

# **GE SNPX Device Driver Help**

© 2009 Kepware Technologies

# Table of Contents

<b>1</b>	<b>Getting Started</b>	<b>3</b>
	Help Contents	3
	Overview	3
<b>2</b>	<b>Device Setup</b>	<b>3</b>
	Device Setup	3
	Modem Setup	4
	Variable Import Settings	4
<b>3</b>	<b>Data Types Description</b>	<b>4</b>
	Data Types Description	4
<b>4</b>	<b>Address Descriptions</b>	<b>5</b>
	Address Descriptions	5
	GE Micro	5
	311 Addressing	6
	313 Addressing	7
	331 Addressing	7
	341 Addressing	8
	350 Addressing	9
	360 Addressing	9
	731 Addressing	10
	732 Addressing	11
	771 Addressing	11
	772 Addressing	12
	781 Addressing	13
	782 Addressing	13
	GE OPEN Addressing	14
	Advanced Addressing	15
<b>5</b>	<b>Automatic Tag Database Generation</b>	<b>16</b>
	Automatic Tag Database Generation	16
	Tag Hierarchy	17
	Import File-to-Server Name Conversions	17
	<b>Importing VersaPro Tags</b>	<b>18</b>
	Importing VersaPro Tags	18
	VersaPro Import Preparation: VersaPro Steps	18
	VersaPro Import Preparation: OPC Server Steps	20
	Highlighting VersaPro Variables	20
	VersaPro Array Tag Import	21
	<b>Importing LogicDeveloper Tags</b>	<b>21</b>
	Importing LogicDeveloper Tags	21
	LogicDeveloper Import Preparation: LogicDeveloper Steps	22
	LogicDeveloper Import Preparation: OPC Server Steps	23
	Highlighting LogicDeveloper Variables	24
	LogicDeveloper Array Tag Import	24
	<b>Importing Proficy Logic Developer Tags</b>	<b>24</b>
	Importing Proficy Logic Developer Tags	24
	Proficy Logic Developer Import Preparation: Logic Developer Steps	24
	Proficy Logic Developer Import Preparation: OPC Server Steps	26
	Highlighting Proficy Logic Developer Variables	27

Proficy Logic Developer Array Tag Import ..... 27

**6 Error Descriptions..... 27**

**Error Descriptions..... 27**

**Address Validation..... 28**

Address Validation..... 28

Missing address..... 28

Device address '<address>' contains a syntax error..... 28

Address '<address>' is out of range for the specified device or register..... 29

Device address '<address>' is not supported by model '<model name>'..... 29

Data Type '<type>' is not valid for device address '<address>'..... 29

Device address '<address>' is Read Only..... 29

Array size is out of range for address '<address>'..... 30

Array support is not available for the specified address: '<address>'..... 30

**Serial Communications..... 30**

Serial Communications..... 30

COMn does not exist..... 30

Error opening COMn..... 30

COMn is in use by another application..... 31

Unable to set comm parameters on COMn..... 31

Communications error on COMn [<error mask>]..... 31

**Device Status Messages..... 31**

Device Status Messages..... 31

Device '<device name>' not responding..... 31

Unable to write to '<address>' on device '<device name>'..... 32

**Device Specific Messages..... 32**

Device Specific Messages..... 32

Invalid tag in block starting at <address> on device <device name>. Block deactivated..... 32

**Automatic Tag Database Generation Messages..... 32**

Automatic Tag Database Generation Messages..... 32

Unable to generate a tag database for device <device name>. Reason: Low memory resources..... 33

Unable to generate a tag database for device <device name>. Reason: Import file is invalid..... 33

Database Error: Tag '<orig. tag name>' exceeds 31 characters. Tag renamed to '<new tag name>'..... 33

Database Error: Array tags '<orig. tag name><dimensions>' exceed 31 characters. Tags renamed..... 33

Database Error: Datatype '<type>' for tag '<tag name>' is currently not supported. Tag not created..... 34

Database Error: Datatype '<type>' for tag '<tag name>' not found in import file. Setting to default..... 34

Database Error: Logic Developer Variable Arrays are currently not supported. Array Tag(s) '..... 34

Database Error: No Reference Address found for tag '<tag name>' in import file. Tag not created..... 34

Database Error: Only variables with Data Source '<data source name>' are imported. Data Source name is not ..... 35

Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created... 35

**Index**

## GE SNPX Device Driver Help

---

Help version 1.011

### CONTENTS

#### [Overview](#)

What is the GE SNPX Device Driver?

#### [Device Setup](#)

How do I configure a device for use with this driver?

#### [Data Types Description](#)

What data types does this driver support?

#### [Address Descriptions](#)

How do I address a data location on a GE SNPX device?

#### [Automatic Tag Database Generation](#)

How can I easily configure tags for the GE SNPX driver?

#### [Error Descriptions](#)

What error messages does the GE SNPX driver produce?

### Overview

---

The GE SNPX Device Driver was designed specifically for use with 32 bit OPC server products running on Intel microprocessor based computers. For operating system (OS) requirements, please refer to the OPC server help documentation.

This driver is intended for use with GE FANUC Programmable Logic Controllers.

### Device Setup

---

#### Supported Devices

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

#### Communication Protocol

GE SNPX

#### Supported Communication Parameters\*

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

\*Not all devices support the listed configurations.

#### Device IDs

Series 90-30 PLCs support up to 6-character strings (i.e. 1, Ge3, Fanuc1).  
Series 90-70 PLCs support up to 7-character strings (i.e. 1, Ge7, GeFanuc).

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

#### Flow Control

When using an RS232/RS485 converter, the type of flow control that is required will depend upon the needs of the converter. Some converters do not require any flow control and others will require RTS flow. Consult the converter's documentation in order to determine its flow requirements. We recommend using an RS485 converted that provides

automatic flow control.

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

## Automatic Tag Database Generation

### [GE SNPX Variable Import Settings](#)

## Cabling

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

## Modem Setup

This driver supports modem functionality. For more information, please refer to the topic "Modem Support" in the OPC Server Help documentation.

## Variable Import Settings

### Tag Import File

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

### Display Descriptions

Check this option in order to have tag descriptions imported. If necessary, a description will be given to tags with long names stating the original tag name.

**See Also:** [Automatic Tag Database Generation](#).

## Data Types Description

Data Type	Description
Boolean	Single bit
Byte	Unsigned 8 bit value bit 0 is the low bit bit 7 is the high bit
Word	Unsigned 16 bit value bit 0 is the low bit bit 15 is the high bit
Short	Signed 16 bit value bit 0 is the low bit bit 14 is the high bit bit 15 is the sign bit
DWord	Unsigned 32 bit value bit 0 is the low bit bit 31 is the high bit
Long	Signed 32 bit value bit 0 is the low bit bit 30 is the high bit bit 31 is the sign bit
BCD	Two byte packed BCD Value range is 0-9999. Behavior is undefined for values

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

## Address Descriptions

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

### [GE Micro](#)

[311](#)

[313](#)

[331](#)

[341](#)

[350](#)

[360](#)

[731](#)

[732](#)

[771](#)

[772](#)

[781](#)

[782](#)

[GE OPEN](#)

## GE Micro

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R0001 to R9999	<b>Word</b> , Short, BCD	Read/Write

	R0001 to R9998	DWord, Long, LBCD, Float	
Analog Inputs	AI0001 to AI1024 AI0001 to AI1023	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ256 AQ0001 to AQ255	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Array Support**

The following data types support arrays: Byte, Word, Short, DWord, Long, Float.

An array is a collection of contiguous elements of a given data type. The maximum array size is 32 DWords (Longs, Floats), 64 Words (Shorts) or 128 Bytes for a total of 1024 bits. There are two ways to specify an array:

**Examples**

Address	Address Breakdown
G1 [4] includes the following byte addresses	G1,G9,G17,G25
Note: G25 indicates the fourth byte beginning at bit 25.	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

**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)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R001 to R512 R001 to R511	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI01 to AI64 AI01 to AI63	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ001 to AQ032	<b>Word</b> , Short, BCD	Read/Write

AQ001 to AQ031	DWord, Long, LBCD, Float
----------------	--------------------------

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

### Advanced Addressing

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 313 Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R001 to R1024 R001 to R1023	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI01 to AI64 AI01 to AI63	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ001 to AQ032 AQ001 to AQ031	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

### Advanced Addressing

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 331 Addressing

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

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

Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R0001 to R2048 R0001 to R2047	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI001 to AI128 AI001 to AI127	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ001 to AQ64 AQ01 to AQ63	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

**341 Addressing**

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M1024 M0001 to M1017 (every 8th bit) M0001 to M1009 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R0001 to R9999 R0001 to R9998	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI1024 AI0001 to AI1023	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ256 AQ0001 to AQ255	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**[Default Data Type Override](#)[String Access to Registers](#)[Array Support](#)**350 Addressing**

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R0001 to R9999 R0001 to R9998	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI2048 AI0001 to AI2047	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ001 to AQ512 AQ001 to AQ511	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**[Default Data Type Override](#)[String Access to Registers](#)[Array Support](#)**360 Addressing**

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

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

	M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	Byte Word, Short, BCD	
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R32768 R00001 to R32767	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI2048 AI0001 to AI2047	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ512 AQ0001 to AQ511	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

**731 Addressing**

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M2048 M0001 to M2041 (every 8th bit) M0001 to M2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 732 Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I001 to I512 I001 to I505 (every 8th bit) I001 to I497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q512 Q001 to Q505 (every 8th bit) Q001 to Q497 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G1280 G0001 to G1273 (every 8th bit) G0001 to G1265 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M2048 M0001 to M2041 (every 8th bit) M0001 to M2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

### Advanced Addressing

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 771 Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit)	<b>Boolean</b> Byte	Read/Write

	T001 to T241 (every 8th bit)	Word, Short, BCD	
Status References (Same for SA, SB, SC)	S001 to S128 S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

### Advanced Addressing

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 772 Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I0001 to I2048 I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q0001 to Q2048 Q0001 to Q2041 (every 8th bit) Q0001 to Q2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M0001 to M4096 M0001 to M4089 (every 8th bit) M0001 to M4081 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

### Advanced Addressing

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 781 Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I00001 to I12288 I00001 to I12281 (every 8th bit) I00001 to I12273 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q00001 to Q12288 Q00001 to Q12281 (every 8th bit) Q00001 to Q12273 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M12288 M00001 to M12281 (every 8th bit) M00001 to M12273 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R16384 R00001 to R16383	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

### Advanced Addressing

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## 782 Addressing

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I00001 to I12288 I00001 to I12281 (every 8th bit) I00001 to I12273 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q00001 to Q12288 Q00001 to Q12281 (every 8th bit) Q00001 to Q12273 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G0001 to G7680 G0001 to G7673 (every 8th bit) G0001 to G7665 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M12288 M00001 to M12281 (every 8th bit) M00001 to M12273 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T001 to T256 T001 to T249 (every 8th bit) T001 to T241 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Status References	S001 to S128	<b>Boolean</b>	Read Only

(Same for SA, SB, SC)	S001 to S121 (every 8th bit) S001 to S113 (every 8th bit)	Byte Word, Short, BCD	
Register References	R00001 to R16384 R00001 to R16383	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI0001 to AI8192 AI0001 to AI8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ0001 to AQ8192 AQ0001 to AQ8191	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

**GE OPEN Addressing**

The GE OPEN model selection has been provided to supply support for any GE SNPX compatible device that is not currently listed in the standard model selection menu. The 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 your specific PLC's capability, the SNPX driver will respect all messages from the PLC regarding memory range limits.

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

Device Address	Range	Data Type*	Access
Discrete Inputs	I0001 to I32768 I0001 to I32761 (every 8th bit) I0001 to I32753 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q00001 to Q32768 Q00001 to Q32761 (every 8th bit) Q00001 to Q32753 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G00001 to G32768 G00001 to G32761 (every 8th bit) G00001 to G32753 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M32768 M00001 to M32761 (every 8th bit) M00001 to M32753 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T00001 to T32768 T00001 to T32761 (every 8th bit) T00001 to T32753 (every 8th bit)	<b>Boolean</b> 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)	<b>Boolean</b> Byte Word, Short, BCD	Read Only
Register References	R00001 to R32768 R00001 to R32767	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Inputs	AI00001 to AI32768 AI00001 to AI32767	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write
Analog Outputs	AQ00001 to AQ32768 AQ00001 to AQ32767	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float	Read/Write

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

**Advanced Addressing**

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

## Advanced Addressing

### Default Data Type Override

The default data types for each device type are shown in the table above. These 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.)

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

### String Access to Registers

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

### Examples

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

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

### Array Support

The following data types support arrays: Byte, Word, Short, DWord, Long, Float.

An array is a collection of contiguous elements of a given data type. The maximum array size is 32 DWords (Longs, Floats), 64 Words (Shorts) or 128 Bytes for a total of 1024 bits. There are two ways to specify an array:

### Examples

Address	Address Breakdown
G1 [4] includes the following byte addresses	G1,G9,G17,G25
Note: G25 indicates the fourth byte beginning at bit 25.	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

### Automatic Tag Database Generation

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

1. Cimplicity Machine Edition - Logic Developer
2. VersaPro

There are two parts to Automatic Tag Database Generation. The first is to create a variable import file from the application in use. The second is to generate tags based on this variable import file, from the OPC Server. This help topic discusses the second part; it provides an overview of Automatic Tag Database Generation and a detailed look at configuring the OPC Server for Automatic Tag Database Generation. For more information on creating variable import files, refer to [Importing VersaPro Tags](#) or [Importing LogicDeveloper Tags](#). It is recommended that users become familiar with the second part before dealing with the first part.

#### Overview

If the target device supports its own local tag database, the driver will read the device's tag information and use this data to generate OPC tags within the OPC Server. If the device does not natively support its own named tags, the driver will create a list of tags based on information specific to the driver. An example of these two conditions may be as follows:

A data acquisition system that supports its own local tag database. The driver will use the tags names found in the device to build the OPC Server's OPC tags.

1. An Ethernet I/O system that supports detection of I/O module type. The driver in this case will automatically generate OPC tags in the OPC Server that are based on the types of I/O modules plugged into the Ethernet I/O rack.
2. The OPC tags generated are given meaningful names in the OPC Server and are based on the variables imported. These tags are also placed in meaningful tag groups to provide a structured and manageable interface to the tags. The end result is a well-organized OPC Server project that directly reflects the variable import file.

For further information on...	See..
OPC Tag/Group layout in OPC Server	<a href="#">Tag Hierarchy</a>
OPC Tag/Group name creation in OPC Server	<a href="#">Tag Hierarchy</a> <a href="#">Import File-To-Server Name Conversions</a>
Variable name alteration to meet OPC Server name requirements.	<a href="#">Import File-To-Server Name Conversions</a>

### 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. Recall the selection Perform the following action in the Database Creation dialog under Device Properties. The options available are Delete on create, Overwrite as necessary, Do not overwrite and Do not overwrite, log error. After the first OPC Server import/database creation is done, check that the action is set to Do not overwrite or Do not overwrite, log error for future imports. This will allow tags to be imported without deleting or overwriting tags that have been imported previously.

## Tag Hierarchy

The tags created via automatic tag generation follow a specific hierarchy.

The root level groups (or subgroup levels of the group specified in "Add generated tags to the following group") are determined by the variable addresses (i.e. R, G, M, etc.) referenced. For example, every variable that is of address type "R", will be placed in a root level group called "R". Each array tag is provided in its own subgroup of the parent group. The name of the array subgroup provides a description of the array. For instance an array R10[6] defined in the import file would have a subgroup name "R10\_x"; x signifies dimension 1 exists.

### Tags in "R10\_x" Group

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

## Import File-to-Server Name Conversions

### Leading Underscores, Percents, Pound, and Dollar Signs

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

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

### Invalid Characters In Name

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

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

### Long Names

The GE SNPX driver is limited to 31 character group and tag names. Therefore, if a tag name exceeds 31 characters, it must be clipped. Names are clipped as follows:

### Non-Array

1. Determine a 5-digit Unique ID for this tag.
2. Given a tag name: ThisIsALongTagNameAndProbablyExceeds31

3. Clip tag at 31: ThisIsALongTagNameAndProbablyEx
4. Room is made for the Unique ID: ThisIsALongTagNameAndProba#####
5. Insert this ID: ThisIsALongTagNameAndProba00000

### Array

1. Determine a 5-digit Unique ID for this array.
2. Given an array tag name: ThisIsALongTagNameAndProbablyExceeds31\_23\_45\_8
3. Clip tag at 31 while holding on to the element values: ThisIsALongTagNameAndPr\_23\_45\_8
4. Room is made for the Unique ID: ThisIsALongTagName#####\_23\_45\_8
5. Insert this ID: ThisIsALongTagName00001\_23\_45\_8

## Importing VersaPro Tags

---

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

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

See [VersaPro Import Preparation: VersaPro Steps](#)

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

See [VersaPro Import Preparation: OPC Server Steps](#)

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

See [Generating Tag Database While Preserving Previously Generated Tag Databases](#)

### How do I highlight variables in VersaPro?

See [Highlighting VersaPro Variables](#)

### How are VersaPro array variables imported?

See [VersaPro Array Tag Import](#)

## VersaPro Import Preparation: VersaPro Steps

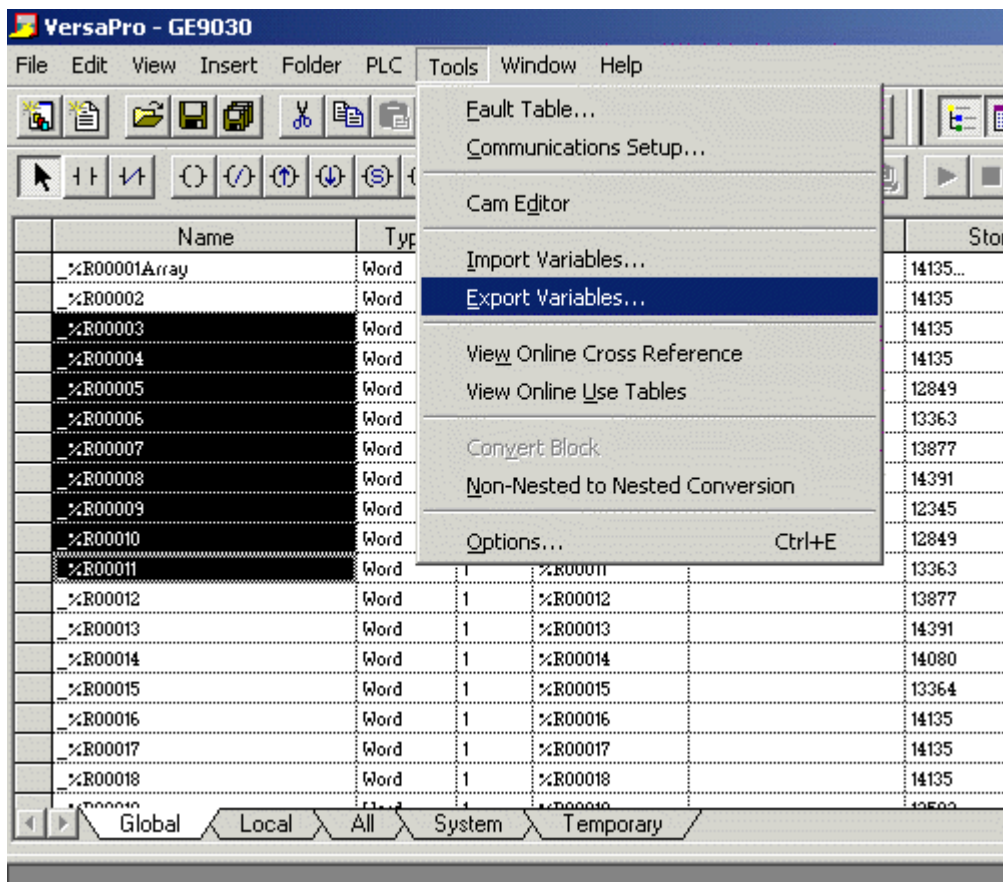
---

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

Name	Type	Len	Address	Description	Start
%R00001Array	Word	30	%R00001		14135...
%R00002	Word	1	%R00002		14135
%R00003	Word	1	%R00003		14135
%R00004	Word	1	%R00004		14135
%R00005	Word	1	%R00005		12849
%R00006	Word	1	%R00006		13363
%R00007	Word	1	%R00007		13877
%R00008	Word	1	%R00008		14391
%R00009	Word	1	%R00009		12345
%R00010	Word	1	%R00010		12849
%R00011	Word	1	%R00011		13363
%R00012	Word	1	%R00012		13877
%R00013	Word	1	%R00013		14391
%R00014	Word	1	%R00014		14080
%R00015	Word	1	%R00015		13364
%R00016	Word	1	%R00016		14135
%R00017	Word	1	%R00017		14135
%R00018	Word	1	%R00018		14135
%R00019	Word	1	%R00019		14500

Global Local All System Temporary

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



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

## VersaPro Import Preparation: OPC Server Steps

1. Open up the **Device Properties** for the device of interest for which tags will be generated.
2. Select the **Database Settings** tab.
3. Enter or browse for the location of the **VersaPro SNF file** newly created.
4. Select the **Database Creation** tab and utilize as instructed above.
5. The OPC Server will state in the event log that it is attempting to perform a tag import. When finished, it will state that the tag import has completed. All variable exported out of VersaPro will appear in the OPC Server in the layout discussed in [Tag Hierarchy](#).

**See Also:** [Variable Import Settings](#)

## Highlighting VersaPro Variables

Variables can be highlighted in VersaPro using the actions.

### Single Variable Selecting

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

### Pick-n-Choose

N/A.

### Selecting a Range of Variables

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

### Selecting All Variables

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

## VersaPro Array Tag Import

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

### Array Tag

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

### Individual Elements

Element tags are simply the base address + element number. This has the following form, where  $n = \text{Length} - 1$ .

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

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

### Example

#### Variable Imported:

MyArrayTag, Length = 10, Address = R1

#### Result as Array Tag:

MyArrayTag [10]

#### Result as Individual Elements:

R1  
R2  
R3  
R4  
R5  
R6  
R7  
R8  
R9  
R10

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

## Importing LogicDeveloper Tags

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

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

See [LogicDeveloper Import Preparation: LogicDeveloper Steps](#)

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

See [LogicDeveloper Import Preparation: OPC Server Steps](#)

### How do I highlight variables in LogicDeveloper?

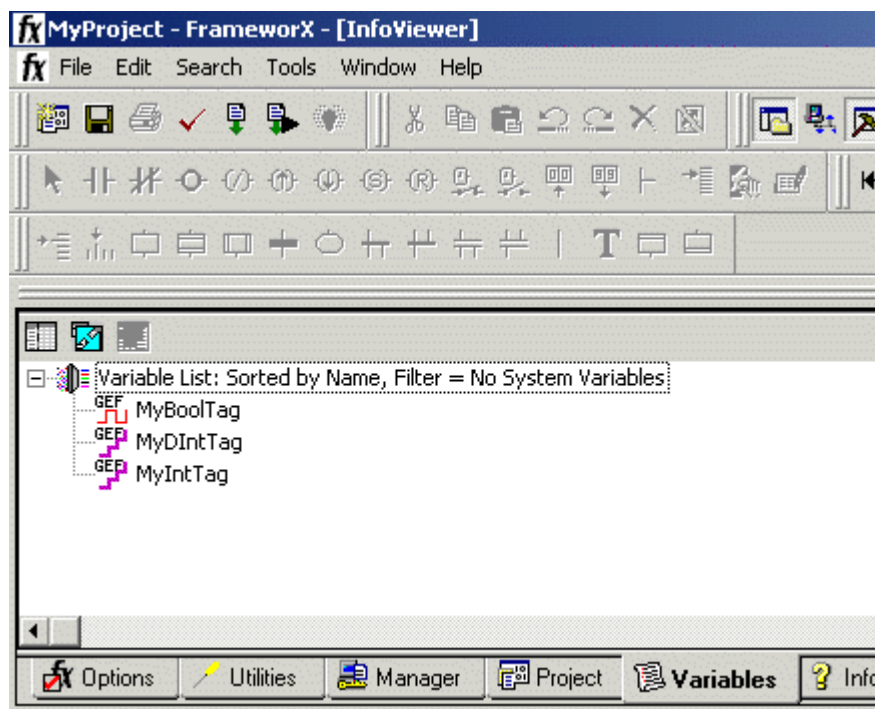
See [Highlighting LogicDeveloper Variables](#)

### How are LogicDeveloper array variables imported?

See [LogicDeveloper Array Tag Import](#)

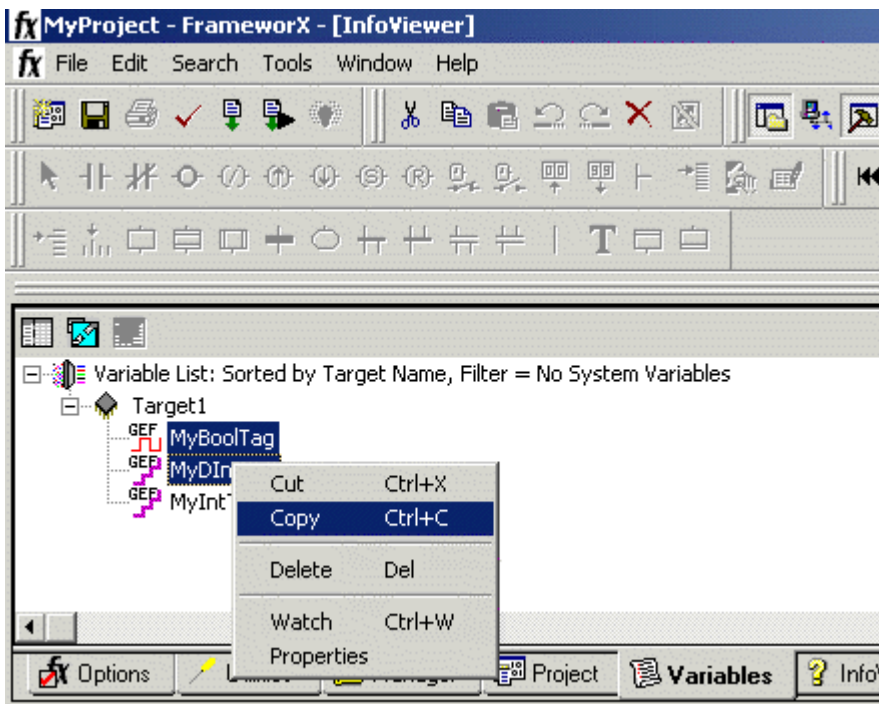
## LogicDeveloper Import Preparation: LogicDeveloper Steps

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



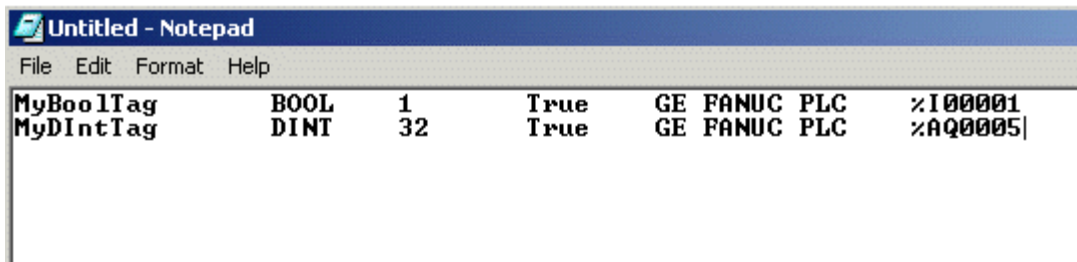
**Important Note about Targets:** Each FrameworkX project contains one or more targets. A target is essentially the device on which the application will run. Variables are created on the target-level, so in order to specify the variables to export; the target of interest must first be decided. In order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

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



7. Open a word processing program such as Notepad or Wordpad. This will be the place where the variables will ultimately be saved for importing. Click **Edit | Paste**.

8. The variables on the Clipboard will now be pasted to the document, TAB delimited. Do not modify the contents. Modifications may cause the import to fail.



9. Save this text document with the **TXT extension** (i.e. .txt) in **ANSI form**.

10. The variables highlighted and copied in Logic Developer are now contained within this text document to be imported into the OPC Server.

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

1. Open up the **Device Properties** for the device of interest for which tags will be generated.
2. Select the **Database Settings** tab.
3. Enter or browse for the location of the **Logic Developer TXT** file newly created.
4. Select the **Database Creation** tab and utilize as instructed above.
5. The OPC Server will state in the event log that it is attempting to perform a tag import. When finished, it will state that the tag import has completed. All variable exported out of Logic Developer will appear in the OPC Server in the layout discussed in [Tag Hierarchy](#).

**See Also:** [Variable Import Settings](#).

## Highlighting LogicDeveloper Variables

---

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

### Single Variable Selecting

Left-click on the variable of interest.

### Pick-n-Choose

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

### Selecting a Range of Variables

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

### Selecting All Variables

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

## LogicDeveloper Array Tag Import

---

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

## Importing Proficy Logic Developer Tags

---

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

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

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

See [Proficy Logic Developer Import Preparation: Logic Developer Steps](#)

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

See [Proficy Logic Developer Import Preparation: OPC Server Steps](#)

### How do I highlight variables in Proficy Logic Developer?

See [Highlighting Proficy Logic Developer Variables](#)

### How are Proficy Logic Developer array variables imported?

See [Proficy Logic Developer Array Tag Import](#)

## Proficy Logic Developer Import Preparation: Logic Developer Steps

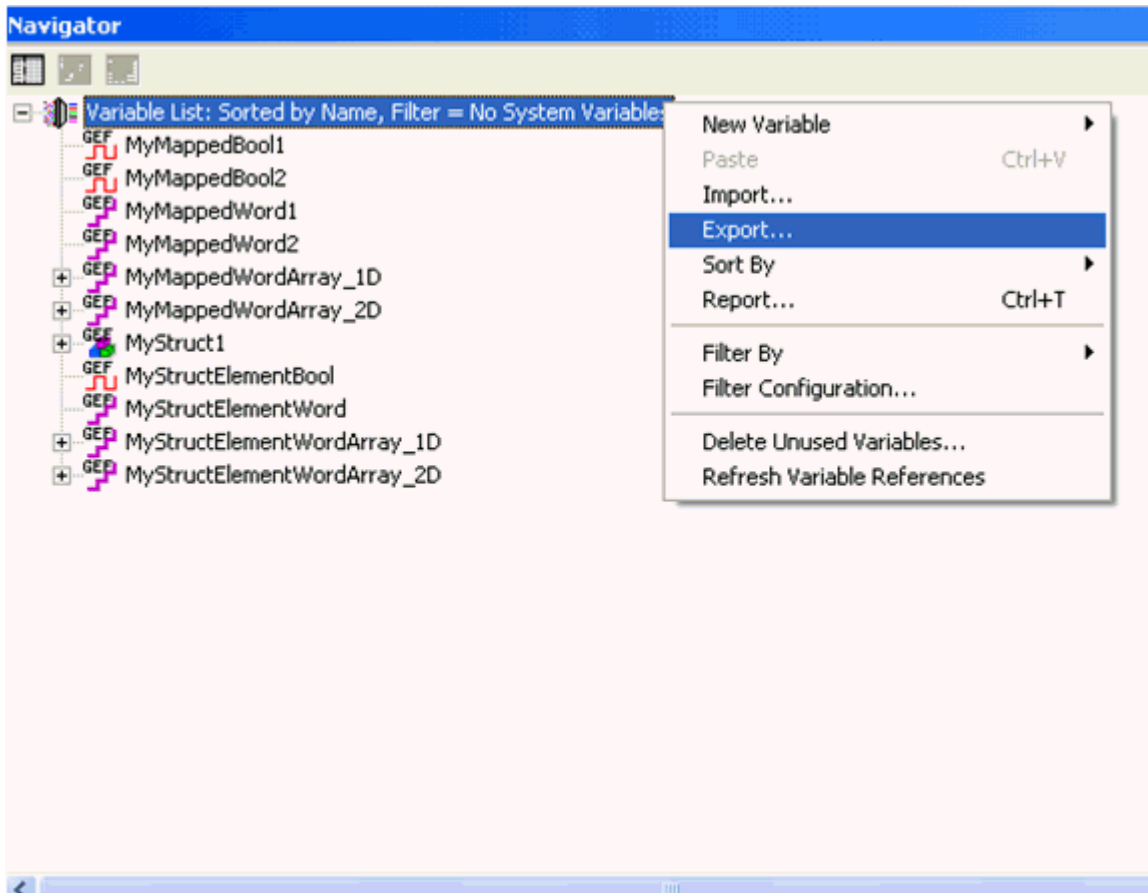
---

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

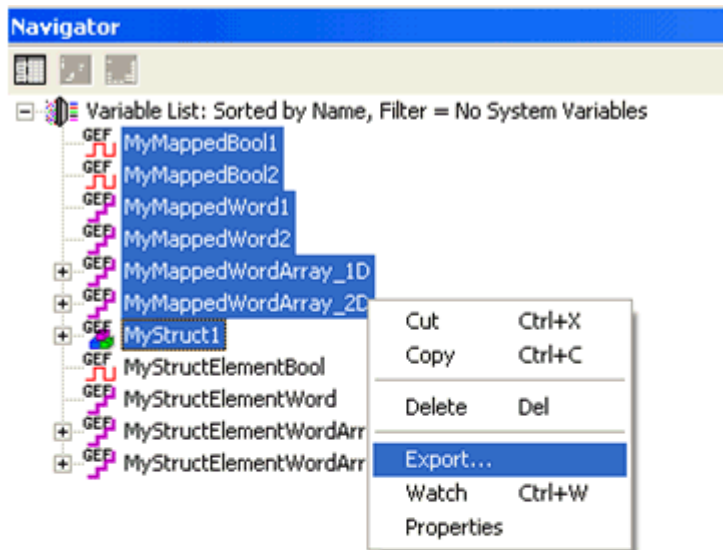
**Important Note about Targets:** Each Logic Developer project contains one or more targets. A target is essentially the device on which the application will run. Variables are created on the target-level, so in order to specify the variables to export, the target of interest must first be defined. In order for the target variables to be imported, the

variables must have GE FANUC PLC as the Data Source. This can be verified by left clicking on the variable and looking at its Data Source properties in the Inspector window. Internal variables will not be imported.

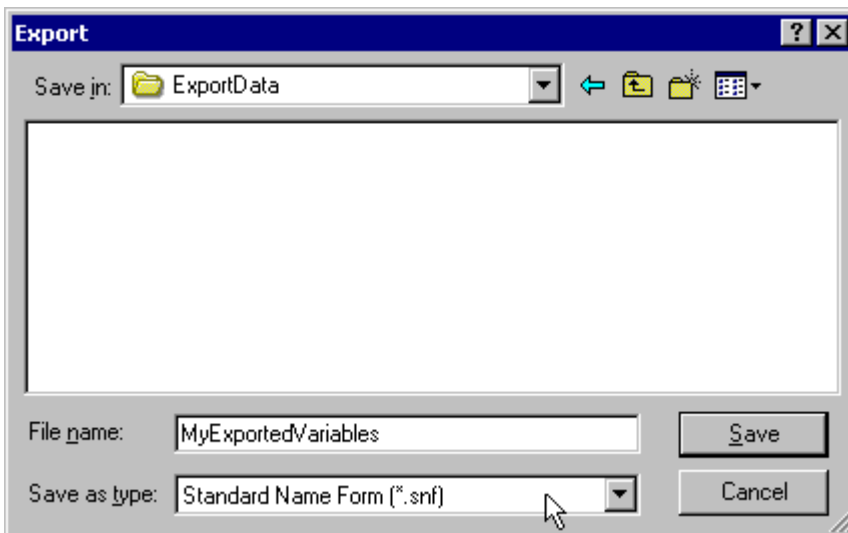
5. To export **all of the variables**, right-click on the Variable List tree root and click the **Export...** pop-up menu option.



6. To export only **selected variables**, highlight the tags of interest in the target of interest. Then, right-click on one of the selected variables and click the **Export...** pop-up menu option.



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



### **Proficiency Logic Developer Import Preparation: OPC Server Steps**

1. Open up the **Device Properties** for the device of interest for which tags will be generated.
2. Select the **Variable Import Settings** tab.

3. Enter or browse for the location of the export file newly created (\*.snf or \*.csv).
4. Select the **Database Creation** tab and click **Auto Create** in order to import variables now. There are more setting on that page which can be used to automatically create the database later.
5. The OPC Server will state in the event log that it is attempting to perform a tag import. When finished, it will state that the tag import has completed. All variables exported out of Logic Developer will appear in the OPC Server in the layout discussed in [Tag Hierarchy](#).

## **Highlighting Proficy Logic Developer Variables**

---

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

### **Single Variable Selecting**

Left-click on a variable of interest.

### **Pick-n-Choose**

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

### **Selecting a Range of Variables**

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

### **Selecting All Variables**

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

## **Proficy Logic Developer Array Tag Import**

---

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

### **Referenced Variable Arrays**

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

### **Symbolic Variable Arrays**

A single array tag will be generated for each symbolic variable array in the import file. All symbolic variable array tags will be 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.

## **Error Descriptions**

---

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

### **Address Validation**

#### [Missing address](#)

[Device address '<address>' contains a syntax error](#)

[Address '<address>' is out of range for the specified device or register](#)

[Device address '<address>' is not supported by model '<model name>'](#)

[Data Type '<type>' is not valid for device address '<address>'](#)

[Device address '<address>' is Read Only](#)

[Array size is out of range for address '<address>'](#)

[Array support is not available for the specified address: '<address>'](#)

### **Serial Communications**

[COMn does not exist](#)

[Error opening COMn](#)

[COMn is in use by another application](#)  
[Unable to set comm parameters on COMn](#)  
[Communications error on COMn \[<error mask>\]](#)

#### Device Status Messages

[Device '<device name>' is not responding](#)  
[Unable to write to '<address>' on device '<device name>'](#)

#### Device Specific Messages

[Invalid tag in block starting at <address> on device <device name>. Block deactivated](#)

#### Automatic Tag Database Generation Messages

[Unable to generate a tag database for device <device name>. Reason: Low memory resources](#)  
[Unable to generate a tag database for device <device name>. Reason: Import file is invalid or corrupt](#)  
[Database Error: Tag '<orig. tag name>' exceeds 31 characters. Tag renamed to '<new tag name>'](#)  
[Database Error: Array tags '<orig. tag name><dimensions>' exceed 31 characters. Tags renamed to '<new tag name><dimensions>'](#)  
[Database Error: Data type '<type>' for tag '<tag name>' not found in import file. Setting to Default Type '<type>'](#)  
[Database Error: Data type '<type>' for tag '<tag name>' is currently not supported. Tag not created](#)  
[Database Error: Logic Developer Variable Arrays are currently not supported. Array Tag\(s\) '<array tag name>' not created](#)  
[Database Error: No Reference Address found for tag '<tag name>' in import file. Tag not created](#)  
[Database Error: Only variables with Data Source '<data source name>' are imported. Data Source '<data source name>' is not supported. Tag '<tag name>' not created](#)  
[Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created](#)

### **Address Validation**

---

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

#### Address Validation

[Missing address](#)  
[Device address '<address>' contains a syntax error](#)  
[Address '<address>' is out of range for the specified device or register](#)  
[Device address '<address>' is not supported by model '<model name>'](#)  
[Data Type '<type>' is not valid for device address '<address>'](#)  
[Device address '<address>' is Read Only](#)  
[Array size is out of range for address '<address>'](#)  
[Array support is not available for the specified address: '<address>'](#)

#### Missing address

---

##### Error Type:

Warning

##### Possible Cause:

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

##### Solution:

Re-enter the address in the client application.

#### Device address '<address>' contains a syntax error

---

##### Error Type:

Warning

**Possible Cause:**

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

**Solution:**

Re-enter the address in the client application.

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

Modify the requested data type in the client application.

---

**Device address '<address>' is Read Only**

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

Change the access mode in the client application.

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

**Serial Communications**

---

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

**Serial Communications**

[COMn does not exist](#)

[Error opening COMn](#)

[COMn is in use by another application](#)

[Unable to set comm parameters on COMn](#)

[Communications error on COMn \[<error mask>\]](#)

---

**COMn does not exist**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

Verify that the proper COM port has been selected.

---

**Error opening COMn**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

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

**COMn is in use by another application**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

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

**Unable to set comm parameters on COMn**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

Verify the serial parameters and make any necessary changes.

**Communications error on COMn [<error mask>]**

---

**Error Type:**

Serious

**Error Mask Definitions:**

**B** = Hardware break detected.

**F** = Framing error.

**E** = I/O error.

**O** = Character buffer overrun.

**R** = RX buffer overrun.

**P** = Received byte parity error.

**T** = TX buffer full.

**Possible Cause:**

1. The serial connection between the device and the host PC is bad.
2. The communications parameters for the serial connection are incorrect.

**Solution:**

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

**Device Status Messages**

---

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

**Device Status Messages**

[Device '<device name>' is not responding](#)

[Unable to write to '<address>' on device '<device name>'](#)

**Device '<device name>' not responding**

---

**Error Type:**

Serious

**Possible Cause:**

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

**Solution:**

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

---

**Unable to write to '<address>' on device '<device name>'****Error Type:**

Serious

**Possible Cause:**

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

**Solution:**

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

---

**Device Specific Messages**

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

**Device Specific Messages**

[Invalid tag in block starting at <address> on device <device name>. Block deactivated](#)

---

**Invalid tag in block starting at <address> on device <device name>. Block deactivated****Error Type:**

Serious

**Possible Cause:**

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

**Solution:**

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

---

**Automatic Tag Database Generation Messages**

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

**Automatic Tag Database Generation Messages**

[Unable to generate a tag database for device <device name>. Reason: Low memory resources](#)

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

[Database Error: Tag '<orig. tag name>' exceeds 31 characters. Tag renamed to '<new tag name>'](#)

[Database Error: Array tags '<orig. tag name><dimensions>' exceed 31 characters. Tags renamed to '<new tag name><dimensions>'](#)

[Database Error: Data type '<type>' for tag '<tag name>' not found in import file. Setting to Default Type '<type>'](#)

[Database Error: Data type '<type>' for tag '<tag name>' is currently not supported. Tag not created](#)

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

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

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

[Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created](#)

---

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

#### **Error Type:**

Warning

#### **Possible Cause:**

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

#### **Solution:**

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

---

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

#### **Error Type:**

Warning

#### **Possible Cause:**

The file specified as the Tag Import File (in the Database Settings device properties page) 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 Preparation](#)

---

### **Database Error: Tag '<orig. tag name>' exceeds 31 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 31-character limitation and will be renamed to one that is valid.

#### **Solution:**

None.

#### **See Also:**

[Import File-to-Server Name Conversions](#)

---

### **Database Error: Array tags '<orig. tag name><dimensions>' exceed 31 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 31-character limitation and will be renamed to one that is valid. <Dimensions> define the number of dimensions for the given array tag: XXX for 1 dimension, XXX\_YYY for 2. The number of 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: Datatype '<type>' for tag '<tag name>' is currently not supported. Tag not created**

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

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

Warning

**Possible Cause:**

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

**Solution:**

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

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

Warning

**Possible Cause:**

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

**Solution:**

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

**Note:**

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

# Index

## - 3 -

311 Addressing	6
313 Addressing	7
331 Addressing	7
341 Addressing	8
350 Addressing	9
360 Addressing	9

## - 7 -

731 Addressing	10
732 Addressing	11
771 Addressing	11
772 Addressing	12
781 Addressing	13
782 Addressing	13

## - A -

Address '<address>' is out of range for the specified device or register	29
Address Descriptions	5
Address Validation	28
Advanced Addressing	15
Automatic Tag Database Generation	16
Automatic Tag Database Generation Messages	32

## - B -

Boolean	4
Byte	4

## - C -

Char	4
Communications error on COMn [<error mask>]	31
COMn does not exist	30
COMn is in use by another application	31

## - D -

Data Types Description	4
Database Error	
Array tags '<orig. tag name><dimensions>' exceed 31 characters. Tags renamed to '<new tag name><dimensions>'	33
Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created	35
Datatype '<type>' for tag '<tag name>' is currently not supported. Tag not created	34
Datatype '<type>' for tag '<tag name>' not found in import file. Setting to Default Type '<type>'	34
Logic Developer Variable Arrays are currently not supported. Array Tag(s) '<array tag name>' not created.	34
No Reference Address found for tag '<tag name>' in import file. Tag not created	34
Only variables with Data Source '<data source name>' are imported. Data Source '<data source name>' is not...	35
Tag '<orig. tag name>' exceeds 31 characters. Tag renamed to '<new tag name>'	33
Device '<device name>' not responding	31
Device address '<address>' contains a syntax error	28
Device ID	3
Device Specific Messages	32
Device Status Messages	31

## - E -

Error Descriptions	27
Error opening COMn	30

## - F -

framing	31
---------	----

## - G -

GE Micro	5
GE OPEN Addressing	14
GE SNPX Variable Import Settings	4

**- H -**

Highlighting LogicDeveloper Variables 24  
Highlighting VersaPro Variables 20

**- I -**

Import File-to-Server Name Conversions 17  
Importing LogicDeveloper Tags 21  
Importing VersaPro Tags 18

**- L -**

LogicDeveloper Array Tag Import 24  
LogicDeveloper Import Preparation  
LogicDeveloper Steps 22  
LogicDeveloper Import Preparation OPC Server  
Steps 23

**- M -**

mask 31  
Modem Setup 4

**- N -**

Network 3

**- O -**

overrun 31  
Overview 3

**- P -**

parity 31

**- S -**

Serial Communications 30  
Short 4

**- T -**

Tag Hierarchy 17

**- U -**

Unable to generate a tag database for device  
<device name>. Reason  
    Import file is invalid or corrupt 33  
    Low memory resources 33  
Unable to set comm parameters on COMn 31  
Unable to write tag '<address>' on device '<device  
name>' 32

**- V -**

VersaPro Array Tag Import 21  
VersaPro Import Preparation OPC Server Steps  
20  
VersaPro Import Preparation VersaPro Steps  
18

**- W -**

Word 4