

# **Modbus RTU Serial Driver Help**

© 2009 Kepware Technologies

# Table of Contents

<b>1</b>	<b>Getting Started.....</b>	<b>3</b>
	Help Contents.....	3
	Overview.....	3
<b>2</b>	<b>Device Setup.....</b>	<b>3</b>
	Device Setup.....	3
	Cable Diagram.....	4
	Modem Setup.....	4
	Settings.....	4
	Block Sizes.....	7
	Variable Import Settings.....	8
	Framing.....	9
	Error Handling.....	11
<b>3</b>	<b>Automatic Tag Database Generation.....</b>	<b>12</b>
	Automatic Tag Database Generation.....	12
	Exporting Variables from Concept.....	13
	Exporting Variables from ProWORX.....	14
<b>4</b>	<b>Data Types Description.....</b>	<b>16</b>
	Data Types Description.....	16
<b>5</b>	<b>Address Descriptions.....</b>	<b>17</b>
	Address Descriptions.....	17
	Address Descriptions: Modbus.....	17
	Address Descriptions: Magnetek.....	20
	Address Descriptions: Elliot.....	21
	Address Descriptions: Daniel S500.....	21
	Address Descriptions: Dynamic Fluid Meter.....	22
	Address Descriptions: Omni.....	23
	Address Descriptions: Omni.....	23
	Omni Custom Packets.....	26
	Omni Raw Data Archive.....	27
	Omni Text Reports.....	32
	Omni Text Archive.....	33
<b>6</b>	<b>Error Descriptions.....</b>	<b>35</b>
	Error Descriptions.....	35
	Address '<address>' is out of range for the specified device or register.....	35
	Array size is out of range for address '<address>'.....	36
	Array support is not available for the specified address: '<address>'.....	36
	Bad address in block [<start address> to <end address>] on device '<device name>'.....	36
	Bad array spanning [<address> to <address>] on device '<device>'.....	36
	Communications error on COMn [<error mask>].....	37
	COMn does not exist.....	37
	COMn is in use by another application.....	37
	Could not read Omni text buffer due to memory allocation problem.....	37
	Could not read Omni text report '<address>' on device '<device name>' due to packet.....	38
	Data Type '<type>' is not valid for device address '<address>'.....	38
	Description truncated for import file record number <record>.....	38
	Device address '<address>' contains a syntax error.....	38
	Device address '<address>' is not supported by model '<model name>'.....	38

Device address '<address>' is Read Only..... 39

Device '<device name>' is not responding..... 39

Error opening COMn..... 39

Error parsing import file record number <record>, field <field>..... 39

Error writing Omni text data to file for '<tag name>' on device '<device name>'..... 40

File exception encountered during tag import..... 40

Imported tag name '<tag name>' is invalid. Name changed to '<tag name>'..... 40

Missing address..... 40

No Omni text archive data available in specified date range on device '<device name>'..... 40

Omni text output file specified for '<tag name>' on device '<device name>'..... 41

Received block length of '<received length>' does not match expected length of '<expected length>' for.... 41

Tag import failed due to low memory resources..... 41

Tag '<tag name>' could not be imported because data type '<data type>' is not supported..... 41

Unable to set comm parameters on COMn..... 42

Unable to write to address '<address>' on device '<device>': Device responded with exception code..... 42

Unable to write to '<address>' on device '<device name>'..... 42

Write failed for '<tag name>' on device '<device name>'. Maximum path length of '<number>'..... 42

Write to Omni text report '<address>' on device '<device name>' truncated..... 42

Modbus Exception Codes..... 43

    Modbus Exception Codes..... 43

**Index**

## **Modbus RTU Serial Driver Help**

---

Help version 1.029

### **CONTENTS**

#### **Overview**

What is the Modbus RTU Serial 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 Modbus RTU Serial driver?

#### **Data Types Description**

What data types does this driver support?

#### **Address Descriptions**

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

#### **Error Descriptions**

What error messages are produced by the Modbus RTU Serial driver?

### **Overview**

---

The Modbus RTU Serial Driver provides an easy and reliable way to connect Modbus RTU Serial devices to OPC Client applications, including HMI, SCADA, Historian, MES, ERP and countless custom applications. It intended for use with serial devices that support the Modbus RTU protocol. The Modbus RTU Serial driver has been developed to support a wide range of Modbus RTU compatible devices. This driver's special features allow users to control the following: the amount of data requested from a device in a single request, the word ordering of 32 bit double register values, the byte ordering of 16 and 32 bit register values and address base adjustment. The Modbus RTU driver can also control the operation of the RTS line for use with radio modems that require specific RTS timing. The driver also supports sending broadcast messages on Device ID 0.

### **Device Setup**

---

#### **Supported Devices**

Modbus compatible devices  
Elliott Flow Computer  
Magnetek GPD 515 Drive  
Omni Flow Computer  
Daniel S500 Flow Computer  
Dynamic Fluid Meter (DFM) SFC3  
TSXCUSBMBP USB Adapter

#### **Communication Protocol**

Modbus RTU Protocol.

#### **Supported Communication Parameters\***

Baud Rate: 1200, 2400, 9600, 19200  
Parity: Odd, Even, None  
Data Bits: 8  
Stop Bits: 1,2

\*Not all of the listed configurations may be supported in every device.

#### **Maximum Number of Channels and Devices**

The maximum number of channels supported by this driver is 100. The maximum number of devices supported is 255.

#### **Ethernet Encapsulation**

This driver supports Ethernet Encapsulation, which allows the driver to communicate with serial devices attached to an Ethernet network using a terminal server such as the Lantronix DR1. To enable Ethernet Encapsulation for the channel, click **Use Ethernet Encapsulation** in the Communications dialog in Channel Properties. For more information, refer to the main OPC Server help file.

### Device ID (PLC Network Address)

Modbus RTU Serial devices are assigned Device IDs in the range 0 to 255. When using Modbus Device ID 0, the driver will send only broadcast Write messages to remote stations. When configuring a device under the channel, setting the Device ID to 0 will place that device in broadcast mode. Only Writes will occur from this device. Reads from the broadcast device will always return zero. All other Device IDs (1-255) will read and write data to/from the remote Modbus RTU device.

### Flow Control

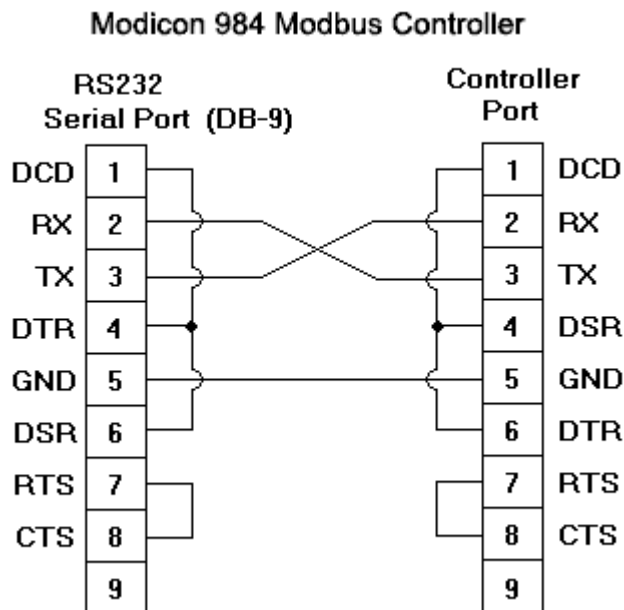
When using an RS232/RS485 converter, the type of flow control that is required will depend on the converter's needs. Some do not require any flow control and others will require RTS flow. Consult the converter's documentation in order to determine its flow requirements. We recommend using an RS485 converted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

**Note:** When using the manufacturer's supplied communications cable, it is sometimes necessary to choose a flow control setting of **RTS** or **RTS Always** under the Channel Properties.

The Modbus RTU driver supports the RTS Manual flow control option. This selection is used to configure the driver for operation with radio modems that require special RTS timing characteristics. For more information on RTS Manual flow control, refer to the main OPC Server help file topic "Channel Wizard."

### Cable Diagram

For recommended wiring and cable diagrams, refer to the Modbus device manufacturer's documentation. For example, the Modicon 984 Modbus Controller cable diagram is shown below.



### Modem Setup

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

### Settings

----- Data Access Group -----

### Zero vs. One Based Addressing

If the address numbering convention for the device starts at one as opposed to zero, it can be specified when defining the parameters for the device. By default, user entered addresses will have one subtracted from them when frames are constructed to communicate with a Modbus device. If the device doesn't follow this convention, then the **Use zero based addressing** check box should be unchecked in Device Properties. For information on the appropriate application to obtain information on setting Device Properties, refer to the online help documentation. The default behavior follows Modicon PLCs conventions.

### Zero vs One Based Bit Addressing Within Registers

Memory types that allow bits within Words can be referenced as a Boolean. The addressing notation for doing this is as follows:

<address>.<bit>

where <bit> represents the bit number within the word.

Zero Based Bit Addressing within registers provides two ways of addressing a bit within a given word: Zero Based and One Based. Zero Based Bit addressing within registers simply means the first bit begins at 0. One Based Bit addressing means that the first bit begins at 1.

#### Zero Based Bit Addressing within registers (Default Setting / Checked)

Data Type	Bit Range
Word	Bits 0-15

#### One Based Bit Addressing within registers (Unchecked)

Data Type	Bit Range
Word	Bits 1-16

### Holding Register Bit Mask Writes

When writing to a bit location within a holding register, the driver should only modify the bit of interest. Some devices support a special command to manipulate a single bit within a register (Function code hex 0x16 or decimal 22). If the device does not support this feature, the driver will need to perform a Read/Modify/Write operation to ensure that only the single bit is changed.

Check this box if the device supports holding register bit access. By default, it is left unchecked. If this setting is selected, then the driver will use function code 0x16, irrespective of the setting for **Use Modbus function 06 for single register writes**. However, if this setting is not selected, then the driver will use either function code 0x06 or 0x10 depending on the selection for 'Use Modbus function 06 for single register writes.'

**Note:** When Modbus byte order is deselected, the byte order of the masks sent in the command will be Intel byte order.

### Use Modbus Function 06 or 16

The Modbus driver has the option of using two Modbus protocol functions to write holding register data to the target device. In most cases, the driver switches between these two functions based on the number of registers being written. When writing a single 16 bit register, the driver will generally use the Modbus function 06. When writing a 32 bit value into two registers, the driver will use Modbus function 16. For the standard Modicon PLC, the use of either of these functions is not a problem. There are, however, a large number of 3rd party devices that have implemented the Modbus protocol. Many of these devices support only the use of Modbus function 16 to write to Holding registers, regardless of the number of registers to be written.

The **Use Modbus function 06** selection is used to force the driver to use only Modbus function 16 if needed. By default, this selection is checked which allows the driver to operate as it has historically, switching between 06 and 16 as needed. If a device requires all writes be done using only Modbus function 16, uncheck this selection.

**Note:** For bit within word writes, the **Holding Register Bit Mask Writes** property takes precedence over this property (Use Modbus Function 06). If Holding Register Bit Mask Writes is selected, then function code 0x16 is used no matter what the selection for this property. However, if Holding Register Bit Mask Writes is not selected, then depending upon the selection of this property either function code 0x06 or 0x10 will be used for bit within word writes.

### Use Modbus Function 05 or 15

The Modbus driver has the option of using two Modbus protocol functions to write Output coil data to the target device. In most cases the driver switches between these two functions based on the number of coils being written. When writing a single coil, the driver will use the Modbus function 05. When writing an array of coils, the driver will use Modbus function 15. For the standard Modicon PLC the use of either of these functions is not a problem. There are, however, a large number of third party devices that have implemented the Modbus protocol. Many of these devices support only the use of Modbus function 15 to write to output coils regardless of the number of coils to be written.

The Use Modbus function 05 selection is used to force the driver to use only Modbus function 15 if needed. By default, this selection is checked which allows the driver to operate as it has historically, switching between 05 and 15 as needed. If a device requires all writes be done using only Modbus function 15, uncheck this selection.

## ----- Data Encoding Group -----

### Modbus Byte Order

The Ethernet driver's byte order can be changed from the default Modbus byte ordering to Intel byte ordering by using this selection. This selection will be checked by default, which is the normal setting for Modbus compatible devices. If the device uses Intel byte ordering, deselecting this selection will enable the Modbus driver to properly read Intel formatted data.

**Note:** This setting does not apply to the Omni model. It always uses Modbus byte order.

### First Word Low in 32 Bit Data Types

Two consecutive registers' addresses in a Modbus device are used for 32 bit data types. Users can specify whether the driver should assume the first word is the low or the high word of the 32 bit value. The default, first word low, follows the convention of the Modicon Modsoft programming software.

**Note:** This setting does not apply to the Omni model. It always uses Modbus byte order.

### First DWord Low in 64 Bit Data Types

Four consecutive registers' addresses in a Modbus device are used for 64 bit data types. Users can specify whether the driver should assume the first DWord is the low or the high DWord of the 64 bit value. First DWord low (the default) follows the default convention of 32 bit data types.

**Note:** This setting does not apply to the Omni model. It always uses Modbus byte order.

### Use Modicon Bit Ordering

When checked, the driver will reverse the bit order on reads and writes to registers to follow the convention of the Modicon Modsoft programming software. For example, a write to address 40001.0/1 will affect bit 15/16 in the device when this option is enabled. This option is disabled (unchecked) by default.

**Note:** For the following example, the 1st through 16th bit signifies either 0-15 bits or 1-16 bits depending on if the driver is set at Zero Based or One Based Bit Addressing within registers.

MSB = Most Significant Bit

LSB = Least Significant Bit

#### Use Modicon Bit Ordering Checked

MSB								LSB							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

#### Use Modicon Bit Ordering Unchecked (Default Setting)

MSB								LSB							
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

### Data Encoding Options Details

The following summarizes usage of the Data Encoding options.

- Use default Modbus byte order option sets the data encoding of each register/16 bit value.

- First word low in 32 bit data types option sets the data encoding of each 32 bit value and each double word of a 64 bit value.
- First DWord low in 64 bit data types option sets the data encoding of each 64 bit value.

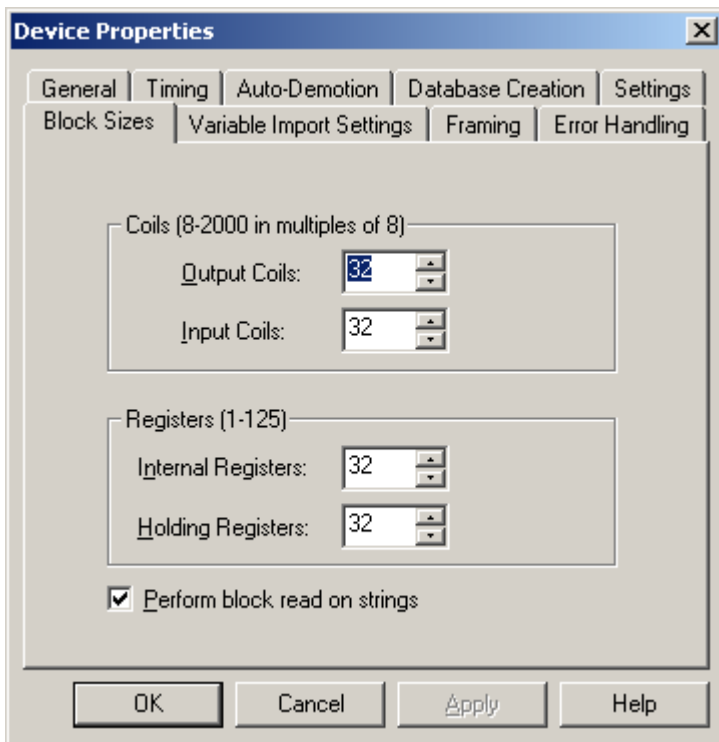
Data Types	Use default Modbus byte order Applicable	First word low in 32 bit data types Applicable	First DWord low in 64 bit data types Applicable
Word, Short, BCD	Yes	No	No
Float, DWord, Long, LBCD	Yes	Yes	No
Double	Yes	Yes	Yes

If needed, use the following information and the particular device's documentation to determine the correct settings of the Data Encoding options. The default settings are correct for the majority of Modbus devices.

Data Encoding Group Option	Data Encoding	
Use default Modbus byte order Checked	High Byte (15..8)	Low Byte (7..0)
Use default Modbus byte order Unchecked	Low Byte (7..0)	High Byte (15..8)
First word low in 32 bit data types Unchecked	High Word (31..16)	Low Word (15..0)
	High Word(63..48) of Double Word in 64 bit data types	Low Word (47..32) of Double Word in 64 bit data types
First word low in 32 bit data types Checked	Low Word (15..0)	High Word (31..16)
	Low Word (47..32) of Double Word in 64 bit data types	High Word (63..48) of Double Word in 64 bit data types
First DWord low in 64 bit data types Unchecked	High Double Word (63..32)	Low Double Word (31..0)
First DWord low in 64 bit data types Checked	Low Double Word (31..0)	High Double Word (63..32)

## Block Sizes

---



### Coil Block Sizes

Coils can be read from 8 to 2000 points (bits) at a time. A higher block size means more points will be read from the device in a single request. The block size can be reduced in order to read data from non-contiguous locations within the device.

### Register Block Sizes

Registers can be read from 1 to 125 locations (words) at a time. A higher block size means more register values will be read from the device in a single request. The block size can be reduced in order to read data from non-contiguous locations within the device.

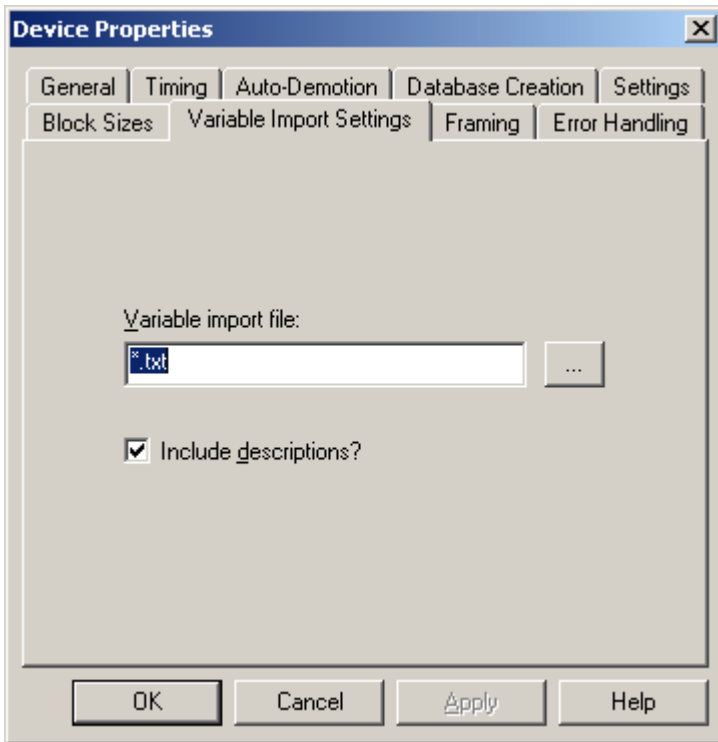
**Caution:** If the Register Block sizes value is set above 120 and a 32 or 64 bit data type is used for any tag, then a "Bad address in block" error could occur. To prevent this error, decrease the block size value to 120.

### Perform Block Read on Strings

When checked, this option will block read string tags (which are normally read individually). String tags will also be grouped together depending on the selected block size. Block reads can only be performed for Modbus model string tags.

### Variable Import Settings

The Variable Import Settings parameters specify the location of the variable import file that will be used when Automatic Tag Database Generation is enabled.



Descriptions of the parameters are as follows:

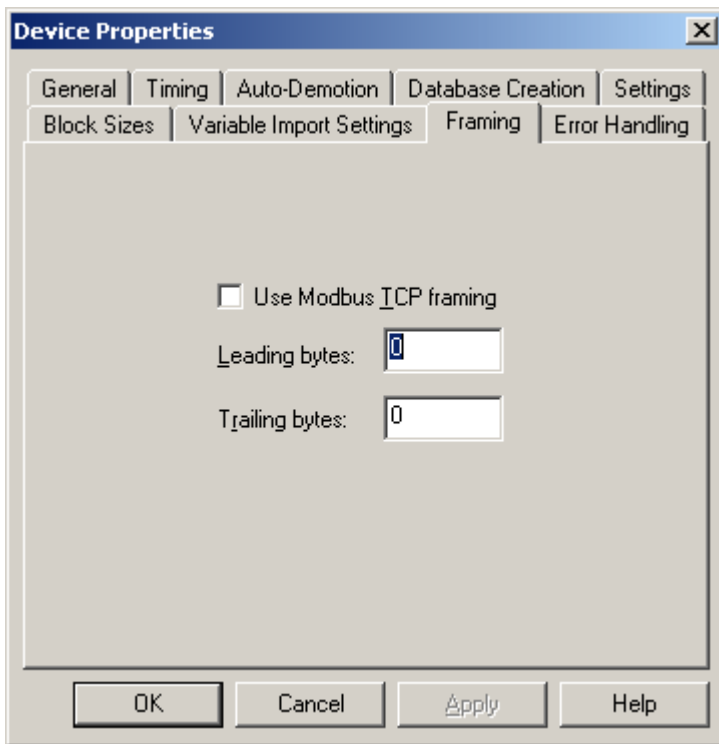
- **Variable Import File:** This parameter is used to browse to the exact location of the Concept or ProWORX variable import file that the driver will use during Automatic Tag Database Generation.
- **Include Descriptions:** When checked, imported tag descriptions will be used if present in the file.

**Note:** For more information on configuring the Automatic Tag Database Generation feature (and how to create a variable import file), refer to [Automatic Tag Database Generation](#).

## **Framing**

---

Since some terminal server devices will add additional data to Modbus frames, the Framing parameters are used to configure the driver to ignore the additional bytes in response messages.



Descriptions of the parameters are as follows:

- **Use Modbus TCP Framing:** When checked, this parameter is used to communicate with native Modbus TCP devices using Ethernet Encapsulation.
- **Leading bytes:** This parameter is used to specify the number of bytes to be attached to the beginning of Modbus responses. Values may range from 0 to 8.
- **Trailing bytes:** This parameter is used to specify the number of bytes to be attached to the end of Modbus responses. Values may range from 0 to 8.

### Using Ethernet Encapsulation

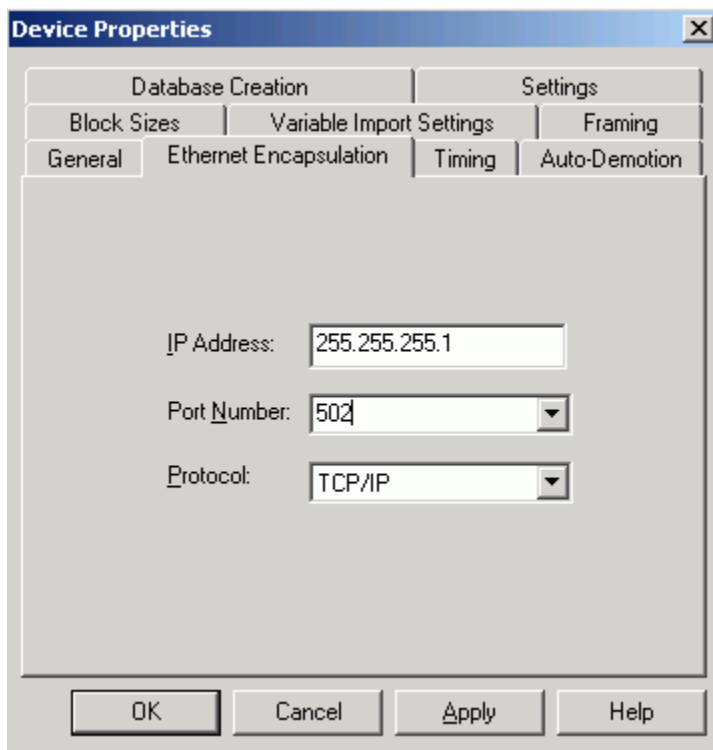
Ethernet Encapsulation must be enabled in order for Framing to be available; otherwise, the selection **Use Modbus TCP Framing** will be grayed out in the dialog box. To invoke Ethernet Encapsulation, use the following steps.

1. Go to the device's **Channel Properties** page.
2. In the **Communications** dialog, click **Use Ethernet Encapsulation**. This enables Ethernet Encapsulation for the channel.
3. Next, open the device's **Device Properties** page. Fill out the Ethernet Encapsulation parameters as shown below.

**IP Address:** Enter the device's IP address.

**Port Number:** Generally, 502 is entered for Modbus TCP devices.

**Protocol:** TCP/IP

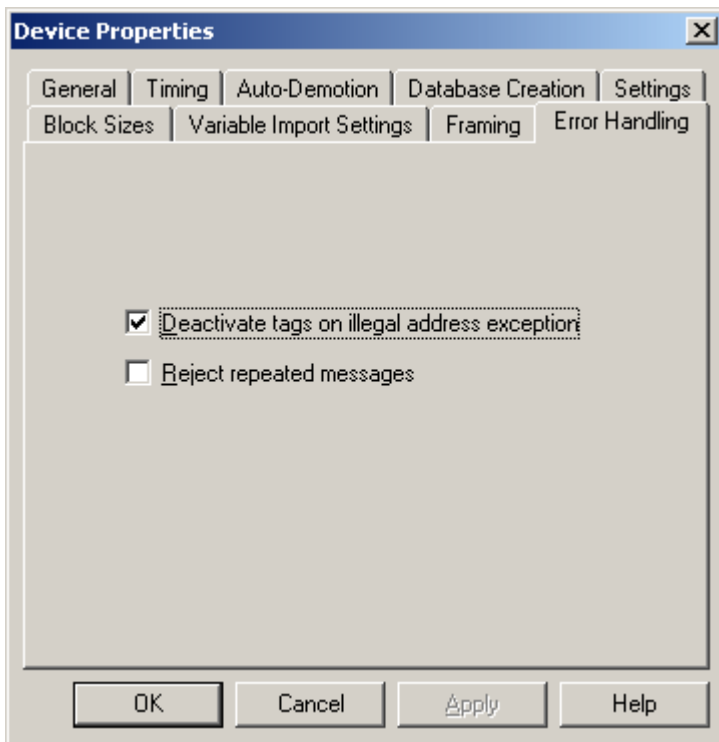


See Also: [Device Setup](#)

## **Error Handling**

---

The Error Handling parameters determine how to deal with errors from the device.



Descriptions of the parameters are as follows:

- **Deactivate Tags on Illegal Address Exception:** When checked, the driver will stop polling for a block of data if the device returns Modbus exception code 2 (illegal address) or 3 (illegal data, such as number of points) in response to a read of that block. This is the default setting. Uncheck this setting in order to read addresses that can become accessible in a dynamic fashion in the device. The default setting is checked.
- **Reject Repeated Messages:** When checked, the driver will expect repeated messages. When unchecked, the driver will interpret a repeated message as an invalid response and will retry the request. The default setting is unchecked.

**Note:** Some message-relay equipment will echo Modbus requests back to the driver.

## Automatic Tag Database Generation

---

### Overview

The Modbus RTU Serial driver makes use of the OPC Server's Automatic Tag Database Generation feature, which enables drivers to automatically create tags that access data points used by the device's ladder program. While it is sometimes possible to query a device for the information needed to build a tag database, this driver must use a **Variable Import File** instead. Variable import files can be generated using the Concept and ProWORX device programming applications.

### Creating the Variable Import File

The import file must be in semicolon delimited **Concept .TXT format**. This is the default export file format of the Concept device programming application. The ProWORX programming application can also export variable data in this format. For application-specific information on creating the variable import file, refer to [Exporting Variables from Concept](#) and [Exporting Variables from ProWORX](#).

### OPC Server Configuration

The automatic tag database generation feature can be customized to fit an application's specific needs. The primary control options can be set during the Database Creation step of the Device Wizard or later by selecting **Device Properties | Database Creation**. For more information, refer to the OPC Server's Help Documentation.

This driver requires additional settings in addition to the basic settings that are common to all drivers that support automatic tag database generation. The specialized settings include the name and location of the variable import file, which can be specified during the Variable Import Settings step of the Device Wizard or later by selecting **Device**

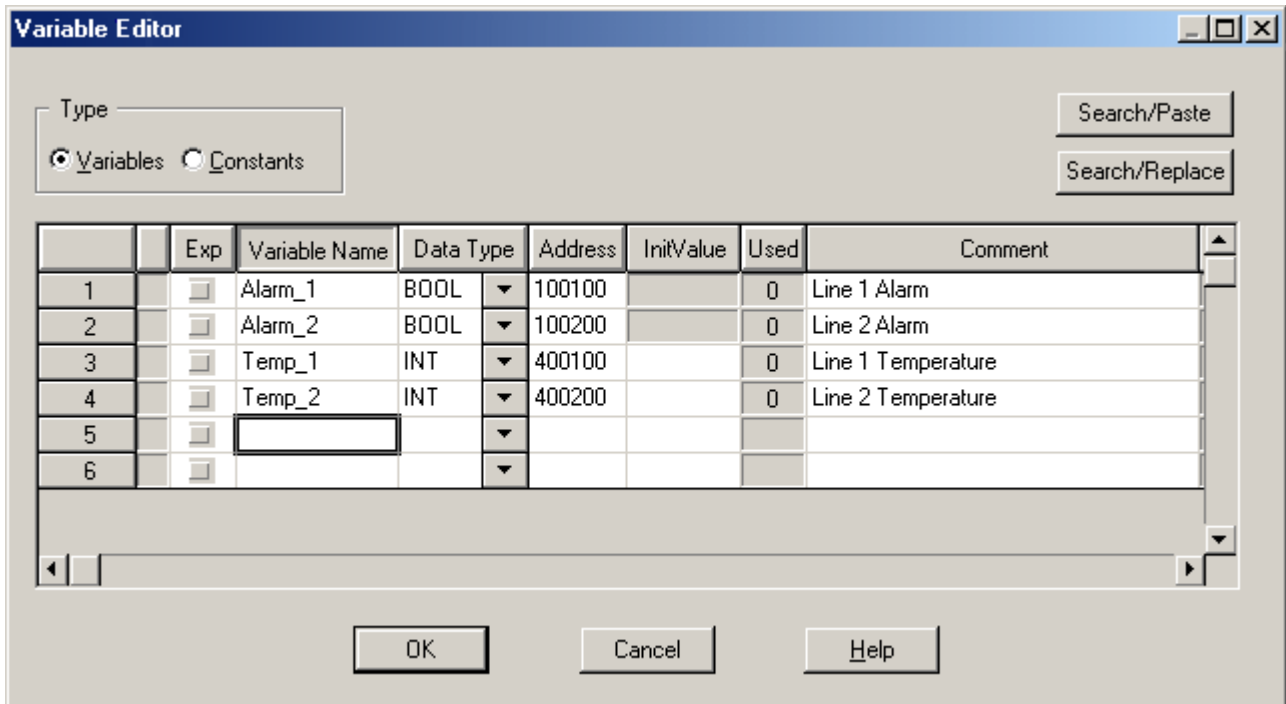
**Properties** | **Variable Import Settings** tab. For more information, refer to [Variable Import Settings](#).

### Operation

Depending on the configuration, tag generation may start automatically when the OPC Server project starts or be initiated manually at some other time. The OPC Server's event log will show when the tag generation process started, any errors that occurred while processing the variable import file and when the process completed.

### Exporting Variables from Concept

As the ladder program is created, symbolic names for various data points referenced can be defined using the **Variable Editor**. Additional symbols and constants that are not used by the ladder program can also be defined.



**Note:** Though Concept is used to define variable names that begin with an underscore, such names are not allowed by the OPC server. The driver will modify invalid imported tag names as needed, and inform users of any such name changes in the server's event log.

User defined data types are not currently supported by this driver. Records in the export file containing references to such types will be ignored. The table below displays the supported simple data types.

Concept Data Type	Generated Tag Data Type
BOOL	Boolean
Byte	Word
DINT	Long
INT	Short
REAL	Float
TIME	DWord
UDINT	DWord
UINT	Word
Word	Word

**Note 1:** Unlocated variables, which do not correspond to a physical address in the device, will be ignored by the driver.

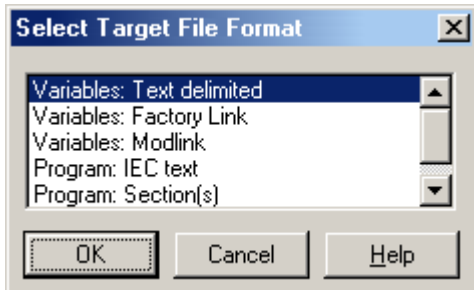
**Note 2:** Comments are allowed and can be included as the generated tag descriptions or not. For more information,

refer to [Variable Import Settings](#).

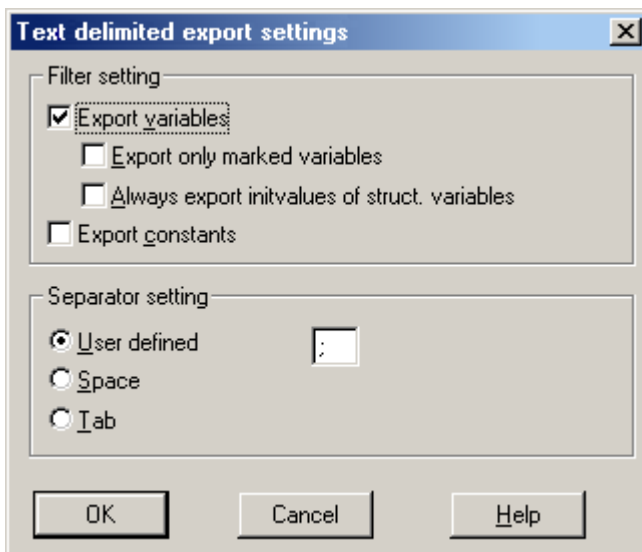
### Exporting Variable from Concept

After the variables have been defined, they must be exported from Concept.

1. Click **File | Export**. Then, select the **Variables: Text delimited** format.



2. Click **OK**. Next, specify the filter and separator settings.



3. Although any filter setting can be chosen, this driver will only be able to read the exported data if the default semicolon separator is used. Click **OK** to generate the file.

### Exporting Variables from ProWORX

In order for ProWORX to export the necessary variable information, **Symbols** must be checked. To do so, click **File | Preferences** and make sure that it is selected.

**Note:** As the ladder program is created, symbolic names for the various data points reference can be defined using the **Document Editor**.

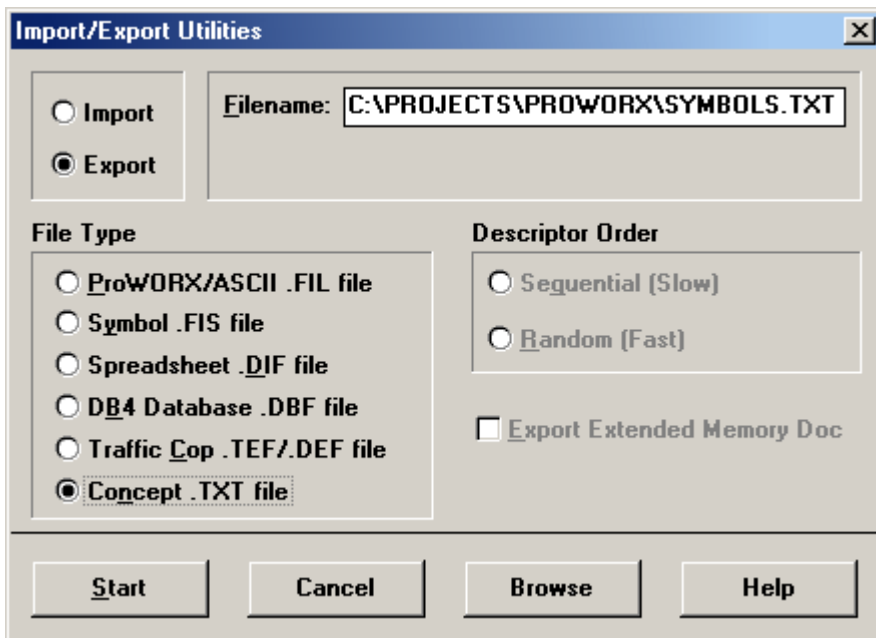
**Note:** ProWORX does not place many restrictions on variable names. However, the OPC Server requires that tag names consist of only alphanumeric characters and underscores, and that the first character not be an underscore. The driver will modify invalid imported tag names as needed, and inform users of any such name changes in the server's event log.

ProWORX will assign a data type of either BOOL or INT to the exported variables. The driver will create tags of type Boolean and Short respectively. In order to generate tags with other data types, users should manually edit the exported file and use any of the supported Concept data types. For a list of supported types, refer to [Exporting Variables from Concept](#).

### Exporting Variables from ProWORX

Once the variables have been defined, they must be exported from ProWORX.

1. Click **File | Utilities | Import/Export**.
2. Select **Export** and the **Concept .TXT file** format.
3. Descriptors are allowed and may be included as the generated tag descriptions. For more information, refer to [Variable Import Settings](#).



4. Click **OK** to generate the file.

## Data Types Description

Data Type	Description
Boolean	Single 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.
String	Null terminated ASCII string Supported on Modbus Model, includes HiLo LoHi byte order

	selection. 8 Byte and 16 Byte Omni Flow Computer string data
Double*	64 bit floating point value  The driver interprets four consecutive registers as a double precision value by making the last two registers the high DWord and the first two registers the low DWord.
Double Example	If register 40001 is specified as a double, bit 0 of register 40001 would be bit 0 of the 64 bit data type and bit 15 of register 40004 would be bit 63 of the 64 bit data type.
Float*	32 bit floating point value  The driver interprets two consecutive registers as a single precision value by making the last register the high word and the first register the low word.
Float Example	If register 40001 is specified as a float, bit 0 of register 40001 would be bit 0 of the 32 bit data type and bit 15 of register 40002 would be bit 31 of the 32 bit data type.

\*The descriptions assume the default first DWord low data handling of 64 bit data types, and first word low data handling of 32 bit data types.

## 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.

### [Modbus Addressing](#)

### [Magnetek GPD 515 Drive Addressing](#)

### [Elliott Flow Computer Addressing](#)

### [Daniels S500 Flow Computer Addressing](#)

### [Dynamic Fluid Meter Addressing](#)

### [Omni Flow Computer Addressing](#)

## Address Descriptions: Modbus

The default data types for dynamically defined tags are shown in **bold** where appropriate.

### Modbus Addressing Decimal Format

Address	Range	Data Type	Access*
Output Coils [Function Codes (decimal): 01, 05, 15]	000001-065536	<b>Boolean</b>	Read/Write
Input Coils [Function Code (decimal): 02]	100001-165536	<b>Boolean</b>	Read Only
Internal Registers [Function Code (decimal): 04]	300001-365536 300001-365535 300001-365533  3xxxxx.0/1-3xxxxx.15/16  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	<b>Word</b> , Short, BCD Float, DWord, Long, LBCD Double  Boolean	Read Only
Internal Registers As String with HiLo Byte Order [Function Codes (decimal): 04]	300001.2H-365536.240H  .Bit is string length, range 2 to 240	<b>String**</b>	Read Only

	bytes.		
Internal Registers As String with LoHi Byte Order	300001.2L-365536.240L	<b>String**</b>	Read Only
[Function Codes (decimal): 04]	.Bit is string length, range 2 to 240 bytes.		
Holding Registers	400001-465536	<b>Word</b> , Short, BCD	Read/Write
[Function Codes (decimal): 03, 06, 16]	400001-465535	Float, DWord, Long, LBCD	
	400001-465533	Double	
	4xxxxx.0/1-4xxxxx.15/16	Boolean	
[Function Codes (decimal): 03, 06, 16, 22]	(See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)		
Holding Registers As String with HiLo Byte Order	400001.2H-465536.240H	<b>String**</b>	Read/Write
[Function Codes (decimal): 03, 16]	.Bit is string length, range 2 to 240 bytes.		
Holding Registers As String with LoHi Byte Order	400001.2L-465536.240L	<b>String**</b>	Read/Write
[Function Codes (decimal): 03, 16]	.Bit is string length, range 2 to 240 bytes.		

### Modbus Addressing Hexadecimal Format

Address	Range	Data Type	Access
Output Coils	H000001-H0FFFF	<b>Boolean</b>	Read/Write
[Function Codes (decimal): 01, 05, 15]			
Input Coils	H100001-H1FFFF	<b>Boolean</b>	Read Only
[Function Code (decimal): 02]			
Internal Registers	H300001-H310000	<b>Word</b> , Short, BCD	Read Only
[Function Code (decimal): 04]	H300001-H3FFFF	Float, DWord, Long, LBCD	
	H300001-H3FFFF	Double	
	H3xxxxx.0/1-H3xxxxx.F/10	Boolean	
	(See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)		
Internal Registers As String with HiLo Byte Order	H300001.2H-H3FFFF.240H	<b>String**</b>	Read Only
[Function Codes (decimal): 04]	.Bit is string length, range 2 to 240 bytes.		
Internal Registers As String with LoHi Byte Order	H300001.2L-H3FFFF.240L	<b>String**</b>	Read Only
[Function Codes (decimal): 04]	.Bit is string length, range 2 to 240 bytes.		
Holding Registers	H400001-H410000	<b>Word</b> , Short, BCD	Read/Write
[Function Codes (decimal): 03, 06, 16]	H400001-H4FFFF	Float, DWord, Long, LBCD	
	H400001-H4FFFF	Double	
	H4xxxxx.0/1-H4xxxxx.F/10	Boolean	
	(See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)		

Holding Registers As String with HiLo Byte Order	H400001.2H-H4FFFF.240H	<b>String**</b>	Read/Write
[Function Codes (decimal): 03, 16]	.Bit is string length, range 2 to 240 bytes.		
Holding Registers As String with LoHi Byte Order	H400001.2L-H4FFFF.240L	<b>String**</b>	Read/Write
[Function Codes (decimal): 03, 16]	.Bit is string length, range 2 to 240 bytes.		

### \*Write Only Access

All Read/Write addresses may be set as Write Only by prefixing a "W" to the address such as "W40001", which will prevent the driver from reading the register at the specified address. Any attempts by the client to read a Write Only tag will result in obtaining the last successful write value to the specified address. If no successful writes have occurred, the client will receive 0/NULL for numeric/string values for an initial value.

**Caution:** Setting the Client Access privileges of Write Only tags to Read Only will cause writes to these tags to fail and the client to always receive 0/NULL for numeric/string values.

### \*\*String Support

The Modbus model supports reading and writing holding register memory as an ASCII string. When using holding registers for string data, each register will contain two bytes of ASCII data. The order of the ASCII data within a given register can be selected when the string is defined. The length of the string can be from 2 to 240 bytes and is entered in place of a bit number. The length must be entered as an even number. The byte order is specified by appending either a "H" or "L" to the address.

### String Examples

- To address a string starting at 40200 with a length of 100 bytes and HiLo byte order, enter:  
40200.100H
- To address a string starting at 40500 with a length of 78 bytes and LoHi byte order, enter:  
40500.78L

**Note:** The string's length may be limited by the maximum size of the write request that the device will allow. If, while utilizing a string tag, an error message of "Unable to write to address <address> on device <device>: Device responded with exception code 3" is received in the server event window, this means the device did not like the string's length. If possible, try shortening the string.

### Normal Address Examples

- The 255'th output coil would be addressed as '0255' using decimal addressing or 'H0FF' using hexadecimal addressing.
- Some documentation refers to Modbus addresses by function code and location. For instance, function code 3; location 2000 would be addressed as '42000' or 'H47D0' (the leading '4' represents holding registers or function code 3).
- Some documentation refers to Modbus addresses by function code and location. For instance, setting function code 5 location 100 would be addressed as '0100' or 'H064' (the leading '0' represents output coils or function code 5). Writing 1 or 0 to this address would set or reset the coil.

### Array Support

Arrays are supported for internal and holding register locations for all data types except for Boolean and strings. Arrays are also supported for input and output coils (Boolean data types). There are two methods of addressing an array. Examples are given using holding register locations.

4xxxx [rows] [cols]  
4xxxx [cols] this method assumes rows is equal to one.

For arrays, rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register/coil type. For register arrays of 32 bit data types, rows multiplied by cols multiplied by 2 cannot exceed the block size.

### Packed Coil Address Type

The Packed Coil address type allows access to multiple consecutive coils as an analog value. This feature is available for both input coils and output coils, polled mode only. The only valid data type is Word. The syntax is:

Output coils: 0xxxxx#nn Word Read/Write

Input coils: 1xxxxx#nn Word Read Only

where xxxxx is the address of the first coil (decimal and hex values allowed), and nn is the number of coils to be packed into an analog value (1-16, decimal only).

The bit order will be such that the start address will be the LSB (least significant bit) of analog value.

## Address Descriptions: Magnetek GPD 515

This table provides the general ranges of data available from the Magnetek GPD 515 Drive. Consult the Magnetek Modbus RTU Technical Manual, part number TM4025, for information on how specific Drive parameters can be accessed using Modbus RTU addressing. In all cases, the letter H (used to signify Hex addressing) should precede the desired address. The default data types for dynamically defined tags are shown in **bold** where appropriate.

### Magnetek GPD 515 Addressing Hexadecimal Format

Address	Range	Data Type	Access
Command Registers	H40001-H4000F	<b>Word</b> , Short	Read/Write
Bit Level Access	H4xxxx.0/1-H4xxxx.F/10  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	Boolean	
Monitor Registers	H40010-H4001A	<b>Word</b> , Short	Read Only
Bit Level Access	H4xxxx.0/1-H4xxxx.F/10  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	Boolean	
Drive Parameter Registers (Monitor Only)	H40020-H40097	<b>Word</b> , Short	Read Only
Bit Level Access	H4xxxx.0/1-H4xxxx.F/10  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	Boolean	
Drive Parameter Registers	H40100-H4050D	<b>Word</b> , Short	Read/Write
Bit Level Access	H4xxxx.0/1-H4xxxx.F/10  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	Boolean	
Special Registers	H4FFDD ACCEPT H4FFFD ENTER	Word, Short	Write Only

### Example

To access the Driver's Operation Status, address 02BH, enter the following address:  
H4002B

**Note:** When adding a Magnetek Device to the OPC Server project, users must make sure that the setting "Use Zero Based Addressing" is not checked. If this parameter is not set correctly, the Modbus RTU driver will offset all of the Magnetek addresses by 1.

### Array Support

Arrays are supported for holding register locations for all data types except Boolean. There are two methods of addressing an array. Examples are given using holding register locations.

4xxxx [rows] [cols]  
 4xxxx [cols] this method assumes rows is equal to one.

Rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register type.

### Address Descriptions: Elliot

The default data types for dynamically defined tags are shown in **bold** where appropriate.

Address	Range	Data Type	Access
Output Coils	000001-065536	<b>Boolean</b>	Read/Write
Input Coils	100001-165536	<b>Boolean</b>	Read Only
Internal Registers	300001-365536 300001-365535	<b>Word</b> , Short, BCD Float, DWord, Long, LBCD	Read Only
	3xxxxx.0/1-3xxxxx.15/16  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	Boolean	
Holding Registers	400001-465536 400001-465535	<b>Word</b> , Short, BCD (1) Float, DWord, Long, LBCD	Read/Write
	4xxxxx.0/1-4xxxxx.15/16  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	Boolean	

**Note:** Address ranges 405001 to 405315 and 407001 to 407315 are 32 bit registers. Addresses in the range of 405001 to 405315 use a default data type of Long. Addresses in the range of 407001 to 407315 use a default data type of Float. Since these address registers are 32 bit, only Float, DWord, Long or LBCD data types are allowed. Arrays are not allowed for these special address ranges.

### Array Support

Arrays are supported for internal and holding register locations for all data types except Boolean. There are two methods of addressing an array. Examples are given using holding register locations.

4xxxx [rows] [cols]  
 4xxxx [cols] this method assumes "rows" is equal to one.

Rows multiplied by cols cannot exceed the block size that has been assigned to the device for the register type. For arrays of 32 bit data types, rows multiplied by cols multiplied by 2 cannot exceed the block size.

### Address Description: Daniel S500

The default data types for dynamically defined tags are shown in **bold** where appropriate.

Address	Hex Range	Decimal Range	Data Type	Function Codes	Access
Totals	000-0FF	4096-4351	Double	03	Read Only
Calculated / Measured Variables	100-24F	4352-4687	Float	03, 16	Read/Write
Calculation Constants	250-28F	4688-4751	Float	03, 16	Read/Write
Keypad Default Values	290-2AF	4752-4783	Float	03, 16	Read/Write
Alarm and Scaling Constants	2B0-5FF	4784-5631	Float	03, 16	Read/Write

Status /Control	700-7FF	5888-6143	<b>Boolean</b>	02, 5	Read/Write
Alarms	800-FFF	6144-8191	<b>Boolean</b>	02	Read Only

### Address Descriptions: Dynamic Fluid Meter SFC3

The default data types for dynamically defined tags are shown in **bold** where appropriate.

#### Dynamic Fluid Meter Addressing Decimal Format

Address	Range	Data Type	Access
Holding Registers (16 bit)	400000-407000	<b>Word</b> , Short, BCD	Read/Write
	400000-406999	Float, DWord, Long, LBCD	
	408001-465535	<b>Word</b> , Short, BCD	
	408001-465534	Float, DWord, Long, LBCD	
	4xxxxx.0/1-4xxxxx.15/16	Boolean	
	(See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)		
Holding Registers (32 bit)	407001-408000	<b>Float</b>	Read/Write
Holding Registers As String with HiLo Byte Order	400000.2H-407000.240H	<b>String</b>	Read/Write
	408001.2H-465535.240H		
	.Bit is string length, range 2 to 240 bytes.		
Holding Registers As String with LoHi Byte Order	400000.2L-407000.240L	<b>String</b>	Read/Write
	408001.2L-465535.240L		
	.Bit is string length, range 2 to 240 bytes.		

#### Dynamic Fluid Meter Addressing Hexadecimal Format

Address	Range	Data Type	Access
Holding Registers (16 bit)	H400000-H401B58	<b>Word</b> , Short, BCD	Read/Write
	H400000-H401B57	Float, DWord, Long, LBCD	
	H401F41-H40FFFF	<b>Word</b> , Short, BCD	
	H401F41-H40FFFE	Float, DWord, Long, LBCD	
	H4xxxxx.0/1-H4xxxxx.F/10	Boolean	
	(See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)		
Holding Registers (32 bit)	H401B59-H401F40	<b>Float</b>	Read/Write
Holding Registers As String with HiLo Byte Order	H400000.2H-H401B58.240H	<b>String</b>	Read/Write
	H401F41.2H-H40FFFF.240H		
	.Bit is string length, range 2 to 240 bytes.		
Holding Registers As String with LoHi Byte Order	H400000.2L-H401B58.240L	<b>String</b>	Read/Write
	H401F41.2L-H0FFFF.240L		
	.Bit is string length, range 2 to 240 bytes.		

**Note:** This driver requires that all addresses begin with "4" for the Dynamic Fluid Meter model. This 4 may not always be written explicitly in the Dynamic Fluid Meter documentation. For example, users may see a reference to "Unit ID at address 3001". This value must be addressed in the server as "403001".

## String Support

The Dynamic Fluid Meter model supports reading and writing holding register memory as an ASCII string. When using holding registers for string data, each register will contain two bytes of ASCII data. The order of the ASCII data within a given register can be selected when the string is defined. The length of the string can be from 2 to 240 bytes and is entered in place of a bit number. The length must be entered as an even number. The byte order is specified by appending either a "H" or "L" to the address.

## String Examples

1. To address a string starting at 40200 with a length of 100 bytes and HiLo byte order, enter:  
40200.100H
2. To address a string starting at 40500 with a length of 78 bytes and LoHi byte order, enter:  
40500.78L

**Note:** The string's length may be limited by the maximum size of the write request that the device will allow. If, while utilizing a string tag, an error message of "Unable to write to address <address> on device <device>: Device responded with exception code 3" is received in the server event window, this means the device did not like the string's length. If possible, try shortening the string.

## Address Descriptions: Omni

The default data types for dynamically defined tags are shown in **bold** where appropriate.

Address	Range	Data Type	Access
Digital I/O Point	1001-1024	<b>Boolean</b>	Read/Write
Programmable Boolean Point	1025-1088	<b>Boolean</b>	Read/Write
Meter Run Status and Alarm Points	1n01-001n59 1n76-1n99 n=Number of Meter Run	<b>Boolean</b>	Read/Write
Micro Motion Alarm Status Points	1n60-1n75 n=Number of Meter Run	<b>Boolean</b>	Read/Write
User Scratch Pad Boolean Points	1501-1599 1601 -1649	<b>Boolean</b>	Read/Write
User ScratchPad One Shot Points	1650-1699	<b>Boolean</b>	Read/Write
Command Boolean Points/Variables	1700-1798	<b>Boolean</b>	Read/Write
Meter Station Alarm and Status Points	1801-1899	<b>Boolean</b>	Read/Write
Prover Alarm and Status Points	1901-1967	<b>Boolean</b>	Read/Write
Meter Totalizer Roll-over Flags	2n01-2n37 n=Number of Meter Run	<b>Boolean</b>	Read/Write
Misc. Meter Station Alarm and Status	2601-2623	<b>Boolean</b>	Read/Write
Station Totalizer Roll-over Flags	2801-2851	<b>Boolean</b>	Read/Write
Station Totalizer Decimal Resolution	2852-2862 2865-2999	<b>Boolean</b>	Read/Write

16 Bit Integer Data Addresses	Range	Data Type	Access
Custom Data Packet #1	3001-3040	<b>Short</b> , Word, BCD	Read/Write
Custom Data Packet #2	3041-3056	<b>Short</b> , Word, BCD	Read/Write
Custom Data Packet #3	3057-3096	<b>Short</b> , Word, BCD	Read/Write
Misc. 16 bit Integer Data	3097-3099 3737-3799 3875-3899	<b>Short</b> , Word, BCD	Read/Write
Meter Run 16 bit Integer Data	3n01-3n52 n=Number of Meter Run	<b>Short</b> , Word, BCD	Read/Write
Scratchpad 16 bit Integer Data	3501-3599	<b>Short</b> , Word, BCD	Read/Write
User Display #1	3601-3608	<b>Short</b> , Word, BCD	Read/Write

User Display #2	3609-3616	<b>Short</b> , Word, BCD	Read/Write
User Display #3	3617-3624	<b>Short</b> , Word, BCD	Read/Write
User Display #4	3625-3632	<b>Short</b> , Word, BCD	Read/Write
User Display #5	3633-3640	<b>Short</b> , Word, BCD	Read/Write
User Display #6	3641-3648	<b>Short</b> , Word, BCD	Read/Write
User Display #7	3649-3656	<b>Short</b> , Word, BCD	Read/Write
User Display #8	3657-3664	<b>Short</b> , Word, BCD	Read/Write
Access Raw Data Archive Records	3701-3736	<b>Short</b> , Word, BCD	Read/Write
Meter Station 16 bit Integer Data	3800-3842	<b>Short</b> , Word, BCD	Read/Write
Meter #1 Batch Sequence	3843-3848	<b>Short</b> , Word, BCD	Read/Write
Meter #2 Batch Sequence	3849-3854	<b>Short</b> , Word, BCD	Read/Write
Meter #3 Batch Sequence	3855-3860	<b>Short</b> , Word, BCD	Read/Write
Meter #4 Batch Sequence	3861-3866	<b>Short</b> , Word, BCD	Read/Write
Flow Computer Time/Date	3867-3874	<b>Short</b> , Word, BCD	Read/Write
Prover 16 bit Integer Data	3901-3999	<b>Short</b> , Word, BCD	Read/Write

<b>8 Character ASCII String Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Meter Run ASCII Data	4n01-4n39 n=Number of Meter Run	<b>String</b>	Read/Write
Scatch Pad ASCII Data	4501-4599	<b>String</b>	Read/Write
User Display Definition Variables	4601 -4640	<b>String</b>	Read/Write
Station Auxiliary Input Variables	4707-4710	<b>String</b>	Read/Write
Meter Station ASCII Data	4801-4851	<b>String</b>	Read/Write
Meter #1 Batch ID	4852-4863	<b>String</b>	Read/Write
Meter #2 Batch ID	4864-4875	<b>String</b>	Read/Write
Meter #3 Batch ID	4876-4887	<b>String</b>	Read/Write
Meter #4 Batch ID	4888-4899	<b>String</b>	Read/Write
Prover ASCII String Data	4901-4942	<b>String</b>	Read/Write

<b>32 Bit Integer Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Meter Run 32 bit Integer Data	5n01-5n99 n=Number of Meter Run	<b>Long</b> , DWord, LBCD, Float	Read/Write
Scratch Pad 32 bit Integer Data	5501-5599	<b>Long</b> , DWord, LBCD, Float	Read/Write
Station 32 bit Integer Data	5801-5818	<b>Long</b> , DWord, LBCD, Float	Read/Write
Meter #1 Batch Size	5819-5824	<b>Long</b> , DWord, LBCD, Float	Read/Write
Meter #2 Batch Size	5825-5830	<b>Long</b> , DWord, LBCD, Float	Read/Write
Meter #3 Batch Size	5831-5836	<b>Long</b> , DWord, LBCD, Float	Read/Write
Meter #4 Batch Size	5837-5842	<b>Long</b> , DWord, LBCD, Float	Read/Write
Additional 32 bit Meter Run Data	5843-5899	<b>Long</b> , DWord, LBCD, Float	Read/Write
Prover 32 bit Integer Data	5901-5973	<b>Long</b> , DWord, LBCD, Float	Read/Write
Compact Prover TDVOL/TDFMP Pulses	5974-5999	<b>Long</b> , DWord, LBCD, Float	Read/Write

<b>32 Bit IEEE Floating Point Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Reserved Data	6001-7000	<b>Float</b> , Long, DWord, LBCD	Read/Write
Digital to Analog Outputs	7001-7024	<b>Float</b> , Long, DWord, LBCD	Read/Write
User Variables	7025-7088	<b>Float</b> , Long, DWord, LBCD	Read/Write
Programmable Accumulator	7089-7099	<b>Float</b> , Long, DWord, LBCD	Read/Write
Meter Run Data	7n01 - 7n99 n=Number of Meter Run	<b>Float</b> , Long, DWord, LBCD	Read/Write

Scatch Pad Data	7501-7599	<b>Float</b> , Long, DWord, LBCD	Read/Write
PID Control Data	7601-7623	<b>Float</b> , Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run Data	7624-7699	<b>Float</b> , Long, DWord, LBCD	Read/Write
Miscellaneous Variables	7701-7799	<b>Float</b> , Long, DWord, LBCD	Read/Write
Meter Station Data	7801-7899	<b>Float</b> , Long, DWord, LBCD	Read/Write
Prover Data	7901-7918	<b>Float</b> , Long, DWord, LBCD	Read/Write
Configuration Data for Prover	7919-7958	<b>Float</b> , Long, DWord, LBCD	Read/Write
Last Prove Data	7959-7966	<b>Float</b> , Long, DWord, LBCD	Read/Write
Data Rejected During Prove	7967-7990	<b>Float</b> , Long, DWord, LBCD	Read/Write
Prove Run Data	7991-8050	<b>Float</b> , Long, DWord, LBCD	Read/Write
Prove Average Data	8051-8079	<b>Float</b> , Long, DWord, LBCD	Read/Write
Prove Run-Master Meter Data	8080-8199	<b>Float</b> , Long, DWord, LBCD	Read/Write
Proving Series Data	8200-8223	<b>Float</b> , Long, DWord, LBCD	Read/Write
Data of Meter Being Proved	8224-8230	<b>Float</b> , Long, DWord, LBCD	Read/Write
Mass Prove Data	8231-8500	<b>Float</b> , Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #1	8501-8599	<b>Float</b> , Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #2	8601-8699	<b>Float</b> , Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #3	8701-8799	<b>Float</b> , Long, DWord, LBCD	Read/Write
Miscellaneous Meter Run #4	8801-8899	<b>Float</b> , Long, DWord, LBCD	Read/Write
Station Previous Batch Average Data	8901-8999	<b>Float</b> , Long, DWord, LBCD	Read/Write

<b>16 Bit Integer Configuration Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Meter Run #1	13001-13013	<b>Short</b> , Word, BCD	Read/Write
Meter Run #2	13014-13026	<b>Short</b> , Word, BCD	Read/Write
Meter Run #3	13027-13039	<b>Short</b> , Word, BCD	Read/Write
Meter Run #4	13040-13052	<b>Short</b> , Word, BCD	Read/Write
Prover Configuration	13053-13073	<b>Short</b> , Word, BCD	Read/Write
General Flow Configuration	13074-13084	<b>Short</b> , Word, BCD	Read/Write
Serial Port Configuration	13085-13128	<b>Short</b> , Word, BCD	Read/Write
PID Configuration	13129-13160	<b>Short</b> , Word, BCD	Read/Write
PLC Data	13161-13299	<b>Short</b> , Word, BCD	Read/Write
Peer to Peer Setup	13300-13499	<b>Short</b> , Word, BCD	Read/Write
Raw Data Archive	13500-13999	<b>Short</b> , Word, BCD	Read/Write

<b>16 Character ASCII String Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Flow Computer Configuration	14001-14499	<b>String</b>	Read/Write

<b>32 Bit Integer Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Flow Computer Configuration	15001-16999	<b>Long</b> , DWord, LBCD, Float	Read/Write

<b>32 Bit IEEE Floating Point Data</b>	<b>Range</b>	<b>Data Type</b>	<b>Access</b>
Flow Computer Configuration	17001-18999	<b>Float</b> , Long, DWord, LBCD	Read/Write

### Supported Extended Omni Types

[Custom Packets](#)

[Raw Data Archive](#)

[Text Reports](#)

[Text Archive](#)

## Omni Custom Packets

The Omni Flow Computer allows users to map various ranges of memory to a single data structure that can be read with a single, highly efficient read command. These data structures are called Custom Packets.

### Packet Configuration

Each Custom Packet may contain up to twenty groups of data points. Each group is defined by its starting index and the number of data points. The total size of the custom packet must not exceed 250 bytes. The addresses used to define the Custom Packets are listed below.

#### Custom Packet 1 (address 1)

3001 Group 1-Starting index  
3002 Group 2-Number of points

to

3039 Group 20-Starting index  
3040 Group 20-Number of points

#### Custom Packet 2 (address 201)

3041 Group 1-Starting index  
3042 Group 2-Number of points

to

3055 Group 20-Starting index  
3056 Group 20-Number of points

#### Custom Packet 3 (address 401)

3057 Group 1-Starting index  
3058 Group 2-Number of points

to

3095 Group 20-Starting index  
3096 Group 20-Number of points

**Note:** Data is returned from the device as 16 bit registers. Digital I/O must be mapped in blocks of 16 bits.

### Custom Packet Address Syntax

Tags can be created to access data at a given offset within a Custom Packet. The address syntax is as follows; the default data types are shown in **bold**.

Address	Range	Data Type	Access
CPn_o	n = Packet Number (1-3) o = Word offset (0-125)	Word, <b>Short</b> , BCD, DWord, Long, LBCD, Float, String	Read Only
CPn_o.b	n = Packet Number (1-3) o = Word offset (0-125) b = Bit number (0/1-15/16)  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	<b>Boolean</b>	Read Only

**Note 1:** Only 8 character ASCII string data is supported.

**Note 2:** If a 16 character ASCII string data address is contained in group configuration, then data can be read as two 8 character ASCII string data items.

### Example:

Define Custom Packet #1 to map to: 16 bits of Digital I/O (1001-1016), 15 32 bit integers of Meter Run 1 Batch data (5101 -5115), 12 32 bit floats of Analog Outputs (7001-7012), 4 8 character ASCII strings of Meter Run (4101-4104), 6 8 character ASCII strings of Meter Station (4808-4813) and 2 16 character ASCII strings of Flow Configuration data (14001-14002), for a total of 222 bytes. The Custom Packet configuration registers would have the following values:

3001 = 1001  
3002 = 16  
3003 = 5101  
3004 = 15  
3005 = 7001  
3006 = 12  
3007 = 4101  
3008 = 4  
3009 = 4808  
3010 = 6  
3011 = 14001  
3012 = 2

Tags to access the Digital I/O data would have the following addresses (all 16 values contained in word 0):

CP1\_0.0 (Word 0 of Custom Packet 1, bit 0-mapped to 1001)  
CP1\_0.1 (Word 0 of Custom Packet 1, bit 1-mapped to 1002)  
...  
CP1\_0.15 (Word 0 of Custom Packet 1, bit 15-mapped to 1015)

Tags to access the Meter Run 1 Batch data would have the following addresses (each 32 bit value uses 2 words):

CP1\_1 (Word 1 of Custom Packet 1-mapped to 5101)  
CP1\_3 (Word 3 of Custom Packet 1-mapped to 5102)  
...  
CP1\_29 (Word 29 of Custom Packet 1-mapped to 5115)

Tags to access the Analog Output data would have the following addresses (each 32 bit value uses 2 words):

CP1\_31 (Word 31 of Custom Packet 1-mapped to 7001)  
CP1\_33 (Word 33 of Custom Packet 1-mapped to 7002)  
...  
CP1\_53 (Word 53 of Custom Packet 1-mapped to 7012)

Tags to access the Meter Run 8 character ASCII String data would have the following addresses (each String value uses 4 words):

CP1\_55 (Word 55 of Custom Packet 1-mapped to 4101)  
...  
CP1\_67 (Word 67 of Custom Packet 1-mapped to 4104)

Tags to access the Meter Station 8 character ASCII String data would have the following addresses (each String value uses 4 words):

CP1\_71 (Word 71 of Custom Packet 1-mapped to 4808)  
...  
CP1\_91 (Word 91 of Custom Packet 1-mapped to 4813)

Tags to access the Flow Configuration 16 character ASCII String data would have the following addresses (each String value uses 4 words):

CP1\_95 (Word 95 of Custom Packet 1-mapped to 14001 characters 1-8)  
CP1\_99 (Word 99 of Custom Packet 1-mapped to 14001 characters 9-16)  
CP1\_103 (Word 103 of Custom Packet 1-mapped to 14002 characters 1-8)  
CP1\_107 (Word 107 of Custom Packet 1-mapped to 14002 characters 9-16)

---

## **Omni Raw Data Archive**

The Omni Flow Computer may be configured to map various ranges of memory to a single data structure, and store that structure in an archive when triggered. Up to ten archives may be configured by the user. There are two additional fixed format archives for Alarm and Audit data. Each archive is a circular buffer, where each new record replaces the oldest record.

### **Record Configuration and Retrieval**

The record structure of Raw Data Archives 1 to 10 may be configured by the user. Archives 11 and 12 are of fixed format and contain Alarm and Audit data respectively. For a full discussion of Raw Data Archives, refer to Omni Technical Bulletin 96073.

Each record may contain up to sixteen groups of data points. Each group is defined by its starting index and the number of data points. The addresses used to define the archive records are listed below. The total size of the record

must not exceed 250 bytes. The device will use the first 6 bytes for date and time stamp data (see below for format), leaving 244 bytes for raw data. Each record will have its own Boolean trigger. Data will be stored when the trigger goes from low to high.

Before a group starting index, number of points in group, or trigger for a raw data archive can be changed, archiving must halt. The **Allow Archive Configuration Flag** must be set in the device. Be aware that doing this will likely cause the data archive in the device to be reinitialized. This includes all Raw Data Archives and the Text Archive.

13920 Archive Run-0=stop, 1= start

13921 Reconfigure Archives-0=no configuration changes allowed, 1=configuration changes allowed

This driver may be used to read a Raw Data Archive one record at a time. To read a record, first write the desired record index to the "Requested record" register. Once this value is set, users may read the record with an "RA" tag. Make sure the specified record index does not exceed the maximum number of records allowed for that archive. If the "Last record updated" value is zero, there have been no records saved in the archive since it was last initialized.

#### **Raw Data Archive 1 (address 701)**

13500 Group 1-Starting Index

13501 Group 1-Number of Points

to

13530 Group 16-Starting Index

13531 Group 16-Number of points

13900 Trigger Boolean

3701 Maximum number of records

3702 Last record updated

3703 Requested record

#### **Raw Data Archive 2 (address 702)**

13540 Group 1-Starting Index

13541 Group 1-Number of Points

to

13570 Group 16-Starting Index

13571 Group 16-Number of points

13901 Trigger Boolean

3704 Maximum number of records

3705 Last record updated

3706 Requested record

#### **Raw Data Archive 3 (address 703)**

13580 Group 1-Starting Index

13581 Group 1-Number of Points

to

13610 Group 16-Starting Index

13611 Group 16-Number of points

13902 Trigger Boolean

3707 Maximum number of records

3708 Last record updated

3709 Requested record

#### **Raw Data Archive 4 (address 704)**

13620 Group 1-Starting Index

13621 Group 1-Number of Points

to

13650 Group 16-Starting Index  
13651 Group 16-Number of points

13903 Trigger Boolean

3710 Maximum number of records  
3711 Last record updated  
3712 Requested record

**Raw Data Archive 5 (address 705)**

13660 Group 1-Starting Index  
13661 Group 1-Number of Points

to

13690 Group 16-Starting Index  
13691 Group 16-Number of points

13904 Trigger Boolean

3713 Maximum number of records  
3714 Last record updated  
3715 Requested record

**Raw Data Archive 6 (address 706)**

13700 Group 1-Starting Index  
13701 Group 1-Number of Points

to

13730 Group 16-Starting Index  
13731 Group 16-Number of points

13905 Trigger Boolean

3716 Maximum number of records  
3717 Last record updated  
3718 Requested record

**Raw Data Archive 7 (address 707)**

13740 Group 1-Starting Index  
13741 Group 1-Number of Points

to

13770 Group 16-Starting Index  
13771 Group 16-Number of points

13906 Trigger Boolean

3719 Maximum number of records  
3720 Last record updated  
3721 Requested record

**Raw Data Archive 8 (address 708)**

13780 Group 1-Starting Index  
13781 Group 1-Number of Points

to

13810 Group 16-Starting Index  
13811 Group 16-Number of points

13907 Trigger Boolean

3722 Maximum number of records  
 3723 Last record updated  
 3724 Requested record

#### Raw Data Archive 9 (address 709)

13820 Group 1-Starting Index  
 13821 Group 1-Number of Points

to

13850 Group 16-Starting Index  
 13851 Group 16-Number of points

13908 Trigger Boolean

3725 Maximum number of records  
 3726 Last record updated  
 3727 Requested record

#### Raw Data Archive 10 (address 710)

13860 Group 1-Starting Index  
 13861 Group 1-Number of Points

to

13890 Group 16-Starting Index  
 13891 Group 16-Number of points

13909 Trigger Boolean

3728 Maximum number of records  
 3729 Last record updated  
 3730 Requested record

#### Raw Data Archive 11 Alarm (address 711)

Not configurable

3731 Maximum number of records  
 3732 Last record updated  
 3733 Requested record

#### Raw Data Archive 12 Archive (address 712)

Not configurable

3734 Maximum number of records  
 3735 Last record updated  
 3736 Requested record

**Note:** Data is returned from the device as 16 bit registers. Digital I/O must be mapped in blocks of 16 bits.

#### Raw Data Archive Address Syntax

Tags can be created to access data at a given offset within a Raw Data Archive record. The address syntax is as follows. The default data types are shown in **bold**.

Address	Range	Data Type	Access
RA <sub>n</sub> _o	n = Archive Number (1-12) o = Word offset (0-125)	Word, <b>Short</b> , BCD, DWord, Long, LBCD, Float, String	Read Only
RA <sub>n</sub> _o.b	n = Archive Number (1-12) o = Word offset (0-125) b = Bit number (0/1-15/16)  (See <a href="#">Settings</a> Option Use zero based Bit Addressing within registers)	<b>Boolean</b>	Read Only

**Note 1:** Only 8 character ASCII string data is supported.

**Note 2:** If a 16 character ASCII string data address is contained in group configuration, then data can be read as two 8 character ASCII string data items.

### Timestamp Format

The first 6 bytes of each record contains the time and date that the record was placed in the archive.

Byte	Description
1	Month (1-12)* Day (1-31)
2	Day (1-31)* Month (1-12)
3	Year (0-99)
4	Hour (0-23)
5	Minute (0-59)
6	Seconds (0-59)

\*Date format is set with register 3842 (0=dd/mm/yy, 1= mm/dd/yy)

### Alarm/Event Log Record Structure (Address 711)

Field	Data Type	Description
1	3-Byte Date	dd/mm/yy or mm/dd/yy
2	3-Byte Time	hh/mm/ss
3	16 bit Integer	Modbus Index # of alarm or event
4	1 Byte	Alarm Type (see below)
5	1 Byte	0=OK, 1=Alarm
6	IEEE Float	Value of transducer variable at the time of alarm or event
7	32 bit Integer	Volume totalizer at the time of the alarm or event
8	32 bit Integer	Mass totalizer at the time of the alarm or event

### Alarm Types

Type	Description
0	Log event, sound beeper and display in LCD any edge change in bit identified by filed #3.
1	Log event, sound beeper and display in LCD rising edge changes in bit identified by filed #3.
2	Event log any edge change in bit identified by field #3. No beeper or LCD display action.
3	Event log rising edge changes in bit identified by field #3. No beeper or LCD display action.

### Audit Event Log Record Structure (Address 712)

Field	Data Type	Description
1	3-Byte Date	dd/mm/yy or mm/dd/yy
2	3-Byte Time	hh/mm/ss
3	16 bit Integer	Event number, increments for each event, rolls at 65535.
4	16 bit Integer	Modbus index of variable changed.
5	IEEE Float	Numeric variable value before change-old value.
6	IEEE Float	Numeric variable value after change-new value.
7	16-Char ASCII	String variable value before change-old value.
8	16-Char ASCII	String variable value after change-new value.
9	32 bit Integer	Volume totalizer at time of change.
10	32 bit Integer	Mass totalizer at the time of the change.

Fields 5 and 6 are set to 0.0 when the variable type changed is a string. Fields 7 and 8 contain null characters when the

variable type change is not a string. When fields 7 and 8 contain 8 character strings, the remaining 8 characters are padded with nulls.

## Omni Text Reports

The Omni Flow computer can generate several different types of text reports. Each of these reports can be read by this driver and sent to the OPC Client as a string value.

### Text Report Types

There are a number of report types that can be retrieved from the Omni Flow Computer. They may be read using a "TR" tag. The report types are as follows.

#### Custom Report Templates

9001 Report Template-Snapshot / Interval  
 9002 Report Template-Batch  
 9003 Report Template-Daily  
 9003 Report Template-Prove

#### Previous Batch Reports

9101 Batch Report-Last  
 9102 Batch Report-Second from Last  
 ...  
 9108 Batch Report-Eighth from last

#### Previous Prove Reports

9201 Prove Report-Last  
 9202 Prove Report-Second from last  
 ...  
 9208 Prove Report-Eighth from last

#### Previous Daily Reports

9301 Previous Day's Report-Last  
 9302 Previous Day's Report-Second from last  
 ...  
 9308 Previous Day's Report-Eighth from last

#### Last Snapshot Report

9401 Last Local Snapshot / Interval Report

#### Miscellaneous Report Buffer

9402 Miscellaneous Report Buffer

### Text Report Address Syntax

Address	Range	Data Type	Access
TRn	n = Report address	String	Read/Write
TRn T (triggered read)	(9001-9402)		

### Example

To read or write to the Snapshot Report Template (address 9001), create a tag with address "TR9001".

**Note:** Because it can take several seconds to read a Text Report, the "TR" tags should be kept inactive in the OPC client. Alternatively, triggered reads can be used instead. No other tags on the channel can be read or written to while the driver is reading or writing a Text Report.

### Triggered Text Report Reads

As noted above, it is recommended that the Text Report tag be kept inactive, even though it is not always possible. A triggered read capability has been added as an alternative, allowing the Text Archive tag to remain active. It also controls when the actual device reads occur with an auxiliary trigger tag.

A triggered read may not begin immediately depending on when in the Text Report tag's update cycle the trigger is set. After the read attempt has been completed, the driver will clear the trigger state. The Text Report tag will show the value and data quality that resulted from the last triggered read attempt.

### Text Report Read Trigger Address Syntax

Address	Range	Data Type	Access
TRIG_TRn	n = Report address (9001-9402)	Boolean	Read/Write

#### Example

To read the Last Batch Report (address 9101) on trigger, create two tags. The first is a Text Report tag with address "TR9101 T", and the second is a Text Report Read Trigger tag with address "TRIG\_TR9101".

**Note:** The Text Report tag address looks like a normal Text Report address followed by a space and the letter "T" for "triggered read". This "T" must be present in the address for triggered reads to work.

To trigger a read, set the trigger tag value to true (non-zero). After the read attempt has been completed, the driver will set the trigger value to false (0). If the read was successful, the Text Report tag's data quality will be Good. If the read failed, the Text Report tag's data quality will be Bad, and the value will be the last value successfully read.

### Saving Text Report Data To Disk

The driver has the ability to save Text Report data to disk. This feature is enabled by using Text Report Path tags. These tags are used to write file path strings to the driver's memory. Each report type has its own path buffer. After a successful Text Report read, the driver will check the associated path buffer. If a valid path is stored there, the driver will save the report data as ASCII text in that file. The file will be created if needed. The file will be overwritten on subsequent Text Report reads.

The path buffers are initialized to empty strings upon server start up. The driver will not write Text Report data to file until a valid path is saved in the associated path buffer. Path data is not persistent. The path strings must be rewritten each time the server is restarted. The path values can be changed at any time, allowing users to save data to different files on each read if desired.

Path strings may be up to 255 characters long.

### Text Report Path Address Syntax

Address	Range	Data Type	Access
PATH_TRn	n = Report address (9001-9402)	String	Read/Write

#### Example

To read the Last Batch Report (address 9101) and save the result to disk, create two tags. The first is a Text Report tag with address "TR9101", and the second is a path tag with address "PATH\_TR9101".

In order to save the report data in a file called "LastBatch.txt" (which is to be created in the folder "C:\OmniData\BatchReports") set up the client so that the first thing that it does is write "C:\OmniData\BatchReports\LastBatch.txt" to the path tag. Once this is done, read the Text Report tag. If the path is not set before the first read of the Text Report, the driver will not be able to save the data to disk.

**Note:** To disable this feature, simply write an empty string to the path tag.

## Omni Text Archive

In addition to normal text reports, the Omni Flow Computer can also store reports in an archive. This driver can read a range of reports from the archive and send them to the OPC client as a string value.

### Reading the Text Archive

Before the text archive can be read, two settings must be made in the device: the archive start date, and the number of days to retrieve. These 32 bit integer values are at addresses 15128 and 15127 respectively. The date format may be specified using the value at address 3842 (0 = dd/mm/yy, 1 = mm/dd/yy). Shortly after the number of days is set, the device will begin preparing the data. When the data is ready to be read, the number of days value will become negative. The Text Archive can be read at any time after the number of days is set. The driver will wait for the value to become negative.

### Text Archive Address Syntax

Address	Range	Data Type	Access
---------	-------	-----------	--------

TA TA T (triggered read)	N/A	String	Read Only
-----------------------------	-----	--------	-----------

**Note:** Because it can take several minutes to read a Text Archive, the "TA" tag should be kept inactive in the OPC client. Alternatively, triggered reads can be used instead. This tag should only be read using asynchronous reads, since the maximum synchronous read timeout cannot be increased high enough in the server to read a typical text archive request. No other tags on the channel can be read or written to while the Text Archive is being read.

If a Text Archive read fails midway, users should reset the device's read buffer by writing 999 to the number of days register (15127), and then repeat the normal Text Archive read procedure. Otherwise, the driver may not get the first part of the requested archive range.

### Triggered Text Archive Reads

It is recommended that the Text Archive tag be kept inactive even though it is not always possible. A triggered read capability has been added as an alternative, thus allowing the Text Archive tag to remain active. It also controls when the actual device reads occur with an auxiliary trigger tag. The trigger value is stored in the driver's memory and may be read and set using a tag with the address syntax described below.

A triggered read may not begin immediately depending on when in the Text Archive tag's update cycle the trigger is set. After the read attempt has been completed, the driver will clear the trigger state. The Text Archive tag will show the value and data quality that resulted from the last triggered read attempt.

### Text Archive Read Trigger Address Syntax

Address	Range	Data Type	Access
TRIG_TA	N/A	Boolean	Read/Write

### Example

To read the Text Archive on trigger, create two tags. The first is a Text Archive tag with address "TA T", and the second is a Text Archive Read Trigger tag with address "TRIG\_TA". Users will also have to create start date and number of days tags.

**Note:** The Text Archive tag address looks like a normal Text Archive address followed by a space and the letter "T" for "triggered read". This "T" must be present in the address for triggered reads to work.

To trigger a read, set the trigger tag value to true (non-zero). After the read attempt has been completed, the driver will set the trigger value to false (0). If the read was successful, the Text Archive tag's data quality will be Good. If the read failed, the Text Archive tag's data quality will be Bad and the value will be the last value successfully read.

### Saving Text Archive Data To Disk

The driver has the ability to save Text Archive data to disk. This feature is enabled using a Text Archive Path tag. This tag is used to write a file path string to the driver's memory. After a successful Text Archive read, the driver will check the associated path buffer. If a valid path is stored there, the driver will save the Text Archive data as ASCII text in that file. The file will be created if needed. The file will be overwritten on subsequent Text Archive reads.

The path buffer is initialized to an empty string upon server start up. The driver will not write Text Archive data to file until a valid path is saved in the associated path buffer. Path data is not persistent. Users will have to rewrite the path string each time the server is restarted. The path value can be changed at any time, allowing the data to be saved to different files on each read (if desired).

The path string may be up to 255 characters long.

### Text Archive Path Address Syntax

Address	Range	Data Type	Access
PATH_TA	N/A	String	Read/Write

### Example

To read the Text Archive and save the result to disk, create two tags. The first is a Text Archive tag with address "TA", and the second is a path tag with address "PATH\_TA". Users will also have to create start date and number of days tags as described above.

In order to save the Text Archive data in a file called "TextArchive.txt" (which is to be created in the folder "C:\OmniData\ArchiveData") set up the client so that the first thing that it does is write "C:

\\OmniData\ArchiveData\TextArchive.txt" to the path tag. Once this is done, read the Text Archive tag. If the path is not set before the first read of the Text Archive, the driver will not be able to save the data to disk.

**Note:** To disable this feature, simply write an empty string to the path tag.

## **Error Descriptions**

---

The following error/warning messages may be generated. Click on the link for a description of the message. The errors are listed in alphabetical order.

[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>'](#)

[Bad address in block \[ <start address> to <end address> \] on device '<device name>'](#)

[Bad array spanning \[ <address> to <address> \] on device '<device>'](#)

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

[COMn does not exist](#)

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

[Could not read Omni text buffer due to memory allocation problem](#)

[Could not read Omni text report '<address>' on device '<device name>' due to packet number limit](#)

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

[Description truncated for import file record number <record>](#)

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

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

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

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

[Error opening COMn](#)

[Error parsing import file record number <record>, field <field>](#)

[Error writing Omni text data to file for '<tag name>' on device '<device name>' because <reason>](#)

[File exception encountered during tag import](#)

[Imported tag name '<tag name>' is invalid. Name changed to '<tag name>'](#)

[Missing address](#)

[No Omni text archive data available in specified date range on device '<device name>'](#)

[Omni text output file specified for '<tag name>' on device '<device name>' could not be opened because <reason>](#)

[Received block length of '<received length>' does not match expected length of '<expected length>' for address '<address>' on device '<device>'](#)

[Tag '<tag name>' could not be imported because data type '<data type>' is not supported](#)

[Tag import failed due to low memory resources](#)

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

[Unable to write to address '<address>' on device '<device>': Device responded with exception code '<code>'](#)

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

[Write failed for '<tag name>' on device '<device name>'. Maximum path length of '<number>' characters exceeded](#)

[Write to Omni text report '<address>' on device '<device name>' truncated](#)

**See Also:**

[Modbus Exception Codes](#)

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

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

---

**Error Type:**

Warning

**Possible Cause:**

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

**Solution:**

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

---

**Bad address in block [<start address> to <end address>] on device '<device name>'**

---

**Error Type:**

Serious

**Possible Cause:**

1. An attempt has been made to reference a nonexistent location in the specified device.
2. An attempt has been made to read more registers than allowed by the protocol.

**Solution:**

1. Verify the tags assigned to addresses in the specified range on the device and eliminate ones that reference invalid locations.
2. Decrease the Register block size value to 125.

**See Also:**

[Error Handling](#)

[Block Sizes](#)

---

**Bad array spanning [<address> to <address>] on device '<device>'**

---

**Error Type:**

Serious

**Possible Cause:**

1. An attempt has been made to reference a nonexistent location in the specified device.
2. An attempt has been made to read more registers than allowed by the protocol.

**Solution:**

1. Verify that all the register addresses requested in the array exist in the device and reduce the array size such that only valid addresses (that exist in the device) are requested by the array.
2. Reduce the array size value to 125.

**See Also:**[Error Handling](#)[Block Sizes](#)

---

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

---

**Error Type:**

Serious

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

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

**Solution:**

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

---

**COMn does not exist**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

Verify that the proper COM port has been selected.

---

**COMn is in use by another application**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

1. Verify that the correct port has been assigned to the channel.
2. Verify that only one copy of the current project is running.

---

**Could not read Omni text buffer due to memory allocation problem**

---

**Error Type:**

Serious

**Possible Cause:**

The driver could not allocate memory required for an Omni Text Record or Text Archive read operation.

**Solution:**

Shutdown all unnecessary applications and then retry.

**Could not read Omni text report '<address>' on device '<device name>' due to packet limit**

---

**Error Type:**

Serious

**Possible Cause:**

Text reports are expected to be 8192 bytes or less. This is a limit imposed by the protocol. The driver read 8192 bytes before encountering the expected end of file character.

**Solution:**

Verify that the report template used by the device will generate reports of 8192 bytes or less.

**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.

**Description truncated for import file record number <record>**

---

**Error Type:**

Warning

**Possible Cause:**

The tag description given in specified record is too long.

**Solution:**

The driver will truncate the description as needed. To prevent error in the future, edit the variable import file to change the description if possible.

**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.

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

---

**Error Type:**

Serious

**Possible Cause:**

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

**Solution:**

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

**Error opening COMn**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

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

**Error parsing import file record number <record>, field <field>**

---

**Error Type:**

Serious

**Possible Cause:**

The specified field in the variable import file could not be parsed because it is longer than expected or invalid.

**Solution:**

Edit the variable import file to change the offending field if possible.

---

**Error writing Omni text data to file for '<tag name>' on device '<device name>' because <reason>**

---

**Error Type:**

Warning

**Possible Cause:**

The driver could not write the Omni text data to disk for the indicated reason.

**Solution:**

The operating system supplies the reason indicated. The reason should determine appropriate corrective measures.

---

**File exception encountered during tag import**

---

**Error Type:**

Serious

**Possible Cause:**

The variable import file could not be read.

**Solution:**

Regenerate the variable import file.

---

**Imported tag name '<tag name>' is invalid. Name changed to '<tag name>'**

---

**Error Type:**

Warning

**Possible Cause:**

The tag name encountered in the variable import file contained invalid characters.

**Solution:**

The driver will construct a valid name based on the one from the variable import file. To prevent this error in the future, and to maintain name consistency, change the name of the exported variable (if possible).

---

**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.

---

**No Omni text archive data available in specified date range on device '<device name>'**

---

**Error Type:**

Warning

**Possible Cause:**

No data was stored in the text archive for the date range specified by the Start Date register (15128) and the Number of Days register (15127).

**Solution:**

This is not necessarily an error.

**Omni text output file specified for '<tag name>' on device '<device name>' could not be opened because <reason>**

---

**Error Type:**

Warning

**Possible Cause:**

The file specified in an Omni Text Path tag could not be created or opened.

**Solution:**

The operating system supplies the reason indicated. The reason should determine appropriate corrective measures. The most likely cause is an invalid path.

**See Also:**

[Omni Text Reports](#)

[Omni Text Archive](#)

**Received block length of '<received length>' does not match expected length of '<expected length>' for address '<address>' on device '<device>'**

---

**Error Type:**

Warning

**Possible Cause:**

The driver attempted to read a block of memory but the PLC did not provide the driver with the requested size of data. No error code was returned.

**Solution:**

N/A

**Tag import failed due to low memory resources**

---

**Error Type:**

Serious

**Possible Cause:**

The driver could not allocate memory required to process variable import file.

**Solution:**

Shut down all unnecessary applications and then retry.

**Tag '<tag name>' could not be imported because data type '<data type>' is not supported**

---

**Error Type:**

Warning

**Possible Cause:**

The data type specified in the variable import file is not one of the types supported by this driver.

**Solution:**

If possible, change the data type specified in variable import file to one of the supported types. If the variable is for a structure, manually edit file to define each tag required for the structure, or manually configure the required tags in the OPC Server.

**See Also:**

[Exporting Variables from Concept](#)

---

**Unable to set comm parameters on COMn**

---

**Error Type:**

Fatal

**Possible Cause:**

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

**Solution:**

Verify the serial parameters and make any necessary changes.

---

**Unable to write to address '<address>' on device '<device>': Device responded with exception code '<code>'**

---

**Error Type:**

Warning

**Possible Cause:**

See [Modbus Exception Codes](#) for a description of the exception code.

**Solution:**

See [Modbus Exception Codes](#).

---

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

---

**Error Type:**

Serious

**Possible Cause:**

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

**Solution:**

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

---

**Write failed for '<tag name>' on device '<device name>'. Maximum path length of '<number>' exceeded**

---

**Error Type:**

Warning

**Possible Cause:**

Path length is limited to the indicated number of characters.

**Solution:**

Devise a shorter path.

---

**Write to Omni text report '<address>' on device '<device name>' truncated**

---

**Error Type:**

Serious

**Possible Cause:**

An attempt was made to write more than 8192 bytes to a text report. This is a limit imposed by the protocol.

**Solution:**

Do not write strings greater than the 8192 byte limit. If the string is longer, only the first 8192 characters will be written to the device.

**Modbus Exception Codes**

The following data is from Modbus Application Protocol Specifications documentation.

Code Dec/Hex	Name	Meaning
01/0x01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the server (or slave). This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server (or slave) is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
02/0x02	ILLEGAL DATA ADDRESS	The data address received in the query is not an allowable address for the server (or slave). More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed, a request with offset 96 and length 5 will generate exception 02.
03/0x03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for server (or slave). This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.
04/0x04	SLAVE DEVICE FAILURE	An unrecoverable error occurred while the server (or slave) was attempting to perform the requested action.
05/0x05	ACKNOWLEDG E	The slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the master. The master can next issue a Poll Program Complete message to determine if processing is completed.
06/0x06	SLAVE DEVICE BUSY	The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free.
07/0x07	NEGATIVE ACKNOWLEDG E	The slave cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 decimal. The master should request diagnostic or error information from the slave.
08/0x08	MEMORY PARITY ERROR	The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device.
10/0x0A	GATEWAY PATH UNAVAILABLE	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. This usually means that the gateway is misconfigured or overloaded.
11/0x0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

**Note:** For this driver, the terms Slave and Unsolicited are used interchangeably.

# Index

## - A -

Address Descriptions 17  
Address Descriptions Elliot 20, 21  
Address Descriptions Magnetek 20  
Address Descriptions Modbus 17  
Array size is out of range for address '<address>' 36  
Array support is not available for the specified address: '<address>' 36  
Automatic Tag Database Generation 12

## - B -

Bad address 36  
Bad array spanning [<address> to <address>] on device '<device>' 36  
BCD 16  
Block Size 3  
Block Sizes 7  
Boolean 16

## - C -

Cable Diagram 4  
Channels  
    maximum 3  
Coils 17  
Communications error on COMn [<error mask>] 37  
COMn does not exist 37  
COMn is in use by another application 37  
Could not read Omni text buffer due to memory allocation problem 37  
Could not read Omni text report '<address>' on device '<device name>' due to packet 38

## - D -

Data Type '<type>' is not valid for device address '<address>' 38  
Data Types Description 16  
Description truncated for import file record number <record> 38

Device '<device name>' is not responding 39  
Device address '<address>' contains a syntax error 38  
Device address '<address>' is not supported by model '<model name>' 38  
Device address '<address>' is Read Only 39  
Device ID 3  
Devices  
    maximum 3  
DWord 16

## - E -

Error Descriptions 35  
Error Handling 11  
Error opening COMn. 39  
Error parsing import file record number <record> field <field> 39  
Error writing Omni text data to file for '<tag name>' on device '<device name>' because 40  
Exporting Variables from Concept 13  
Exporting Variables from ProWORX 14

## - F -

File exception encountered during tag import 40  
Float 16  
Framing 9, 37

## - I -

Imported tag name <tag name> is invalid. Name changed to <tag name> 40

## - L -

LBCD 16  
Long 16

## - M -

mask 37  
Missing address 40  
Modbus Exception Codes 43  
Modem Setup 4

**- N -**

Network 3

No Omni text archive data available in specified date range on device '<device name>' 40

**- O -**

Omni Custom Packets 26

Omni Flow Computer Addressing 23

Omni Raw Data Archive 27

Omni Text Archive 33

Omni text output file specified for '<tag name>' on device '<device name>' could 41

Omni Text Reports 32

overrun 37

Overview 3

**- P -**

parity 37

**- R -**

Received block length of '<received length>' does not match expected length of '<expected length>' for address '<address>' 41

Registers 17

**- S -**

Settings 4

Short 16

**- T -**

Tag <tag name> could not be imported because data type <data type> is not supported 41

Tag import failed due to low memory resources 41

**- U -**

Unable to set comm parameters on COMn 42

Unable to write to '<address>' on device '<device name>' 42

Unable to write to address '<address>' on device '<device>': Device responded with exception code '<code>' 42

**- V -**

Variable Import Settings 8

**- W -**

Word 16

Write failed for '<tag name>' on device '<device name>'. Maximum path length of '<number>' 42

Write to Omni text report '<address>' on device '<device name>' truncated 42