication	Possible Solution
10091	Indicates that the underlying network sub-system is not ready for network communication.	Wait a few seconds and restart the driver.
10067	Limit on the number of tasks supported by the Windows Sockets implementation has been reached.	Close one or more applications that may be using Winsock and restart the driver.

Winsock V1.1 or higher must be installed to use the GE Ethernet device driver

Error Type:
Fatal

Possible Cause:
The version number of the Winsock DLL found on the system is less than 1.1.

Solution:
Upgrade Winsock to version 1.1 or higher.

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 Dimensions. The number of X's and Y's 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: 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. Furthermore, 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 or the individual elements will be generated. String data type is currently not supported by the driver. Thus, avoid using String data type if possible.

Database Error: Data type '<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 driver. The tag was not automatically generated.

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

Database Error: Data type '<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 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 tags 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 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 Ethernet PLC. The tag was not automatically generated.

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

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

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

Error Type:
Warning

Possible Cause:
The file specified as the Tag Import File (in the Variable Import Settings property group) is a corrupt import file (*.snf or*.csv) or 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

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 cancelled.

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

Index

3

7

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W