

# **GE Ethernet Driver Help**

**© 2013 Kepware Technologies**

# Table of Contents

<b>Table of Contents</b> .....	<b>2</b>
<b>GE Ethernet Driver Help</b> .....	<b>5</b>
<b>Overview</b> .....	<b>5</b>
<b>Device Setup</b> .....	<b>6</b>
<b>Communications Parameters</b> .....	<b>7</b>
<b>Variable Import Settings</b> .....	<b>8</b>
<b>PLC Settings</b> .....	<b>8</b>
<b>Automatic Tag Database Generation</b> .....	<b>10</b>
<b>Tag Hierarchy</b> .....	<b>10</b>
<b>Import File-to-Server Name Conversions</b> .....	<b>11</b>
<b>Importing VersaPro Tags</b> .....	<b>12</b>
<b>VersaPro Import Preparation: VersaPro Steps</b> .....	<b>12</b>
<b>VersaPro Import Preparation: OPC Server Steps</b> .....	<b>14</b>
<b>Highlighting VersaPro Variables</b> .....	<b>14</b>
<b>VersaPro Array Tag Import</b> .....	<b>15</b>
<b>Importing Cimplicity Logic Developer Tags</b> .....	<b>15</b>
<b>Cimplicity Logic Developer Import Preparation: Logic Developer Steps</b> .....	<b>16</b>
<b>Cimplicity Logic Developer Import Preparation: OPC Server Steps</b> .....	<b>17</b>
<b>Highlighting Cimplicity Logic Developer Variables</b> .....	<b>18</b>
<b>Cimplicity Logic Developer Array Tag Import</b> .....	<b>18</b>
<b>Importing Proficy Logic Developer Tags</b> .....	<b>18</b>
<b>Proficy Logic Developer Import Preparation: Logic Developer Steps</b> .....	<b>18</b>
<b>Proficy Logic Developer Import Preparation: OPC Server Steps</b> .....	<b>21</b>
<b>Highlighting Proficy Logic Developer Variables</b> .....	<b>21</b>
<b>Proficy Logic Developer Array Tag Import</b> .....	<b>21</b>
<b>Optimizing Your GE Ethernet Communications</b> .....	<b>23</b>
<b>Data Types Description</b> .....	<b>25</b>
<b>Address Descriptions</b> .....	<b>26</b>
<b>PACSystems Addressing</b> .....	<b>26</b>
<b>Symbolic Variables</b> .....	<b>27</b>
<b>311 Addressing</b> .....	<b>30</b>
<b>313 Addressing</b> .....	<b>31</b>
<b>331 Addressing</b> .....	<b>32</b>
<b>341 Addressing</b> .....	<b>32</b>
<b>350 Addressing</b> .....	<b>33</b>
<b>360 Addressing</b> .....	<b>34</b>
<b>731 Addressing</b> .....	<b>35</b>
<b>732 Addressing</b> .....	<b>36</b>
<b>771 Addressing</b> .....	<b>37</b>

---

772 Addressing.....	38
781 Addressing.....	39
782 Addressing.....	40
GE OPEN Addressing.....	41
Horner OCS Addressing.....	42
VersaMax Addressing.....	43
Advanced Addressing.....	44
Special Items.....	45
<b>Error Descriptions.....</b>	<b>48</b>
<b>Address Validation Error Messages.....</b>	<b>48</b>
Address '<address>' is out of range for the specified device or register.....	49
Array size is out of range for address '<address>'.....	49
Array support is not available for the specified address: '<address>'.....	49
Data Type '<type>' is not valid for device address '<address>'.....	49
Device address '<address>' contains a syntax error.....	49
Device address '<address>' is not supported by model '<model name>'.....	50
Device address '<address>' is Read Only.....	50
Missing address.....	50
<b>Device Status Messages.....</b>	<b>50</b>
Attempting to reacquire symbolic variable mapping information for device '<device name>'. Cannot read or write symbolic variables until complete.....	50
Device '<device name>' not responding.....	50
Unable to write to '<address>' on device '<device name>'.....	51
<b>Driver Error Messages.....</b>	<b>51</b>
Bit access is invalid for the device data type '<native data type>' at address '<address>' in device '<device>'.....	51
Bit index '<bit index>' is not valid for the device data type '<native data type>' at address '<address>' in device '<device>'.....	52
Device '<device name>' returned error code <error num> reading n byte(s) starting at <address>.....	52
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.....	52
The array size of address '<address>' on device '<device name>' must be the same as symbolic variable's array size of [rows][cols].....	52
The data type of the symbolic variable addressed by '<address>' on device '<device name>' is not compatible with tag.....	52
The symbolic discrete array size of '<size in bytes>' bytes at address '<address>' on device '<device>' exceeds the configured '<block size>' maximum request bytes.....	53
The symbolic variable name in tag address '<address>' does not exist in current configuration of device '<device name>'.....	53
Winsock initialization failed (OS Error = n).....	53
Winsock V1.1 or higher must be installed to use the GE Ethernet device driver.....	54
<b>Automatic Tag Database Generation Messages.....</b>	<b>54</b>
Database Error: Array tags '<orig. tag name><dimensions>' exceed 31 characters. Tags renamed to '<new.'.....	

tag name><dimensions>'.....	54
Database Error: Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created.....	54
Database Error: Data type '<type>' for tag '<tag name>' is currently not supported. Tag not created.....	55
Database Error: Data type '<type>' for tag '<tag name>' not found in import file. Setting to default.....	55
Database Error: Logic Developer Variable Arrays are currently not supported. Array Tag(s) '<array tag name>' not created.....	55
Database Error: No Reference Address found for tag '<tag name>' in import file. Tag not created.....	55
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.....	56
Database Error: Tag '<orig. tag name>' exceeds 31 characters. Tag renamed to '<new tag name>'.....	56
Unable to generate a tag database for device <device name>. Reason: Import file is invalid or corrupt....	56
Unable to generate a tag database for device <device name>. Reason: Low memory resources.....	56
<b>Index</b> .....	<b>57</b>

## GE Ethernet Driver Help

---

Help version 1.061

### CONTENTS

#### [Overview](#)

What is the GE Ethernet Driver?

#### [Device Setup](#)

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

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

How can I easily configure tags for the GE Ethernet Driver?

#### [Optimizing Your GE Ethernet Communications](#)

How do I get the best performance from the GE Ethernet Driver?

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

What data types does this driver support?

#### [Address Descriptions](#)

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

#### [Error Descriptions](#)

What error messages does the GE Ethernet Driver produce?

### Overview

---

The GE Ethernet Driver provides an easy and 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.

## Device 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.

### Connection Timeout

This parameter specifies the time that the driver will wait for a connection to be made with a device. Depending on network load, the connection time may vary with each attempt. The valid range is 1 to 60 seconds. The default setting is 3 seconds.

### Request Timeout

This parameter specifies the time that the driver will wait on a response from the device before giving up and going on to the next request. Longer timeouts will only affect performance if a device is not responding. The valid range is 50 to 30000 milliseconds. The default setting is 1000 milliseconds.

### Retry Attempts

This parameter specifies the number of times that the driver will retry a message before giving up and going on to the next message. The valid range is 1 to 10. The default setting is 3 retries.

### Device IDs

Up to 1024 devices may be defined on a given channel. Each device on the channel must be uniquely identified by its own IP address. The LogicMaster 90 TCP software supplied by GE can be used to configure the IP address of an Ethernet module. In general, the Device ID has the following format: `YYY.YYY.YYY.YYY`, where `YYY` designates the device IP address. Each `YYY` byte should be in the range of 0 to 255.

### Automatic Tag Database Generation

For more information, refer to [Variable Import Settings](#).

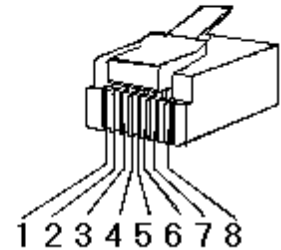
### Cable Diagrams

## Patch Cable (Straight Through)

TD + 1	OR/WHT	OR/WHT	1 TD +
TD - 2	OR	OR	2 TD -
RD + 3	GRN/WHT	GRN/WHT	3 RD +
4	BLU	BLU	4
5	BLU/WHT	BLU/WHT	5
RD - 6	GRN	GRN	6 RD -
7	BRN/WHT	BRN/WHT	7
8	BRN	BRN	8

RJ45 RJ45

10 BaseT



## Crossover Cable

TD + 1	OR/WHT	GRN/WHT	1 TD +
TD - 2	OR	GRN	2 TD -
RD + 3	GRN/WHT	OR/WHT	3 RD +
4	BLU	BLU	4
5	BLU/WHT	BLU/WHT	5
RD - 6	GRN	OR	6 RD -
7	BRN/WHT	BRN/WHT	7
8	BRN	BRN	8


RJ45 RJ45

8-pin RJ45

**Communications Parameters**

This dialog is used to define the device's port number and the maximum number of bytes of data that can be received in a single request.

**New Device - Communications Parameters** X



Set TCP/IP Port number of the device. The default is 18245.

Max request size refers to the number of bytes of data that may be received in a single request.

You can improve performance on some ethernet modules by selecting a smaller request size.

TCP/IP port:

Max. bytes per request:

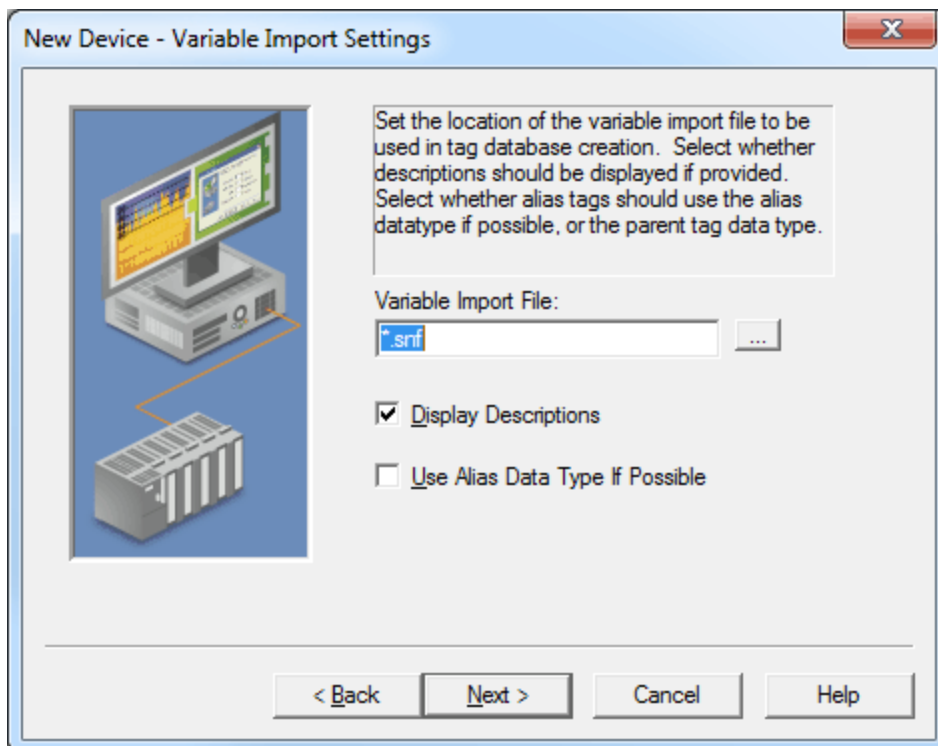
< Back
Next >
Cancel
Help

Descriptions of the parameters are as follows:

- **TCP/IP port:** This parameter specifies the TCP/IP port number that the remote device is configured to use. The default setting is 18245.
- **Max. bytes per request:** This parameter specifies 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.

## Variable Import Settings

This dialog is used to locate the Variable Import File, specify the tag description import setting, and define how to use an alias data type.



Descriptions of the parameters are as follows:

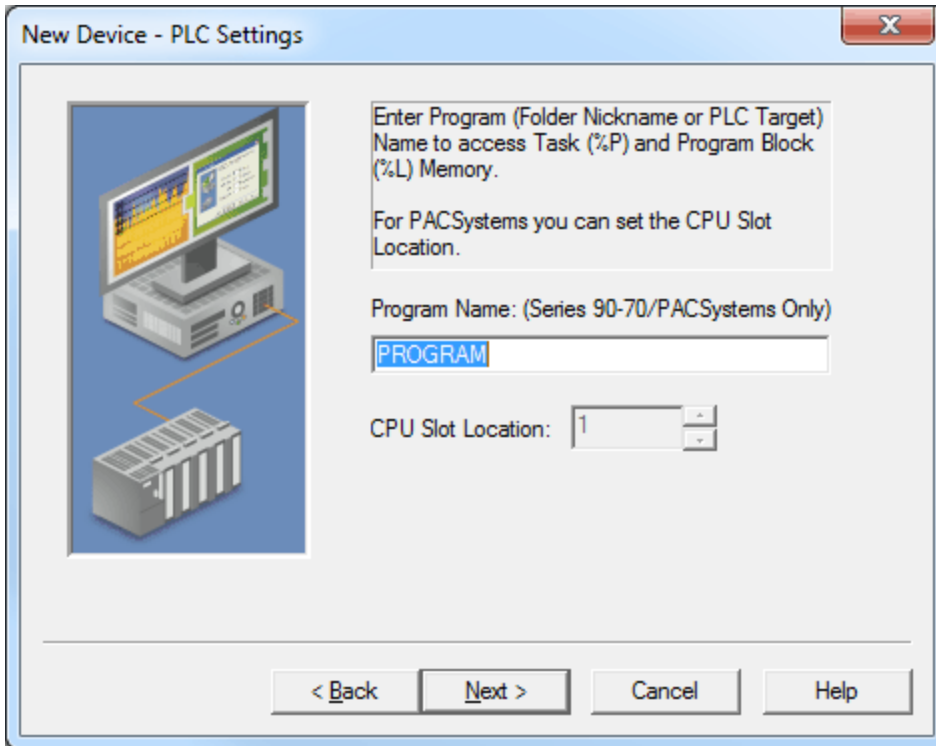
- **Variable Import File:** This parameter specifies the exact location of the variable import file (.snf or .csv file extension) or Logic Developer variable import file (.txt or other file extension) from which variables will be imported. It is this file that will be used when Automatic Tag Database Generation is instructed to create the tag database. All tags will be imported and expanded according to their respective data types.
- **Display Descriptions:** When checked, this option will import tag descriptions. If necessary, a description will be given to tags with long names that state the original tag name.
- **Use Alias Data Type If Possible:** When checked, this option will use the data type assigned to an alias tag in the import file. If the alias data type is incompatible with the source tag data type, the source tag data type will be used instead.

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

## PLC Settings

This dialog is used to define the Program Name and the CPU Slot Location.





Descriptions of the parameters are as follows:

- Program Name:** This parameter is required in order to access program block registers (P) or sub-program block registers (L). It is also referred to as the Control Program Task Name, and must be seven characters or less in length. The name will default to uppercase. It may be obtained by accessing `_PROGRAM` or from the following programming packages:

**Proficy Machine Edition - Logic Developer**  
**Cimplicity Machine Edition - Logic Developer**

**For 90-70 PLCs:** The program name is referred to as the PLC Target Name. It is located under Target Properties (Inspector).

**For PACSystems:** The program name is located under Properties for "Program Block \_MAIN".

**Note:** In VersaPro, the program name is referred to as the Folder Nickname. It is located under **PLC | Status Info**.

- CPU Slot Location:** This parameter specifies the physical location of the device on the rack. 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.

**See Also:** [Special Items](#)

## Automatic Tag Database Generation

The GE Ethernet Device Driver generates its tags offline based on variables imported from a text file. It is offline 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. It is recommended that users become familiar with the second part before starting the first part.

**Note:** For information on creating variable import files, refer to [Importing VersaPro Tags](#), [Importing Proficy Logic Developer Tags](#) or [Importing Cimplicity Logic Developer Tags](#).

### 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 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 driver will use the tag names found in the device to build the server's OPC tags.
2. If an Ethernet I/O system supports detection of its own available I/O module types, the driver will automatically generate OPC tags in the server. These tags will be based on the types of I/O modules plugged into the Ethernet I/O rack.

The generated tags are assigned names in the OPC server that are based on the variables imported. These tags are also placed in meaningful tag groups to provide a structured and manageable interface. The end result is a well-organized OPC server project that directly reflects the variable import file.

**See Also:** [Tag Hierarchy](#) and [Import File-To-Server Name Conversions](#).

### 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 Database Creation dialog under Device Properties, click on the selection **Perform the following action**. 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 completes, users should 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 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" parameter) 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 31 characters. Names that exceed 31 characters will be truncated as follows:

#### Non-Array

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

### **Importing VersaPro Tags**

---

This driver uses Shared Name Files (SNF) that are generated from VersaPro in order 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.

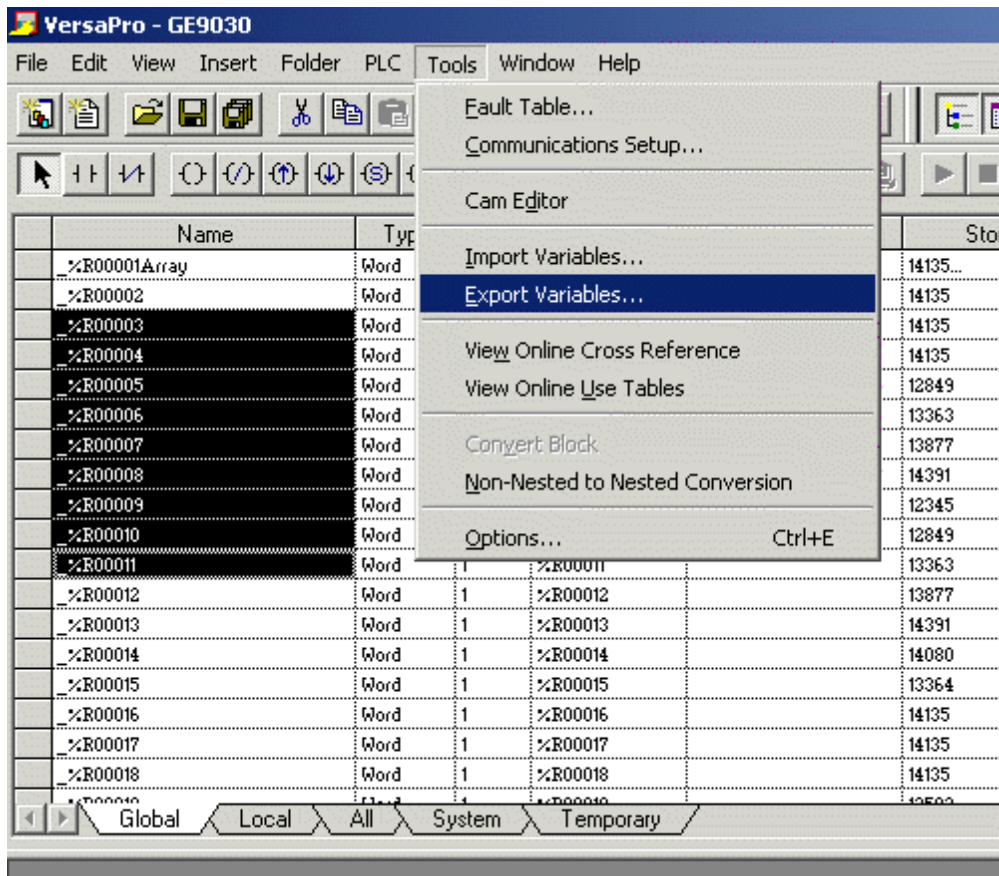
VersaPro - GE9030

File Edit View Insert Folder PLC Tools Window Help

Name	Type	Len	Address	Description	SI
%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		13500

Global Local All System Temporary

- 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**.



- 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

- To start, right-click on the device for which tags will be generated and select **Device Properties**. Then, open the **Variable Import Settings** tab.
- In **Variable Import File**, enter or browse for the location of the VersaPro \*.snf file newly created. Then, click **Apply**.
- Next, select the **Database Creation** tab. Then, click **Auto-Create**.
- 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](#).
- 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 in order to be used in HMIs 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 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

---

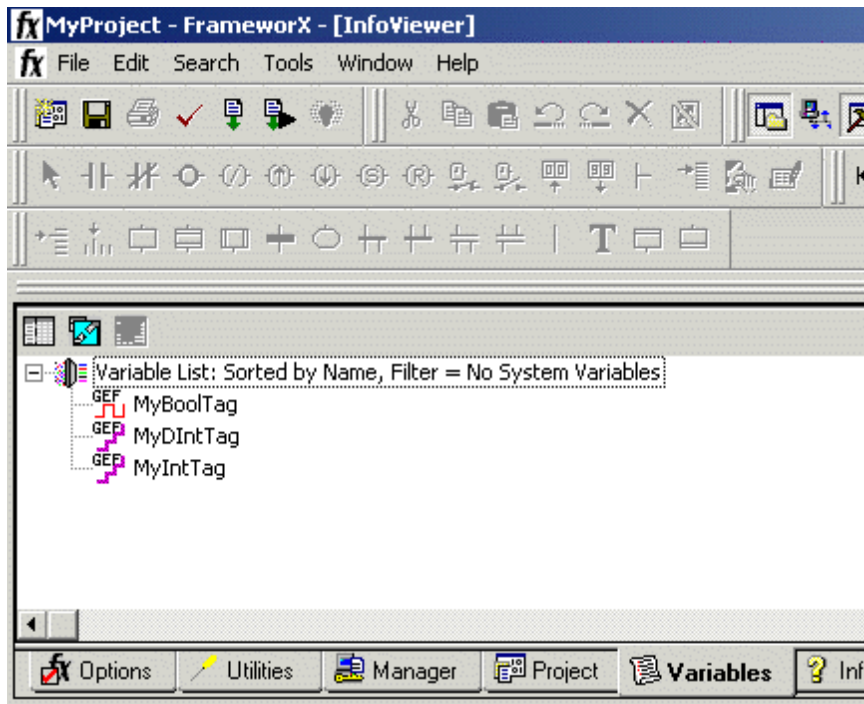
This driver uses user-generated ASCII text files from Logic Developer to generate the tag database. Certain aspects of the Automatic Tag Database Generation process depend on the application from which variables are imported. For more information on a specific aspect of Logic Developer tag import, select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

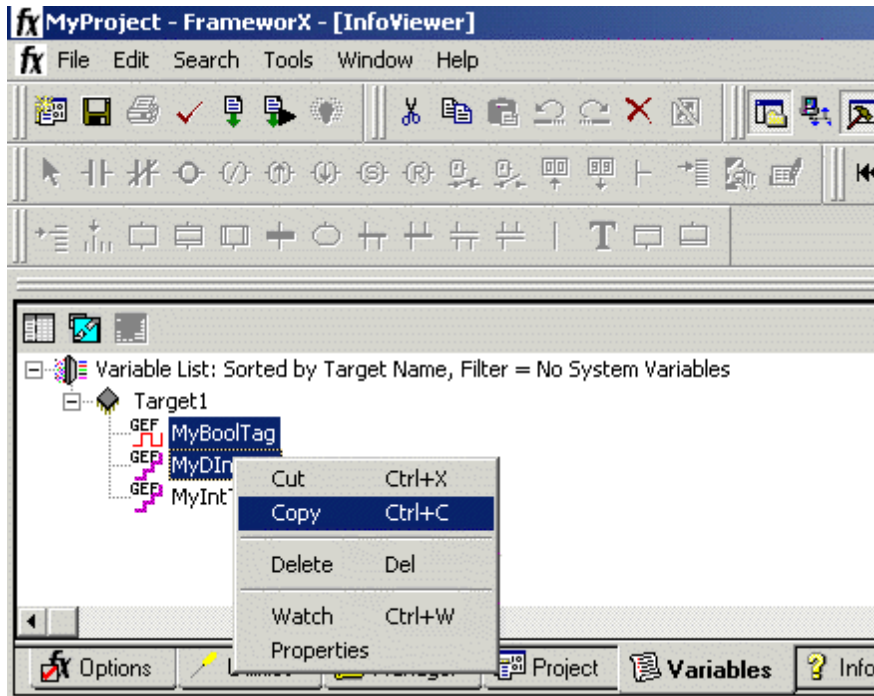
1. To start, open the FrameworkX project containing the tags (variables) that will be ported to the OPC server. Then, open the **Navigator** window by pressing **Shift+F4**.
2. Next, click on the **Variables** tab and select **Variable List View**.



**Note:** Each FrameworkX 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**.

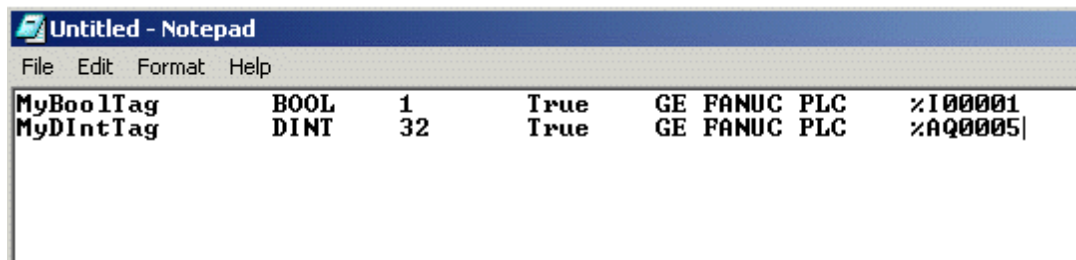




**Note:** The highlighted variables should now be copied to the Clipboard.

- 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.



- Save the text document with the text extension (.txt) in ANSI form.
- 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**

- To start, right-click on the device for which tags will be generated and select **Device Properties**. Then, open the **Variable Import Settings** tab.
- In **Variable Import File**, enter or browse for the location of the Logic Developer \*.txt file newly created. Then, click **Apply**.
- Next, select the **Database Creation** tab. Then, click **Auto-Create**.
- 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](#).
- 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

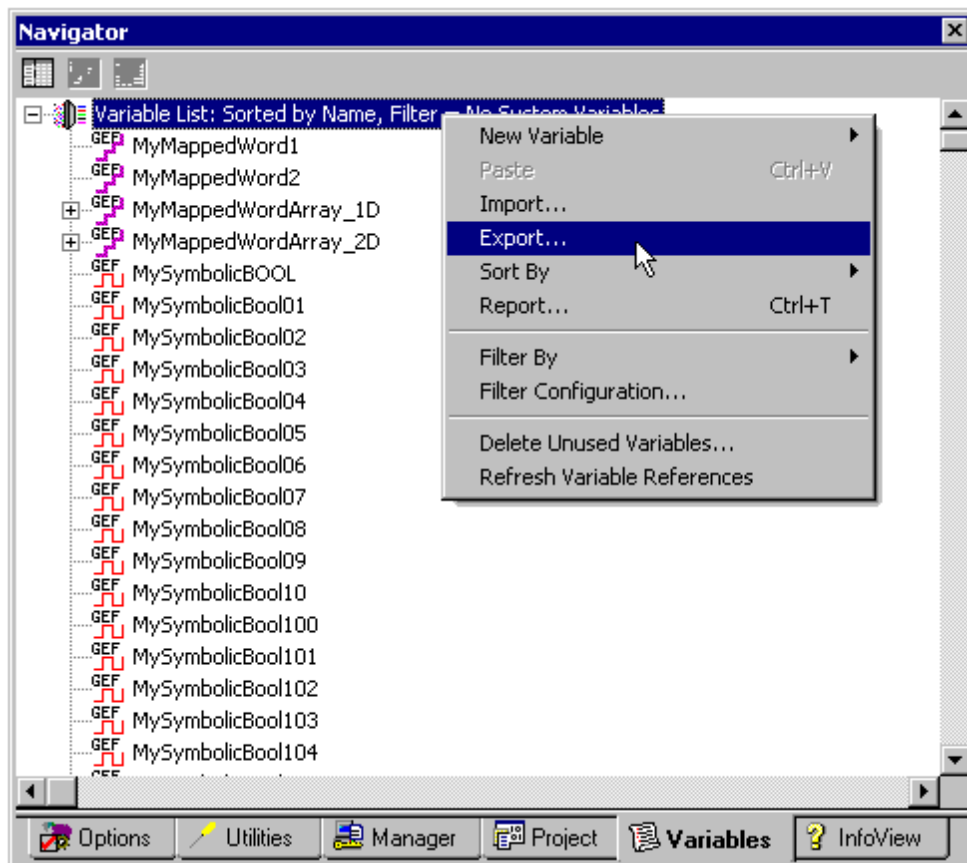
---

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**.

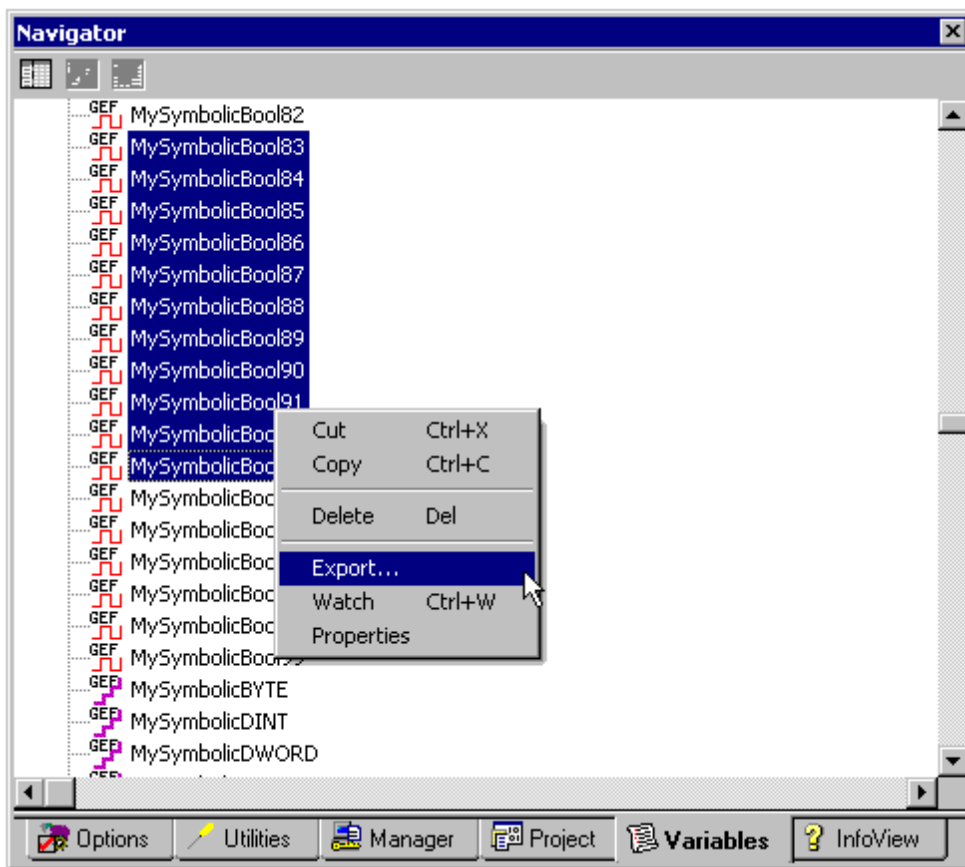
**Note:** Each Logic Developer project contains one or more targets, which is essentially the device on which the application will run. Because variables are created on the target level, the target of interest must be specified before the variables that will be exported may be. In order for the target 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**.

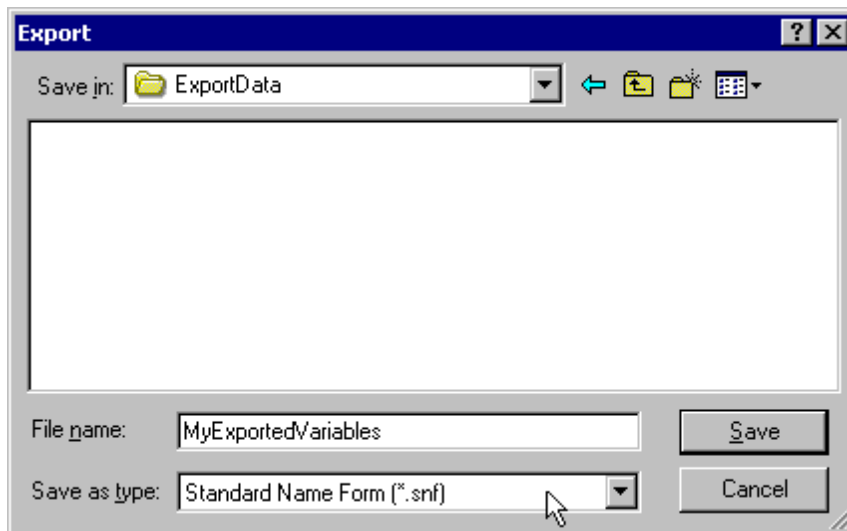
4. To export all of the variables, right-click on the **Variable List** and click **Export**.

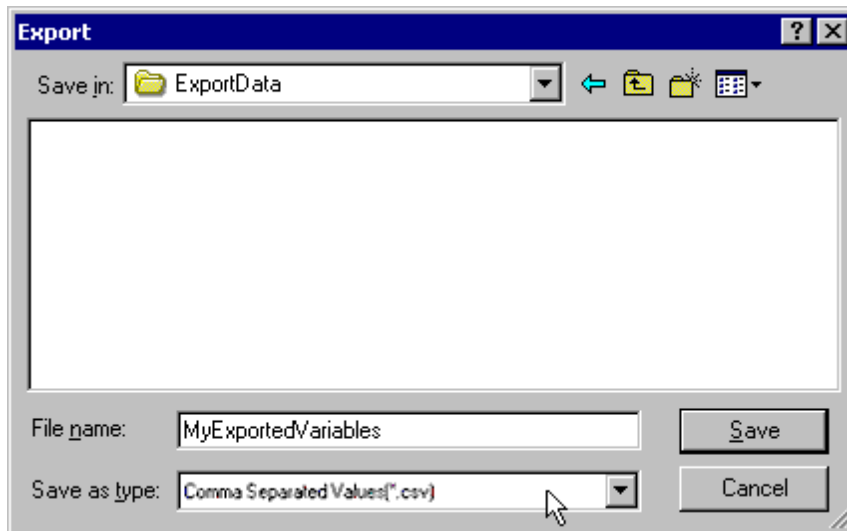


5. To export selected variables, highlight the tags of interest in the target of interest. Then, right-click on one of the selected variables and click **Export**.



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** tab.
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 **Database Creation** tab and then click **Auto Create** to import variables. Alternatively, use the other settings in that dialog 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.

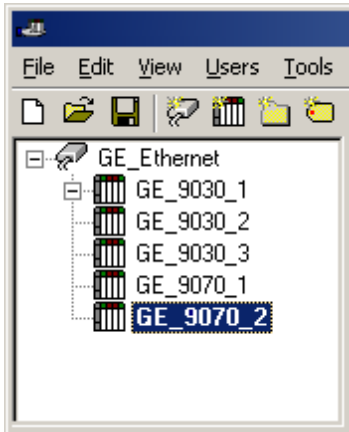
**See Also:** [Symbolic Variables](#)

---

## Optimizing Your GE Ethernet Communications

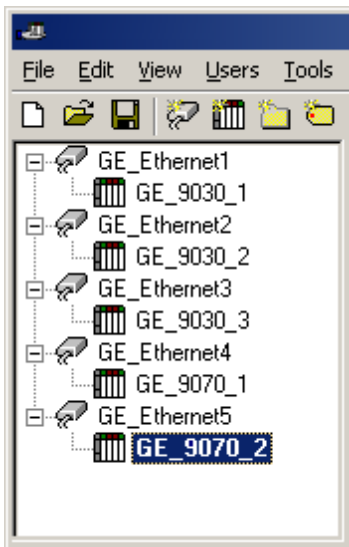
The GE Ethernet Driver has been designed to provide the best performance with the least amount of impact on the system's overall performance. While the GE Ethernet Driver is fast, there are a couple of guidelines that can be used in order to control and optimize the application and gain maximum performance.

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

The performance will improve even if the application has more than 256 devices. While 256 or 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.

The GE Ethernet Driver's performance can also be affected by 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 Block Size 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 parameter limits the number of bytes returned for all the blocks.

**See Also:** [Communications Parameters](#)

**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](#)

[Symbolic Variables](#)

[311](#)

[313](#)

[331](#)

[341](#)

[350](#)

[360](#)

[731](#)

[732](#)

[771](#)

[772](#)

[781](#)

[782](#)

[GE OPEN](#)

[Horner OCS](#)

[VersaMax Addressing](#)

[Advanced Addressing](#)

[Special Items](#)

## 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)	<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 G7680 G00001 to G7673 (every 8th bit) G00001 to G7665 (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	T0001 to T1024 T0001 to T1017 (every 8th bit) T0001 to T1009 (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	R1.0 to R32640.15 R00001 to R32640 R00001 to R32639 R00001 to R32637	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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>	<b>Boolean</b>  <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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."

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

**See Also:** [Symbolic Variables](#)

### Advanced Addressing

[Bit Access to Registers](#)

[Default Data Type Override](#)

[String Access to Registers](#)

[Array Support](#)

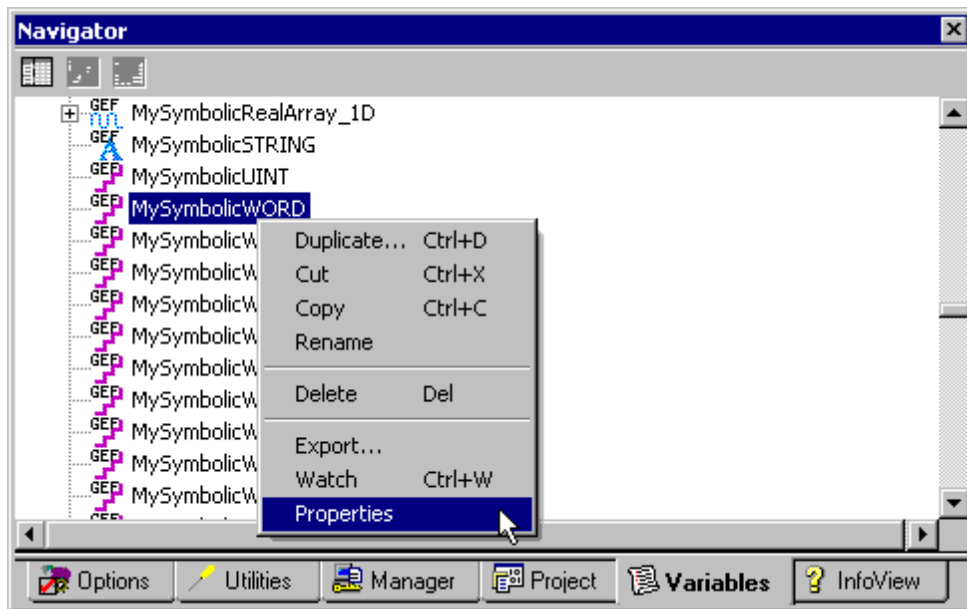
## Symbolic Variables

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:

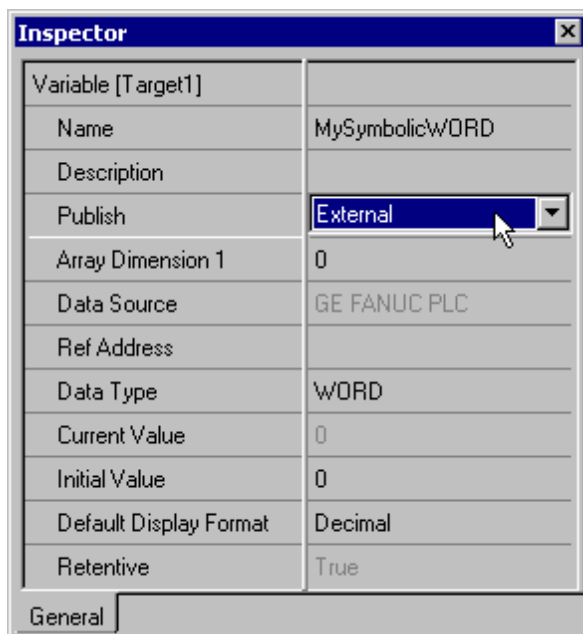
- **Reference Variable:** This variable is mapped to a specific I/O or internal register location in the PLC (such as I00001, R00100, and so forth).
- **Symbolic Variable:** This variable is not mapped to a specific register. The PLC will allocate memory for symbolic variables as needed from its managed 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.

### 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** setting in Device Properties. For more information, refer to [Device Setup](#).

**Important:** In order for a symbolic variable to be visible to this driver, **Publish** must be set to **External**.

- Next, download the modified project to the PLC.
- 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** parameter 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 parameter.

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 parameter 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 parameter. Unlike block reads, the variables' location in the controller's memory is of little consequence. Writes can further be optimized with the **Write Optimizations** setting 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)	<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	R1.0 to R512.15 R001 to R512 R001 to R511 R001 to R509	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ32.15 AQ001 to AQ032	<b>Boolean</b> <b>Word</b> , Short, BCD	Read/Write

	AQ001 to AQ031 AQ001 to AQ029	DWord, Long, LBCD, Float Double	
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)	<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	R1.0 to R1024.15 R001 to R1024 R001 to R1023 R001 to R1021	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R2048.15 R0001 to R2048 R0001 to R2047 R0001 to R2045	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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)	<b>Boolean</b> Byte	Read/Write



	M0001 to M1009 (every 8th bit)	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	R1.0 to R9999.15 R0001 to R9999 R0001 to R9998 R0001 to R9996	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R9999.15 R0001 to R9999 R0001 to R9998 R0001 to R9996	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R32768.15 R00001 to R32768 R00001 to R32767 R00001 to R32765	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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>	<b>Boolean</b>   <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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](#)

### 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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Reg- isters**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15  L00001<SUBPRGM> to L16384<SUBPRGM>  L00001<SUBPRGM> to L16383<SUBPRGM>  L00001<SUBPRGM> to L16381<SUBPRGM>	<b>Boolean</b>   <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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>	<b>Boolean</b>   <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD,	Read/Write

		Float Double	
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)	<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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Reg- isters**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15  L00001<SUBPRGM> to L16384<SUBPRGM>  L00001<SUBPRGM> to L16383<SUBPRGM>  L00001<SUBPRGM> to L16381<SUBPRGM>	<b>Boolean</b>   <b>Word</b> , Short, BCD  DWord, Long, LBCD, Float  Double	Read/Write
Analog Inputs	AI1.0 to AI8192.15 AI0001 to AI8192	<b>Boolean</b> <b>Word</b> , Short, BCD	Read/Write

	AI0001 to AI8191 AI0001 to AI8189	DWord, Long, LBCD, Float Double	
Analog Outputs	AQ1.0 to AQ8192.15 AQ0001 to AQ8192 AQ0001 to AQ8191 AQ0001 to AQ8189	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write
SubProgram Block Reg- isters**	L00001<SUBPRGM>.0 to L16384<SUBPRGM>.15  L00001<SUBPRGM> to L16384<SUBPRGM>  L00001<SUBPRGM> to	<b>Boolean</b>   <b>Word</b> , Short, BCD   DWord, Long, LBCD,	Read/Write

	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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R16384.15 R00001 to R16384 R00001 to R16383 R00001 to R16381	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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>	<b>Boolean</b>  <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<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	R1.0 to R32768.15 R00001 to R32768 R00001 to R32767 R00001 to R32765	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Program Block Registers***	P00001 to P32768 P00001 to P32767 P00001 to P32765	<b>Word</b> , 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>	<b>Boolean</b>   <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write

\*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	<b>Boolean</b>	Read/Write

	I0001 to I2041 (every 8th bit) I0001 to I2033 (every 8th bit)	Byte Word, Short, BCD	
Discrete Outputs	Q00001 to Q20488 Q00001 to Q2041 (every 8th bit) Q00001 to Q2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Internal Coils	M00001 to M2048 M00001 to M2041 (every 8th bit) M00001 to M2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Temporary Coils	T00001 to T2048 T00001 to T2041 (every 8th bit) T00001 to T2033 (every 8th bit)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Globals	G1 to G32768 G1 to G32761 G1 to G32753	<b>Boolean</b> <b>Byte</b> Word, Short, BCD	Read/Write
Status References (Same for SA, SB, and SC)	S1 to S32768 S1 to S32761 S1 to S32753	<b>Boolean</b> <b>Byte</b> Word, Short, BCD	Read/Write
Register References	R1.0 to R32768.15 R00001 to R32768 R00001 to R32767 R00001 to R32765	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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	<b>Boolean</b> <b>Word</b> , 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)	<b>Boolean</b> Byte Word, Short, BCD	Read/Write
Discrete Outputs	Q001 to Q2048 Q001 to Q2041 (every 8th bit) Q001 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 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	R1.0 to R2048.15	<b>Boolean</b>	Read/Write

	R001 to R2048 R001 to R2047 R001 to R2045	<b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	
Analog Inputs	AI1.0 to AI128.15 AI01 to AI128 AI01 to AI127 AI01 to AI125	<b>Boolean</b> <b>Word</b> , Short, BCD DWord, Long, LBCD, Float Double	Read/Write
Analog Outputs	AQ1.0 to AQ128.15 AQ001 to AQ128 AQ001 to AQ127 AQ001 to AQ125	<b>Boolean</b> <b>Word</b> , 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" setting 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 Variables	512 DWords (longs, floats), 1024 words (shorts) or 1024 bytes 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  <b>Note:</b> 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	<b>Boolean</b>	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 0 = PLC Fault table is empty	Boolean	Read Only
_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
_SWEETIME	Time taken by the last complete sweep for the main control program task.  The value returned is a whole number representing the number of tenths of a millisecond. The Sweep Time in milliseconds is 1/10 of this value. For example, if _SWEETIME is 123, the actual Sweep Time is 12.3 milliseconds.	Word, Short	Read Only

_SNPID	CPU Controller ID.	<b>String</b>	Read Only
_PROGNAME	Name of control program task in the PLC CPU.	<b>String</b>	Read Only
_SNUM_PROGS	Number of control program tasks defined for the PLC CPU.	<b>Byte, Char</b>	Read Only
_SPROG_FLAGS	Bit flags indicating which control programs tasks have programmers attached. Bit 0 = control program 1.	<b>Byte, Char</b>	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)	<b>String</b>	Read Only
_SECOND	PLC Time: Seconds (0-59)	<b>Byte, Char</b>	Read Only
_MINUTE	PLC Time: Minutes (0-59)	<b>Byte, Char</b>	Read Only
_HOUR	PLC Time: Hours (0-23)	<b>Byte, Char</b>	Read Only
_DAY	PLC Date: Day (1- 31)	<b>Byte, Char</b>	Read Only
_MONTH	PLC Date: Month (1-12)	<b>Byte, Char</b>	Read Only
_YEAR	PLC Date: Year (0-99)	<b>Byte, Char</b>	Read Only
_DOW	PLC Date: Day of week (1-Sunday, 2-Monday, etc)	<b>Byte, Char</b>	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 31 characters. Tags renamed to '<new tag name><dimensions>'](#)

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

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

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

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

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

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

## Address Validation Error Messages

---

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

---

### **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.

---

**Device Status Messages**

---

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

**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>'](#)

---

**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:**

Warning

---

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.

---

**Driver Error Messages**

---

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

**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](#)

---

**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.

**Note:**

The native data types' 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 parameter (located in **Device Properties | Communications Parameters**) 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 in order 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).

**Note:**

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 [Proficiency Logic Developer Array Tag Import](#).

**See Also:**

[Communications Parameters](#)

---

**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 subsystem 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.

### **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**

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

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

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

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

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

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

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

### **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 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 GE Ethernet 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 GE Ethernet 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 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](#)

---

**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](#)

---

**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.

---



# Index

## 3

311 Addressing.....	30
313 Addressing.....	31
331 Addressing.....	32
341 Addressing.....	32
350 Addressing.....	33
360 Addressing.....	34

## 7

731 Addressing.....	35
732 Addressing.....	36
771 Addressing.....	37
772 Addressing.....	38
781 Addressing.....	39
782 Addressing.....	40

## A

Address '<address>' is out of range for the specified device or register.....	49
Address Descriptions.....	26
Address Validation Error Messages.....	48
Advanced Addressing.....	44
Array size is out of range for address '<address>'.....	49
Array support is not available for the specified address: '<address>'.....	49
Attempting to reacquire symbolic variable mapping information for device '<device name>'.....	50
Cannot read or write symbolic variables until complete.....	
Automatic Tag Database Generation.....	10
Automatic Tag Database Generation Messages.....	54

## B

BCD.....	25
Bit access is invalid for the device data type '<native data type>' at address '<address>' in ... device '<device>'.....	51
Bit index '<bit index>' is not valid for the device data type '<native data type>' at address ... '<address>' in device '<device>'.....	52
Block Size.....	6
Boolean.....	25
Byte.....	25

C

Cimplicity Logic Developer Array Tag Import ..... 18  
 Cimplicity Logic Developer Import Preparation: Logic Developer Steps ..... 16  
 Cimplicity Logic Developer Import Preparation: OPC Server Steps ..... 17  
 Communications ..... 6  
 Communications Parameters ..... 7

D

Data Type '<type>' is not valid for device address '<address>' ..... 49  
 Data Types Description ..... 25  
 Database Error ..... 54  
**Data type '<type>' arrays are currently not supported. Tag '<array tag name>' not created** 54  
 Database Error: Array tags '<orig. tag name><dimensions>' exceed 31 characters. Tags ..... 54  
 renamed to '<new tag name><dimensions>' .....  
 Database Error: Data type '<type>' for tag '<tag name>' is currently not supported. Tag not ... 55  
 created .....  
 Database Error: Data type '<type>' for tag '<tag name>' not found in import file. Setting to ... 55  
 Default Type '<type>' .....  
 Database Error: Logic Developer Variable Arrays are currently not supported. Array Tag(s) .. 55  
 '<array tag name>' not created .....  
 Database Error: No Reference Address found for tag '<tag name>' in import file. Tag not .... 55  
 created .....  
 Database Error: Only variables with Data Source '<data source name>' are imported. Data ... 56  
 Source '<data soure name>' is not.....  
 Database Error: Tag '<orig. tag name>' exceeds 31 characters. Tag renamed to '<new tag ... 56  
 name>' .....  
 device ..... 48-49  
 Device '<device name>' returned error code <error num> reading in byte(s) starting at ..... 52  
 <address> .....  
 Device '<device name>' not responding ..... 50  
 Device address '<address>' contains a syntax error ..... 49  
 Device address '<address>' is not supported by model '<model name>' ..... 50  
 Device address '<address>' is Read Only ..... 50  
 Device ID ..... 6  
 Device Setup ..... 6  
 Device Status Messages ..... 50  
 Driver Error Messages ..... 51  
 DWord ..... 25

E

Error Descriptions ..... 48

**F**

Float ..... 25

**G**

GE OPEN Addressing ..... 41

**H**

Highlighting LogicDeveloper Variables ..... 18

Highlighting Proficy Logic Developer Variables ..... 21

Highlighting VersaPro Variables ..... 14

Horner OCS Addressing ..... 42

**I**

Import File-to-Server Name Conversions ..... 11

Importing LogicDeveloper Tags ..... 15

Importing Proficy Logic Developer Tags ..... 18

Importing VersaPro Tags ..... 12

**L**

LBCD ..... 25

Long ..... 25

**M**

Missing address ..... 50

**N**

Network ..... 6

Nickname ..... 9

---

**O**

Optimizing GE Ethernet Communications ..... 23  
 Overview ..... 5

**P**

PACSystems Addressing ..... 26  
 PLC Settings ..... 8  
 Proficy Logic Developer Array Tag Import ..... 21  
 Proficy Logic Developer Import Preparation: Logic Developer Steps ..... 18  
 Proficy Logic Developer Import Preparation: OPC Server Steps ..... 21  
 protocol ..... 6

**S**

Short ..... 25  
 Special Items ..... 45  
 Symbolic Variables ..... 27

**T**

Tag Hierarchy ..... 10  
 Target Name ..... 9  
 The address length of the symbolic variable addressed by '<tag name>' on device '<device>' is bigger ..... 52  
 The array size of address '<address>' on device '<device name>' must be the same as symbolic variable's array size of [rows][cols] ..... 52  
 The data type of the symbolic variable addressed by '<address>' on device '<device name>' is not compatible with tag ..... 52  
 The symbolic discrete array size of '<size in bytes>' bytes at address '<address>' on device '<device>' exceeds the configured '<block size>' maximum request bytes ..... 53  
 The symbolic variable name in tag address '<address>' does not exist in current configuration of device '<device name>' ..... 53

**U**

Unable to generate a tag database for device <device name>. Reason ..... 56  
 Unable to generate a tag database for device <device name>. Reason: Import file is invalid or corrupt ..... 56  
 Unable to generate a tag database for device <device name>. Reason: Low memory ..... 56

resources.....	
Unable to write tag '<address>' on device '<device name>'.....	51

## V

Variable Import Settings.....	8
VersaMax Addressing.....	43
VersaPro Array Tag Import.....	15
VersaPro Import Preparation: VersaPro Steps.....	12
VersaPro Import Preparation: OPC Server Steps.....	14

## W

Winsock initialization failed (OS Error = n).....	53
Winsock V1.1 or higher must be installed to use the GE Ethernet device driver.....	54
Word.....	25

---