Modbus ASCII Driver

Table of Contents

Modbus ASCII Driver

Help version 1.035

CONTENTS

Overview

The Modbus ASCII Driver provides a reliable way to connect Modbus ASCII serial devices to OPC Client applications, including HMI, SCADA, Historian, MES, ERP, and countless custom applications. It is intended for use with serial devices that support the Modbus ASCII protocol. The driver's special features provide control over 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 32-bit and 16-bit register values and address base adjustment. The driver can also control the RTS line operation for use with radio modems that require specific RTS timing.

Channel Setup

Communication Serialization

The Modbus ASCII Driver supports communication serialization, which specifies whether data transmissions should be limited to one channel at a time. *For more information, refer to "Channel Properties - Advanced" in the server help file.*

Device Setup

Supported Devices
Modbus ASCII compatible devices
Flow Computers using the Daniels/Omni/Elliot register addressing

Communication Protocol
Modbus ASCII Protocol

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

Note: Some devices may not support the listed configurations.

Maximum Number of Channels and Devices
The maximum number of channels supported by this driver is 100. The maximum number of supported devices is 247.

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. It may be invoked through the COM ID dialog in Channel Properties. For more information, refer to the server help file.

Device ID (PLC Network Address)
Modbus Serial devices are assigned Device IDs in the range of 1 to 247.

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

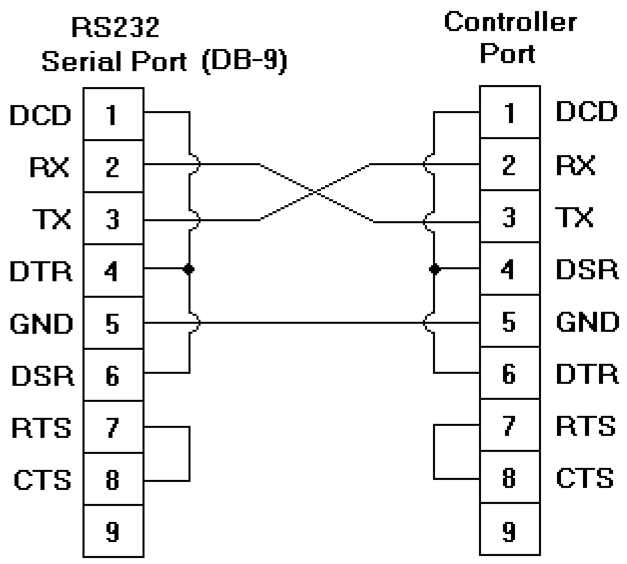
Notes:

1. 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.

2. The Modbus ASCII 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 server help file.

Cable Diagram

Cable Connections (Modbus Controller)



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, users can specify it when defining the device's parameters. 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, users can uncheck the **Use zero based addressing** check box in Device Properties. For the appropriate application that can be used to obtain information on setting device properties, refer to the online help documentation. The default behavior follows the convention of the Modicon PLCs.

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 addressing within registers 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. The default setting is unchecked. If this setting is selected, then the driver will use function code 0x16 regardless of the setting for "Use Modbus function 06 for single register writes." 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 not selected, 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 in most cases 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 third 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.

Use Modbus function 06 can be used to force the driver to use only Modbus function 16 if needed. This selection is checked by default. It allows the driver to switch between 06 and 16 as needed. If the device requires all writes to 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 on 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.

Use Modbus function 05 can be used to force the driver to use only Modbus function 15 if needed. This selection is checked by default. It allows the driver to switch between 05 and 15 as needed. If a device requires all writes to 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 election 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.

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.

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. The default, first DWord low, follows the default convention of 32-bit data types.

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.

Note: 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) High Word(63..48) of Double Word in 64-bit data types	Low Word (15..0) 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) Low Word (47..32) of Double Word in 64-bit data types	High Word (31..16) 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. Block size can be reduced if data needs to be read from non-contiguous locations within the device.

Register Block Sizes

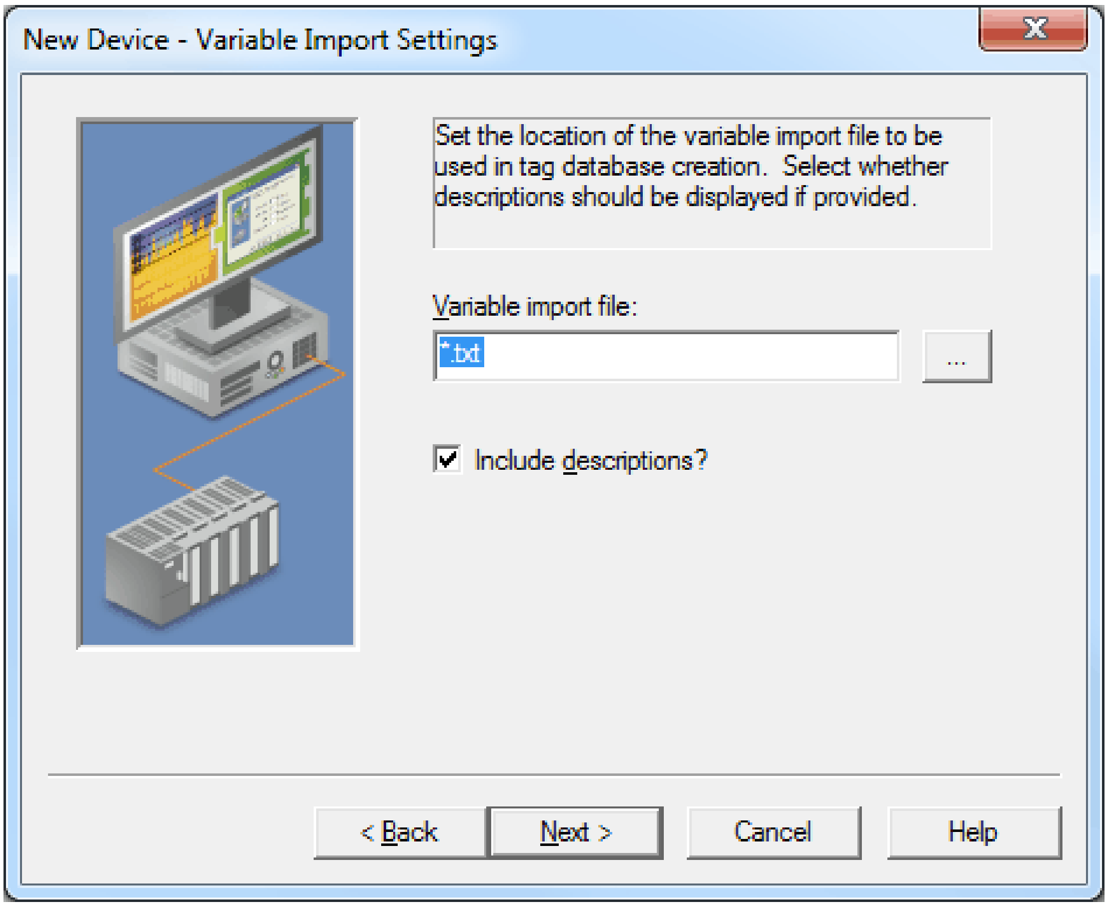
Registers can be read from 1 to 100 locations (words) at a time. A higher block size means more register values will be read from the device in a single request. Block size can be reduced if data needs to be read from non-contiguous locations within the device.

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 the error from occurring, decrease the block size value to 120.

Perform Block Read on Strings

Check this option to block read string tags, which are normally read individually. When this option is selected, string tags will be grouped together depending on the selected block size. Block reads can only be performed for Modbus model string tags.

Variable Import Settings



Set the location of the variable import file to be used in tag database creation. Select whether descriptions should be displayed if provided.

Variable import file: *.txt

☑ Include descriptions?

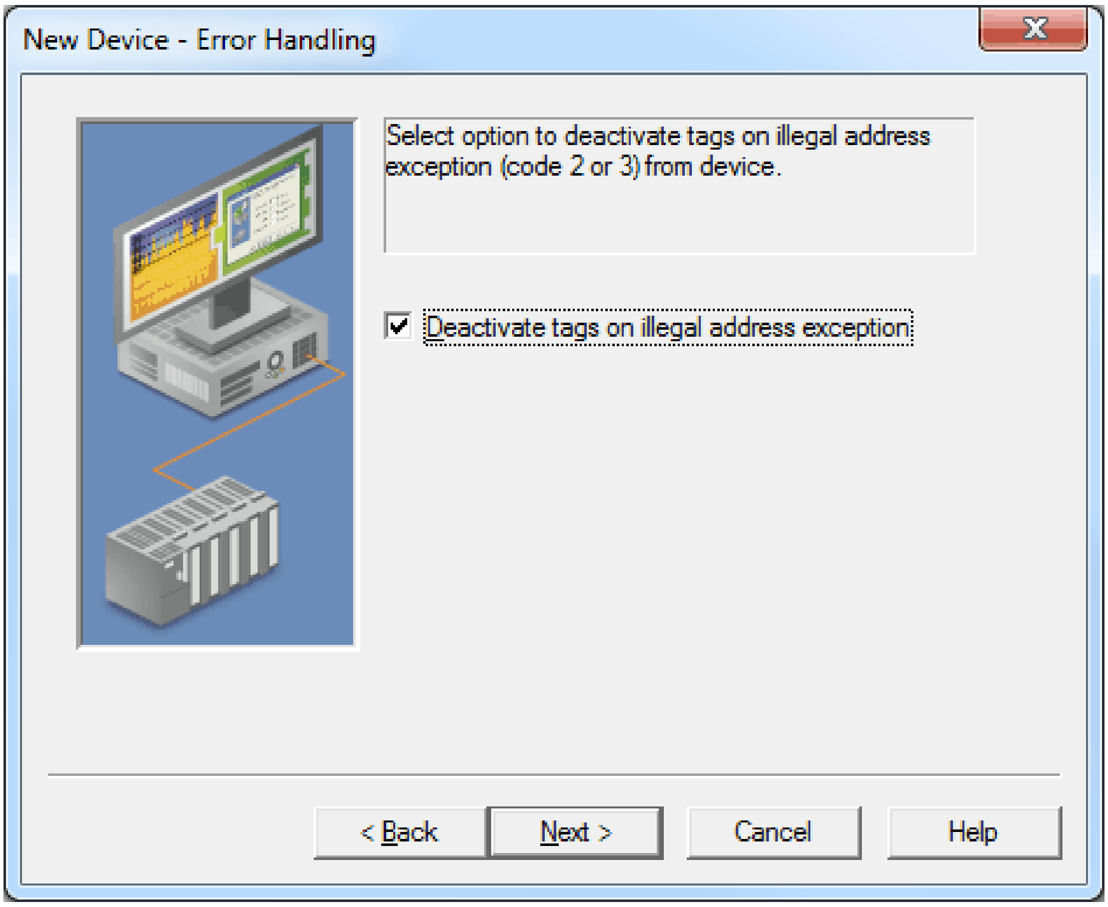
Descriptions of the parameters are as follows:

- **Variable import file:** This parameter specifies the exact location of the delimited text file the driver can use when Automatic Tag Database Generation is enabled. Variable import files can be created from Concept or ProWORX applications.
- **Include descriptions:** When checked, imported tag descriptions will be used if present in file. The default setting is checked.

Note: For more information on configuring the Automatic Tag Database Generation feature and creating a variable import file, refer to **Automatic Tag Database Generation**.

See Also:
Exporting Variables from Concept
Exporting Variables from ProWORX

Error Handling



Select option to deactivate tags on illegal address exception (code 2 or 3) from device.

[✓] Deactivate tags on illegal address exception

Description of the parameter is as follows:

- **Deactivate tags on illegal address exception:** When checked, the driver will stop polling for a block of data when 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. When unchecked, the driver will continue to poll that data block. Users will not need to restart the server to activate a deactivated block. The default setting is checked.

Automatic Tag Database Generation

The Modbus ASCII Driver makes use of the Automatic Tag Database Generation feature. This 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 .TXT format, which 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 the application's needs. The primary control options can be set during the Database Creation step of the Device Wizard or later by selecting the **Device Properties** | **Database Creation**. For more information, refer to the OPC Server's help documentation.

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

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 Hi-Lo Lo-Hi byte order selection.
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 above assume the default settings; that is, 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 ASCII Addressing
Flow Computer Addressing
Flow Automation Addressing

Modbus ASCII Addressing

5-Digit Addressing vs. 6-Digit Addressing

In Modbus addressing, the first digit of the address specifies the primary table. The remaining digits represent the device's data item. The maximum value is a two byte unsigned integer (65,535). Six digits are required to represent the entire address table and item. As such, addresses that are specified in the device's manual as 0xxxx, 1xxxx, 3xxxx, or 4xxxx will be padded with an extra zero once applied to the Address field of a Modbus tag.

Primary Table	Description
0	Output Coils
1	Input Coils
3	Internal Registers
4	Holding Registers

Modbus ASCII Addressing

The default data types for dynamically defined tags are shown in **bold**. For notes and restrictions, refer to **Packed Coil Tags**, **String Support**, and **Array Support.**

Address	Range	Data Type	Access	Function Codes*
Output Coils	000001-065536 000001#1-065521#16	**Boolean** Word (Packed Coil Tag)	Read/Write	01, 05, 15**
Input Coils	100001-165536 100001#1-165521#16	**Boolean** Word (Packed Coil Tag)	Read Only	02**
Internal Registers	300001-365536 300001-365535 300001-365533 3xxxxx.0/1-3xxxxx.15/16***	**Word**, Short, BCD Float, DWord, Long, LBCD Double Boolean	Read Only	04
Internal Registers As String with HiLo Byte Order	300001.2H-365536.240H .Bit is string length, range 2 to 240 bytes.	**String**	Read Only	04
Internal Registers As String with LoHi Byte Order	300001.2L-365536.240L .Bit is string length, range 2 to 240 bytes.	**String**	Read Only	04
Holding Registers	400001-465536 400001-465535 400001-465533 4xxxxx.0/1-4xxxxx.15/16***	**Word**, Short, BCD Float, DWord, Long, LBCD Double Boolean	Read/Write	03, 06, 16 03, 06, 16, 22
Holding Registers As String with HiLo Byte Order	400001.240H-465536.2H .Bit is string length, range 2 to 240 bytes.	**String**	Read/Write	03, 16
Holding Registers As	400001.2L-465536.240L	**String**	Read/Write	03, 16

Address	Range	Data Type	Access	Function Codes*
String with LoHi Byte Order	.Bit is string length, range 2 to 240 bytes.			

*The supported Function Codes are displayed in decimal. For more information, refer to **Function Codes Description**.
For more information, refer to **Packed Coil Tags.
***For more information, refer to the "Use Zero-Based Bit Addressing Within Registers" subtopic in **Settings**.

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, then 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.

Packed Coil Tags

The Packed Coil address type allows access to multiple consecutive coils as an analog value. This feature is available for the Modbus ASCII model only. The only valid data type is Word. The syntax is as follows.

Output coils: 0xxxxx#nn Word Read/Write
Input coils: 1xxxxx#nn Word Read Only

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

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

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.

Note: For information on how to perform block read on string tags for the Modbus model, refer to **Block Sizes**.

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

Normal Address Examples

1. The 255th output coil would be addressed as '0255' using decimal addressing.
2. Some documentation refers to Modbus addresses by function code and location. For instance, function code 3; location 2000 would be addressed as '42000' (the leading '4' represents holding registers or function code 3).
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' (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.

Function Codes Description

Decimal	Hexadecimal	Description
01	0x01	Read Coil Status
02	0x02	Read Input Status
03	0x03	Read Holding Registers
04	0x04	Read Internal Registers
05	0x05	Force Single Coil
06	0x06	Preset Single Register
15	0x0F	Force Multiple Coils
16	0x10	Preset Multiple Registers
22	0x16	Masked Write Register

Flow Computer Addressing
The default data types for dynamically defined tags are shown in **bold**.

Address	Range	Data Type	Access
Output Coils	000001-065536	Boolean	Read/Write
Input Coils	100001-165536	Boolean	Read Only
Internal Registers	300001-365536 300001-365535	**Word**, Short, BCD Float, DWord, Long, LBCD	Read Only
Holding Registers	400001-465536 400001-465535	**Word**, Short, BCD*, Float, DWord, Long, LBCD	Read/Write
Flow Computer Registers	405000-406800 407000-407800	**Long**, DWord, LBCD **Float**, Long, DWord	Read/Write

*****Address ranges 405000 to 406800 and 407000 to 407800 are 32-bit registers. Addresses in the range of 405000 to 406800 use a default data type of Long. Addresses in the range of 407000 to 407800 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.

Arrays
Arrays are supported for internal and holding register locations for all data types except for 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.

Flow Automation Addressing
The default data types for dynamically defined tags are shown in **bold**.

Address	Range	Data Type	Access
Flow Computer Registers	40001-465535	**Float**	Read/Write

The Flow Automation Flow Computer treats all data as a 32-bit floating point value. All addresses in the holding register space of the device will be read as 32-bit floating point numbers. A complete memory map of the flow automation control is provided in the custom report section of the flow automation manual.

Error Descriptions

The following categories of messages may be generated. Click on the link for a list of related messages.

Address Validation
Serial Communications
Device Status Messages
Device-Specific Messages
Automatic Tag Database Generation Messages
Modbus Exception Codes

Address Validation

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

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

Missing address.

Error Type:
Warning

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

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

Device address <address> contains a syntax error.

Error Type:
Warning

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

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

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

Error Type:
Warning

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

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

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

Error Type:
Warning

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

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

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

Error Type:
Warning

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

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

Device address <address> is read only.

Error Type:
Warning

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

Solution:
Change the access mode in the client application.

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

Error Type:
Warning

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

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

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

Error Type:
Warning

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

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

Serial Communications

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

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

COMn does not exist.

Error Type:
Fatal

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

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

Error opening COMn.

Error Type:
Fatal

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

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

COMn is in use by another application.

Error Type:
Fatal

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

Solution:

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

Unable to set comm parameters on COMn.

Error Type:
Fatal

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

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

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

Error Type:
Serious

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.

See Also:
[Error Mask Definitions]()

Device Status Messages

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

Device <device name> is not responding.
Unable to write to <address> on device <device name>.
Unable to write to address <array address> on device <device>: Device responded with exception code.

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 Device ID.
4. The response from the device took longer to receive than the amount of time specified in the "Request Timeout" device setting.

Solution:

1. Verify the cabling between the PC and the PLC device.
2. Verify that the specified communications parameters match those of the device.
3. Verify that the Device ID given to the named device matches that of the actual device.
4. Increase the Request Timeout setting so that the entire response can be handled.

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

Error Type:
Serious

Possible Cause:

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

Solution:

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

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**.

Device Specific Messages

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

Bad address in block [<start address> to <end address>] on device <device name>.
Bad array spanning [<address> to <address>] on device <device name>.

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

Error Type:
Serious

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

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

Bad array spanning [<address> to <address>] on device <device name>.

Error Type:
Fatal

Possible Cause:
An array of addresses was defined that spans past the end of the address space.

Solution:
Verify the size of the device's memory space and then redefine the array length accordingly.

Automatic Tag Database Generation Messages

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

Tag import failed due to low memory resources.
File exception encountered during tag import.
Error parsing import file record number <record>, field <field>.
Description truncated for import file record number <record>.
Imported tag name <tag name> is invalid. Name changed to <tag name>.
Tag <tag name> could not be imported because data type <data type> is not supported.

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:
Shutdown all unnecessary applications and retry.

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.

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.

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 this error in the future, edit the variable import file to change the description if possible.

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.

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

Modbus Exception Codes

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	ACKNOWLEDGE	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 ACKNOWLEDGE	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.

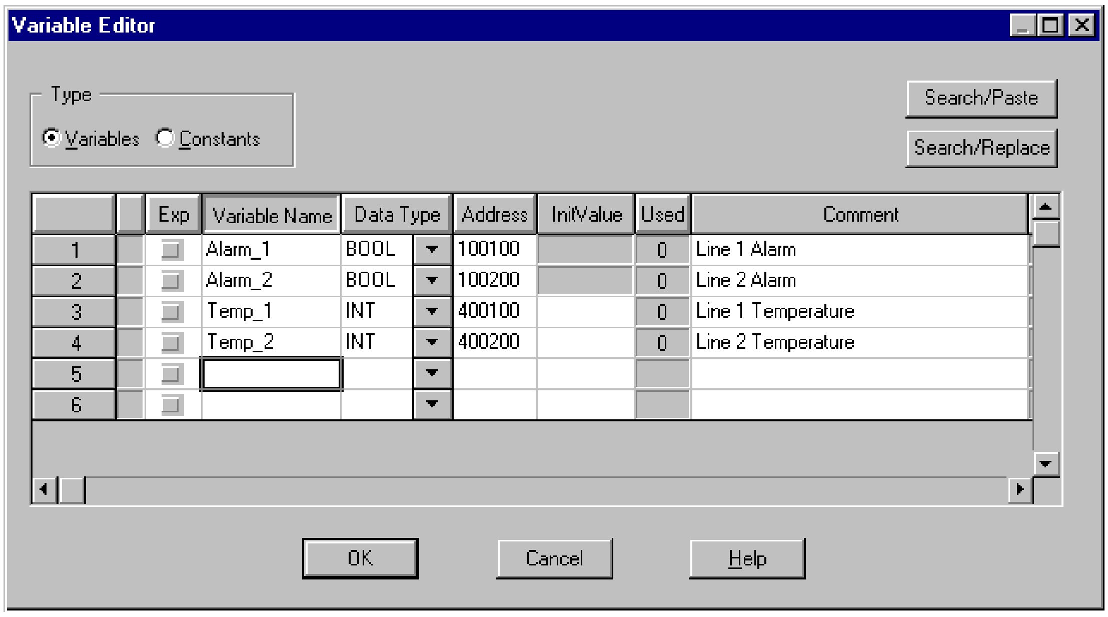
Note: For this driver, the terms Slave and Unsolicited are used interchangeably.

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

Appendix — Exporting Variables from Concept

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



	Exp	Variable Name	Data Type	Address	InitValue	Used	Comment
1		Alarm_1	BOOL	100100		0	Line 1 Alarm
2		Alarm_2	BOOL	100200		0	Line 2 Alarm
3		Temp_1	INT	400100		0	Line 1 Temperature
4		Temp_2	INT	400200		0	Line 2 Temperature
5							
6							

Note: Though Concept can be 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 will inform the user 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 following simple data types are supported:

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

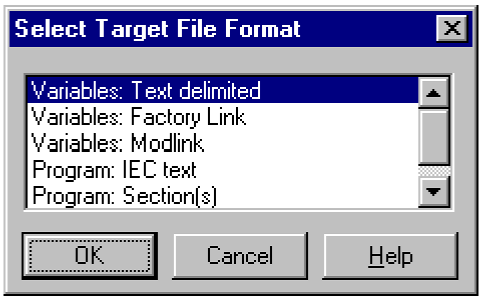
Notes:

1. Unlocated variables, which do not correspond to a physical address in the device, will be ignored by the driver.
2. Comments are allowed. Users can choose whether or not to include these as the generated tag descriptions. For more information, refer to **Variable Import Settings**.

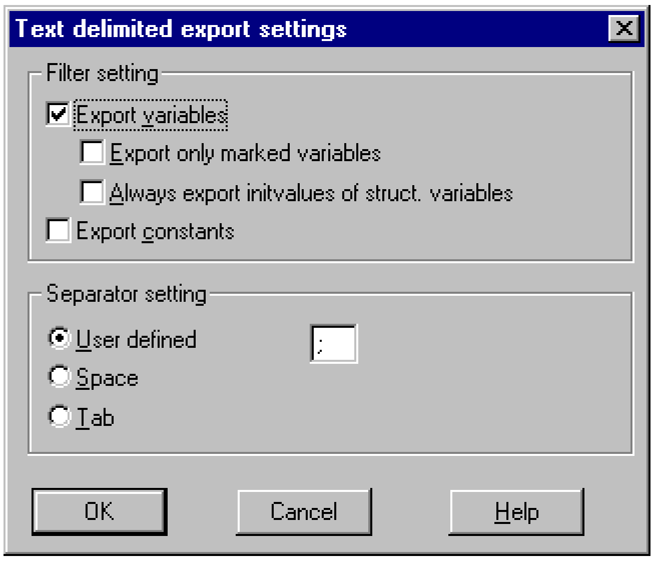
Exporting Data from Concept

Once the variables have been defined, the data must be exported from Concept. To do so, follow the instructions below.

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



2. Click **OK**. Next, specify the **Filter Setting** and **Separator Setting**.


Text delimited export settings
Filter setting
Export variables
Export only marked variables
Always export initvalues of struct. variables
Export constants
Separator setting
User defined
Space
Tab
OK
Cancel
Help

Filter setting: [✓] Export variables; [ ] Export only marked variables; [ ] Always export initvalues of struct. variables; [ ] Export constants

Separator setting: [✓ User defined] ; Space; Tab

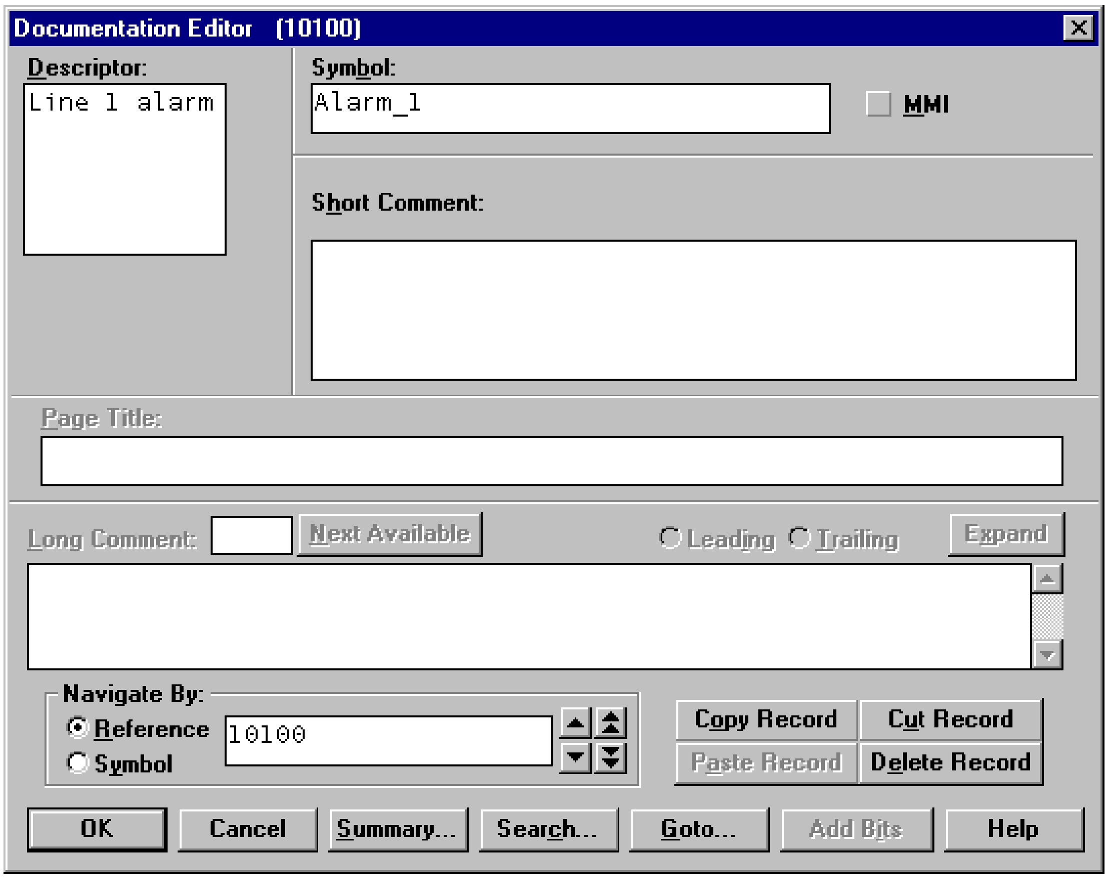
Note: Although any filter settings may be chosen, this driver will only be able to read the exported data if the default semicolon separator is used.

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

Appendix — Exporting Variables from ProWORX

For ProWORX to export the necessary variable information, make sure that the **Symbols** option is checked under **File** | **Preferences**.

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


Documentation Editor (10100)

Descriptor: Line 1 alarm

Symbol: Alarm_1

[ ] MMI

Short Comment:

Page Title:

Long Comment:

Navigate By: [✓ Reference] Symbol 10100

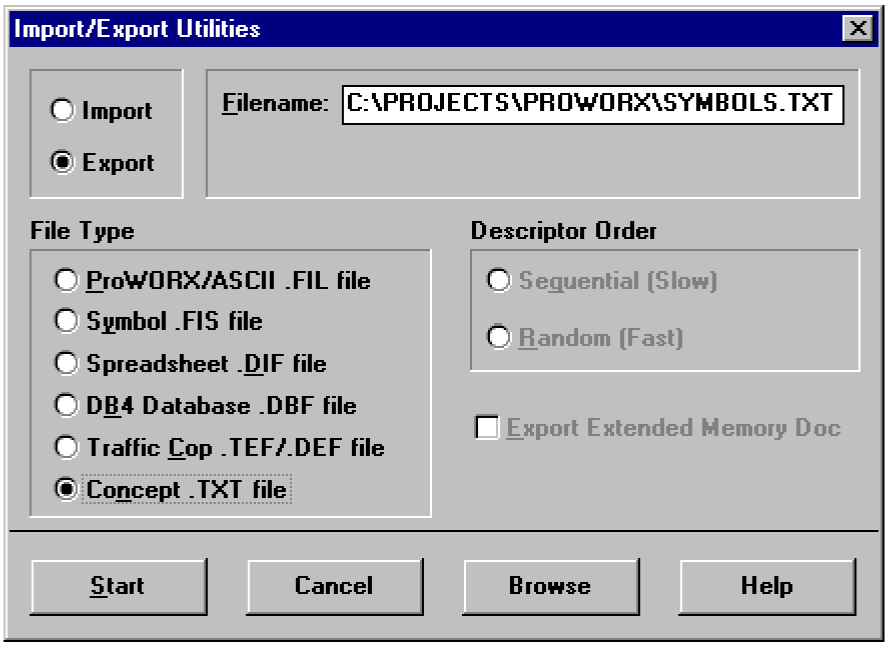
Notes:

1. Although ProWORX does not place many restrictions on variable names, the OPC Server requires that tag names consist of 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 of any such name changes in the server's event log.
2. 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. 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 Datas from ProWORX

Once the variables have been defined, the data must be exported from ProWORX. To do so, follow the instructions below.

1. Click **File** | **Utilities** | **Import/Export**.
2. Select the **Export** and the **Concept .TXT file** format.
3. **Note:** Descriptors are allowed and can be included as the generated tag descriptions or not. For more information, refer to **Variable Import Settings**.


Import/Export Utilities

( ) Import [● Export]

Filename: C:\PROJECTS\PROWORX\SYMBOLS.TXT

File Type
- ( ) ProWORX/ASCII .FIL file
- ( ) Symbol .FIS file
- ( ) Spreadsheet .DIF file
- ( ) DB4 Database .DBF file
- ( ) Traffic Cop .TEF/.DEF file
- [● Concept .TXT file]

Descriptor Order
- ( ) Sequential (Slow)
- ( ) Random (Fast)

☐ Export Extended Memory Doc

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

Index