



# MBX OPC Driver Agent Help

---

*OPC Server Driver Agent for  
Modicon Networks*

---

Version 9

## **MBX OPC DRIVER AGENT HELP**

**For All Modicon networks and programmable controller families**

### **Version 9**

Copyright © 1994-2017, Cyberlogic® Technologies Inc. All rights reserved.

This document and its contents are protected by all applicable copyright, trademark and patent laws and international treaties. No part of this document may be copied, reproduced, stored in a retrieval system or transmitted by any means, electronic, mechanical, photocopying, recording or otherwise, without the express written permission of Cyberlogic Technologies Inc. This document is subject to change without notice, and does not necessarily reflect all aspects of the mentioned products or services, their performance or applications. Cyberlogic Technologies Inc. is not responsible for any errors or omissions in this presentation. Cyberlogic Technologies Inc. makes no express or implied warranties or representations with respect to the contents of this document. No copyright, trademark or patent liability or other liability for any damages is assumed by Cyberlogic Technologies Inc. with respect to the use of the information contained herein by any other party.

Cyberlogic®, DHX®, MBX®, WinConX® and Intelligent • Powerful • Reliable® are registered trademarks and DirectAccess™, OPC Crosslink™, OPC Datacenter™, DevNet™ and C-logic™ are trademarks of Cyberlogic Technologies Inc. All other trademarks and registered trademarks belong to their respective owners.

Document last revision date May 1, 2019

## TABLE OF CONTENTS

<b>Introduction .....</b>	<b>5</b>
Compatibility and Compliance.....	5
Blending MBX-Supported Networks.....	6
<b>What Should I Do Next?.....</b>	<b>8</b>
Learn How the OPC Server Works.....	8
Read a Quick-Start Guide .....	8
Get Detailed Information on the Configuration Editors .....	8
Verify That It’s Working or Troubleshoot a Problem .....	8
Print a Copy of This Document .....	8
Contact Technical Support.....	8
<b>Theory of Operation .....</b>	<b>9</b>
Main Server Features .....	10
Network Connections Tree .....	11
Network Automatic Configuration .....	11
Standard Modbus Protocol vs Modbus Extensions.....	11
Address Space Tree .....	12
Access Paths.....	13
Unsolicited Message Filters .....	13
Data Items .....	13
Bit Order.....	14
Conversions Tree.....	14
Simulation Signals Tree.....	14
Alarm Definitions Tree .....	14
<b>Quick-Start Guide.....</b>	<b>15</b>
<b>Configuration Editor Reference .....</b>	<b>16</b>
Network Connections .....	17
Creating and Deleting .....	17
Editing the MBX Driver Agent.....	23
Editing Network Connections.....	34
Editing Network Nodes .....	37
Address Space.....	49
Device Folders .....	49
Devices.....	52
Folders .....	67
Data Items .....	70
Conversions .....	80
Simulation Signals .....	81
Alarm Definitions .....	81
Database Operations .....	82
OPC Crosslinks .....	82
Saving and Undoing Configuration Changes .....	82
Configuration Import/Export.....	83
Options.....	83
<b>Validation &amp; Troubleshooting .....</b>	<b>84</b>
Data Monitor .....	84
Cyberlogic OPC Client .....	84
Performance Monitor .....	84
DirectAccess.....	84

Event Viewer .....	90
MBX Driver Agent Messages .....	90
Frequently Asked Questions .....	92
<b>Appendix A: Register Addresses .....</b>	<b>93</b>
Modicon Address Notations .....	93
Cyberlogic Address Extensions.....	95
<b>Appendix B: Data Item Duplication Wizard .....</b>	<b>97</b>
Example 1: Simple Addressing.....	97
Example 2: Addressing Bits Within Registers .....	102
<b>Appendix C: Address Wizard .....</b>	<b>108</b>
Example 1: Simple Modicon PLC Addressing .....	108
Example 2: Arrays of Registers .....	113
Example 3: OMNI Flow Controller Bit Field .....	118
<b>Appendix D: Unsolicited Message Programming .....</b>	<b>126</b>
<b>Appendix E: Configuring Maximum Concurrent Requests .....</b>	<b>128</b>
<b>Appendix F: Writing to PLC Inputs .....</b>	<b>131</b>
<b>Appendix G: MBX Architecture and Companion Products.....</b>	<b>135</b>
MBX Driver .....	135
Ethernet MBX Driver .....	136
Serial MBX Driver.....	136
MBX Gateway Driver .....	137
Mbx.Net Gateway Driver .....	137
Virtual MBX Driver .....	137
MBX Bridge .....	138
MBX OPC Server .....	138
MbpStat.Net.....	139
Mbx.Net Server.....	139
MBX SDK.....	140

## INTRODUCTION

The Cyberlogic OPC Server provides OPC Data Access, Alarms & Events and XML Data Access functions, and has a modular structure that supports a variety of industrial devices and communication networks. The various communication subsystems, which we call driver agents, are plug-ins that you can easily add as required. As a result, the server maintains a set of common features, but has the flexibility to allow additional features as required by the specific driver agent.

The MBX Driver Agent is one of these plug-in modules. It allows the Cyberlogic OPC Server to communicate to 184, 384, 484, 584, 884, 984, Micro 84, Modcell, Quantum and Momentum controllers or any Modbus-compatible device over Modbus (RTU and ASCII), Modbus Plus and Ethernet networks. In addition to the standard Modbus protocol, the MBX Driver Agent is also compatible with Modbus extensions that support 32-bit and 64-bit registers. That includes the popular Enron Modbus protocol.

The MBX Driver Agent is included in the MBX OPC Server Suite, MBX OPC Premier Suite and MBX OPC Enterprise Suite.

**Note**

This document includes only the information that is specific to the MBX Driver Agent. For information on the common features of the Cyberlogic OPC Server, refer to the [Cyberlogic OPC Server Help](#).

## Compatibility and Compliance

The MBX Driver Agent is compatible with all MBX family products. The industry-standard MBX family of drivers, which includes the MBX Driver, Ethernet MBX Driver, Serial MBX Driver, MBX Gateway Driver and Mbx.Net Gateway Driver, provides the low-level communication services. It is also compatible with the MBX Bridge, which routes messages between Ethernet, Modbus and Modbus Plus networks.

Cyberlogic OPC products provide full compliance with the OPC Foundation specifications for:

- Data Access 3.0, 2.05a and 1.0a
- Alarms & Events 1.1
- XML Data Access 1.0
- Data Access Automation 2.02

These products are tested for compliance to the OPC specifications using the latest test software from the OPC Foundation. All Cyberlogic OPC products are certified for compliance by the OPC Foundation's Independent Testing Laboratory. In addition, they are tested annually for interoperability with other OPC products at the OPC Foundation's Interoperability Workshops.

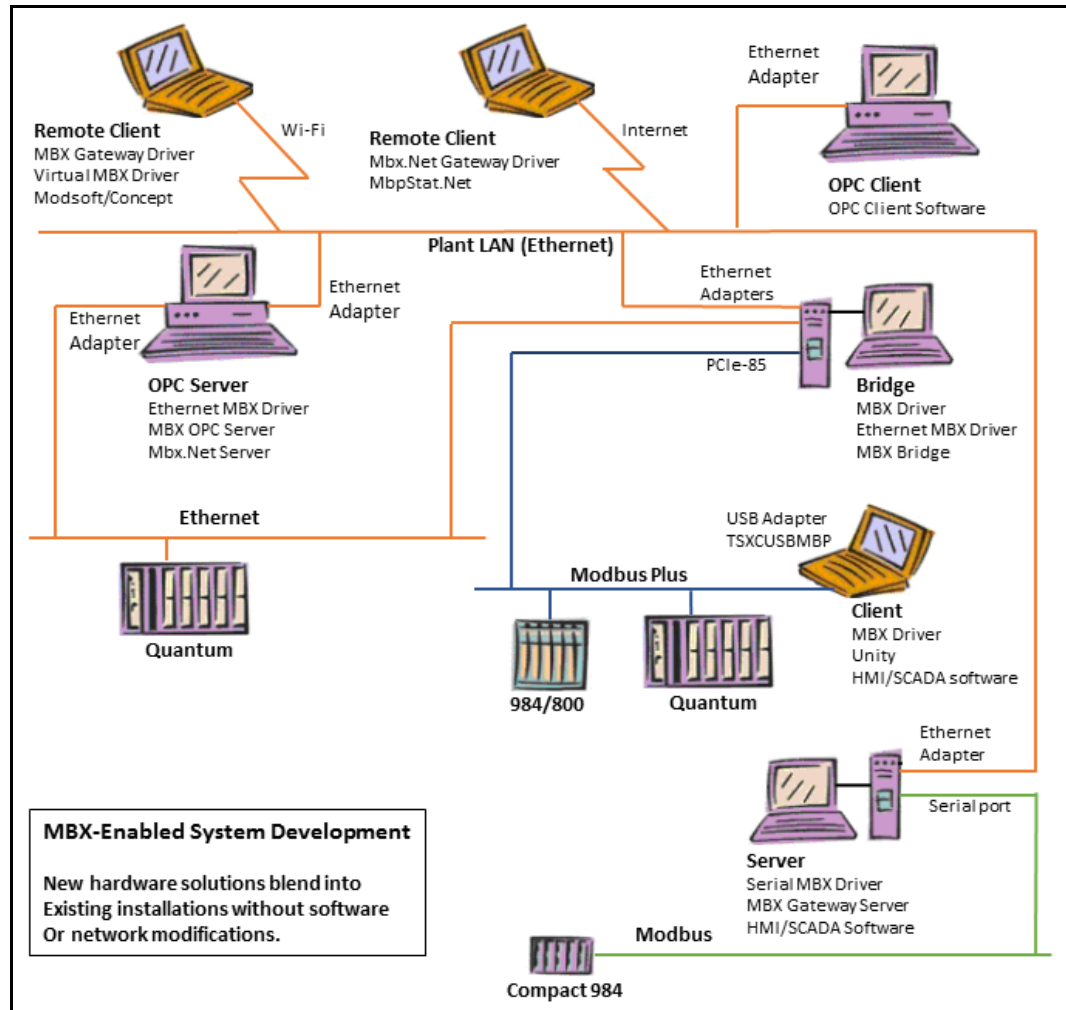
## Blending MBX-Supported Networks

The MBX driver family provides support for all Modicon networks through a common architecture, with identical programming interfaces. This means that an application that operates with one of the MBX family drivers, such as the Ethernet MBX Driver, will work with the rest of them as well. Thus, virtually all Modbus Plus compatible software programs can operate over all Modicon-supported networks with no code modifications. You will find a complete description of the MBX family in the [Appendix G: MBX Architecture and Companion Products](#).

Migration of existing installations to new hardware products does not require the user to discard working, proven software solutions. As depicted in the diagram below, a user can mix Modbus, Modbus Plus and Modbus TCP based hardware products in existing installations without losing software, network or integration investment.

The MBX family of products includes:

- [MBX Driver](#) is Cyberlogic's device driver for Modbus Plus interface adapters.
- [Ethernet MBX Driver](#) provides Modbus TCP communication.
- [Serial MBX Driver](#) provides Modbus RTU/ASCII communication.
- [MBX Gateway Driver](#) works with the other MBX drivers, giving access to Modbus, Modbus Plus and Modbus TCP networks from remote locations.
- [Mbx.Net Gateway Driver](#) works with the other MBX drivers, giving secure access to Modbus, Modbus Plus and Modbus TCP networks from remote locations, including over the Internet.
- [Virtual MBX Driver](#) works with the other MBX drivers to permit 16-bit legacy software to run in current 32-bit Windows operating systems.
- [MBX Bridge](#) allows you to bridge any combination of Modicon networks by routing messages between MBX devices.
- [MBX OPC Server](#) connects OPC-compliant client software applications to data sources over all Modicon networks.
- [MbpStat.Net](#) is the Modbus Plus network diagnostic utility. It provides the same functionality as the original DOS-based MBPSTAT application.
- [Mbx.Net Server](#) a WCF server that provides secure remote access to all MBX devices.
- [MBX SDK](#) is a software development kit for MBXAPI, MBXAPI.Net and NETLIB compliant development.



## WHAT SHOULD I DO NEXT?

The links below will take you directly to the section of this manual that contains the information you need to configure, use and troubleshoot the MBX Driver Agent.

This document describes only the features specific to the MBX Driver Agent. For information on the common features of the Cyberlogic OPC Server, refer to the [Cyberlogic OPC Server Help](#).

### Learn How the OPC Server Works

If you are not familiar with the way that the MBX Driver Agent obtains data, you should begin by reading the [Theory of Operation](#).

### Read a Quick-Start Guide

First-time users of the MBX Driver Agent will want to refer to the [Cyberlogic OPC Server Help](#) for a quick-start guide, which walks through a typical configuration session, step-by-step.

### Get Detailed Information on the Configuration Editors

Experienced users who want specific information on features of the configuration editors will find it in the [Configuration Editor Reference](#) section.

### Verify That It's Working or Troubleshoot a Problem

If you have already configured the server, you should verify that it operates as expected. Refer to the [Validation & Troubleshooting](#) section for assistance. In case of communication problems, this section also provides problem-solving hints.

### Print a Copy of This Document

The content of this document is also provided in PDF format. PDF files can be viewed using the Adobe® Reader program, and can also be used to print the entire document.

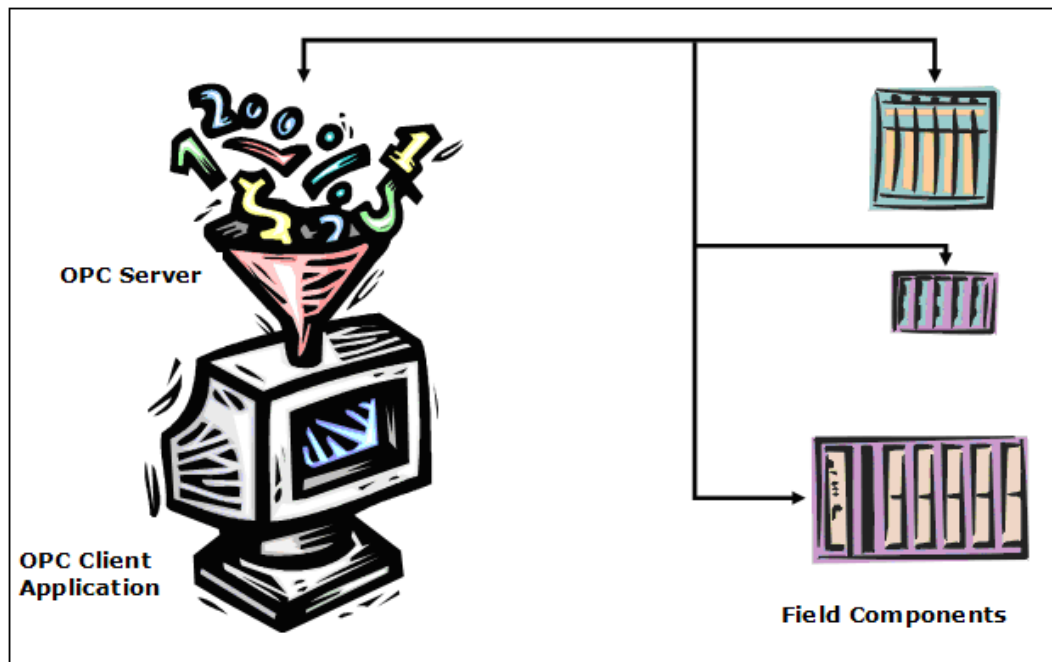
### Contact Technical Support

To obtain support information, open the Windows **Start** menu and go to **Cyberlogic Suites**, and then select **Product Information**.



## THEORY OF OPERATION

This section will familiarize you with the main features of the Cyberlogic OPC Server as they relate to the MBX Driver Agent. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of the common features of the Cyberlogic OPC Server. If you are new to OPC or the Cyberlogic OPC Server, we strongly recommend that you read the OPC Tutorial first. You will find it in the Help section of your product installation.

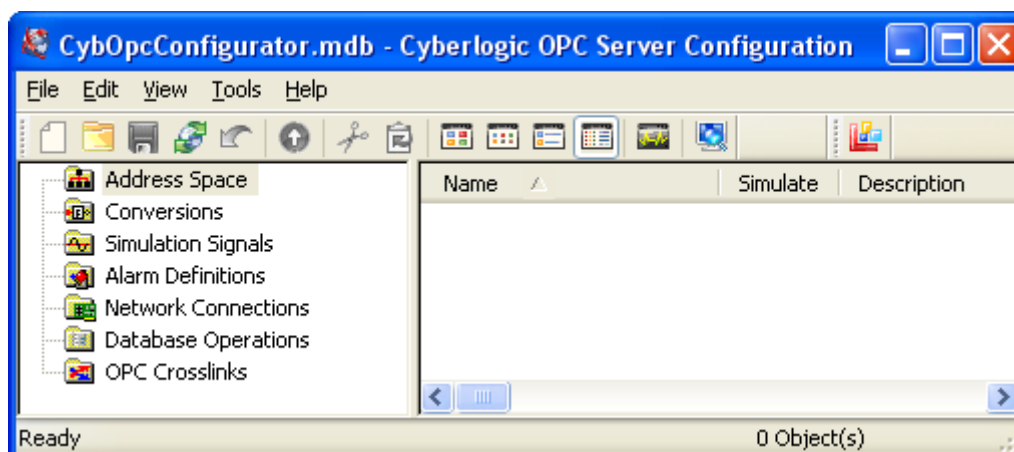


OPC servers obtain data from field components and present it, in a standard way, to OPC client applications. The basic functionality of the OPC server is the same for all types of field components and networks, but some of the communication and data-handling will vary from one family to another. The driver agent is a plug-in software module that accommodates these specific differences.

The MBX Driver Agent handles communications over all Modicon network types—Modbus, Modbus Plus and Ethernet. It also provides the means to obtain and pass data from all Modicon controllers. These include 184, 384, 484, 584, 884, 984, Micro 84, Modcell, Quantum and Momentum controllers and any other Modbus-compatible device. In addition to the standard Modbus protocol, the MBX Driver Agent is compatible with Modbus extensions that support 32-bit and 64-bit registers. That includes the popular Enron Modbus protocol.

The remainder of this theory section will discuss the [Main Server Features](#) you will find in the Cyberlogic OPC Server, as they relate to Modicon communications.

## Main Server Features



When you open the Cyberlogic OPC Server Configuration editor, you will find several main trees. These trees represent the main areas that you will configure. Note that some are for premium features that may not be part of the product you have installed, so they will not appear in your configuration. The trees are:

- The [Address Space Tree](#) is required for most configurations. Here you will create and organize the data items that will be available to the client application, and you will define how they are updated with new information.
- The Conversions Tree is optional. In it, you can define formulas that can be used to convert raw data values obtained from the field equipment into a form that is more useful to the client. For example, you can change a transducer's voltage value into a pressure value in psi. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this tree.
- The Simulation Signals Tree is optional. If you want to be able to use simulated data item values instead of real values, you can create various types of simulated data functions in this tree. Simulations are often useful for troubleshooting client applications. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this tree.
- The Alarm Definitions Tree is another optional tree. It is used when you will interface to Alarms & Events clients. This tree allows you to define the desired alarm conditions and specify what information should be passed as they occur and clear. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this tree.
- The [Network Connections Tree](#) is required for all configurations. This is where you select the networks and interface devices you will use, and configure each of the field components as nodes on those networks.
- The Database Operations Tree is part of the logging feature, which is a premium feature. If this tree is in your product, you can use it to configure databases and data logging operations. Refer to the [Data Logger Help](#) for a full discussion of this tree.
- The OPC Crosslinks Tree is part of OPC Crosslink, which is a premium feature. If this tree is in your product, you can use it to configure data transfers between PLCs, between OPC servers and between PLCs and OPC servers. Refer to the [OPC Crosslink Help](#) for a full discussion of this tree.

The following sections describe these operational features of the server. Because the Network Connections Tree is normally configured first, we will start there.

## Network Connections Tree

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this tree.

The Network Connections tree is used to describe the physical connections to the field components. At the top of this tree are the communication driver agent folders. The **MBX (Modicon)** folder contains the configuration information for the MBX Driver Agent.

The MBX Driver Agent uses various means for connecting to their devices or networks. For a Modbus connection, a serial COM port serves that purpose. In other cases, a Modbus Plus or Ethernet adapter card is used. The Cyberlogic OPC Server refers to all of these using the generic term "network connection". The MBX Driver Agent uses the Cyberlogic MBX family of drivers for low-level communication services to its network connections. Each network connection in the Cyberlogic OPC Server corresponds to an MBX device in the driver configuration and contains all parameters for these devices.

The server refers to each physical device on the network as a "network node". A typical network node might be a Quantum controller on a Modbus TCP network. The server accesses the network nodes through their corresponding network connection. The network node configuration contains the communication parameters for the physical node device.

## Network Automatic Configuration

The Cyberlogic OPC Server Configuration Editor can automatically detect the Modicon devices attached to the network connections and create corresponding network nodes in the server configuration file.

### Caution!

The automatic configuration feature detects network connections and network nodes associated with the MBX devices you have configured. Before you can use automatic configuration, you must use the MBX configuration editor to create the devices that the OPC server will use.

## Standard Modbus Protocol vs Modbus Extensions

The Modbus protocol was originally developed by Modicon to interact with their PLCs. The data memory in these PLCs consists of five consecutive ranges: outputs (**000001-065536**), inputs (**100001-165536**), input registers (**300001-365536**), holding registers (**400001-465536**) and general registers (**600000-665535**). Outputs and inputs are single-bit, while the rest of them are 16-bit integers. The first digit in the register number identifies the register type.

Modbus has been widely accepted and has become a de facto standard industrial communication protocol. It is now a commonly available means of connecting industrial electronic devices from several vendors.

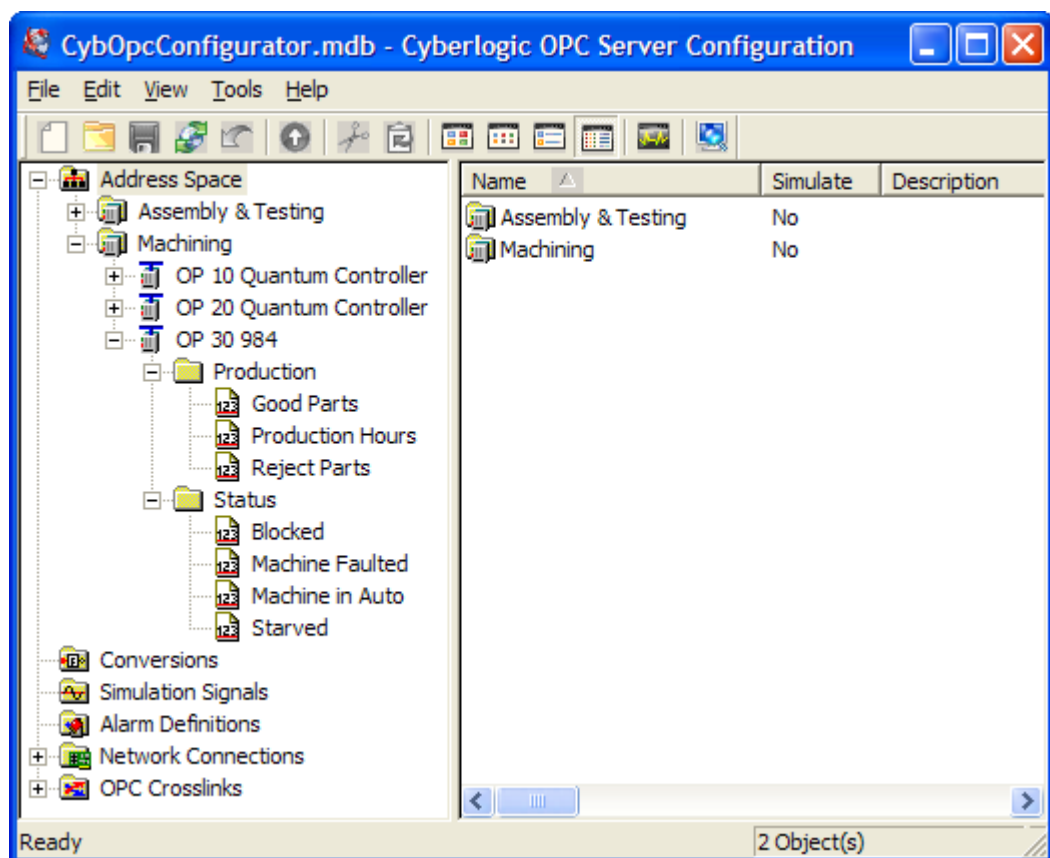
However, outside of discrete inputs and outputs, standard Modbus only allowed support for 16-bit registers. This became a limitation as many devices introduced 32-bit and 64-bit registers supporting both integer and floating-point values. Also, many products did not want to be limited to the five register ranges and the Modicon register numbering scheme. To overcome these limitations, some companies developed their own extensions to the standard Modbus, such as Enron Modbus by Enron Corporation.

The MBX OPC Driver Agent supports devices that use standard Modbus as well as the Modbus extensions. To accommodate the variety of existing devices, the MBX Driver Agent allows the user to describe the internal data memory layout by setting up an address map for each device type. This can be done on the [Address Maps](#) tab of the **MBX (Modicon)** folder located in the Network Connections tree.

## Address Space Tree

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this tree.

The address space tree allows you to organize the data items in a way that makes sense for your project. You can group and name related data items regardless of where they exist in the physical devices.



The branches of the tree are device folders, devices and folders. These establish how the data items are organized. The data items themselves are the “leaves” of the tree. You will begin construction of the tree at the address space root folder, which may contain device folders and devices.

## Access Paths

An access path is a logical connection to a network node. These connections, which are required for solicited communications, link the data items in an address space device with their values in a physical device. They tell the server where and how to obtain these values during solicited data reads and writes.

Each device in the server's address space can have a list of associated access paths. If there are more than one, the access path at the top of the list is the primary access path, and the rest are backups.

You can create and edit access paths on the device's [Access Paths Tab](#).

**Note**

Access paths are required only for solicited communications. If you are planning to use unsolicited data updates instead, you do not have to configure any access paths. Refer to the [Unsolicited Message Filters](#) section for more information.

## Unsolicited Message Filters

In addition to the more common solicited updates, the Cyberlogic OPC Server supports unsolicited data updates. In a solicited update, the server sends a request to a device asking it for data, and the device replies. In an unsolicited update, the device decides when to send data to the server. This helps to minimize the amount of traffic on the network. For information on how to program your programmable controller for unsolicited communication, refer to [Appendix D: Unsolicited Message Programming](#).

Although unrestricted unsolicited updates are possible, the Cyberlogic OPC Server supports a mechanism of unsolicited message filters to prevent data corruption. Unsolicited messages must first pass through the user-defined filters before the server accepts them. These filters guarantee that unsolicited messages are accepted only from trusted sources.

You can create and edit unsolicited message filters on the device's [Unsolicited Message Filters Tab](#).

**Note**

Unsolicited message filters are used only for unsolicited communications. If you are planning on using solicited data updates instead, you do not have to configure any unsolicited message filters. Refer to the [Access Paths](#) section for more information.

## Data Items

A data item represents a register in the physical device, a range of registers, a bit inside a register or a range of bits. The MBX Driver Agent supports all Modicon register types—coils (0x), inputs (1x), input registers (3x), holding registers (4x) and general reference registers (6x)—including global data. It also supports the addresses as defined by the custom address maps (see [Address Maps tab](#)). For more information on the supported

data types, refer to [Appendix A: Register Addresses](#). The user can individually configure each data item for solicited updates, unsolicited updates or both.

## Bit Order

Controllers and other devices will vary in the way they arrange the bits within their data items. For example, some store string data in low byte / high byte order, while others use high byte / low byte. Some have the bits in descending order, others in ascending order.

The MBX Driver Agent provides the ability to manipulate the order of the bits within data items to accommodate the different formats used by data sources and client software. The software can reverse the order of bits within each four-bit nibble, swap nibbles within each byte, swap bytes within each word, swap words within each double word and swap double words within each quad word. Users can specify any combination of these as needed.

For maximum flexibility and ease of use, the user can specify a bit swap pattern for each device, and can override the device setting for each data item, as desired.

For details on how to configure this feature, refer to the [Address Map](#)

This field identifies the address map that represents the data memory layout of the physical devices associated with this MBX Device. You may select from the address maps you configured on the Address Maps Tab in the **MBX (Modicon)** folder. The OPC Server Configuration Editor uses this information to verify the syntax for the Address field on the [Data Tab](#) of a data item. The default selection for this field is <Modicon PLC>.

Bit Order... discussion in the Devices section.

## Conversions Tree

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this branch.

## Simulation Signals Tree

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this branch.

## Alarm Definitions Tree

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion of this branch.

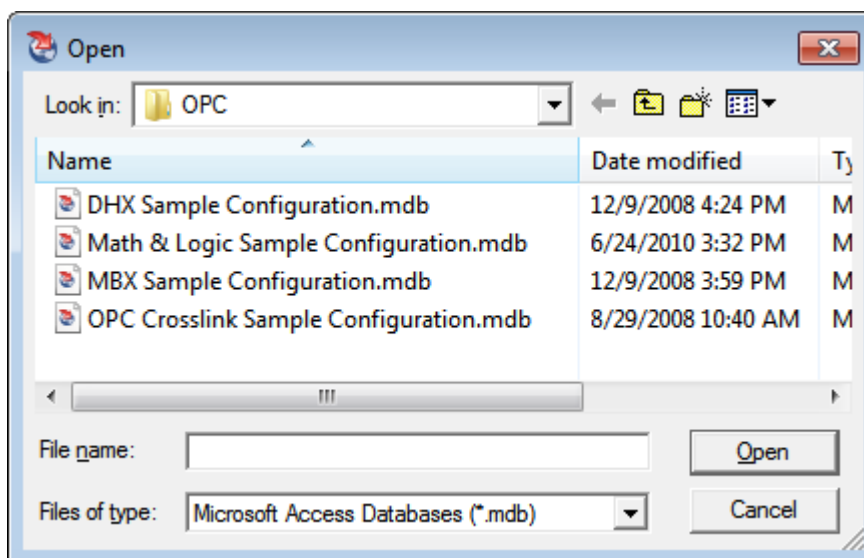
## QUICK-START GUIDE

Before you can use the OPC server, you must configure it by using the OPC Server Configuration Editor. Every server requires configuration of the Network Connections tree, and most users will want to configure the Address Space tree. The remaining trees (Conversions, Simulation Signals and Alarm Definitions) are optional features used by some systems.

### **Sample Configuration Files**

The default installation of all Cyberlogic OPC Server Suites includes a set of sample configuration files. These samples will help you to understand how to configure the OPC server for your project. In addition, the OPC Math & Logic sample provides you with numerous sample programs that you can modify and use in your system.

To open a sample configuration file from the OPC Server Configuration Editor, open the **File** menu and then select **Open Sample...**



A browse window will open to allow you to select the configuration file you want. The available choices will depend on which OPC products you have installed.

The default location of the files is:

C:\Program Files\Common Files\Cyberlogic Shared\OPC.

### **Step-By-Step Example**

For a step-by-step guide through a typical configuration session, refer to the [Cyberlogic OPC Server Help](#). After you have created the basic configuration using that procedure, the [Configuration Editor Reference](#) will explain the details of how to edit your configuration.

## CONFIGURATION EDITOR REFERENCE

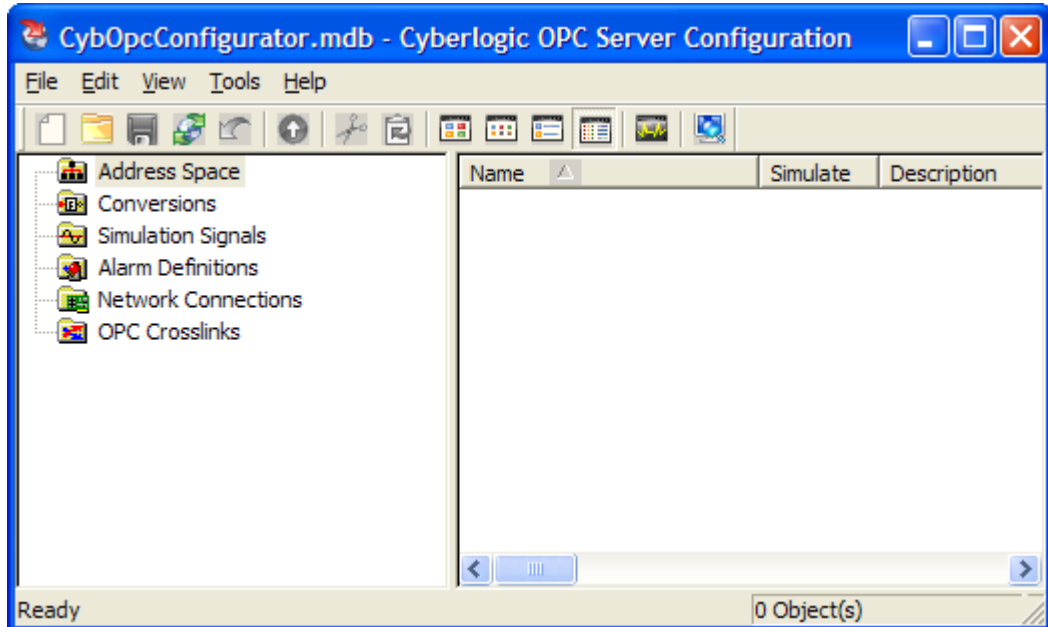
Before you can use the OPC server, you must configure it by using the OPC Server Configuration Editor. Every server requires configuration of the Network Connections tree, and most users will want to configure the Address Space tree. The remaining trees (Conversions, Simulation Signals and Alarm Definitions) are optional features used by some systems.

This section provides a detailed description of each of the configuration editor features. If you are a new user and want a procedure to guide you through a typical configuration session, refer to the Quick-Start Guide in the [Cyberlogic OPC Server Help](#).

The Cyberlogic OPC Server Configuration Editor allows the user to create and modify the configuration file used by the runtime module. It is needed only to generate configuration files and is not otherwise required for the operation of the runtime module.

**Caution!** After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

To launch the editor from the Windows **Start** menu, go to **Cyberlogic Suites**, then open the **Configuration** sub-menu, and then select **OPC Server**.



The left pane of the main workspace window includes the five main configuration trees. Two of these, the [Network Connections](#) and [Address Space](#) trees, are described in detail here, followed by a section on [Saving and Undoing Configuration Changes](#).

For a discussion of the Conversions, Simulation Signals and Alarm Definitions trees and other important configuration topics including Configuration Import/Export, Editor



Options and Connecting to OPC Client Software, please refer to the [Cyberlogic OPC Server Help](#).

## Network Connections

The MBX Driver Agent is a plug-in software module that permits the Cyberlogic OPC Server to communicate with Modicon networks, Modicon controllers and other Modbus-compatible devices. You will do this by configuring appropriate network connections and network nodes in the [Network Connections Tree](#).

The MBX Driver Agent uses the industry-standard MBX drivers for its low-level communication services. A network connection in the OPC server corresponds to an MBX device in the driver architecture. An MBX device may relate to a physical network card, such as a PCIe-85, or an abstract object, such as an Ethernet MBX device, which behaves like a network card. A network node in the OPC server corresponds to a Modicon controller or other compatible equipment that is connected to the OPC server over one of the Modicon networks.

### Creating and Deleting

There are two ways to create the network connections and network nodes: automatic and manual.

#### Caution!

After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

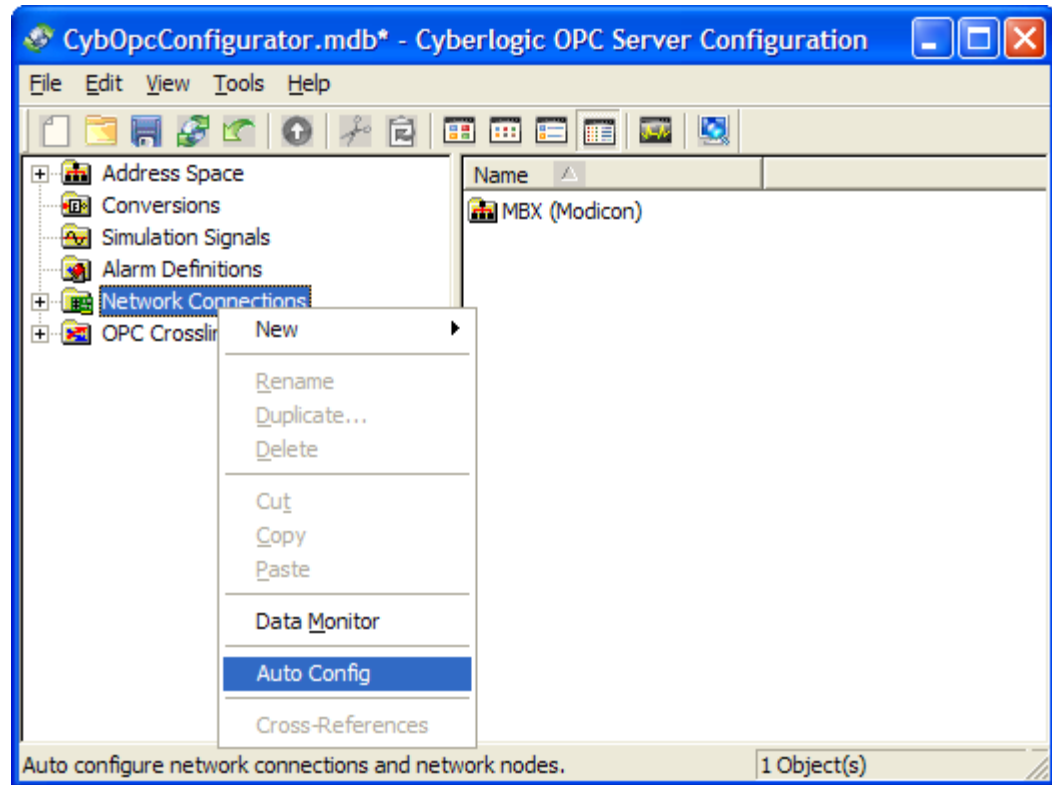
#### *Automatic Configuration*

The simplest method of configuration is automatic configuration. The automatic configuration feature detects network connections and network nodes associated with the MBX devices you have configured.

Before you can use automatic configuration, you must use the MBX configuration editor to create the devices that the OPC server will use. For Ethernet networks, automatic configuration will detect only the nodes that are mapped to Modbus Plus node addresses. Refer to the driver help file for information on how to configure MBX devices.

#### Full Automatic Configuration

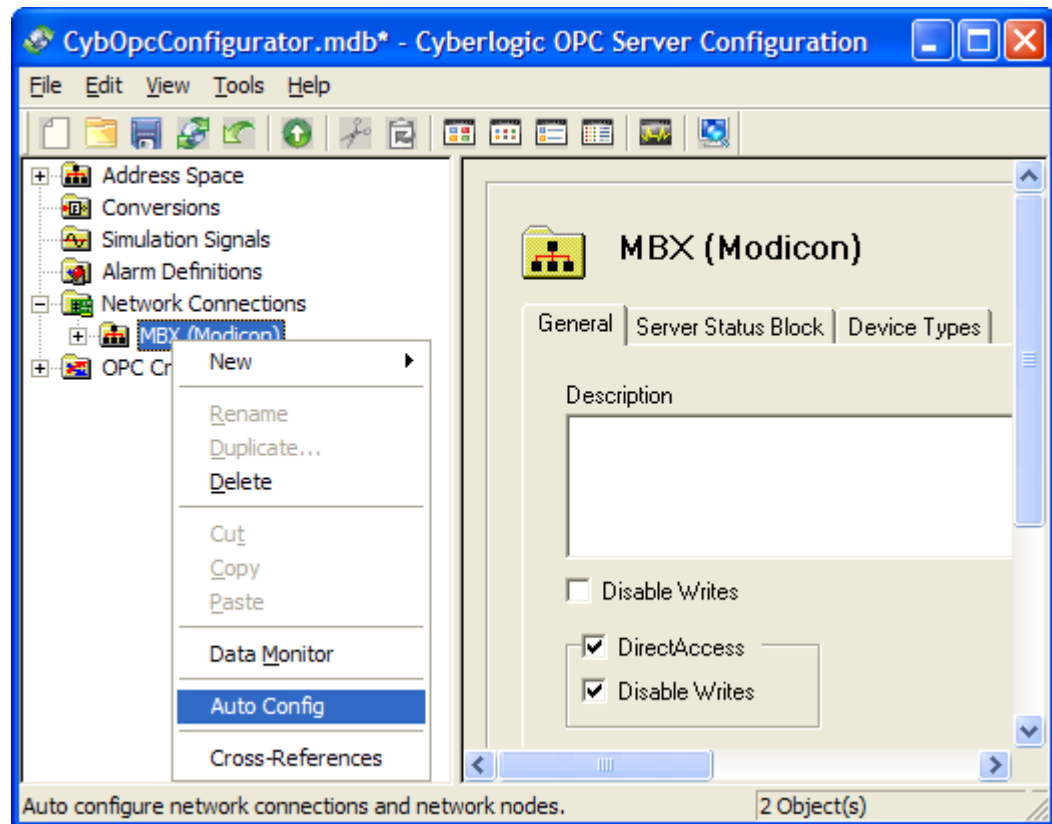
This is the most common automatic configuration procedure. It will find all network connections, and detect and configure all network nodes.



To do this, right-click on the **Network Connections** root folder and select **Auto Config** from the context menu.

Automatic Configuration of a Single Driver Agent

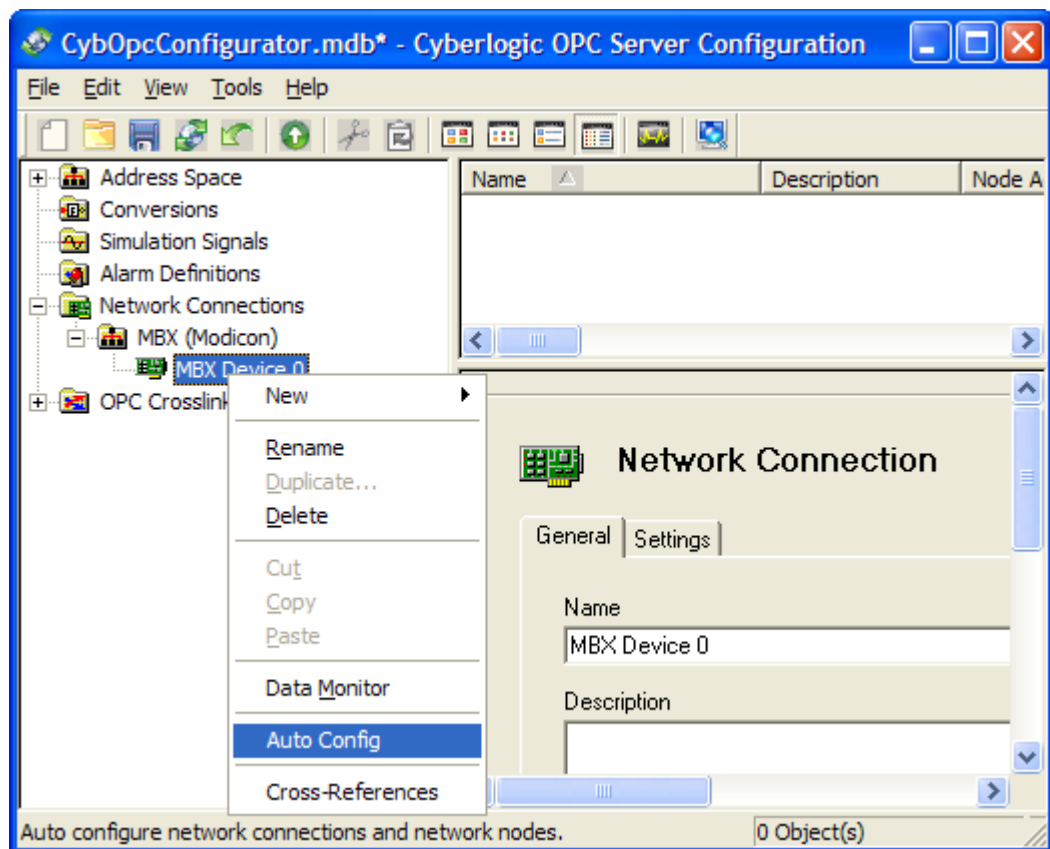
After an MBX Driver Agent folder has been created, you can automatically find all MBX network connections available on your system. Typically, you would do this if you did part of the configuration while not connected to the target network. You can then use this feature to quickly finish the configuration once you are connected.



Right-click the **MBX (Modicon)** folder and select **Auto Config** from the context menu.

#### Automatic Configuration of Network Nodes

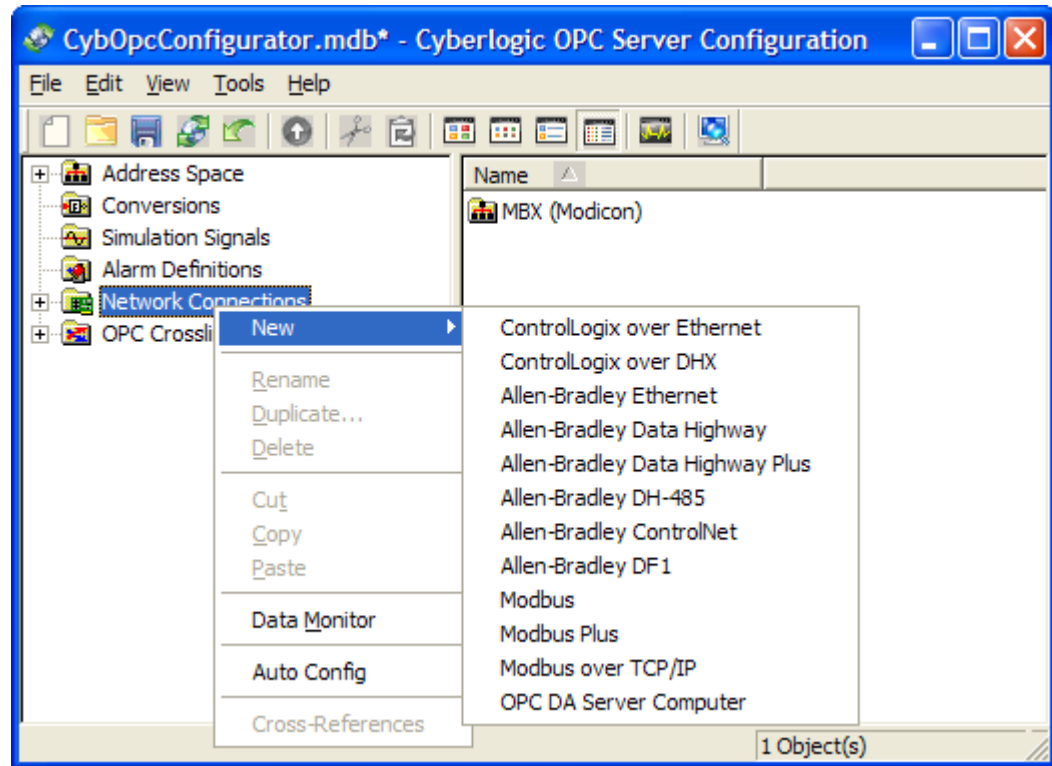
After an MBX network connection has been configured, you can automatically find and configure all of the network nodes attached to that network connection. You might do this, for example, if you add nodes to a network after it is configured and want to quickly update the configuration with the new nodes.



Right-click the specific **MBX network connection** and select **Auto Config** from the context menu.

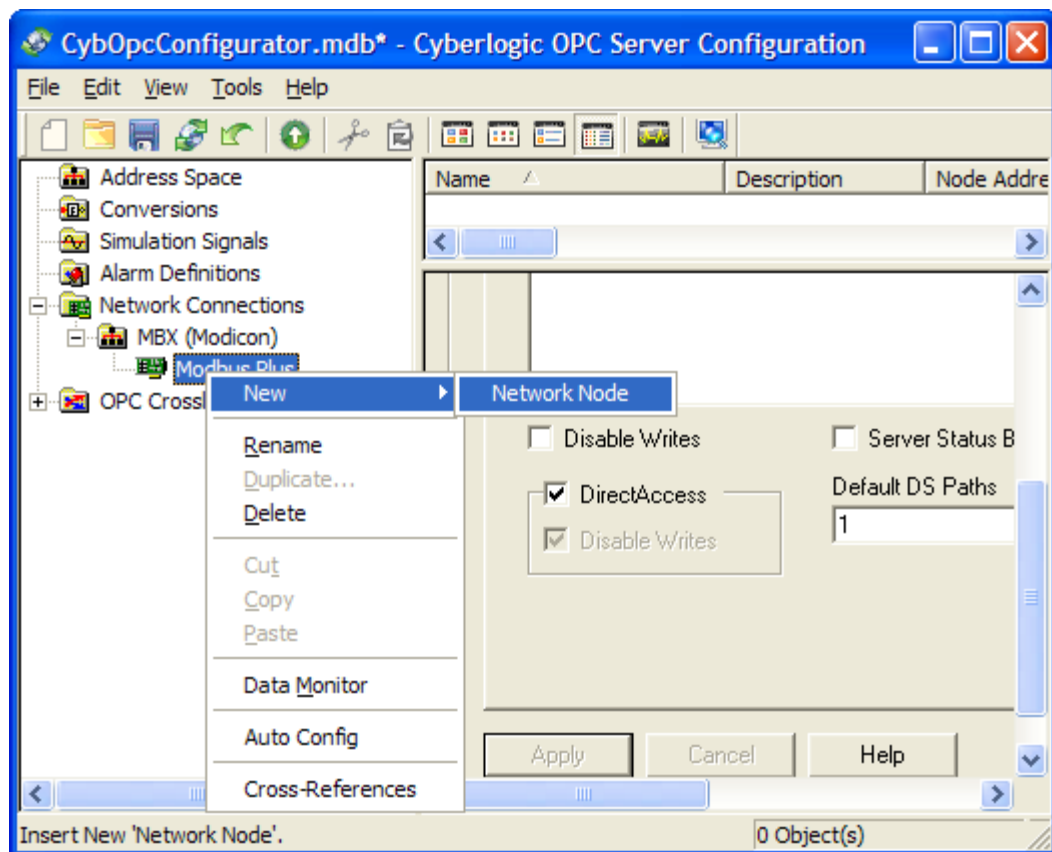
### **Manual Configuration**

You may prefer to configure your communications manually. This will be necessary if you are doing the configuration on a computer that is not connected to the target networks or if you wish to change the default values selected during an automatic configuration. You would also need to manually configure Ethernet MBX nodes that are not mapped to Modbus Plus node addresses.



Right-click on the **Network Connections** root folder and select **New**, then select the desired network type from the context menu.

The Editor will create the proper driver agent and network connection folders.



You can now right-click on the network connection and select **New**, and then **Network Node** to manually create a network node.

### **Deleting**

To delete the MBX Driver Agent folder, a network connection or a network node, select it and press the **Delete key**, or right-click on it and select **Delete** from the context menu.

Deleting the MBX Driver Agent folder will also delete all MBX network connections and network nodes.

Deleting an MBX network connection will also delete all of its network nodes.

### **Note**

You cannot delete a network node that is used as part of an access path for a device. If you wish to delete a network node that is in use, right-click on it and select **Cross-References** to obtain a list of devices that use the network node. You must then edit those devices to remove the access path that contains the network node you want to delete.

## Editing the MBX Driver Agent

Once you have created an MBX network connection, simply click on the **MBX (Modicon)** folder, and the configuration screen will appear on the right side of the editor.

The MBX Driver Agent configuration screen has four tabs: General, Server Status Block, Device Types and Address Maps.

**Caution!** After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

### General Tab

The screenshot shows the 'MBX (Modicon)' configuration dialog box with the 'General' tab selected. The dialog has four tabs: 'General', 'Server Status Block', 'Device Types', and 'Address Maps'. The 'General' tab contains a 'Description' text area, a 'Disable Writes' checkbox (unchecked), and a group box containing 'DirectAccess' and 'Disable Writes' checkboxes (both checked). At the bottom are 'Apply', 'Cancel', and 'Help' buttons.

#### Description

This optional field can be used to describe the driver agent. It can be up to 255 characters long.

#### Disable Writes

If this box is checked, the server will not write data to any of the network nodes that connect through this driver agent.

The default state is unchecked, enabling writes.

*DirectAccess*

If this box is checked, the user is permitted to configure [DirectAccess](#) to the network nodes that connect through this driver agent.

The default state is checked, allowing DirectAccess.

*DirectAccess Disable Writes*

If this box is checked, the server will not write data via DirectAccess to any of the network nodes that connect through this driver agent. This does not affect writes through configured data items.

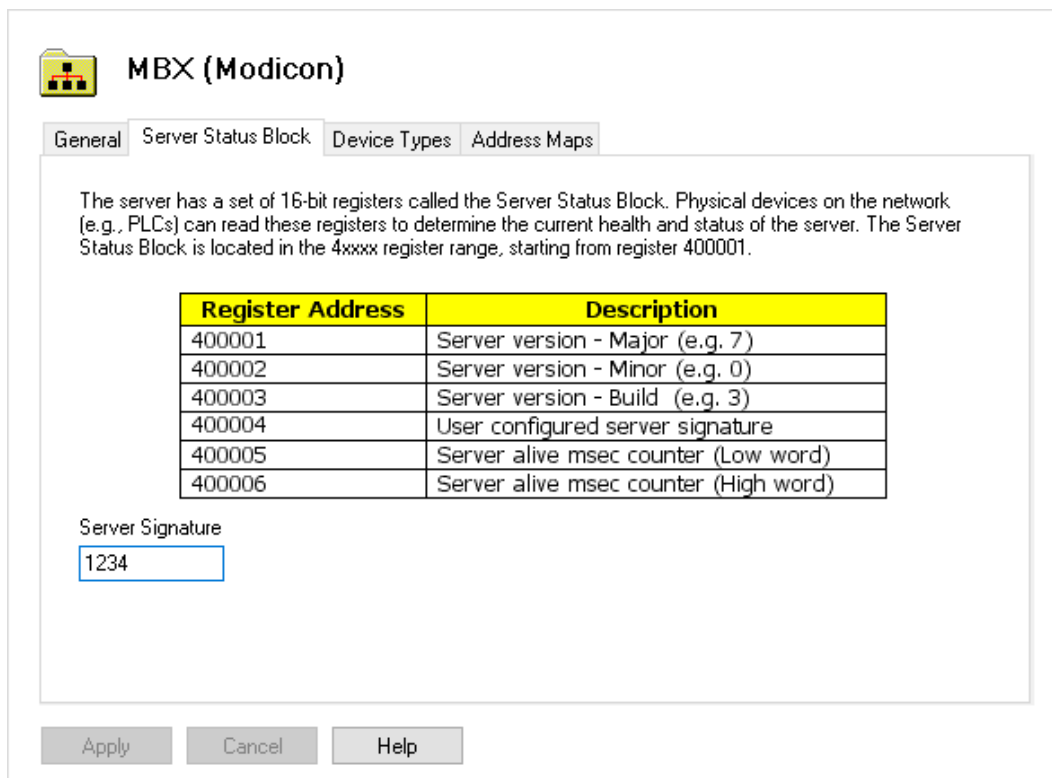
The default state is checked, disabling DirectAccess writes.

***Server Status Block Tab***

The Cyberlogic OPC Server has a block of 16-bit registers called the server status block. Physical devices on the network, such as PLCs, can read these registers to obtain information about the health and current status of the server. The server status block is located in registers 400001 through 400006:

<b>Register Address</b>	<b>Description</b>
400001	Server version – Major (e.g. 7)
400002	Server version – Minor (e.g. 0)
400003	Server version – Build (e.g. 3)
400004	User-configured server signature
400005	Server alive millisecond counter (Low word)
400006	Server alive millisecond counter (High word)





***Server Signature***

Enter a number that will uniquely identify this server. Devices that read this signature value will then be able to identify which server they are communicating with. The default value is 1234.

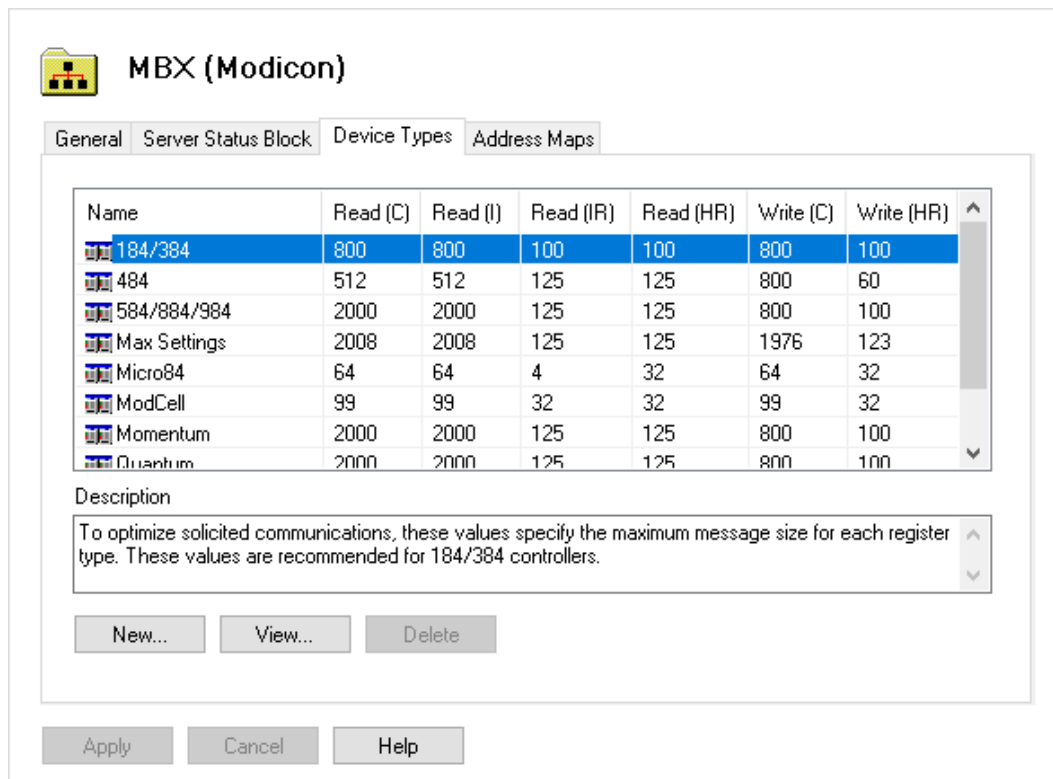
***Device Types Tab***

At run time, the OPC server tries to optimize its solicited communications based on the types of physical devices to which it communicates. One of the ways it does this is by adjusting the maximum message size for different types of registers.

The MBX Driver Agent is preconfigured with several standard device types from which you can choose. You can also create additional custom device types for physical devices that do not match one of the standard types.

**Note** The Safe Settings selection shows a set of conservative settings, which will work with any device type, but which will not provide particularly good performance.

**Note** Most recently developed devices are compatible with large message settings, such as the settings shown for the Momentum or Quantum PLCs. However, we recommend that you contact your device manufacturer to confirm these settings.



New...

Click this button to create a new, custom device type. The [Device Type Editor](#) will open.

Edit...

Select a custom device type and click this button to edit it. The [Device Type Editor](#) will open.

**Note**

Standard device types cannot be edited. When selected, the **Edit...** button is replaced with the **View...** button.

Delete

Select a custom device type and click this button to delete it.

### ***Device Type Editor***

**Device Type** [X]

For each custom device type you must specify the maximum number of registers of each type that may be handled in a single transaction. For Holding and Input registers, the values are expressed in terms of 16-bit registers. The values you select must not exceed the capability of the device you are communicating with.

Name

Description

**Read**

Coils (C) <input type="text" value="2000"/>	Input Reg (IR) <input type="text" value="125"/>
Inputs (I) <input type="text" value="2000"/>	Holding Reg <input type="text" value="125"/>

32/64 bit Registers  
 Modicon Compatible

**Write**

Coils (C) <input type="text" value="800"/>	<input checked="" type="checkbox"/> Supports Modbus Fun 05
Holding Reg (HR) <input type="text" value="100"/>	<input checked="" type="checkbox"/> Supports Modbus Fun 06

OK Cancel Help

#### *Name*

Here you can enter a descriptive name for the device type.

#### *Description*

This optional field can be used to describe the device type. It can be up to 255 characters long.

#### *Read and Write Sections*

Enter the desired maximum message sizes for each of the register types.

**Note**

For Holding and Input registers, the values are always expressed in terms of 16-bit registers, even if your device supports 32/64-bit registers.

***Supports Modbus Fun 05***

Modbus protocol includes two functions that allow writing to coils: function 05 (Force Single Coil) and function 15 (Force Multiple Coils). The MBX OPC Driver Agent supports both functions and uses the most appropriate function as needed. However, some third-party devices support only one of these functions. If your device does not support Modbus function 05, uncheck this checkbox. The driver agent will use Modbus function 15 for all coil writes.

The default state is checked, enabling the use of the Modbus function 05.

**Note**

If your device does not support Modbus function 15 (Force Multiple Coils), set the maximum number of coils that can be written to 1. The driver agent will use Modbus function 05 (Force Single Coil) for all coil writes.

***Supports Modbus Fun 06***

Modbus protocol includes two functions that allow writing to holding registers: function 06 (Preset Single Register) and function 16 (Preset Multiple Registers). The MBX OPC Driver Agent supports both functions and uses the most appropriate function as needed. However, some third-party devices support only one of these functions. If your device does not support Modbus function 06, uncheck this checkbox. The driver agent will use Modbus function 16 for all holding register writes.

The default state is checked, enabling the use of the Modbus function 06.

**Note**

If your device does not support Modbus function 16 (Preset Multiple Registers), set the maximum number of holding registers that can be written to 1. The driver agent will use Modbus function 06 (Preset Single Register) for all holding register writes.

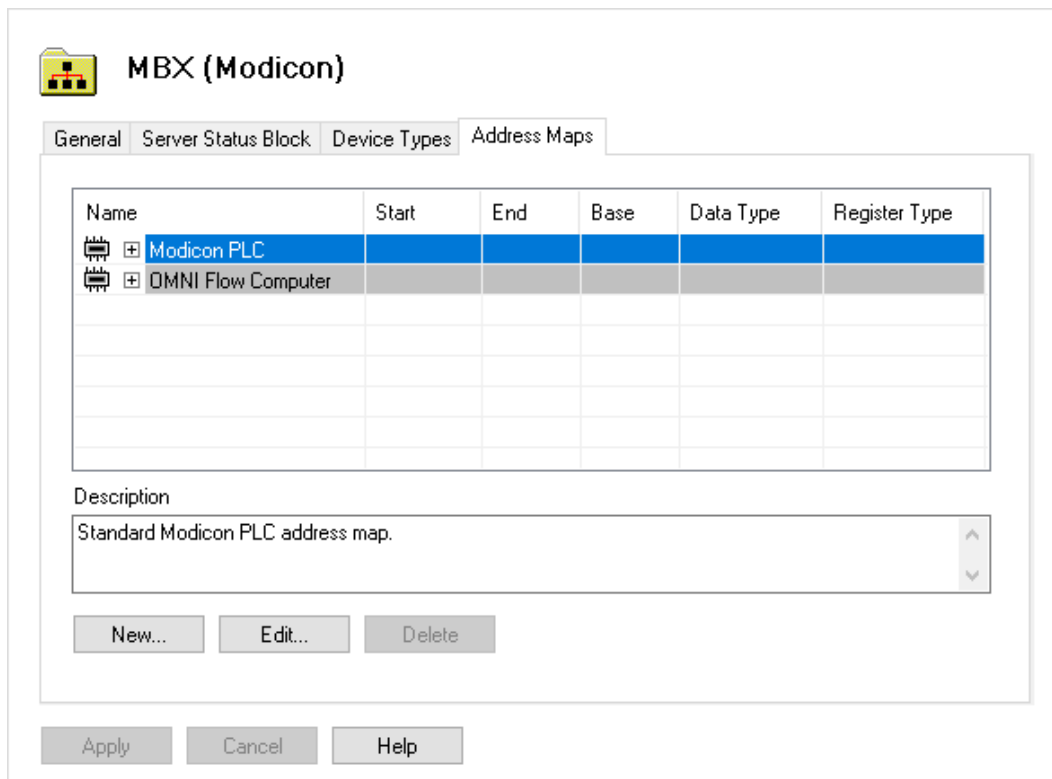
***Modicon Compatible***

Some Modbus requests (e.g. Function 3 Read Holding Registers and Function 16 Preset Multiple Registers) include a count that specifies how many registers/values to transfer. In the standard (Modicon) Modbus protocol, this count always represents the number of 16-bit registers. However, Modbus protocol extensions also allow for larger 32-bit and 64-bit registers. In most cases, the above count represents the number of registers, regardless of the register size. But some implementations still express it as the number of 16-bit values. If that is the case for your custom device, check the Modicon Compatible checkbox.

***Address Maps Tab***

The MBX OPC Driver Agent supports devices that use standard Modbus as well as the Modbus extensions. To accommodate the variety of existing devices, the MBX Driver

Agent allows the user to describe the internal data memory layout by setting up an address map for each device type on the Address Maps tab.



The MBX Driver Agent is preconfigured with a couple of common device maps: one for a generic Modicon PLC and the other for the OMNI Flow Controller from OMNI Flow Controllers, Inc. You can also create additional address maps for physical devices that do not match one of the standard maps.

*New...*

Click this button to create a new, custom address map. You will be given a choice of either starting with an empty map or a map that is based on one of the existing maps. The [Address Map Editor](#) will open.

*Edit...*

Select an address map and click this button to edit it. The [Address Map Editor](#) will open.

**Note** Preconfigured address maps cannot be edited. When selected, the ***Edit...*** button is replaced with the ***View...*** button.

*Delete*

Select an address map and click this button to delete it.

## Address Map Editor

An address map is a collection of address ranges. This editor allows you to add new address ranges to the map and to edit existing address ranges.

**Address Map**

Name: My Modicon PLC

Description: Standard Modicon PLC address map.

Modicon PLC

Start	End	Base	Data Type	Register Type
1	9999	1	BIT (1-bit Boolean)	Coil
10001	19999	10001	BIT (1-bit Boolean)	Input
30001	39999	30001	UINT16 (Unsigned 16-bit Integer)	Input Register
40001	49999	40001	UINT16 (Unsigned 16-bit Integer)	Holding Register
60000	69999	60000	UINT16 (Unsigned 16-bit Integer)	General Register
100001	165536	100001	BIT (1-bit Boolean)	Input
300001	365536	300001	UINT16 (Unsigned 16-bit Integer)	Input Register
400001	465536	400001	UINT16 (Unsigned 16-bit Integer)	Holding Register
600000	699999	600000	UINT16 (Unsigned 16-bit Integer)	General Register

Description:

New... Edit... Delete

OK Cancel Help

### Name

Here you can enter a descriptive name for the address map.

### Description

This optional field can be used to describe the address map. It can be up to 255 characters long.

New...

Click this button to create a new address range. The [Address Range Editor](#) will open.

Edit...

Select an address range and click this button to edit it. The [Address Range Editor](#) will open.

Delete

Select an address map and click this button to delete it.

Modicon PLC

Modicon (and Schneider Electric) PLCs provide a mechanism to read the register range size from the PLC at runtime. Check this checkbox if this address map describes the register layout of a Modicon/Schneider PLC.

**Address Range Editor**

Each address range defines the start and the end address of the range, the base address (used to calculate the address value in the Modbus message frame), register data type, and the register type for the range.

The screenshot shows the 'Address Range Editor' dialog box. It features a title bar with the text 'Address Range' and a close button (X). The main area contains three input fields: 'Start Address' with the value '40001', 'End Address' with '49999', and 'Base Address' with '40001'. Below these are two dropdown menus: 'Data Type' set to 'UINT16 (Unsigned 16-bit Integer)' and 'Register Type' set to 'Holding Register'. At the bottom, there is a 'Description' text area and three buttons: 'OK', 'Cancel', and 'Help'.

### Start Address

This is the start address of the address range.

### End Address

This is the end address of the address range.

**Note** The *End Address* value can be larger than the end register address in the corresponding physical device. If a non-existing register is referenced, the MBX Driver Agent will try to discover the actual end address at runtime. To avoid this discovery process, set the *End Address* to match the physical device.

**Note** A specific register address can belong to only one address range. Register addresses from different address ranges cannot overlap.

### Base Address

The Modbus protocol includes specific messages to interact with each register type. To identify a register of a given type, a Modbus frame uses a 16-bit data address field with a value between 0 and 65,536. To convert the register address to this data address, the *Base Address* value must be subtracted from the register address:

$$\text{Data Address} = (\text{Register Address}) - (\text{Base Address})$$

For example, in a Modicon PLC, for an address range starting with the address 40001, you would set the *Base Address* to 40001. However, in the Enron Modbus, the register addresses are equal to the data addresses. In that case, you would set the *Base Address* to zero.

**Note** Data addresses from different address ranges of the same register type can overlap only when their register data types have the same size (except STRING and WSTRING). Otherwise, data address overlaps are not allowed.

### Data Type

This field selects the data type for each register in the address range. The following data types are supported:



<b>Data Type</b>	<b>Size in bits</b>	<b>Description</b>
BIT	1	1-bit boolean
SINT8	8	Signed 8-bit integer -128 to 127
UINT8	8	Unsigned 8-bit integer 0 to 255
SINT16	16	Signed 16-bit integer -32,768 to 32,767
UINT16	16	Unsigned 16-bit integer 0 to 65,535
SINT32	32	Signed 32-bit integer -2,147,483,648 to 2,847,483,647
UINT32	32	Unsigned 32-bit integer 0 to 4,294,967,295
SINT64	64	Signed 64-bit integer -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
UINT64	64	Unsigned 64-bit integer 0 to 18,446,744,073,709,551,615
FLOAT32	32	IEEE 32-bit floating point number $\pm 3.4e\pm 38$
FLOAT64	64	IEEE 64-bit floating point number $\pm 1.7e\pm 308$
BCD16	16	BCD value; 0 to 9,999
BCD32	32	BCD value; 0 to 99,999,999
STRING	String size * 8	Zero terminated ASCII string of 8-bit characters
WSTRING	String size * 16	Zero terminated UNICODE string of 16- bit characters

Register Type

This is the register type for each register within the address range. The supported register types are: Coil, Input, Input Register, Holding Register, and General Register.

Description

This optional field can be used to describe the address range. It can be up to 255 characters long.

**Editing Network Connections**

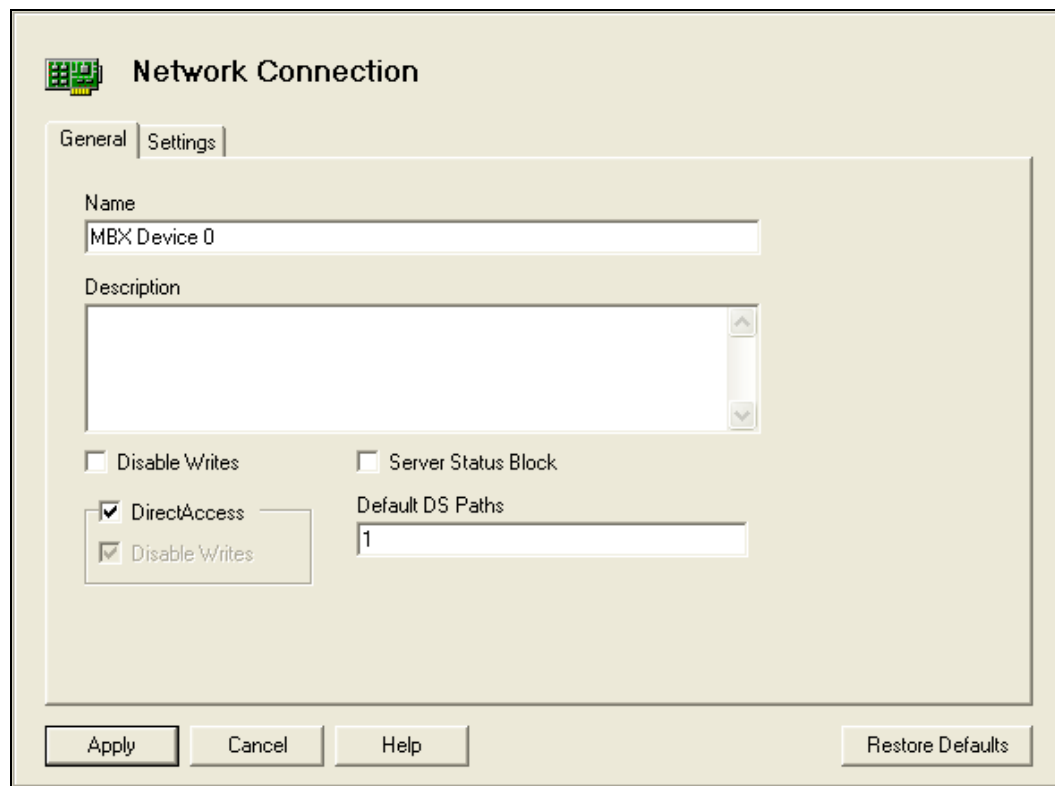
Once you have created an MBX network connection, simply select it and the configuration screen will appear on the right side of the editor.

The MBX network connection configuration has two tabs, General and Settings.

**Caution!**

After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

**General Tab**



Name

The name identifies this network connection. It can be up to 50 characters long, may contain spaces, but must not begin with a space. It also must not contain any periods.

### Description

This optional field further describes the network connection. It can be up to 255 characters long.

### Disable Writes

If this box is checked, the server will not write data to any of the nodes on this network connection.

The default state is unchecked, enabling writes.

**Note** If the Disable Writes checkbox is grayed-out, it indicates that writes have already been disabled at a higher level.

### DirectAccess

If this box is checked, the user is permitted to configure [DirectAccess](#) to the nodes on this network connection. The default state is checked, allowing DirectAccess.

**Note** If the DirectAccess checkbox is grayed-out, it indicates that DirectAccess has already been disabled at a higher level.

When DirectAccess is enabled, a `_Status` folder will appear under this network connection in the client browser window. For more information on this folder and the status items it contains, refer to the [Cyberlogic OPC Server Help](#).

### DirectAccess Disable Writes

If this box is checked, the server will not write data via DirectAccess to any of the nodes on this network connection. This does not affect writes through configured data items. The default state is checked, disabling DirectAccess writes.

**Note** If the DirectAccess Disable Writes checkbox is grayed-out, it indicates that DirectAccess writes have already been disabled at a higher level.

### Server Status Block

If this box is checked, the network nodes under this network connection will be able to read the server status block values. For an explanation of the server status block, refer to the [Server Status Block Tab](#) section.

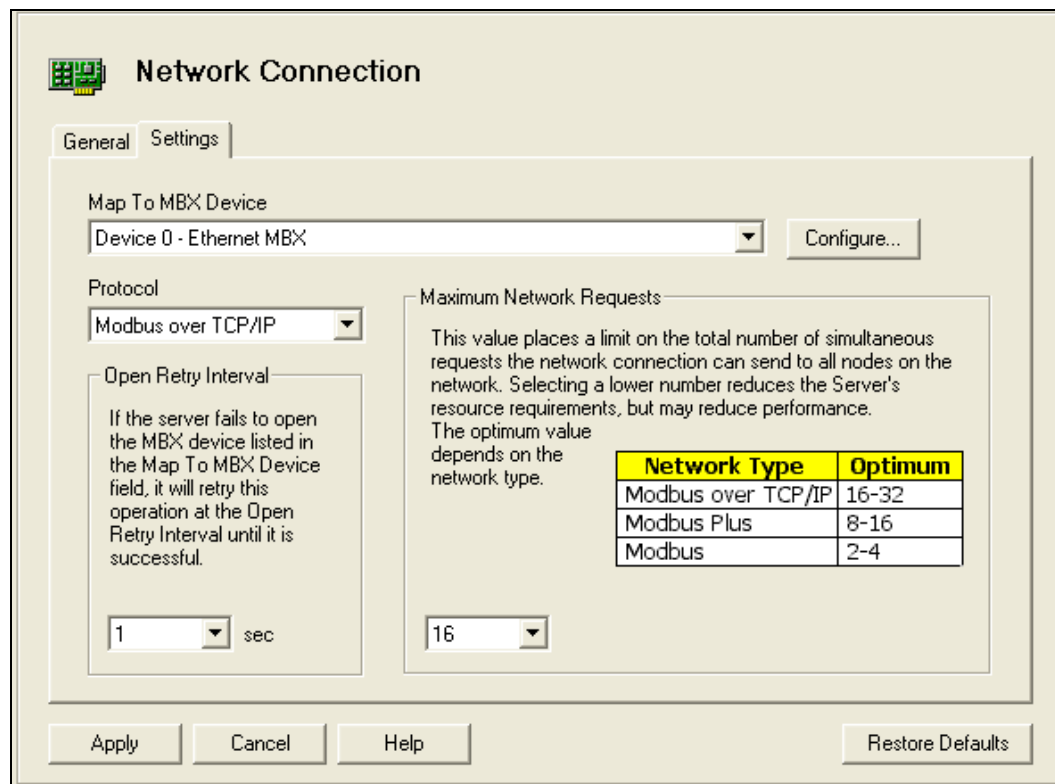
The default state is unchecked, disabling access to the server status block.

Default DS Paths

This field specifies the Data Slave paths on which the server will listen for server status block read requests. The server will also use these paths for unsolicited communication to [Devices](#) that use this network connection and have their Accept All Unsolicited box checked.

The default path is 1.

**Settings Tab**



Map To MBX Device

Click the drop-down button to select the MBX device this network connection will use to communicate.

The data field will tell you if the device is not configured or if it is configured for a different protocol than you have selected for the network connection.

Configure...

Click this button to run the MBX Driver Configuration Editor, allowing you to modify the configuration of the MBX devices and to create new ones. For more information on configuring MBX devices, refer to the help files for each of the MBX family drivers.

Protocol

Click the drop-down button to select the specific protocol used by this network.

Open Retry Interval

The server will try to open the MBX device shown in the Map To MBX Device field. If it fails, it will retry repeatedly until it succeeds. This field specifies the interval at which the server will attempt these retries. This interval is also used when opening DS paths used for unsolicited communications.

The valid range for the Open Retry Interval is 1-60 seconds, and the default is 1 second.

Maximum Network Requests

This value defines the maximum number of simultaneous transactions that the server will allow for the nodes on this network connection. The number of concurrent requests may also be limited for each network node on its [Optimizations Tab](#).

The optimum value to use depends upon the network type. The slowest network in the path to the network node should dictate the range to use. For example, if the OPC server has a 1GB Ethernet connection to a bridge, and the bridge has a 9600 baud serial connection to the network node, the Modbus range should be used, not the Ethernet range.

Use the following table as a guideline when selecting this value.

Network Type	Optimum Range
Modbus TCP	16-32
Modbus Plus	8-16
Modbus	2-4

**Caution!**

Lower values reduce the server’s resource requirements, but may reduce performance. Selecting values above the recommended range consumes more system resources, but typically does not improve the performance of the server, and may actually harm performance.

Refer to [Appendix E: Configuring Maximum Concurrent Requests](#) for more information and examples.

**Editing Network Nodes**

Once you have created an MBX network node, simply select it and the configuration screen will appear on the right side of the editor.

The MBX network node configuration has four tabs, General, Settings, Health Watchdog and Optimizations.

**Caution!** After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

**General Tab**

Name

The name identifies the network node. It can be up to 50 characters long, may contain spaces, but must not begin with a space. It also may not contain any periods.

Description

This optional field further describes the network node. It can be up to 255 characters long.

Disable Writes

If this box is checked, the server will not write data to this node. The default state is unchecked, enabling writes.

**Note** If the Disable Writes checkbox is grayed-out, it indicates that writes have already been disabled at a higher level.

#### *DirectAccess*

If this box is checked, the user is permitted to configure [DirectAccess](#) to this node. The default state is checked, allowing DirectAccess.

**Note** If the DirectAccess checkbox is grayed-out, it indicates that DirectAccess has already been disabled at a higher level.

When DirectAccess is enabled, a `_Status` folder will appear under this network node in the client browser window. For more information on this folder and the status items it contains, refer to the [Cyberlogic OPC Server Help](#).

#### *DirectAccess Disable Writes*

If this box is checked, the server will not write data via DirectAccess to this node. This does not affect writes through configured data items. The default state is checked, disabling DirectAccess writes.

**Note** If the DirectAccess Disable Writes checkbox is grayed-out, it indicates that DirectAccess writes have already been disabled at a higher level.

### ***Settings Tab (Modbus Plus Networks)***

The Settings tab layout is specific to the network type. This is the layout and description when a Modbus Plus network is configured.

For other network types, refer to:

- [Settings Tab \(Ethernet Networks\)](#)
- [Settings Tab \(Modbus Networks\)](#)

**Network Node**

General Settings Health Watchdog Optimizations

Device Type: Quantum

Address Map: <Modicon PLC>

Solicited

Node Address: 15 . 0 . 0 . 0 . 0

Timeout: 3 sec

Retries: 3

Unsolicited

Routing: \* . \* . \* . \*

Source Node: 15

Apply Cancel Add New Help Restore Defaults

### Device Type

This field identifies the physical device type of this network node. You may select from the physical device types you configured on the network connections [Device Types Tab](#). The server uses this information to optimize network communications.

### **Caution!**

If you select Auto Detect, the server uses FNC 17 (11 hex) *Report Slave ID* to determine the device type. All Modicon controllers support this function, but some third-party devices may not. For those devices, the Auto Detect will fail and the server will default to the Safe Settings optimization. Because this setting may not provide the best performance, you should choose a different device type selection for these devices.

If you use global data and you want to read or write the OPC server's global data, you must create a node of type *This Node – Global Data*. This will then allow you to create global data type data items in the address space and connect them to the local node. For more information on global data, refer to [Appendix A: Register Addresses](#).

### Address Map

This field identifies the address map that represents the data memory layout of the physical device associated with this network node. You may select from the address maps you configured on the [Address Maps Tab](#) in the **MBX (Modicon)** folder. The server uses this information to properly build network messages. The default selection for this field is <Modicon PLC>.



### Solicited Node Address

This is the node's network address in the form of a standard five-field Modbus Plus routing array. For more information on node addressing, refer to the MBX Driver help file.

### Solicited Timeout

This is the amount of time that the server will wait to receive a reply to a command message. If the server does not receive a reply within that interval, it cancels the transaction and marks it as timed out. This interval is specified in seconds.

### Solicited Retries

This is the number of times the server will reattempt each transaction that fails. The health watchdog uses this value to determine when to mark the node as unhealthy. Refer to the [Health Watchdog Tab](#) section for more information on this feature.

### Unsolicited Section

For security and safety reasons, you may want to configure your system to permit unsolicited updates only from trusted sources. To do this, you will set up filters when you configure the devices in the address space. These filters allow you to specify which network nodes are permitted to provide unsolicited updates to a particular device. Refer to the [Unsolicited Message Filters Tab](#) section for more details.

For the filters to do their job, the server must have a way to identify the network node that is the origin of each unsolicited message it receives. Every message that the server receives contains a five-byte Modbus Plus routing array and a single-byte source address. The information in the Unsolicited section tells the server how to use the routing array and source address to determine whether or not the message was sent from this network node.

If the Node Address in the Solicited section is of the form *n.0.0.0.0*, where *n* is any node address, then the server knows that this node is located on the local Modbus Plus network. In such a case, the source address appended to the message is all that is needed to positively identify that the message came from this node. The Unsolicited section will be configured for you automatically and will not be available for editing. The routing array value is irrelevant, so the Routing field will be set to *\*,\*,\*,\*,\** because that will accept any routing array value. The Unsolicited Source Node field will be set to the value, *n*, of the first byte in the Solicited section's Node Address field.

If the sending node is located on another Modbus Plus network accessed through one or more bridges, it may not be possible to exactly identify that node. In that case, you can configure the Unsolicited Routing field and the Unsolicited Source Node field to at least narrow down the group of nodes that are permitted to provide unsolicited updates.

### Unsolicited Routing

Here you will specify, as nearly as possible, the five-byte routing path that will be reported to the server for unsolicited messages coming from this node. Each field may

contain a number or an asterisk. If it contains a number, the incoming message must have that value in that position. If it contains an asterisk, any value is acceptable.

Each message received by the server from a Modbus Plus node contains a five-byte routing array as follows.

<b>Byte 1 Server Node Address</b>	<b>Byte 2 Server Slave Path</b>	<b>Byte 3 Normally unused</b>	<b>Byte 4 Normally unused</b>	<b>Byte 5 Normally unused</b>
1 ... 64	1 ... 8	0 ... 255	0 ... 255	0 ... 255

- **Server Node Address.** The first byte will always be the node address of the server on the Modbus Plus network.
- **Server Slave Path.** The second byte will always be the slave path that the server will use to process the message.
- **Normally Unused Bytes.** The third through fifth bytes are normally unused, with values of 0. However, if the sending node does not need all five routing bytes to specify the routing of the message, the extra bytes can be used to provide additional information to identify the source of the message.

For example, suppose the OPC server has Modbus Plus node address 17 and we want to specify slave path 1. The sending node is on another Modbus Plus network and must route the message through a bridge at node address 6 on the other network. The routing it would use would be 6.17.1.0.0. After passing through the bridge, the routing would shift left one place, to become 17.1.0.0.0.

The problem is that this provides no information to help identify the source node. Every unsolicited message the server receives would have 17 in the first byte and a slave path number in the second byte, followed by three zeros. You could, however, have the sending node put a value—let’s say 20—in the first unused byte, to identify itself. The resulting routing would be 6.17.1.20.0 and would route correctly. The routing information received by the server would then be 17.1.20.0.0, giving it some additional information to work with. You could then set the Unsolicited Routing field to 17.1.20.0.0, and it would pass only those messages that originated from this node. Actually, since the value in the third byte is all that is needed to identify the sending node, it might be clearer to set the Unsolicited Routing to \*.\*.20.\*.\*. By using a different identifier byte value for each sending node, you could positively identify the sender.

Unsolicited Source Node

Enter the node address that will be reported to the server as the source node of the unsolicited messages.

The reported address will be the node address of the bridge on the server’s local network that passed the message to the server. All messages coming through that bridge will have the same source node address: the address of the bridge. It is for this reason that the Routing field must be configured to provide unique identification of the node.

### ***Settings Tab (Ethernet Networks)***

The Settings tab layout is specific to the network type. This is the layout and description when an Ethernet network is configured.

For other network types, refer to:

- [Settings Tab \(Modbus Plus Networks\)](#)
- [Settings Tab \(Modbus Networks\)](#)

**Network Node**

General Settings Health Watchdog Optimizations

Device Type: Momentum

Address Map: <Modicon PLC>

IP Address  Lookup Table

IP Address: 192 . 168 . 58 . 56

Destination Index: 0

Timeout: 3 sec Retries: 3

Apply Cancel Add New Help Restore Defaults

#### *Device Type*

This field identifies the device type of this network node. You may select from the device types you configured on the network connections [Device Types Tab](#). The server uses this information to optimize network communications.

**Caution!**

If you select Auto Detect, the server uses FNC 17 (11 hex) *Report Slave ID* to determine the device type. All Modicon controllers support this function, but some third-party devices may not. For those devices, the Auto Detect will fail and the server will default to the Safe Settings optimization. Because this setting may not provide the best performance, you should choose a different device type selection for these devices.

If you use global data and you want to read or write the OPC server's global data, you must create a node of type *This Node – Global Data*. This will then allow you to create

global data type data items in the address space and connect them to the local node. For more information on global data, refer to [Appendix A: Register Addresses](#).

### Address Map

This field identifies the address map that represents the data memory layout of the physical device associated with this network node. You may select from the address maps you configured on the [Address Maps Tab](#) in the **MBX (Modicon)** folder. The server uses this information to properly build network messages. The default selection for this field is <Modicon PLC>.

### IP Address / Lookup Table

You may specify the desired node by entering its IP address or its entry in the Lookup Table. Select which you prefer here.

### Lookup Table Index

This field is available only if you chose the Lookup Table selection, and therefore is not shown in the above illustration. Select the MB+ Node value from the Ethernet MBX Driver's lookup table.

To view the lookup table, select the network connection that this node uses and go to its [Settings Tab](#). Click the Configure... button to open the MBX Driver Configuration Editor. Select the device that the network connection uses and click the Edit button. The lookup table is on the Master Path tab.

### IP Address

This field is available only if you chose the IP Address selection. Enter the node's network address in the form of a standard IP address.

### Destination Index

This field is available only if you chose the IP Address selection. Select the value you wish to use for the destination index.

### Timeout

This is the amount of time that the server will wait to receive a reply to a command message. If the server does not receive a reply within that interval, it cancels the transaction and marks it as timed out. This interval is specified in seconds.

### Retries

This is the number of times the server will reattempt each transaction that fails. The health watchdog uses this value to determine when to mark the node as unhealthy. Refer to the [Health Watchdog Tab](#) section for more information on this feature.

### ***Settings Tab (Modbus Networks)***

The Settings tab layout is specific to the network type. This is the layout and description when a Modbus network is configured.

For other network types, refer to:

- [Settings Tab \(Modbus Plus Networks\)](#)
- [Settings Tab \(Ethernet Networks\)](#)

**Network Node**

General Settings Health Watchdog Optimizations

Device Type: 584/884/984

Address Map: <Modicon PLC>

Node Address: 1

Timeout: 3 sec

Retries: 3

Apply Cancel Add New Help Restore Defaults

#### *Device Type*

This field identifies the device type of this network node. You may select from the device types you configured on the network connections [Device Types Tab](#). The server uses this information to optimize network communications.

**Caution!**

If you select Auto Detect, the server uses FNC 17 (11 hex) *Report Slave ID* to determine the device type. All Modicon controllers support this function, but some third-party devices may not. For those devices, the Auto Detect will fail and the server will default to the Safe Settings optimization. Because this setting may not provide the best performance, you should choose a different device type selection for these devices.

If you use global data and you want to read or write the OPC server's global data, you must create a node of type *This Node – Global Data*. This will then allow you to create

global data type data items in the address space and connect them to the local node. For more information on global data, refer to [Appendix A: Register Addresses](#).

### Address Map

This field identifies the address map that represents the data memory layout of the physical device associated with this network node. You may select from the address maps you configured on the [Address Maps Tab](#) in the **MBX (Modicon)** folder. The server uses this information to properly build network messages. The default selection for this field is <Modicon PLC>.

### Node Address

This is the node's Modbus node number. For more information on node addressing, refer to the Serial MBX Driver help file.

### Timeout

This is the amount of time that the server will wait to receive a reply to a command message. If the server does not receive a reply within that interval, it cancels the transaction and marks it as timed out. This interval is specified in seconds.

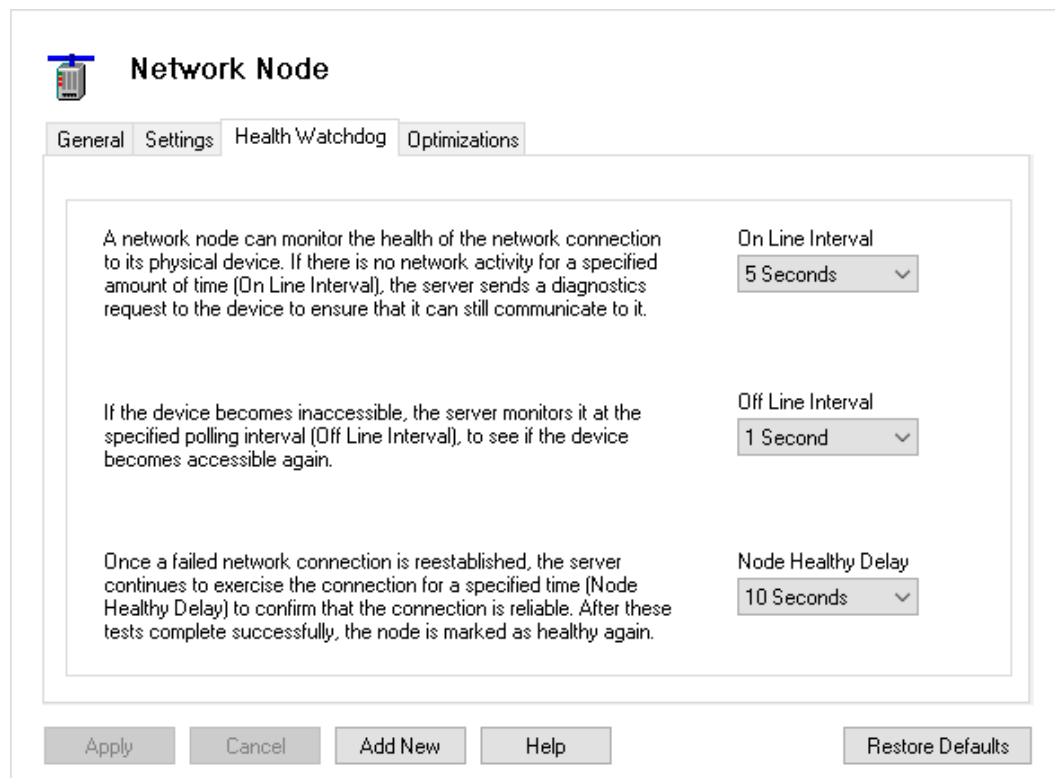
### Retries

This is the number of times the server will reattempt each transaction that fails. The health watchdog uses this value to determine when to mark the node as unhealthy. Refer to the Health Watchdog Tab section for more information on this feature.

### **Health Watchdog Tab**

Each network node monitors the health of the connection to its physical device. If there is no communication for a long time, the server sends a diagnostic request to the device to see if it can still communicate. If it cannot, the server will re-check the connection until communication is reestablished. Once a failed network connection is reestablished, the server continues to exercise the connection until it is satisfied that the connection is reliable. After this, the node is marked as healthy again.

The Health Watchdog tab allows you to configure the time intervals associated with these tests.



### On Line Interval

You may select a value in the range of 1-60 seconds or None.

If there is no traffic to a healthy node for the specified length of time, the server will send a status request to the node to verify that it is still online. Selecting *None* disables the on-line health monitoring.

If you are using a slow network, such as serial Modbus, and you are not using redundant access paths, you might want to disable health monitoring to reduce some of the network traffic. In that case, the OPC server will always report the status of the node as healthy.

### Off Line Interval

You may select a value in the range of 1-60 seconds.

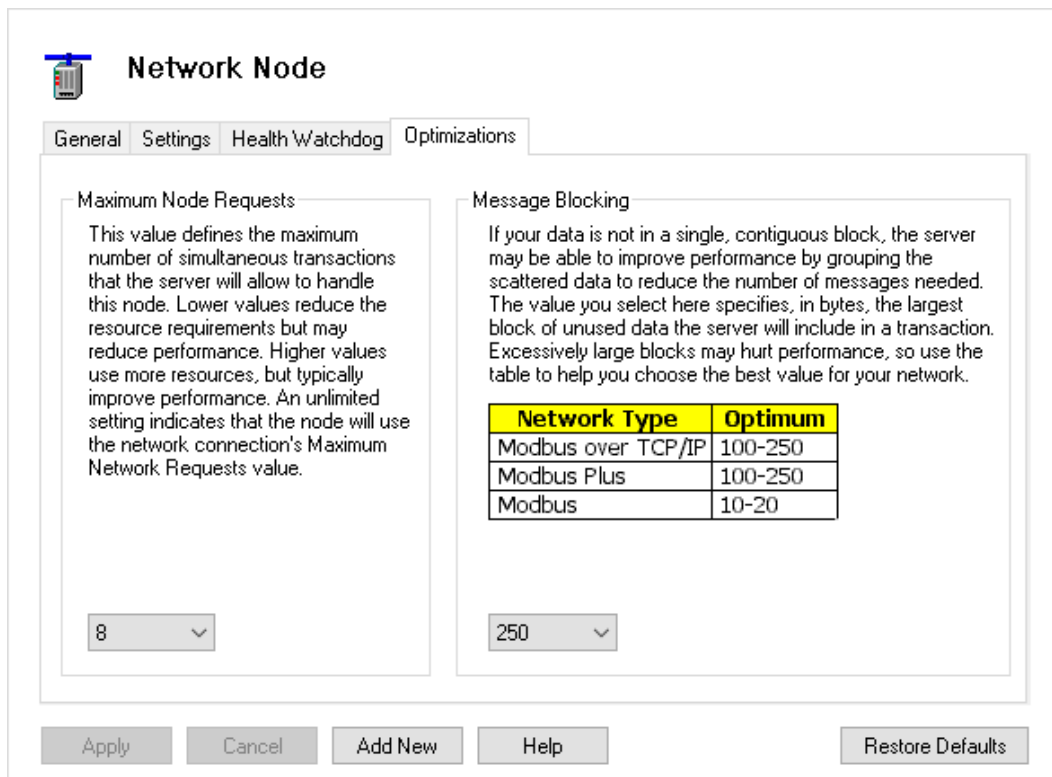
Once communication to a node has failed, the server will send status requests to the node at this interval to determine whether it can communicate.

### Node Healthy Delay

You may select a value in the range of 1-60 seconds.

Once the server reestablishes communication to an unhealthy node, it waits for the communication to stay active for this length of time before considering the node to be healthy. The server then returns the node to service.

## Optimizations Tab



### Maximum Node Requests

This value defines the maximum number of simultaneous transactions that the server will allow to handle this node. The number of concurrent requests is also limited for the network connection on its Settings Tab.

Lower values reduce the resource requirements but may reduce performance. Higher values use more resources, but typically improve performance. However, an excessively high setting may reduce performance. Setting this selection to Unlimited causes the node to use the network connection’s Maximum Network Requests value.

**Note** Setting this value to a lower number may prevent overloading the PLC with messages and allow better operation of other applications, such as PLC programming software, that access the same PLC.

Refer to [Appendix E: Configuring Maximum Concurrent Requests](#) for more information and examples.

### Message Blocking

Typically, the data you will want to access will not be in a single, contiguous block. For the server to access only the information you really require, it may need to make many small transactions. In that case, the server may obtain better performance by grouping the data into fewer, larger blocks. Although this will mean that the system will transfer



unnecessary information, the overall throughput may be improved because of the reduced overhead.

This field allows you to tell the server how to group the scattered data. The value you enter specifies, in bytes, the largest block of extra data that the server will include in a transaction. The optimum value to use will depend upon the type of network. Faster networks can handle larger gaps; slower networks will get better performance with smaller gaps.

The slowest network in the path to the network node should dictate the Message Blocking value to use. For example, if the OPC server has a 1GB Ethernet connection to a bridge, and the bridge has a 9600 baud serial connection to the network node, the Modbus range should be used.

Network Type	Optimum Range
Modbus TCP	100-250
Modbus Plus	100-250
Modbus	10-20

## Address Space

The [Address Space Tree](#) describes the hierarchical address structure of the Cyberlogic OPC Server. The branches of the tree are [Device Folders](#), [Devices](#) and [Folders](#). Its "leaves" are [Data Items](#). The intent of this structure is to permit the user to organize the data items into logical groups.

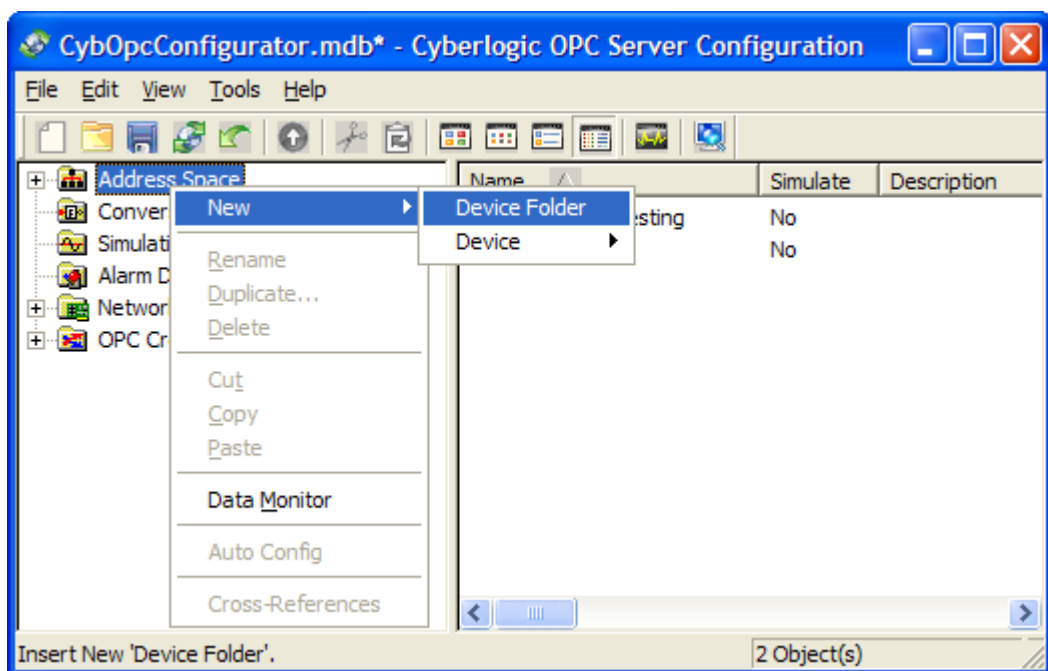
## Device Folders

A device folder groups devices and other device folders. You can place a device folder directly under the Address Space root folder or under another device folder, up to four levels deep.

### Caution!

After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

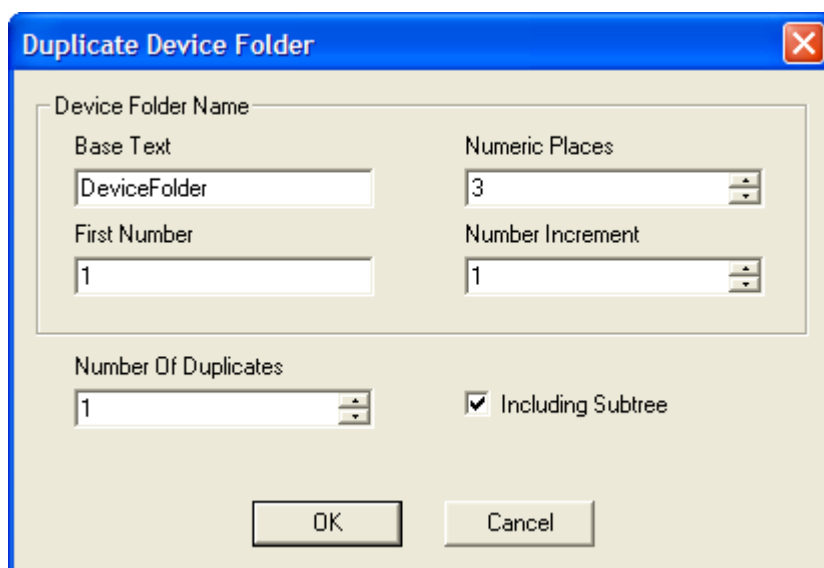
### Creating a New Device Folder



Right-click on the Address Space root folder or an existing device folder. Select **New** and then **Device Folder** from the context menu.

### Duplicating a Device Folder

To speed up the creation of similarly-configured device folders, you can create multiple device folders in a single operation by duplicating an existing one. To do this, right-click on an existing device folder and select **Duplicate...** from the context menu.



The above dialog box opens. You must specify how the duplicates are to be named by entering values for the **Base Text**, **First Number**, **Numeric Places** and **Number Increment** fields. To generate names for the duplicated device folders, the editor

begins with the base text and appends a number to it. The first duplicate uses the selected First Number value with the specified number of digits. This number is then incremented by the specified number for each of the remaining duplicates.

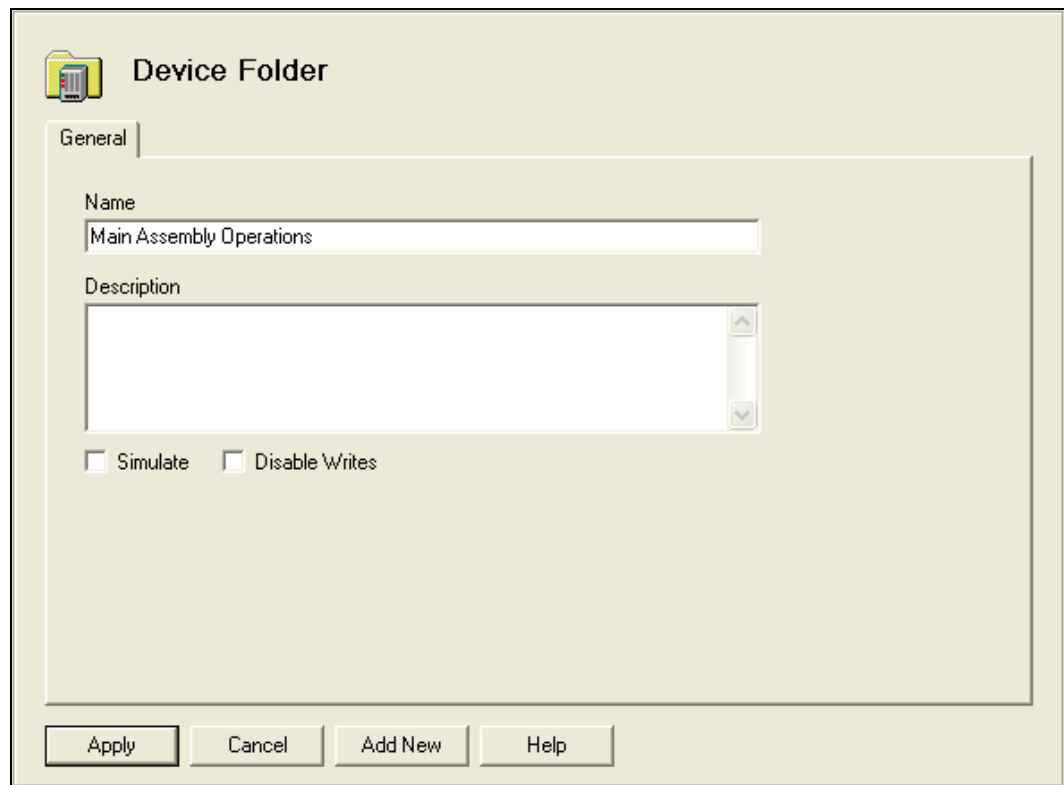
As an example, if Numeric Places is 3 and First Number is 2, the number 002 will be appended to the base text.

Use the **Number Of Duplicates** field to specify the number of device folders you wish to create. If you want to duplicate all branches within the original device folder, check the **Including Subtree** checkbox.

### ***Deleting a Device Folder***

To delete an existing device folder, select it and press the **Delete key**, or right-click on the device folder and select **Delete** from the context menu.

### ***General Tab***



### **Name**

The Name identifies this device folder. It can be up to 50 characters long, may contain spaces, but must not begin with a space. It also may not contain any periods.

### Description

This optional field further describes the device folder. It can be up to 255 characters long.

### Simulate

Checking this box enables data simulation for all data items found at this level or below. This provides a quick way to switch between real and simulated data for a large number of data items. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion about simulating data.

**Note** If the Simulate checkbox is grayed-out, it indicates that simulation has already been selected at a higher level.

### Disable Writes

Checking this box disables write requests for all data items found at this level or below. By default, this box is not checked and writes are enabled.

**Note** If the Disable Writes checkbox is grayed-out, it indicates that writes have already been disabled at a higher level.

## **Devices**

A device in the address space represents a source of data to which the server communicates. This data source may be a PLC, a group of PLCs or other physical data sources. You can place devices directly in the Address Space root folder or in a device folder. In addition to its device-specific functionality, a device operates as a folder. It can contain folders and data items.

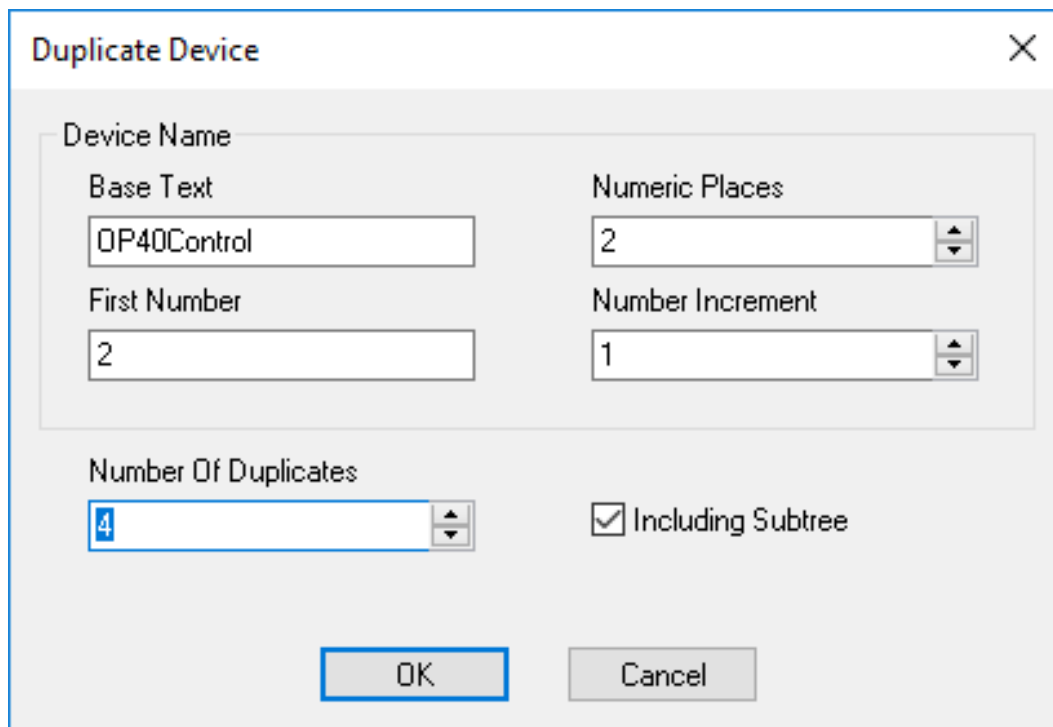
**Caution!** After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

### ***Creating a New Device***

Right-click on the Address Space root folder or an existing device folder. Select **New** and then **Device** from the context menu.

### **Duplicating a Device**

To speed up the creation of similarly-configured devices, you can create multiple devices in a single operation by duplicating an existing one. To do this, right-click on an existing device and select **Duplicate...** from the context menu.



The screenshot shows a dialog box titled "Duplicate Device" with a close button (X) in the top right corner. The dialog is divided into several sections. The top section, labeled "Device Name", contains four input fields: "Base Text" (containing "OP40Control"), "First Number" (containing "2"), "Numeric Places" (containing "2"), and "Number Increment" (containing "1"). Below this section is a "Number Of Duplicates" field (containing "4") and a checked checkbox labeled "Including Subtree". At the bottom of the dialog are two buttons: "OK" and "Cancel".

The above dialog box opens. You must specify how the duplicates are to be named by entering values for the **Base Text**, **First Number**, **Numeric Places** and **Number Increment** fields. To generate names for the duplicated devices, the editor begins with the base text and appends a number to it. The first duplicate uses the selected First Number value with the specified number of digits. This number is then incremented by the specified number for each of the remaining duplicates.

As an example, if Numeric Places is 3 and First Number is 2, the number 002 will be appended to the base text.

Use the **Number Of Duplicates** field to specify the number of devices you wish to create. If you want to duplicate all branches within the original device, check the **Including Subtree** checkbox.

### **Deleting a Device**

To delete an existing device, select it and press the **Delete key**, or right-click on the device and select **Delete** from the context menu.

**General Tab**

**MBX Device**

General Access Paths Unsolicited Message Filters

Name  
NewDevice

Description

Simulate  Disable Writes  Accept All Unsolicited

Address Map  
<Modicon PLC> Bit Order...

DirectAccess  
 Disable Writes

Apply Cancel Add New Help

**Name**

The name identifies the device. It can be up to 50 characters long, may contain spaces, but must not begin with a space. It also may not contain any periods.

**Description**

This optional field further describes the device. It can be up to 255 characters long.

**Simulate**

Checking this box enables data simulation for all data items found at this level or below. This provides a quick way to switch between real and simulated data for a large number of data items. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion about simulating data.

**Note** If the Simulate checkbox is grayed-out, it indicates that simulation has already been selected at a higher level.

**Disable Writes**

Checking this box disables write requests for all data items found at this level or below. By default, this box is not checked and writes are enabled.

**Note** If the Disable Writes checkbox is grayed-out, it indicates that writes have already been disabled a higher level.

### *Accept All Unsolicited*

When this box is checked, the server will ignore the unsolicited message filters and will accept all unsolicited messages. The slave paths that will be used for these unsolicited messages are those specified in the Default DS Paths field of the network connection's [General Tab](#).

By default, this box is not checked and unsolicited messages will be required to pass the filter criteria.

For more information on unsolicited message filters, refer to the [Unsolicited Message Filters Tab](#) section.

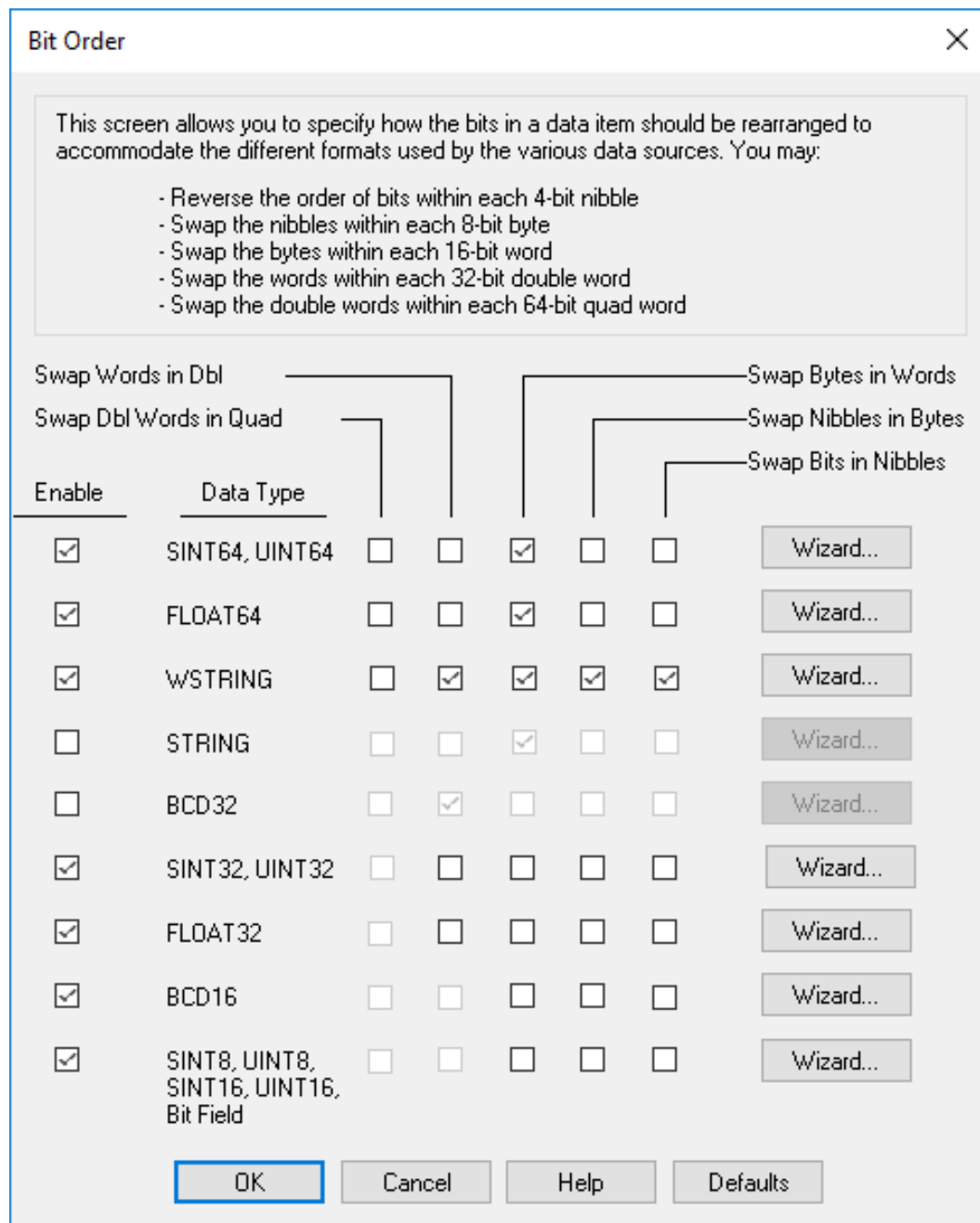
### *Address Map*

This field identifies the address map that represents the data memory layout of the physical devices associated with this MBX Device. You may select from the address maps you configured on the [Address Maps Tab](#) in the **MBX (Modicon)** folder. The OPC Server Configuration Editor uses this information to verify the syntax for the Address field on the [Data Tab](#) of a data item. The default selection for this field is <Modicon PLC>.

### *Bit Order...*

Controllers and other devices will vary in the way they arrange the bits within their data items. For example, some store string data in low byte / high byte order, while others use high byte / low byte. Some have the bits in descending order, others in ascending order. The bit order configuration allows you to tell the server how to rearrange the bits to accommodate these differences.

When you click the Bit Order... button, the Bit Order editing screen opens.



For each data type, you may rearrange groups of bits as needed by checking or unchecking the boxes. When the Enable box is checked, the specified swapping will be applied to that data type. When you uncheck this box, swapping is disabled, but the swap pattern you configured is retained so that it can be turned back on if you wish. Disabled swap patterns will be grayed out.

Each of the swapping check boxes affects the order only at one level. If you check the rightmost box, the server will reverse the order of the four bits within each nibble, but the order of the nibbles within the bytes, words, etc. will not be affected. Checking the second box from the right will swap the order of the nibbles within each 8-bit byte, but



will not affect the order of the bits within the nibbles, nor will it affect the order of the bytes within each word and so on.

You may check and uncheck the boxes as needed to properly arrange the data. Click the **Defaults** button to return the configuration to the default setting that works for most devices.

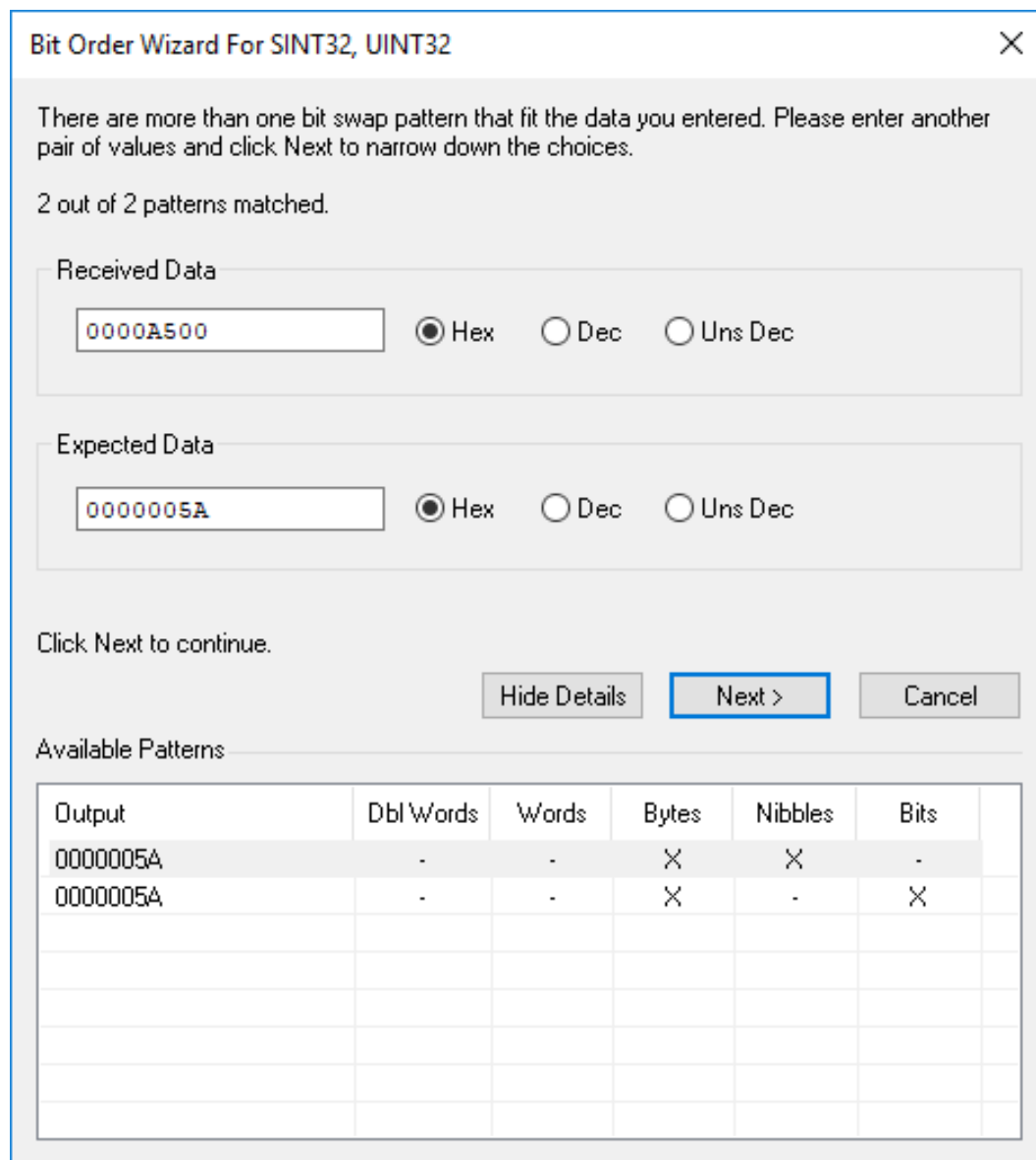
**Note**

A common requirement is to completely reverse the order of all of the bits in a data item. To do this, you must check all of the available boxes for that type.

If you need help in determining the correct swap pattern, click **Wizard...**

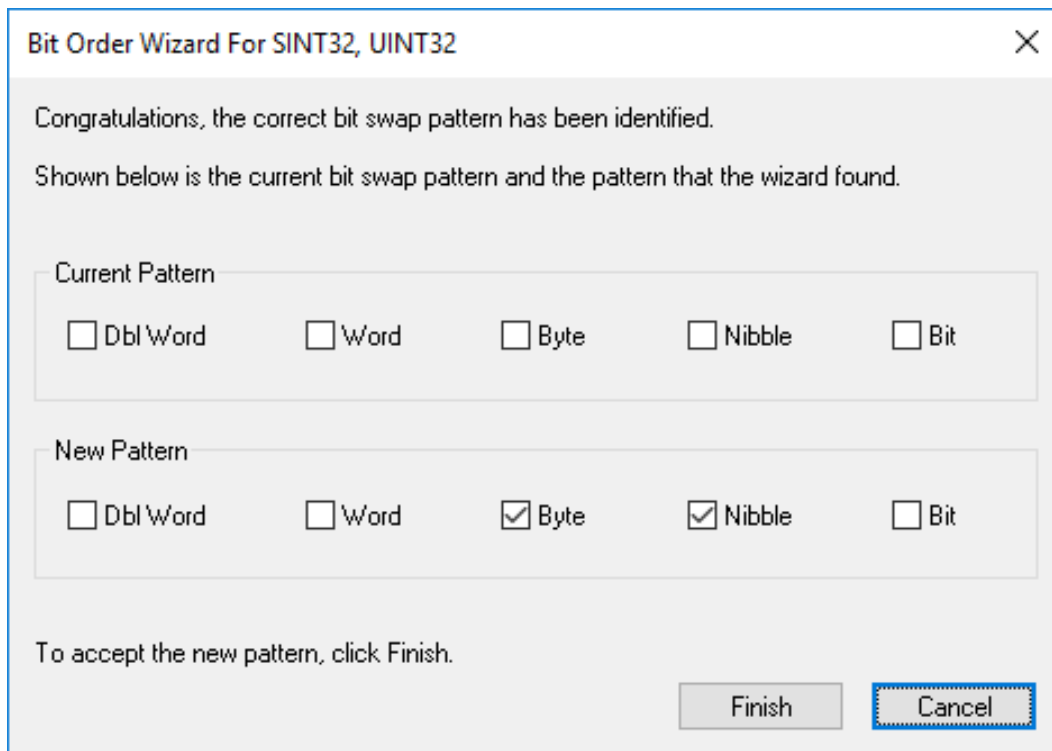
The wizard allows you to enter the data as you see it and as you expected to see it, that is, what you have and what you want. You are given a choice of formats for the data entry. The choices available will depend on the data type you are configuring and include hex, decimal, unsigned decimal, ASCII, Unicode, BCD and floating point.

In the example shown, you see A500, but expected to see 005A. After entering these values, click **Next**. The wizard will check all swap patterns to try to identify which ones will give you 005A.



In this case, it found two swap patterns will work for the given values. Click the **Show/Hide Details** button to open or close the Available Patterns display, which shows you the bit swap patterns that the wizard has identified.

You must identify one specific swap pattern to use, so you must enter another pair of **Received Data** and **Expected Data** values. This time you enter 1600 as the data you see and 0061 as what you expected. Now click **Next**. The wizard will test only the swap patterns that passed the first step to see if they match the current data. By doing this, it identifies those patterns that match both of the data pairs you entered.

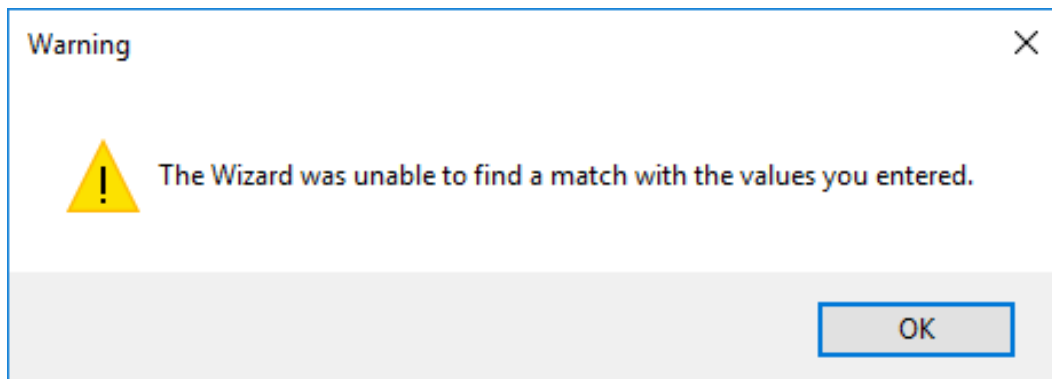


The result screen shows that the wizard has now narrowed down the possibilities to just one. Click **Finish** to exit from the wizard and use this swap pattern.

**Note**

If there had been more than one pattern that matched both sets of data, you would have been returned to the previous screen and asked to provide a third pair of values. This process would continue until the wizard identified just one bit pattern that matched all of the data pairs you entered, after which you would reach this screen.

If none of the possible bit swap patterns match your data, the wizard will notify you.



Click **OK** to continue.

Bit Order Wizard For SINT32, UINT32

There is no bit swap pattern that fits the data you entered. Please enter a different pair of values and click Next to repeat the search.

0 out of 16 patterns matched.

Received Data

1  Hex  Dec  Uns Dec

Expected Data

20  Hex  Dec  Uns Dec

Click Next to continue.

Show Details Next > Cancel

When you enter a new pair of values, the search process starts over and the earlier values are ignored.

#### DirectAccess

If this box is checked, the user is permitted to configure [DirectAccess](#) to the nodes associated with this device. The default state is checked, allowing DirectAccess.

When DirectAccess is enabled, a `_Status` folder will appear under this device in the client browser window. For more information on this folder and the status items it contains, refer to the [Cyberlogic OPC Server Help](#).

#### DirectAccess Disable Writes

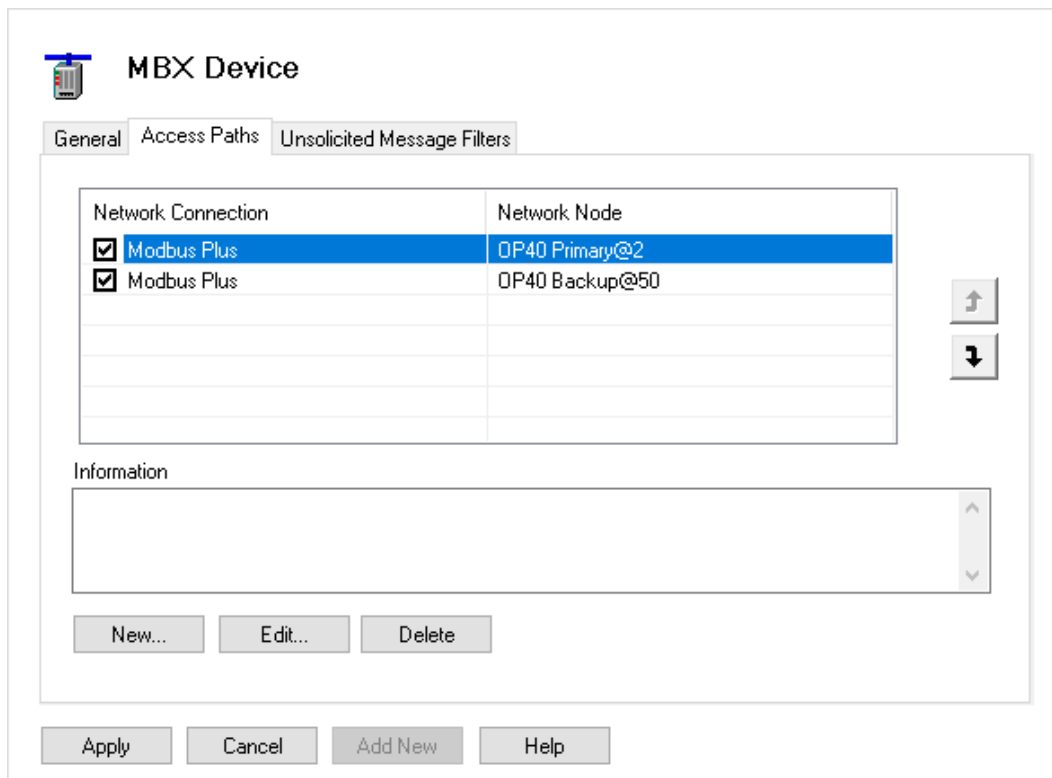
If this box is checked, the server will not write data via DirectAccess to any of the nodes associated with this device. This does not affect writes through configured data items. The default state is checked, disabling DirectAccess writes.

#### **Access Paths Tab**

Each device has an associated list of access paths. Depending upon your network configuration, the access paths may include multiple paths to the same PLC, paths to different PLCs or some combination of the two. The access path at the top of the list is the primary access path; the rest are backups. If the current access path fails or is disabled, the server switches to the highest access path that is available and enabled.

**Note**

Access paths are required only for solicited communications. If you are planning to use only unsolicited data updates with unsolicited message filters, no access paths need to be configured.



Enable Checkbox

To the left of each access path is a checkbox that, when checked, enables the access path. The server uses only enabled access paths.

Network Connection

The Network Connection column displays the network connection associated with each of the access paths.

Network Node

The Network Node column displays the network node associated with each of the access paths.

Priority (Up and Down) Arrows

The server assigns priority to the access paths from top to bottom, with the access path at the top having the highest priority. Use the up and down arrows on the right side of the list box to adjust the priorities as needed.

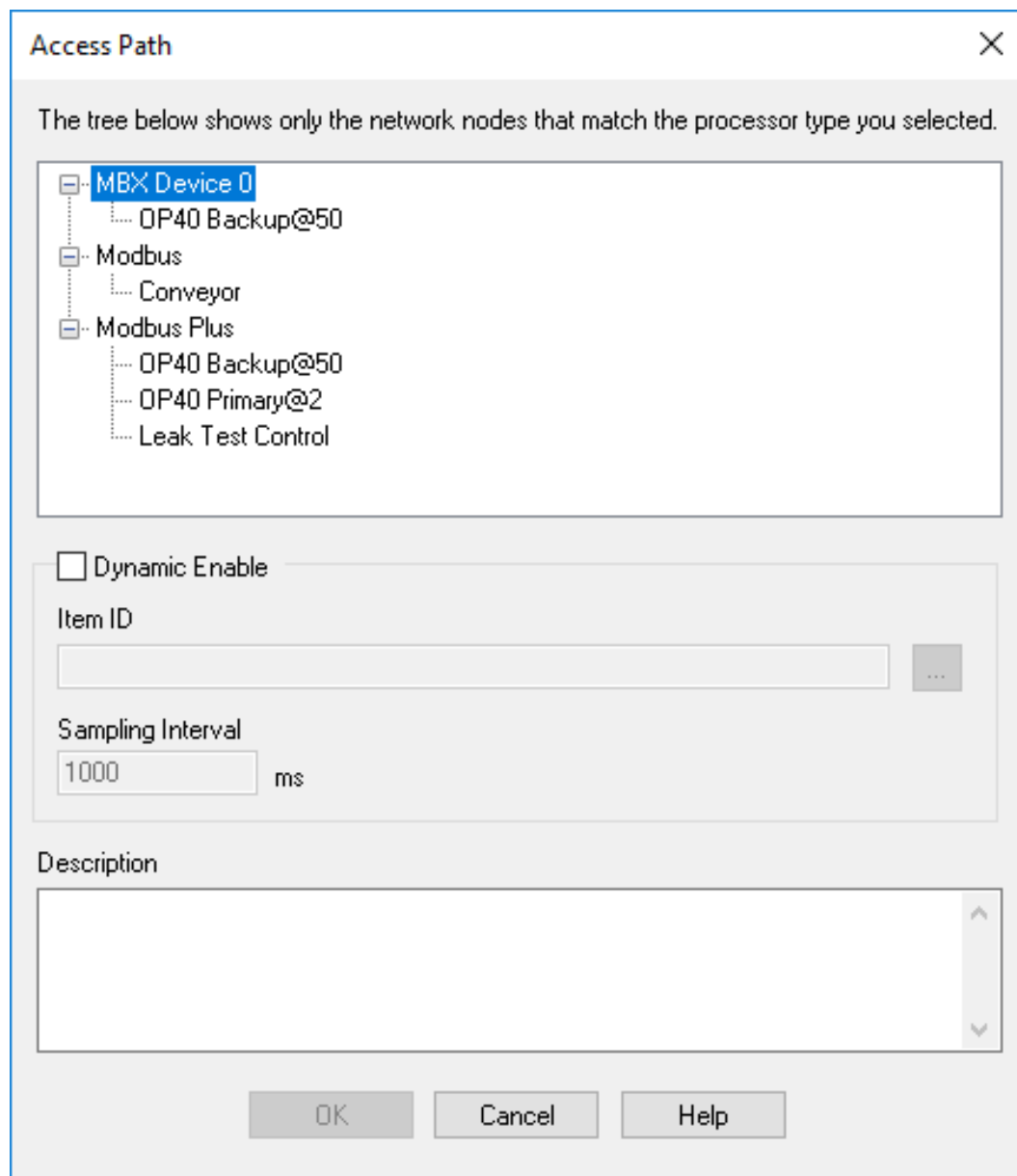
Information

If dynamic enable is configured for the highlighted access path, the enable data item will be shown here. In addition, if a description was entered for the access path, it will be displayed here.

This information is edited on the Access Path dialog box, described below.

New...

Click the **New...** button (or right-click inside the list window and select **New...** from the context menu) to create a new access path. The Access Path dialog box opens.



Select the network node for this access path.

If you check the Dynamic Enable box, you can specify an Item ID that will be used to control the enable status of the access path. Enter a data item or DirectAccess item ID in the box, or click the browse button to the right of it to browse for the desired item. You can use a Math & Logic item, if you wish. The Sampling Interval allows you to specify how often the enable item should be checked.

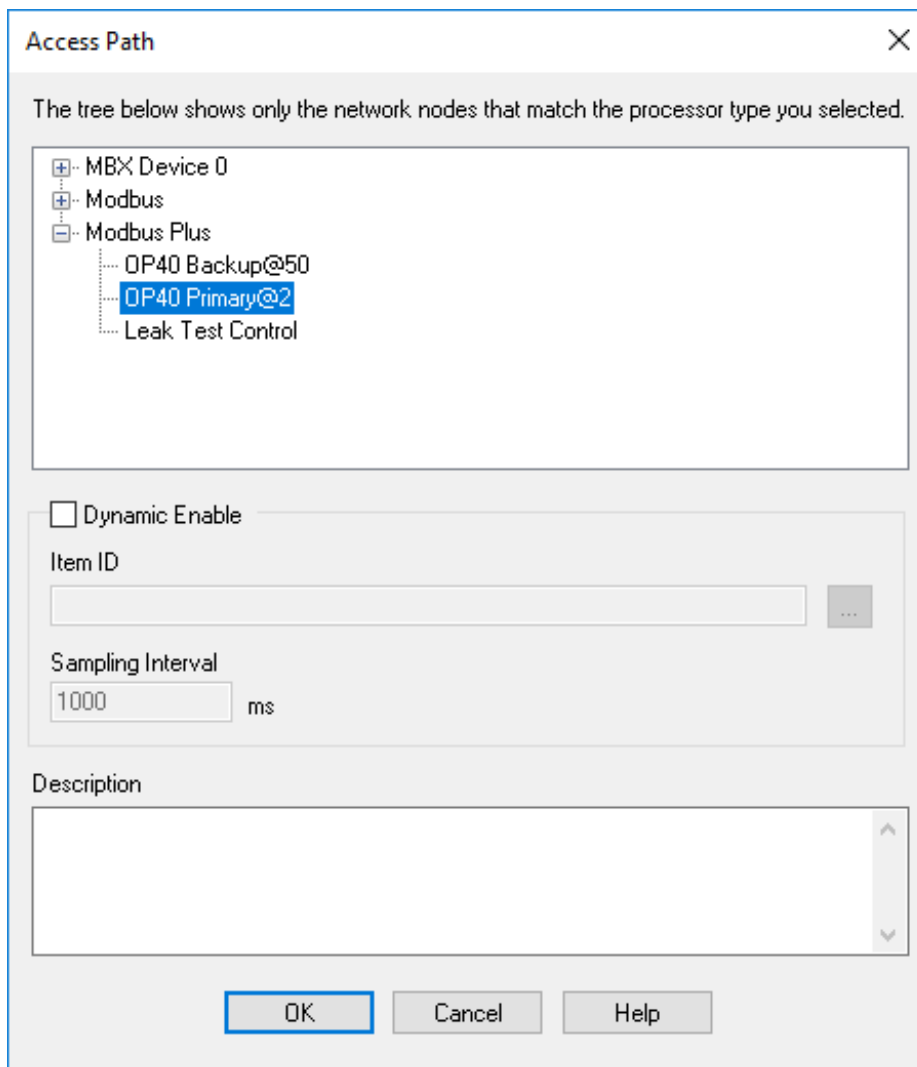
If the value of the enable item is false (Boolean false or a register value of 0), then the access path will be disabled. If the value is true (Boolean true or a nonzero register value), then the access path will be enabled.

You can also enter an optional description text of up to 255 characters.

Click **OK** when you are done.

Edit...

To modify an existing access path, select it and click the **Edit...** button (or right-click on the access path and select **Edit...** from the context menu). The Access Path dialog box opens.



Modify the current selections and click **OK** when you are done.

Delete

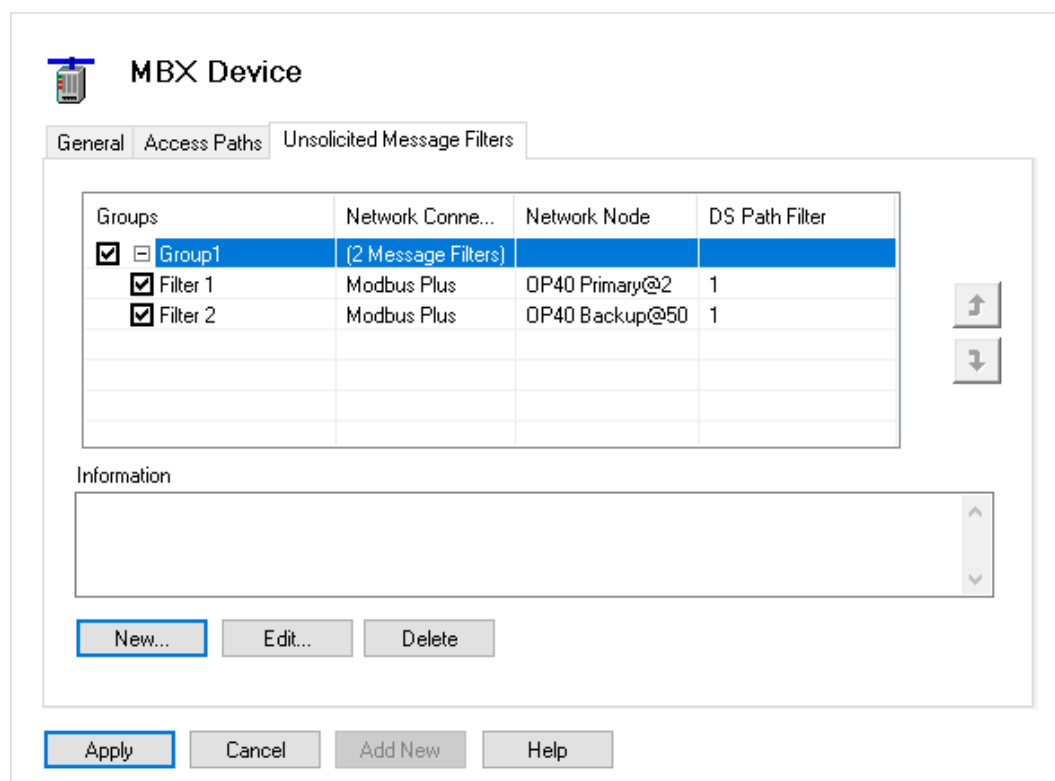
To delete an existing access path, select it and click the **Delete** button (or right-click and select **Delete** from the context menu).

**Unsolicited Message Filters Tab**

If the Accept All Unsolicited box on the General tab is checked, the server will accept all unsolicited messages. Otherwise, unsolicited messages must pass the filter criteria to be accepted. These filters help ensure that unsolicited messages are accepted only from trusted sources. For information on how to program your programmable controller for unsolicited communication, refer to [Appendix D: Unsolicited Message Programming](#).

The [Unsolicited Message Filters](#) are organized in groups. Each group has an equal priority, and a message must pass through at least one of these groups in order to be accepted by the server.

An unsolicited message filter group has a list of trusted network nodes and/or trusted network connections, and supports two modes of operation. In the default non-priority mode, the server will accept messages that pass any of the configured filters. In the alternative, priority mode, the server treats the filter list as a ranked list of preferred and backup data sources. It monitors the connections to each unsolicited message source and accepts messages only from the highest ranked node that has a healthy connection.





### Group

You can arrange the filters in one or more groups. Each group can be configured independently as either a prioritized or a non-prioritized list. There is no implied priority from one group to another.

### Enable Checkbox

To the left of each unsolicited message group or filter is a checkbox that, when checked, enables that group/filter. The server uses only the enabled groups and filters.

### Network Connection

The Network Connection column displays the network connection associated with the filter.

### Network Node

The Network Node column displays the network node associated with the filter.

### Priority (Up and Down) Arrows

The server assigns priority to the filters in a group from top to bottom, with the filter at the top having the highest priority. Use the up and down arrows on the right side of the list box to adjust the priorities as needed.

### New...

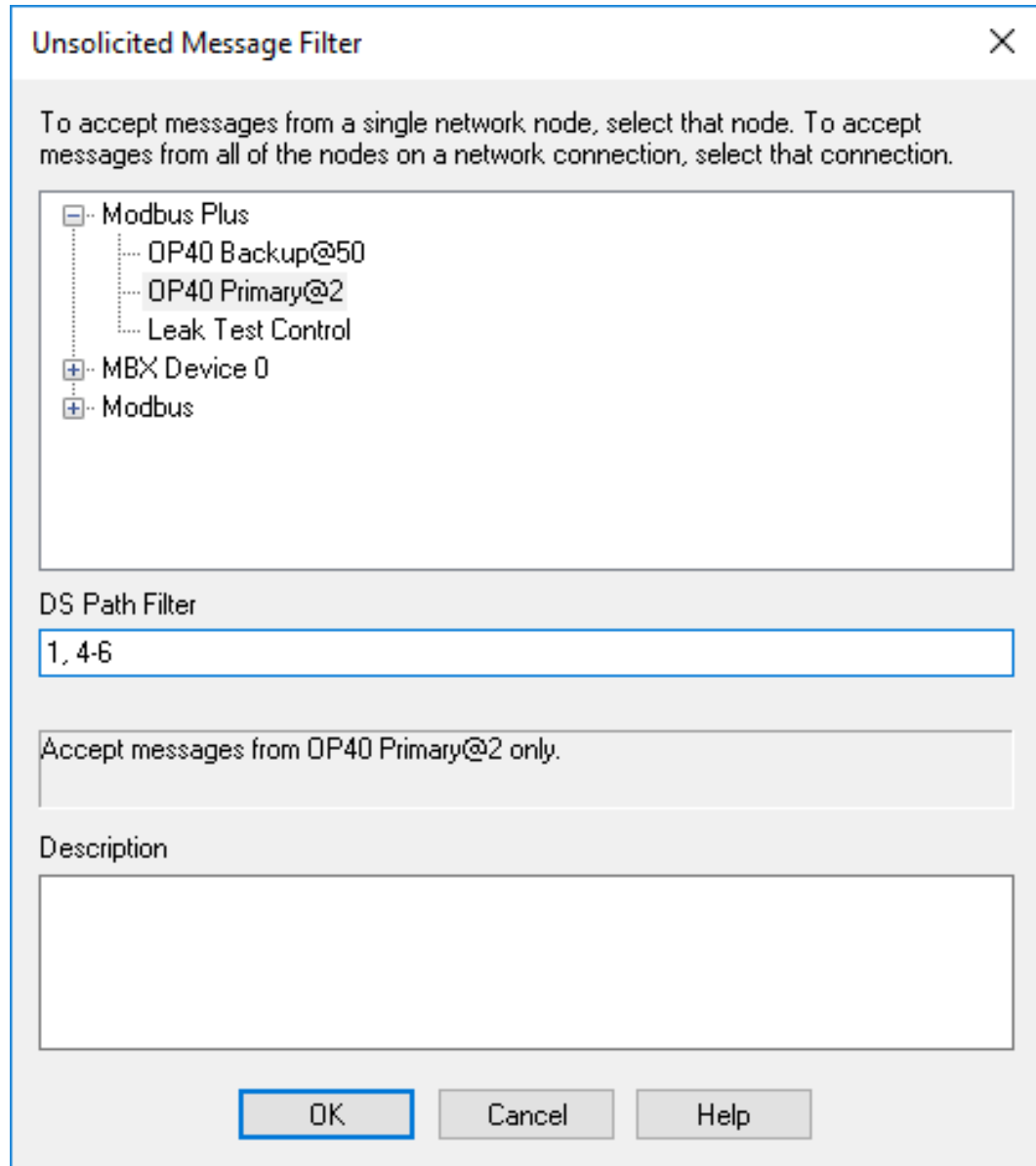
To create a new group, click the **New...** button and select **Group...** (or right-click inside the list window and select **New** then **Group...** from the context menu). The Unsolicited Message Filter Group dialog will open.

The image shows a dialog box titled "Unsolicited Message Filter Group". It has a standard Windows-style title bar with a close button (X) in the top right corner. The dialog contains the following elements:

- A "Name" label followed by a text input field containing the text "Group A".
- A "Description" label followed by a large, empty text area.
- A checkbox labeled "Priority Unsolicited" which is currently unchecked.
- At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Help". The "OK" button is highlighted with a blue border.

Enter the **Name** of the group and an optional **Description**, if desired. If you want this to be a prioritized list of filters, check the **Priority Unsolicited** box. Click **OK** when you are done.

To create a new filter, select the group in which you wish to create the filter. Click the **New...** button and select **Filter...** (or right-click on the desired group and select **New...** then **Filter...** from the context menu). The Unsolicited Message Filter dialog will open.



Select the network connection or network node you wish to use. You may also specify the Data Slave paths that the messages may use, listing these in the **DS Path Filter** field. Finally, you may enter an optional **Description**, if desired. Click the **OK** button when you are done.

The DS Path Filter field can contain a single Data Slave (DS) path number, a range of DS path numbers or a combination of both, separated by commas. The table shows some slave path filter examples.

Slave Path Filter	Slave Paths Used
1	DS1
2,4	DS2, DS4
1-4	DS1, DS2, DS3, DS4
1,3-5,7	DS1, DS3, DS4, DS5, DS7

The same DS path may be used in multiple unsolicited message filters.

*Edit...*

Select an existing group or filter and click the **Edit...** button (or right-click on an existing group or filter, and select **Edit...** from the context menu). The appropriate dialog box will. Modify the current selections and click **OK** when you are done.

*Delete*

Select an existing group or filter and click the **Delete** button (or right-click on the group or filter, and select **Delete** from the context menu).

## Folders

A folder logically groups data items and other folders. You can place folders directly under devices or under other folders, up to four levels deep.

**Caution!**

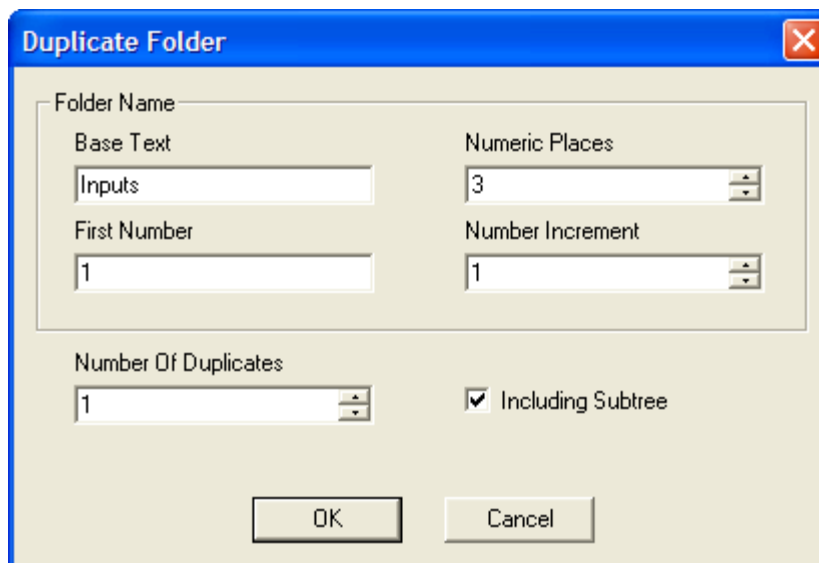
After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

### ***Creating a New Folder***

Right-click on an existing device or folder, and select **New** and then **Folder** from the context menu.

### ***Duplicating a Folder***

To speed up the creation of similarly-configured folders, you can create multiple folders in a single operation by duplicating an existing one. To do this, right-click on an existing folder and select **Duplicate...** from the context menu.



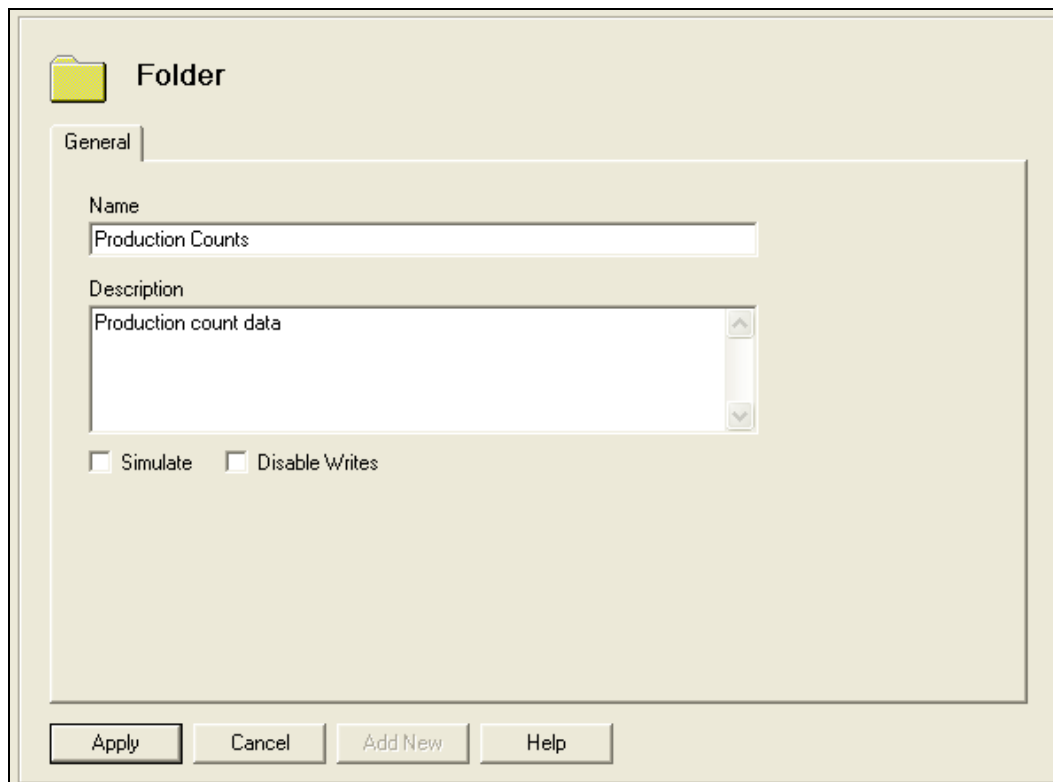
The above dialog box opens. You must specify how the duplicates are to be named by entering values for the **Base Text**, **First Number**, **Numeric Places** and **Number Increment** fields. To generate names for the duplicated folders, the editor begins with the base text and appends a number to it. The first duplicate uses the selected First Number value with the specified number of digits. This number is then incremented by the specified number for each of the remaining duplicates.

As an example, if Numeric Places is 3 and First Number is 2, the number 002 will be appended to the base text.

Use the **Number Of Duplicates** field to specify the number of folders you wish to create. If you want to duplicate all branches within the original folder, check the **Including Subtree** checkbox.

### ***Deleting a Folder***

To delete an existing folder, select it and press the **Delete key**, or right-click on the folder and select **Delete** from the context menu.

**General Tab**

The screenshot shows a 'Folder' dialog box with the following fields and controls:

- Name:** Production Counts
- Description:** Production count data
- Simulate
- Disable Writes
- Buttons: Apply, Cancel, Add New, Help

**Name**

The Name identifies this folder. It can be up to 50 characters long, may contain spaces, but must not begin with a space. It also may not contain any periods.

**Description**

This optional field further describes the folder. It can be up to 255 characters long.

**Simulate**

Checking this box enables data simulation for all data items found at this level or below. This provides a quick way to switch between real and simulated data for a large number of data items. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion about simulating data.

**Note** If the Simulate checkbox is grayed-out, it indicates that simulation has already been selected at a higher level.

### Disable Writes

Checking this box disables write requests for all data items found at this level or below. By default, this box is not checked and writes are enabled.

**Note**

If the Disable Writes checkbox is grayed-out, it indicates that writes have already been disabled a higher level.

## Data Items

A data item represents a register in the physical device, a range of registers, a bit inside a register or a range of bits. Six types of registers are supported: coils, inputs, input registers, holding registers, general registers and global data. The MBX Driver Agent supports all register address notations used by various Modicon products. It also supports the addresses as defined by the custom address maps (see [Address Maps tab](#)). Coils, holding registers and general registers are read/write. Inputs, input registers and bits within registers are read-only.

**Caution!**

After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

### ***Creating a New Data Item***

Right-click on an existing device or folder, and select **New** and then **Data Item** from the context menu.

### ***Duplicating a Data Item***

To speed up the creation of similarly-configured data items, you can easily create multiple data items by duplicating an existing one. To do this, right-click on an existing data item and select **Duplicate...** from the context menu. The Data Item Duplication Wizard will open and will guide you through the duplication process. For details of this operation, refer to [Appendix B: Data Item Duplication Wizard](#).

### ***Deleting a Data Item***

To delete an existing data item, select it and press the **Delete key**, or right-click on the data item and select **Delete** from the context menu.

### General Tab

**Data Item**

General | Data | Simulation | Alarms | Properties

Name  
Good Parts

Description

Simulate    Disable Writes

Data Updates

Solicited Update    Unsolicited Update  
 Unsolicited Late Interval

00:00:00 hh:mm:ss

Apply   Cancel   Add New   Help

#### Name

The Name identifies the data item. It can be up to 50 characters long, may contain spaces, but must not begin with a space. It also may not contain any periods.

#### Description

This optional field further describes the data item. It can be up to 255 characters long.

#### Simulate

Checking this box enables data simulation for this data item. Refer to the [Cyberlogic OPC Server Help](#) for a full discussion about simulating data.

**Note** If the Simulate checkbox is grayed-out, it indicates that simulation has already been selected at a higher level.

#### Disable Writes

Checking this box disables all write requests for this data item. By default, this box is not checked and writes are enabled.

**Note**

If the Disable Writes checkbox is grayed-out, it indicates that writes have already been disabled a higher level.

*Solicited Update*

When checked, this box tells the server to update this data item by polling. The server will use the access paths specified in this item's parent device to retrieve that data.

*Unsolicited Update*

When checked, this box tells the server to allow this data item to be updated through unsolicited messages. The server will use the unsolicited message filters specified in this item's parent device to decide which unsolicited messages will be allowed to modify this value. This box is unchecked by default, disabling unsolicited updates.

**Note**

Normally, you will not want to use both solicited and unsolicited updates at the same time on the same data item. Therefore, you should check one of these boxes and clear the other.

*Unsolicited Late Interval*

This is the maximum interval at which the server will expect to receive unsolicited updates for this data item. If the server does not receive an unsolicited update for this data item within this interval, the item's quality is downgraded to *Uncertain*.



**Data Tab**
Address

This is the address of the data in the physical device.

In general, all addresses, as documented in Modicon manuals, are acceptable. Also, the addresses as defined by the custom address maps (see [Address Maps tab](#)) are allowed, too. In addition, the MBX Driver Agent extends this syntax when it comes to specifying arrays, array elements, register bits and bit fields. For a complete list of all valid register addresses, refer to [Appendix A: Register Addresses](#). The allowed syntax for this field depends on the Address Map setting on the [MBX Device](#) associated with this data item.

Wizard...

If you have difficulties with specifying valid addresses, click this button and the Address Wizard will help you to define the address correctly.

For details on running the wizard, refer to [Appendix C: Address Wizard](#).

Data Type

This drop-down allows you to select the native type of the data as it is found in the physical device. This tells the server how many bytes of data to read and write for this data item. Not all data types are allowed for all register types.

Normally, you will set the type to Default, which selects the usual data format for that type of register. However, in unusual applications where the data is stored in the physical device in a non-standard way, you can specify a different format to simplify the access to the data.

### String Length

If the data type is a string format, this field specifies the number of characters in the string. Remember that Unicode strings require two bytes per character.

### Span Messages

This check box is available only for strings, arrays and bit fields.

If the box is not checked, the server will deliver all of the data for this data item in a single message.

If the box is checked, the server may take more than a single message to obtain all data for this item. This may lead to data tearing if the data changes between the messages.

### **Caution!**

You must not disable span messages for arrays that are larger than the maximum size allowed for a single message. Doing so will cause communication errors. Refer to the Device Type settings for your controller to determine the message size limits for each array type.

For example, suppose you are using a Micro 84 controller and have an array of fifty 4xxx registers. A Micro 84 can transfer only 32 of these registers in a single message. Therefore, the array cannot fit into a single message. You must enable span messages for this array.

### Bit Order Override

The [Bit Order](#) feature of the MBX OPC Server allows you to control how the bits are arranged within a data item. Each of the configured devices can have its own default bit order configuration, but you can override the default for a data item by checking this box.

When the box is checked, some or all of the check boxes and the Wizard... button will be activated. Check boxes that are not activated do not apply to data of the type used for this data item.

If you know the bit-ordering configuration you want to use, you may set it by checking the appropriate boxes.

- **Dbi Word** swaps the 32-bit double words within each 64-bit quad word.
- **Word** swaps the 16-bit words within each 32-bit double word.
- **Byte** swaps the 8-bit bytes within each 16-bit word.
- **Nibble** swaps the 4-bit nibbles within each 8-bit byte.
- **Bit** reverses the order of the bits within each 4-bit nibble.

If you are unsure of the pattern to use, click **Wizard...**, which will help you find the right pattern by comparing the data you see with the data you expected to see.

For more information about setting the bit-ordering pattern and using the wizard, refer to the [Address Map](#)

This field identifies the address map that represents the data memory layout of the physical devices associated with this MBX Device. You may select from the address maps you configured on the Address Maps Tab in the **MBX (Modicon)** folder. The OPC Server Configuration Editor uses this information to verify the syntax for the Address field on the [Data Tab](#) of a data item. The default selection for this field is <Modicon PLC>.

Bit Order... discussion in the Devices section.

### Canonical Data Type

This selection specifies the default variant format (VT\_XXX) in which the data will be sent to OPC clients. Each OPC client can request data in any variant format and consequently override the Canonical Data Type setting.

The most common selection is Default, in which case the server matches the variant type to the item's data type. The following table shows the default mappings.

Type	Size in bits	Default Canonical Data Type	.NET Data Type	Description
Default				Default type based on the data item address
BIT	1	VT_BOOL	bool	1-bit boolean
SINT8	8	VT_I1	sbyte	Signed 8-bit integer -128 to 127
UINT8	8	VT_UI1	byte	Unsigned 8-bit integer 0 to 255
SINT16	16	VT_I2	short	Signed 16-bit integer -32,768 to 32,767
UINT16	16	VT_UI2	ushort	Unsigned 16-bit integer 0 to 65,535
SINT32	32	VT_I4	int	Signed 32-bit integer -2,147,483,648 to 2,147,483,647
UINT32	32	VT_UI4	uint	Unsigned 32-bit integer 0 to 4,294,967,295
SINT64	64	VT_I8	long	Signed 64-bit integer -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
UINT64	64	VT_UI8	ulong	Unsigned 64-bit integer 0 to 18,446,744,073,709,551,615
FLOAT32	32	VT_R4	float	IEEE 32-bit floating point number $\pm 3.4e\pm 38$
FLOAT64	64	VT_R8	double	IEEE 64-bit floating point number $\pm 1.7e\pm 308$
BCD16	16	VT_UI2	ushort	BCD value; 0 to 9,999
BCD32	32	VT_UI4	uint	BCD value; 0 to 99,999,999
STRING	String size * 8	VT_BSTR	string	Zero terminated ASCII string of 8-bit characters
WSTRING	String size * 16	VT_BSTR	string	Zero terminated UNICODE string of 16-bit characters
FIELD	Field size	Best fitting VT_UIx or array of VT_UI1 if size > 64	Best fitting unsigned type or byte[ ] if size > 64	Multiple bit field

### Use Conversion

Check this box to indicate that the selected conversion should be applied to the data before the value is stored in the data item cache. You must select one of the previously-configured [Conversions](#) from the drop-down box.

### **Simulation Tab**

The screenshot shows a dialog box titled "Data Item" with a tabbed interface. The "Simulation" tab is selected. The dialog contains a text box with the following text: "Select the signal you wish to use when the value for this Data Item is simulated. You may choose one of the simulation signals you configured, a Fixed Value, or an Echo function that simply returns the last value written to the Data Item." Below this text, there is a "Signal" label and a dropdown menu currently set to "Fixed Value". Underneath, there is a "Value" label and a text input field containing the number "221". At the bottom of the dialog, there are four buttons: "Apply", "Cancel", "Add New", and "Help".

### Signal

If you enabled simulation on the General tab or at a higher level, you may choose to simulate the data item value with one of the previously-defined [Simulation Signals](#), a fixed value or an echo of the last value written to the item.

### Value

When simulation is enabled and the Signal field is set to Fixed Value, the data item will be set to this value.

## Alarms Tab

**Data Item**

123

General | Data | Simulation | **Alarms** | Properties

Generate Alarms

To generate alarms for this Data Item, you must first configure an appropriate limit or digital alarm. Normally, you should use a limit alarm for analog data and a digital alarm for Boolean data. You can then select that alarm below and specify a prefix to the alarm messages that are generated. The prefix text identifies the data item associated with the alarm.

Message Prefix

Alarm

<Not Assigned>

Apply Cancel Add New Help

### Generate Alarms

If this box is checked, the server will test the alarm conditions for this data item, generating alarms as appropriate.

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion on creating and using alarms.

### Message Prefix

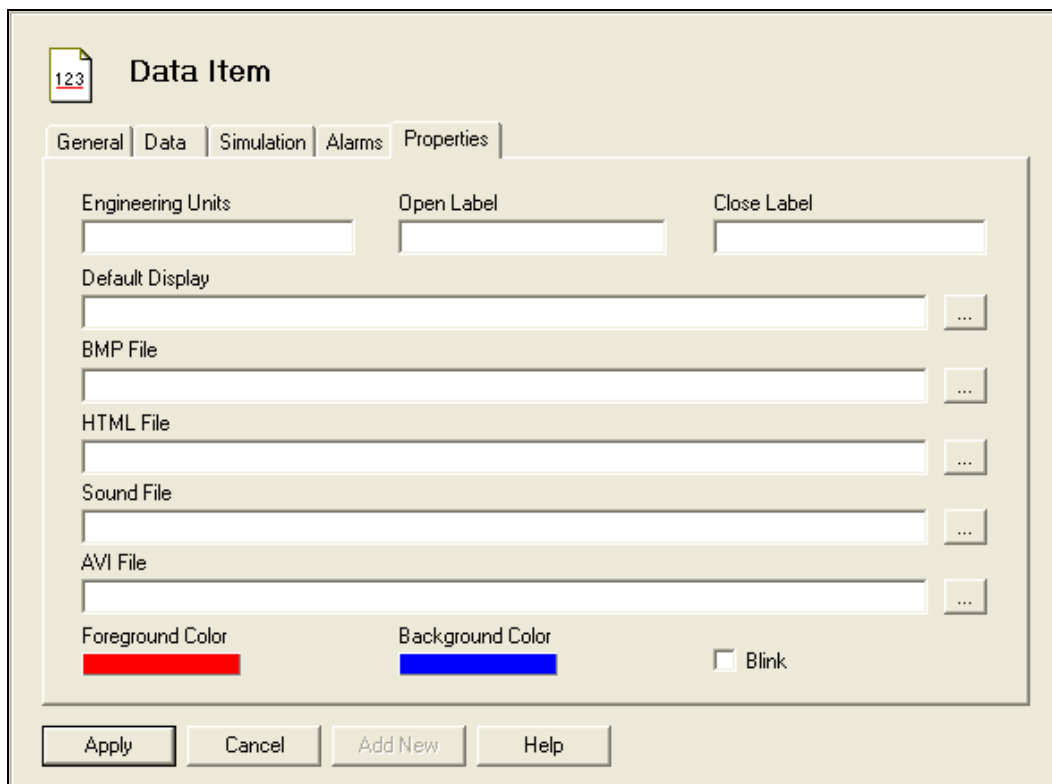
Enter the text for the first part of the alarm message. The second part will be the body text of the specific alarm that is generated.

### Alarm

Select one of the previously-defined [Alarm Definitions](#) to serve as the alarm template for this data item.

## Properties Tab

In addition to the main data item properties—value, quality and timestamp—the OPC specification includes several optional properties that your client application may use. This tab allows you to set these data item properties. These properties are static and do not change while the server is running.



Engineering Units

This is OPC property ID 100. It specifies the engineering units text, such as DEGC or GALLONS. It can be up to 50 characters long.

Open Label

This is OPC property ID 107, and is presented only for discrete data. This text describes the contact when it is in the open (zero) state, such as STOP, OPEN, DISABLE or UNSAFE. It can be up to 50 characters long.

Close Label

This is OPC property ID 106, and is presented only for discrete data. This text describes the contact when it is in the closed (non-zero) state, such as RUN, CLOSE, ENABLE or SAFE. It can be up to 50 characters long.

Default Display

This is OPC property ID 200. It is the name of an operator display associated with this data item. It can be up to 255 characters long.

### BMP File

This is OPC property ID 204. It is the name of a bitmap file associated with this data item, for example C:\MEDIA\FIC101.BMP. It can be up to 255 characters long.

### HTML File

This is OPC property ID 206. It is the name of the HTML file associated with this data item, for example http://mypage.com/FIC101.HTML. It can be up to 255 characters long.

### Sound File

This is OPC property ID 205. It is the name of the sound file associated with this data item, for example C:\MEDIA\FIC101.WAV. It can be up to 255 characters long.

### AVI File

This is OPC property ID 207. It is the name of the AVI file associated with this data item, for example C:\MEDIA\FIC101.AVI. It can be up to 255 characters long.

### Foreground Color

This is OPC property ID 201. Click on the box and select the foreground color used to display the item.

### Background Color

This is OPC property ID 202. Click on the box and select the background color used to display the item.

### Blink

This is OPC property ID 203. Check this box to indicate that displays of the item should blink.

## **Conversions**

The raw data associated with data items may be process values from instruments. In most cases, these measurements are not expressed in engineering units. To simplify operations on the data, the Cyberlogic OPC Server allows you to associate a conversion with each data item.

A user can define many different conversions. A number of data items can then use each conversion. As a result, the user need not define the same conversion many times over.

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion.



## Simulation Signals

The server can simulate the data for each of the data items according to a predefined formula. This makes it easy to perform client-side testing without the need for a physical device.

A user can define many different types of simulation signals. A number of data items can then use each such signal. As a result, the user need not define the same simulation signal many times over.

The server can generate the following types of simulation signals:

- Read count
- Write count
- Random
- Ramp
- Sine
- Square
- Triangle
- Step

Each signal has parameters that define properties such as amplitude, phase and number of steps.

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion.

## Alarm Definitions

The Cyberlogic OPC Server supports the OPC Alarms and Events specification, permitting it to generate alarms based on the value of data items.

The user may define many different alarm conditions. A number of data items can then use each such condition. As a result, the user need not define the same alarm condition many times over.

There are two categories of alarms: digital and limit. Digital alarms are normally used with Boolean data items and limit alarms are normally used with numeric data items, but both types of alarms may be used with either data type. Alarms cannot be used with string or array data items or with bit fields larger than 64 bits.

Refer to the [Cyberlogic OPC Server Help](#) for a full discussion.

**Note**

Configuring alarms is meaningful only if your client software also supports the OPC Alarms & Events specification. Consult your client software documentation to see what specifications it supports.

## Database Operations

In addition to providing data to OPC clients in real time, the Cyberlogic OPC Server can store it in a database. The feature that does this is called Data Logger. Once the data is logged, it is available to any application that can access that database. It need not be an OPC client application.

Refer to the [Data Logger Help](#) for a full discussion.

## OPC Crosslinks

OPC Crosslinks allow you to transfer data from an OPC server or PLC to other OPC servers or PLCs. The data item you read from is called the crosslink input. You may write its value to any number of data items, and these are called crosslink outputs.

Refer to the [OPC Crosslink Help](#) for a full discussion.

## Saving and Undoing Configuration Changes

The Cyberlogic OPC Server Configuration Editor keeps track of recent configuration changes. Until you save these changes, you can revert to the previously-saved configuration. The editor supports two types of save operations. The standard Save operation saves the changes without updating the server or the connected clients. The Save & Update Server operation saves the changes and also updates the server and all connected clients.

**Caution!**

After you edit the configuration, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the changes you have made to take effect. Otherwise, the server will still be running with the old configuration.

### ***Saving Configuration Changes Without Updating the Server***

To save the configuration without updating the server, open the **File** menu and select **Save**, or click the **Save** button on the toolbar. The changes will be saved but the server will still be running with the old configuration.

### ***Saving Configuration Changes and Updating Server***

To save the configuration and update the server, open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** button on the toolbar.

### ***Undoing Configuration Changes***

To undo configuration changes and revert to the previously saved configuration, open the **File** menu and select **Undo Changes**, or click the **Undo Changes** button on the toolbar.

## Configuration Import/Export

The Import/Export feature allows you to export the configuration data to text file format and import configuration data from these exported files and also from comma separated values files from other vendors' OPC servers and programming software.

For details on this important feature and instructions in its operation, refer to the [Cyberlogic OPC Server Help](#).

## Options

The editor has several options that may be set to adjust the operation of the editor to suit your preferences and to set security levels as needed for communication with client software. For a full discussion, refer to the [Cyberlogic OPC Server Help](#).

## VALIDATION & TROUBLESHOOTING

The following sections describe features that will help you to verify and troubleshoot your server's operation. The [Data Monitor](#) and [Cyberlogic OPC Client](#) allow you to view the data as it is received by the server. Microsoft's [Performance Monitor](#) allows you to view relevant performance information. The [DirectAccess](#) feature lets you look at data values even if they have not been configured as data items. The [Event Viewer](#) may provide important status or error messages. Finally, there is a list of [MBX Driver Agent Messages](#) and [Frequently Asked Questions](#) to assist in your troubleshooting.

### Data Monitor

The Data Monitor lets you monitor the values and status of the data items. Its use is described in detail in the [Cyberlogic OPC Server Help](#).

### Cyberlogic OPC Client

The Cyberlogic OPC Client is a simple OPC Data Access client that lets you see how the server interacts with a client and lets you test its response to various loads. Its use is described in detail in the [Cyberlogic OPC Server Help](#).

### Performance Monitor

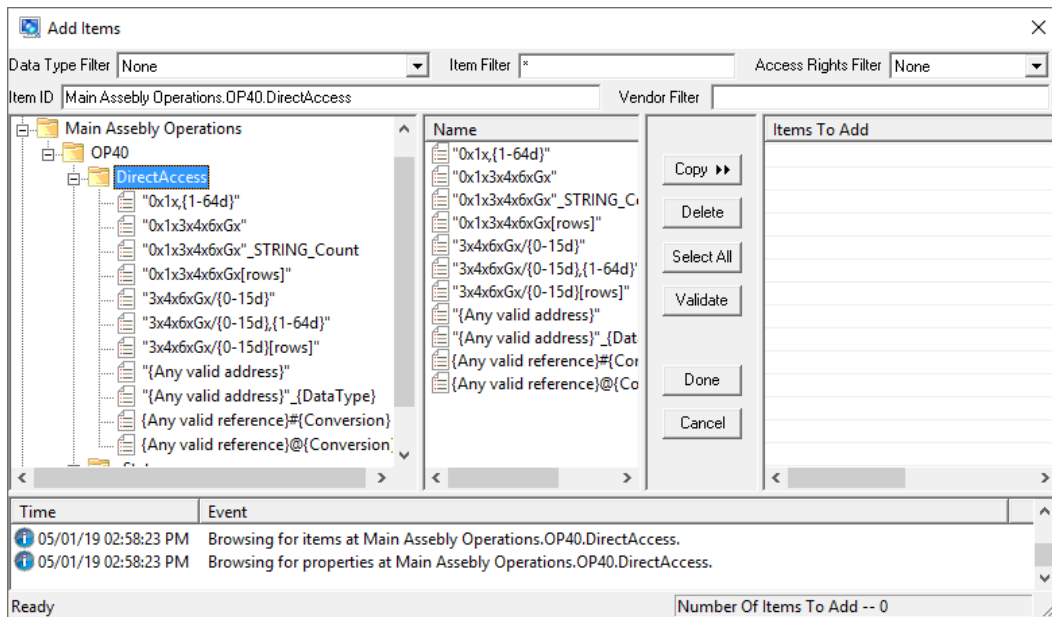
The Performance Monitor is a Microsoft diagnostic tool that the Cyberlogic drivers support. Its use is described in detail in the [Cyberlogic OPC Server Help](#).

### DirectAccess

At run time, in addition to the user-configured branches, the Cyberlogic OPC Server dynamically creates DirectAccess branches in its address space. These are created for both network nodes and devices.

DirectAccess allows read and write operations. However, for extra security, write operations are disabled by default. If writes are permissible, they can be enabled on a node-by-node basis as part of the network node configuration, and on a device-by-device basis as part of the device configuration.

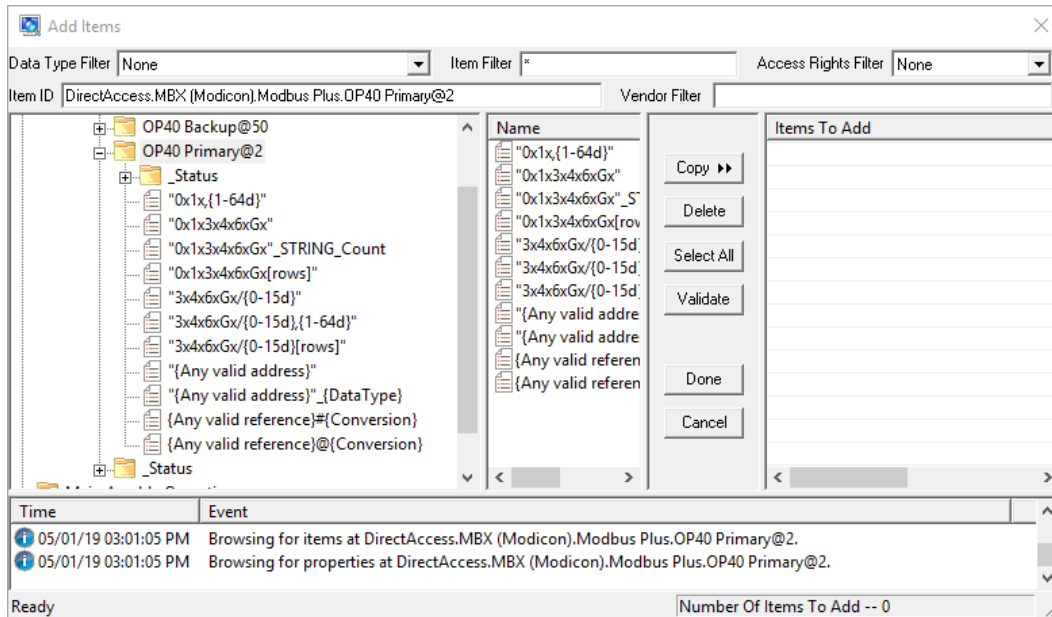
### Device DirectAccess



Each device in the address space will contain all of its configured data items, plus a DirectAccess branch, as shown in the example above. This branch will appear only for devices that have DirectAccess enabled. OPC clients can then use this branch to access any register in the device by directly specifying the register address.

In the DirectAccess branch for a device, the Cyberlogic OPC Server reports a list of hints about the types of data items that may exist on the selected device. These are not valid item addresses. Rather, they are just hints to help the user to specify a proper address. For more details on using these hints, refer to the [Address Hints](#) section.

### Network Node DirectAccess



DirectAccess to network nodes is achieved through a branch called DirectAccess at the root of the address space. This branch acts like a device folder that contains all of the configured network connections.

As you can see in the example above, each network connection branch contains its configured network nodes. However, only network nodes that enable DirectAccess are present. OPC clients can then use this branch to access any bit or register in any configured network node by directly specifying its address.

For network nodes in the DirectAccess branch, the Cyberlogic OPC Server reports a list of hints about the types of data items that may exist on the selected node. These are not valid item addresses. Rather, they are just hints to help the user to specify a proper address.

### ***Address Hints***

In the sample screens shown above, "0x1x3x4x6xGx" is an address hint. This hint lets you know the form of address used to access data from coil bits (0x), input bits (1x), input registers (3x), holding registers (4x), general registers (6x) and global data (Gx). You must replace the hint with the desired address.

Therefore, to access holding register 400007 using DirectAccess to the network node, you would edit the Item ID field at the top of the dialog box to read:

*DirectAccess.MBX (Modicon).Modbus Plus.OP40 Primary@2."400007"*

To access the same register using DirectAccess to the device, you would edit the Item ID to read:

*Main Assembly Operations.OP40.DirectAccess."400007"*

Here is a brief explanation of the other address hints:

*"0x1x,{1-64d}"*

This specifies a bit field and applies only to input and output bits. The {1-64d} hint means that you must specify the number of bits, in decimal form, in the range 1-64. For example, 000007,5 would specify a group of five bits, beginning with coil 000007.

*"0x1x3x4x6xGx[rows]"*

This form specifies an array of values, with [rows] specifying how many elements in the array. Thus, 600015[10] specifies an array of ten general registers, beginning with 600015.

*"0x1x3x4x6xGx" \_STRING\_Count*

You would use this form to specify a string of characters in one or more registers. The \_STRING\_Count portion is a data type override. Use 300010\_STRING\_6 to specify a string of six characters in input registers 300010 – 300015.

*"{Any valid address}"\_{DataType}*

This form tells the server to provide the data for a register in a specific format. The `_{DataType}` portion is a data type override. As an example, `400003_BCD16` would provide the value in holding register 400003, taken in binary-coded decimal form.

*"3x4x6xGx/{0-15d}"*

This form specifies a bit within a register, with `d` indicating that the bit number is in decimal form. The specification `400002/1` would be the second-least significant bit of holding register 400002.

*"3x4x6xGx/{0-15d},{1-64d}"*

This specifies a field of bits within one or more registers. You must identify the first bit in the field by specifying the register and bit position, then you must specify the width of the field. Both the starting bit and the width are decimal numbers. For example, `300016/15,8` specifies a bit field that is eight bits wide, starting with the most significant bit of input register 300016. This means that it would include the first seven bits of 300017.

*"3x4x6xGx/{0-15d}[rows]"*

This form specifies an array of bits within registers. The specification `400007/0[5]` would identify an array of five bits, each the least significant bit of holding registers 400007 – 400011.

*"{Any valid address}"*

In general, all addresses, as documented in Modicon manuals, are acceptable. Also, the addresses as defined by the custom address maps (see [Address Maps tab](#)) are allowed, too. In addition, the MBX Driver Agent extends this syntax when it comes to specifying arrays, array elements, register bits and bit fields. For a complete list of all valid register addresses, refer to [Appendix A: Register Addresses](#).

*{Any valid reference}#{Conversion}*

This form allows you to apply a previously configured data conversion to a raw register value in order to convert it into a form that is more useful to the client. When the conversion name is preceded by a `#` sign, the canonical data type for the requested data will always be VT\_R8 (64-bit floating point).

*{Any valid reference}@{Conversion}*

This form allows you to apply a previously configured data conversion to a raw register value in order to convert it into a form that is more useful to the client. When the conversion name is preceded by an `@` sign, the canonical data type will match the data type of the requested register or the specified `{Data Type Override}`.

**Note**

The address hints are shown enclosed in double-quotes, and the item address you specify in place of the hint must also be enclosed in double-quotes. If a data type override is used, it is not enclosed in the double-quotes.

Previous versions of the Cyberlogic OPC Server did not require the double-quotes, but had the requirement that any periods (.) in the address had to be replaced with a forward slash (/). This format is still valid, for compatibility with existing configurations. However, the double-quote format is preferred for new configurations.

### ***DirectAccess Address Formats***

The listed hints cover only the most common address formats. In fact, any item address in one of the following formats is valid:

- {Register Address}
- {Register Address}\_{Data Type Override}
- {Register Address}#{Conversion}
- {Register Address}\_{Data Type Override}#{Conversion}
- {Register Address}@{Conversion}
- {Register Address}\_{Data Type Override}@{Conversion}

### ***Register Address***

The Register Address portion may be any address that is acceptable in the Address field of a data item. This portion is enclosed in double-quotes.

The PLC addressing requirements are discussed in [Appendix A: Register Addresses](#).

### ***Data Type Override***

This capability permits you to display the value in a format other than the native format of the register. For example, a register might normally contain binary data, but is being used to hold BCD data for a particular application. You could then specify that it should be treated as BCD. The data type override portion is optional, and if used, is not enclosed in double-quotes.

The STRING and WSTRING data types may require the count value, which specifies the maximum number of characters in a string. By default, the count value is 1.

Any data type that is acceptable in the Data Type field of a data item may be specified as the override type. Notice that not all data types are valid for all register addresses. The table below shows all supported data types.

**Caution!**

The Data Type Override field requires you to use the form in the Type column, not the Canonical Data Type. For example, if you want 32-bit floating point format, you must specify **FLOAT32**. The canonical form **VT\_R4** will not work.



Type	Size in bits	Default Canonical Data Type	.NET Data Type	Description
Default				Default type based on the data item address
BIT	1	VT_BOOL	bool	1-bit boolean
SINT8	8	VT_I1	sbyte	Signed 8-bit integer -128 to 127
UINT8	8	VT_UI1	byte	Unsigned 8-bit integer 0 to 255
SINT16	16	VT_I2	short	Signed 16-bit integer -32,768 to 32,767
UINT16	16	VT_UI2	ushort	Unsigned 16-bit integer 0 to 65,535
SINT32	32	VT_I4	int	Signed 32-bit integer -2,147,483,648 to 2,147,483,647
UINT32	32	VT_UI4	uint	Unsigned 32-bit integer 0 to 4,294,967,295
SINT64	64	VT_I8	long	Signed 64-bit integer -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
UINT64	64	VT_UI8	ulong	Unsigned 64-bit integer 0 to 18,446,744,073,709,551,615
FLOAT32	32	VT_R4	float	IEEE 32-bit floating point number $\pm 3.4e\pm 38$
FLOAT64	64	VT_R8	double	IEEE 64-bit floating point number $\pm 1.7e\pm 308$
BCD16	16	VT_UI2	ushort	BCD value; 0 to 9,999
BCD32	32	VT_UI4	uint	BCD value; 0 to 99,999,999
STRING	String size * 8	VT_BSTR	string	Zero terminated ASCII string of 8-bit characters
WSTRING	String size * 16	VT_BSTR	string	Zero terminated UNICODE string of 16-bit characters
FIELD	Field size	Best fitting VT_UIx or array of VT_UI1 if size > 64	Best fitting unsigned type or byte[ ] if size > 64	Multiple bit field

### Conversion

This feature allows you to apply a previously configured data conversion to a raw register value in order to convert it into a form that is more useful to the client. For example, you can change a transducer's voltage value into a pressure value in psi.

The conversion name must be preceded by either a # sign or an @ sign. With the # sign, the canonical data type for the requested data will always be VT\_R8 (64-bit floating point). With the @ sign, the canonical data type will match the data type of the requested register or the specified {Data Type Override}. Here are a couple of examples:

```
DirectAccess.MBX (Modicon).Modbus Plus.OP40 Primary@2."400007"#100%  
Main Assembly Operations.OP40.DirectAccess."33017"_SINT16@P200
```

In the first example, the canonical data type is VT\_R8, while in the second example, it is VT\_I2. The *100%* and *P200* are the names of the conversions.

## Event Viewer

During startup and operation, the Cyberlogic OPC Server may detect problems or other significant events. When a noteworthy event is detected, the server sends an appropriate message to the Windows Event Logger. You can view these messages using the Windows Event Viewer. Its use is described in detail in the [Cyberlogic OPC Server Help](#).

For an explanation of the error messages that may be logged by the MBX Driver Agent, refer to the [MBX Driver Agent Messages](#) section.

## MBX Driver Agent Messages

This section shows Error Log messages that can be generated by the MBX Driver Agent (*MBXDriverClassSrv* in the Source column). The main Cyberlogic OPC Server module can also log error messages (*CybOpcRuntime* in the Source column). For a list of these messages, refer to the [Cyberlogic OPC Server Help](#).

***Registration DLL failed to load. The I/O operations of the server have been disabled. Reinstall the product.***

The installation program should have copied this DLL into the Windows system32 directory. Reinstall the product.

***Registration verification failed. The I/O operations of the server have been disabled. Reinstall the product.***

The server could not access the registration information, which is gathered and stored during the installation process. Reinstall the product.

***This is a <hours>-hour promotional copy of the MBX driver agent. The Cyberlogic OPC Server started at <start time> and the agent will stop at <stop time>.***

This is a time-limited version of the Cyberlogic OPC Server. For information on upgrading to the full version, contact your Cyberlogic distributor or visit Cyberlogic's website at [www.cyberlogic.com](http://www.cyberlogic.com).

***This is a promotional copy of the MBX OPC Server. The allowed operation time has expired. The I/O operations of the server have been disabled.***

This is a time-limited version of the MBX OPC Server and the allowed operation time has expired. For information on upgrading to the full version, contact your Cyberlogic distributor or visit Cyberlogic's website at [www.cyberlogic.com](http://www.cyberlogic.com).

***Memory allocation error in <function name>. Close some applications. Add more memory to your system. Contact the manufacturer's technical support.***

The specified function failed to allocate the needed memory. This is a fatal error. If you are running low on memory, close some applications or add more memory to your system. If the problem continues, contact technical support for more information on a possible solution.

***Memory allocation error in <function name>. The server may not operate correctly. Close some applications. Add more memory to your system. Contact the manufacturer's technical support.***

The specified function failed to allocate the needed memory. The server will continue to operate, but some functions may not work. If you are running low on memory, close some applications or add more memory to your system. If the problem continues, contact technical support for more information on a possible solution.

***Unexpected error in <function name>. Please contact the manufacturer's technical support.***

Indicates a possible programming bug in the server. Contact technical support for more information on a possible solution.

***Unexpected error in <function name> (Error code = <error code>). Please contact the manufacturer's technical support.***

Indicates a possible programming bug in the server. Contact technical support for more information on a possible solution.

***Network node device "<network node name>" does not support Report Slave ID function (0x11) which is required for auto-detecting the device type. Auto Detect cannot be used with this device. Defaults to Safe Settings.***

The OPC server software tried to identify the specified network node, but the device at that node does not support the function call that is used to identify it. Automatic

detection cannot be used for this node. You must edit the network node and select the proper device type manually.

***During auto-detection, network node device "<network node name>" reported an unrecognized device ID (<device id>). Auto Detect cannot be used with this device. Defaults to Safe Settings.***

The OPC server software tried to identify the specified network node, but it reported a device ID that the editor does not recognize. This may indicate a new device type that has not yet been added to the editor's device type database. You must edit the network node and select the proper device type. Please report this error to Cyberlogic's Tech Support so we can include this device type in future releases of the software.

***The Cyberlogic License Server failed to respond with valid license information. The I/O operations of the MBX Driver Agent have been disabled. Contact the manufacturer's technical support.***

The driver agent experienced a problem when it tried to contact the Cyberlogic License Server. If the license server is not running, start it and then try restarting the OPC server. If the license server is already running, contact Cyberlogic Tech Support.

## Frequently Asked Questions

For FAQs common to all driver agents, refer to the [Cyberlogic OPC Server Help](#).

***I have a PCIe-85 card installed on my system, but when I run the MBX demo, it doesn't show any other nodes on the network. I know that the network cable is good because I tested it with another computer.***

Plug-and-Play adapter cards, such as the PCIe-85, default to node address 1. If there is another one of these cards anywhere on your network, it is likely that it is using this default address, resulting in a conflict. Use the MBX Configuration Editor to change the adapter's node address to a different value.

***I have an SA-85 card connected to the Modbus Plus network. I successfully installed the MBX OPC Server, but when I try the automatic configuration, the editor fails to detect any network connections. What is the problem?***

For the automatic configuration to work you must first create at least one MBX device. To do that, at least one of the following MBX family drivers must be installed: MBX Driver, Ethernet MBX Driver, Serial MBX Driver or MBX Gateway Driver. (In your case, you should install at least the MBX Driver.) Refer to the driver-specific help file for information on creating MBX devices. Be sure the driver is running. The drivers default to automatic operation when they are installed, but may have been switched to manual. This selection is done from the driver configuration editor.

## APPENDIX A: REGISTER ADDRESSES

This appendix describes the allowed syntax for the Address field on the [Data Tab](#) of a data item. The MBX Driver Agent supports all register address notations used by various Modicon products. It also supports the addresses as defined by the custom address maps (see [Address Maps tab](#)). In addition, the MBX Driver Agent extends this syntax when it comes to specifying arrays, array elements, bit addresses and bit fields.

### Modicon Address Notations

This section describes the register address notations used in various Modicon products:

#### ***Standard (400001) Notation***

In the standard notation the first digit identifies the register type and the following digits represent the register number.

##### Syntax

0xxxxx – Coil

1xxxxx – Input

3xxxxx – Input register

4xxxxx – Holding register

6xxxxx – General register

##### Examples

400001            First holding register

300002            Second input register

#### ***Separator (4:00001) Notation***

In the separator notation the first digit identifies the register type followed by a colon character and a five-digit number representing the register number.

##### Syntax

0:xxxxx – Coil

1:xxxxx – Input

3:xxxxx – Input register

4:xxxxx – Holding register

6:xxxxx – General register

Examples

4:00001            First holding register  
3:00002            Second input register

**Compact (4:1) Notation**

In the compact notation the first digit identifies the register type followed by a colon character and the register number without leading zeros.

Syntax

0:xxxxx – Coil  
1:xxxxx – Input  
3:xxxxx – Input register  
4:xxxxx – Holding register  
6:xxxxx – General register

Examples

4:1                First holding register  
3:2                Second input register

**IEC (QW00001) Notation**

In the IEC notation the first two letters identify the register type and the following five digits represent the register number.

Syntax

QXxxxxx – Coil  
IXxxxxx – Input  
IDxxxxx – Input register  
QWxxxxx – Holding register

Examples

QW00001            First holding register  
ID00002            Second input register

**Global Data (G01) Notation**

Global data notation consists of a G followed by two decimal digits in the range of 01 - 32. All global data is handled as 16-bit registers.

### Syntax

Gxx – Global data register.

### Examples

G01                    First global data register

G32                    Last global data register

If you use global data and you want to read or write the OPC server's global data, you must create a node of type This Node – Global Data. This will then allow you to create global data type data items in the Address Space and connect them to the local node. For more information on configuring network nodes, refer to the [Editing Network Nodes](#) section.

## Cyberlogic Address Extensions

This section describes the syntax extensions that apply to specifying arrays, array elements, bit addresses and bit fields.

### **Arrays**

Single-dimension arrays of most data types are supported with the exception of strings and bit fields.

### Syntax

{Valid Register Address}[{Number of Elements}]

or

{Valid Register Address}[{Number of Elements},{Lower Bound}]

The Lower Bound specifies the array index value for the first element in an array. If not specified, the Lower Bound defaults to zero. Visual Basic applications may expect the first index to be equal to a one, in which case you would set the Lower Bound value to a 1.

### Examples

400001[5]            Array of five registers starting from address 400001, with a lower bound of 0

0:1[4,1]            Array of four bits starting from address 000001, with a lower bound of 1

4:00010/0[16]      Array of sixteen bits starting from bit 400010/0, with a lower bound of 0

### **Array Elements**

The 16-bit registers may be viewed as arrays of two bytes.

Syntax

{Valid Register Address}{Element Index}

Examples

400001(0)            First byte of register 400001  
 300001(1)            Second byte of register 300001

Notice that it is possible to combine this specification with the array syntax to specify an array of 8-bit bytes beginning at a specific byte.

Example

400001(1)[10]        Array of 10 bytes starting from the second byte of 400001

**Register Bits**

You may specify a single bit within a 16-bit register.

Syntax

{Valid Register Address}/{Bit Number}

The Bit Number is in the range of 0-15.

Examples

400001/1            Bit 1 in register 400001  
 300001/15           Bit 15 in register 300001

**Bit Fields**

You may specify a sequence of bits as a bit field.

Syntax

{Valid Bit Address},{Bit Count}

Examples

400001/1,5           Bit field of five bits starting from bit 400001/1  
 100001,16            Bit field of sixteen bits starting from bit 100001



## APPENDIX B: DATA ITEM DUPLICATION WIZARD

To speed up the creation of similarly configured data items, you can easily create multiple data items by duplicating an existing one. To do this, right-click on an existing data item and select **Duplicate...** from the context menu. The Data Item Duplication Wizard will open.

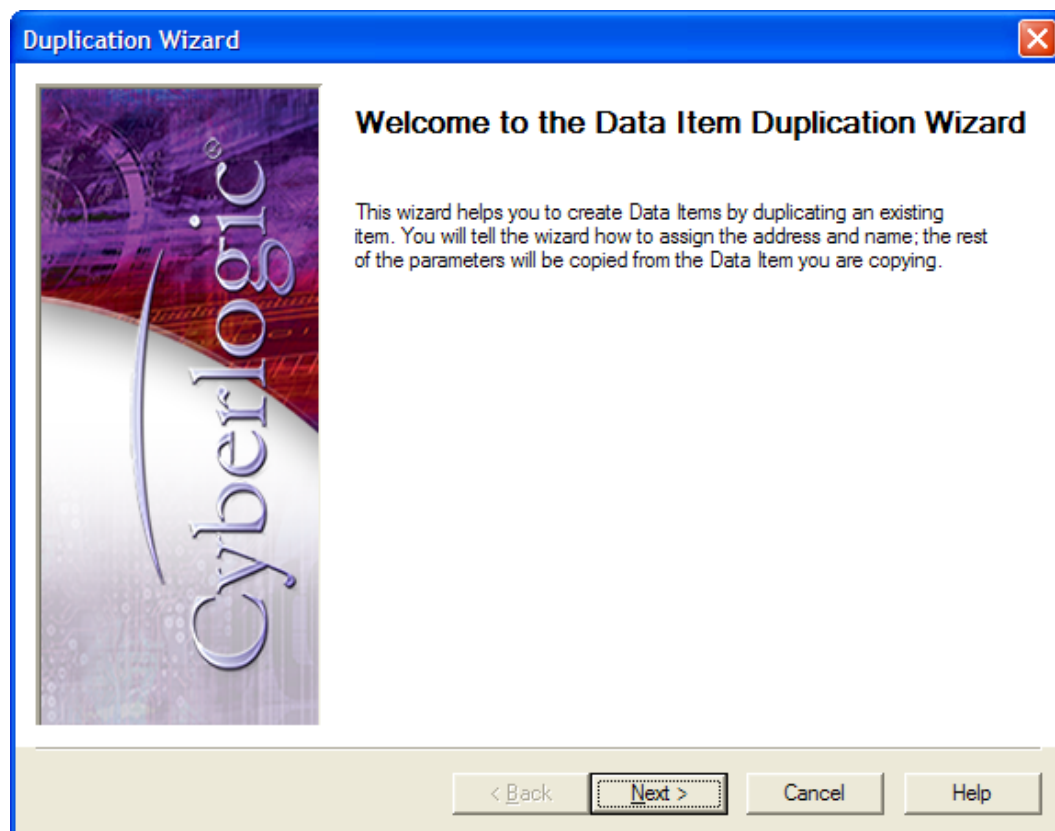
We will present two examples here.

[Example 1: Simple Addressing](#) shows how to duplicate input bits. This same procedure would be used for other data using simple addressing.

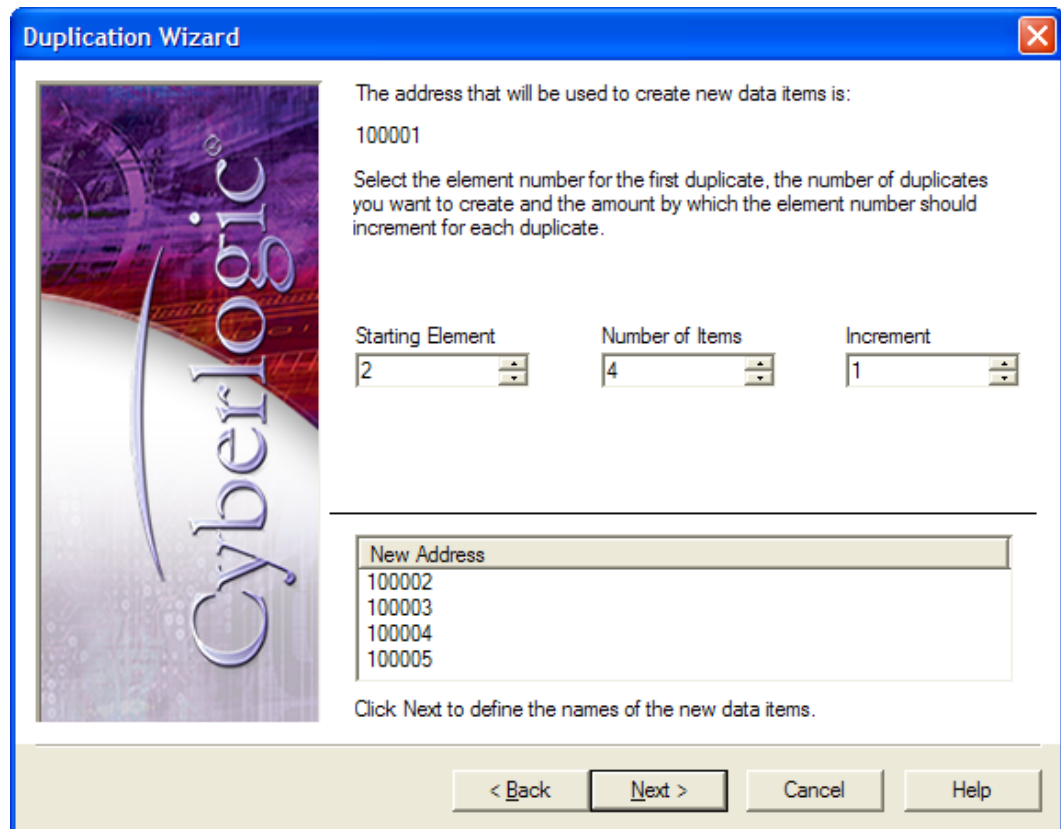
[Example 2: Addressing Bits Within Registers](#) is more complex and shows how to handle data addressed as bits within a register or array.

### Example 1: Simple Addressing

For this example, we selected for duplication a data item called 100001.

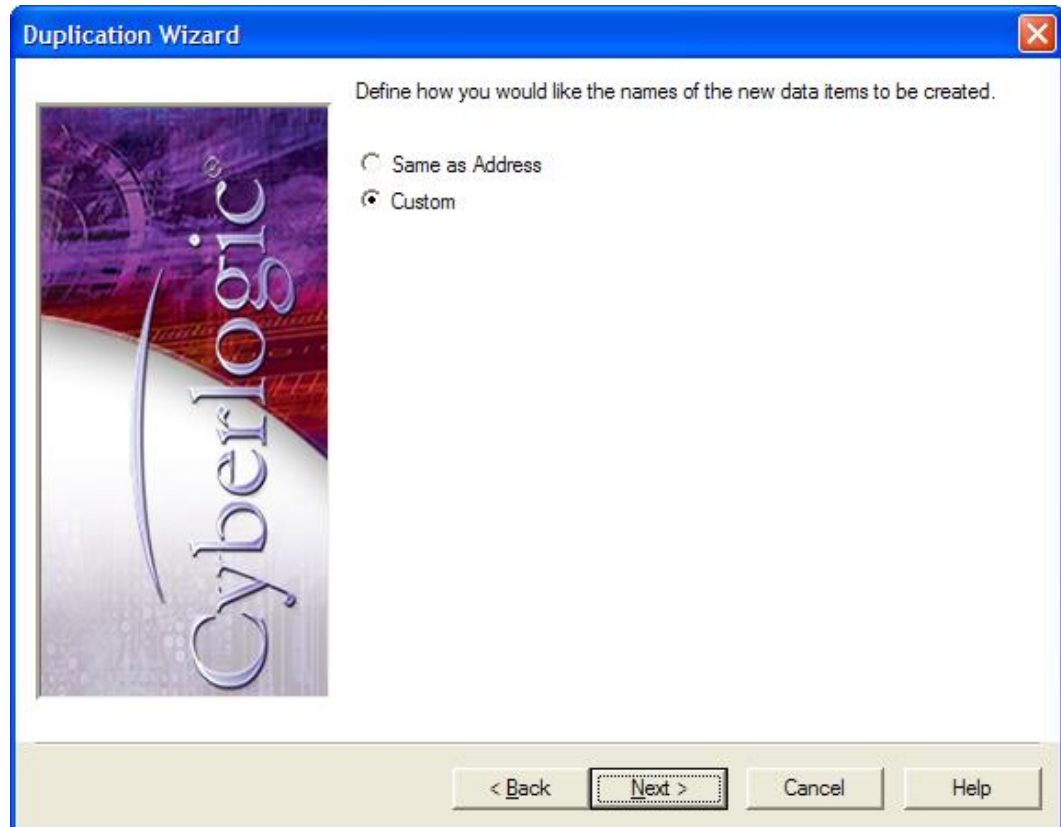


1. On the Welcome screen, click the **Next** button to continue.



On this screen, you will specify the duplicate data items you want to create. Notice that the screen tells you which data item you are duplicating.

2. Enter **2** for the **Starting Element**.
3. Enter **4** for the **Number of Items**.
4. Enter **1** for the **Increment**.
5. The **New Address** box shows you the data items that the wizard will create. Verify that these are correct.
6. Click **Next** to continue.



You must now decide how you wish to name the data items you will create. You may simply use the address as the name or you may create a custom naming scheme.

7. Select **Custom**.
8. Click **Next** to continue.

**Duplication Wizard**

The wizard creates names consisting of a prefix, a numeric value and a suffix. Enter text for the prefix and suffix. For the number, select the starting value, the amount to increment for each item, and the number of digits to display.

Prefix  
1

Starting Value      Increment      Numeric Places  
2      1      5

Suffix

Address	Name
100002	100002
100003	100003
100004	100004
100005	100005

Click Finish to create the new Data Items.

< Back    Next >    Cancel    Help

The wizard will create names for the data items for you. These names will consist of a prefix, a numeric value and a suffix. The first data item we created was named 100001 and we would like the duplicates to have names of the same style.

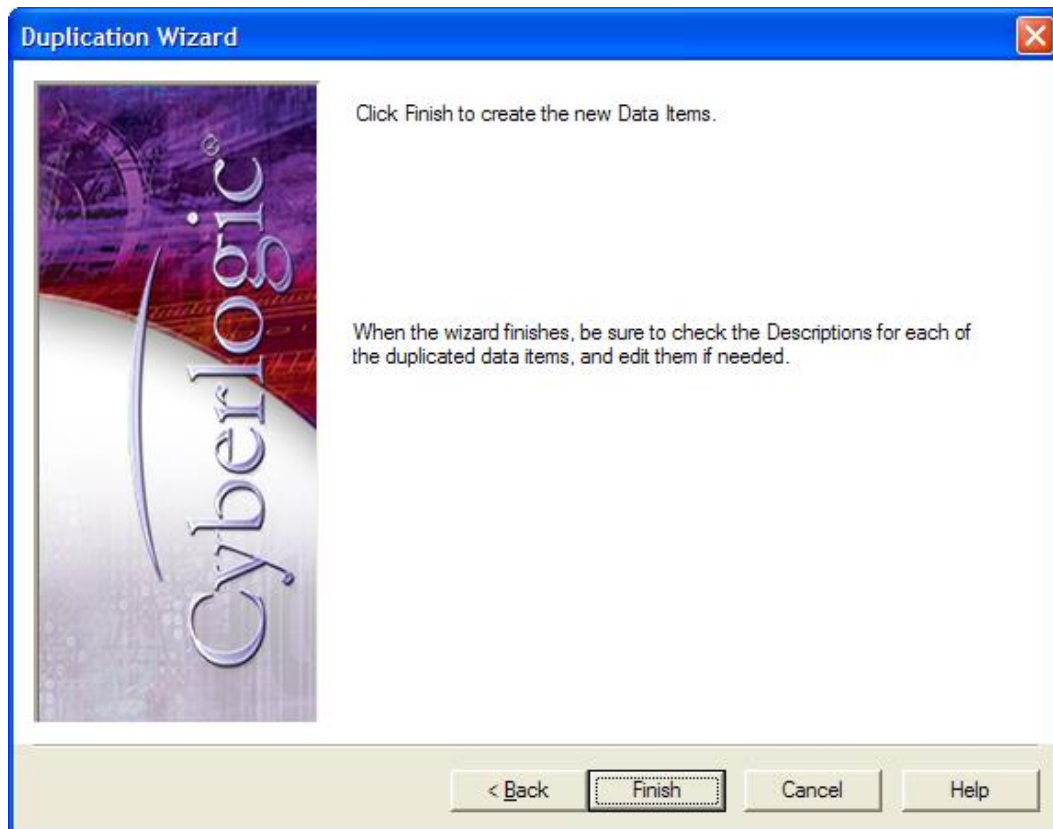
- Enter **1** in the **Prefix** field, to have all of the names start with 1.

The next three fields define the numeric values to be used.

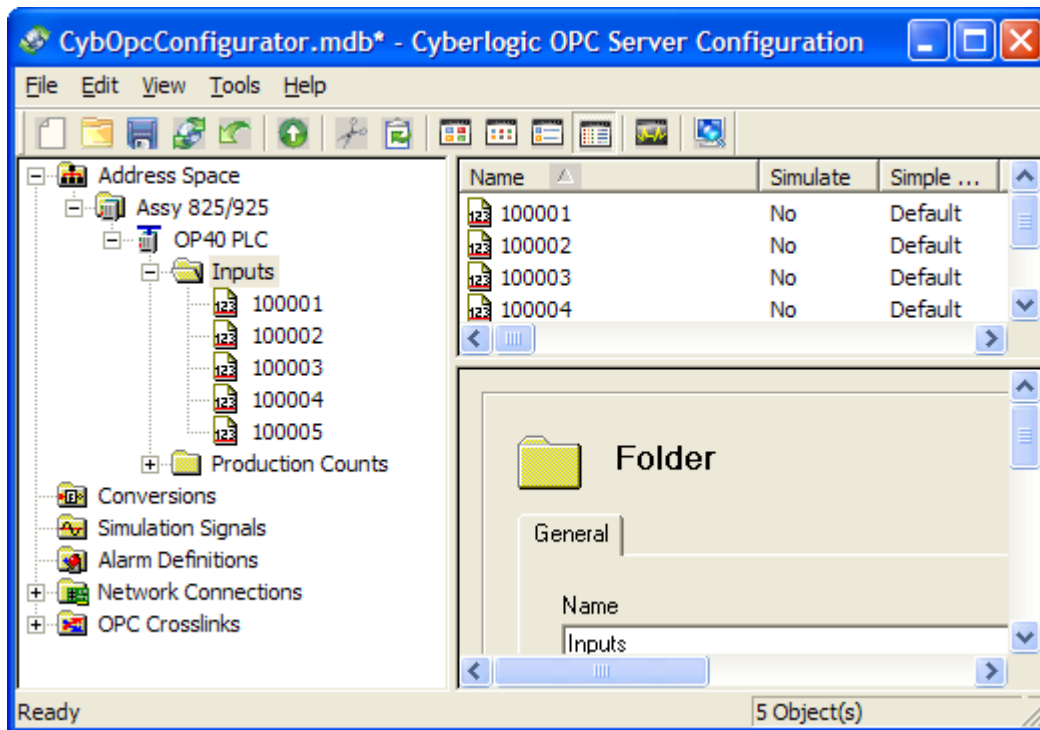
- Enter **2** as the **Starting Value**.
- Enter **1** for the **Increment**.
- Enter **5** for **Numeric Places**.

These selections cause the duplicates to be numbered consecutively, beginning with 2. It also forces the names to use five digits after the prefix, inserting leading zeros as needed.

- Leave the **Suffix** field blank, because no suffix is necessary for this naming scheme.
- The lower window will show you the names that will be used for each data item. Verify that these are correct.
- Click **Next** to continue.



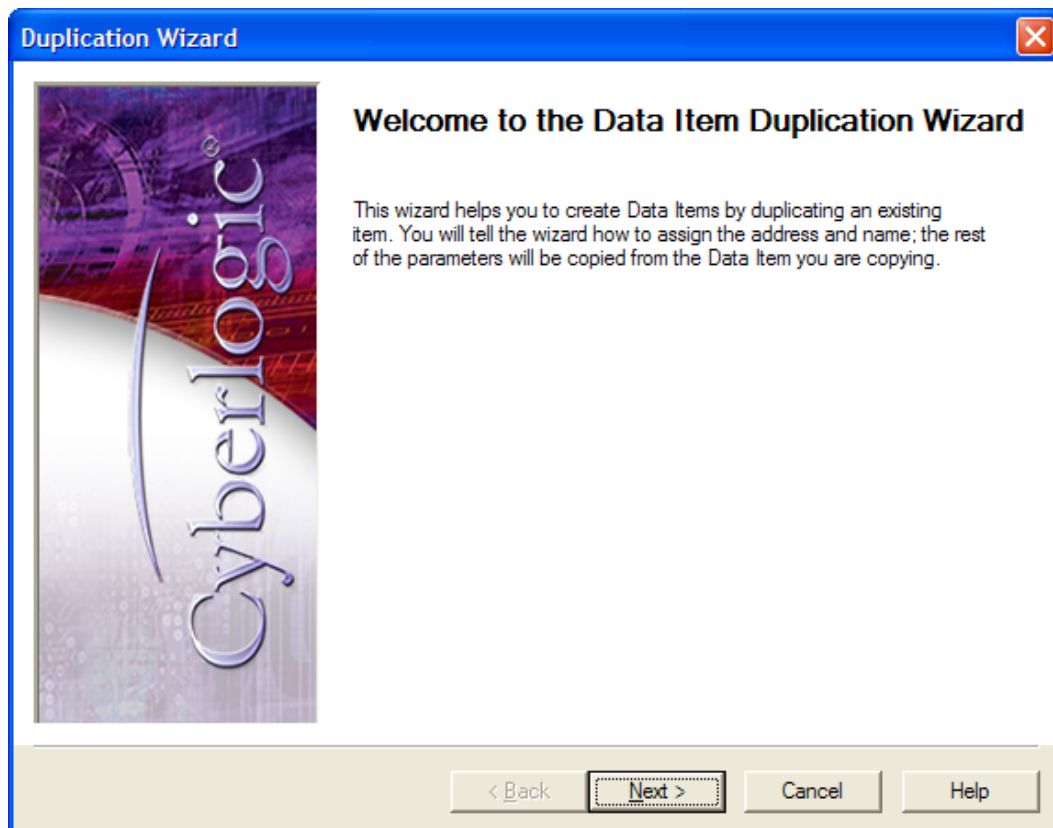
16. Click **Finish** to create the duplicate data items and exit the wizard.



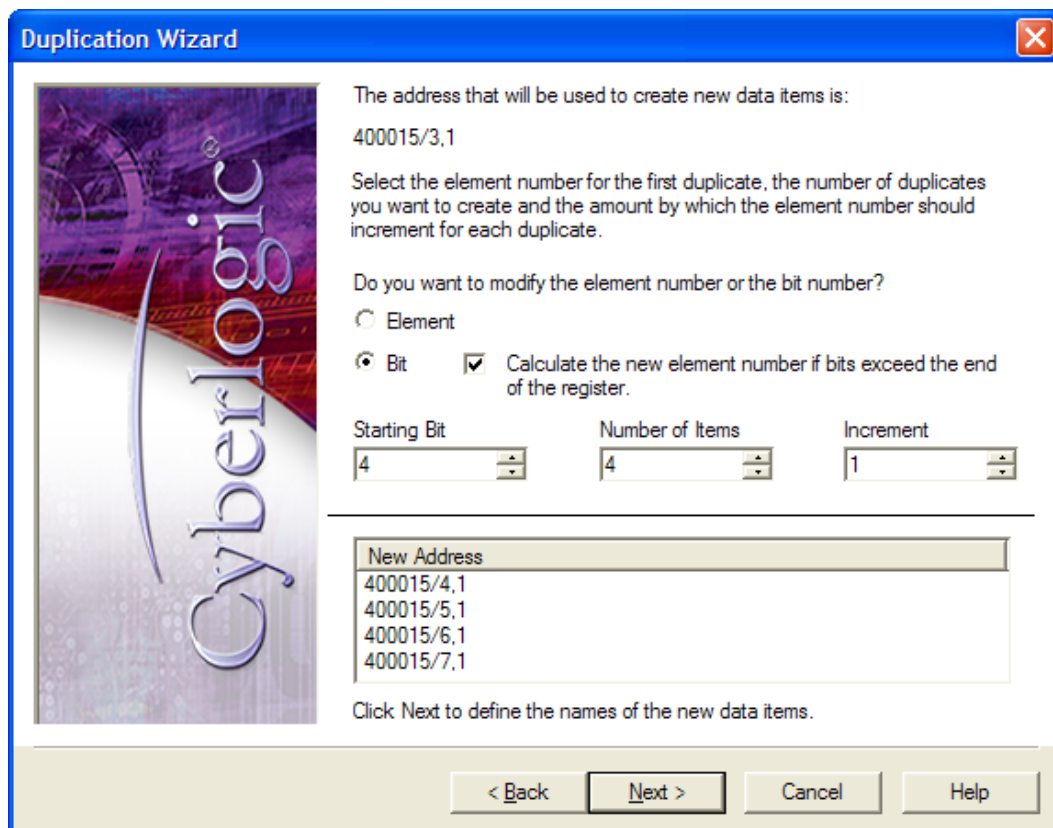
Your duplicates will be created. You will find, however, that the Description field for each duplicate is the same as the original. You may wish to edit these descriptions. You should also check the other parameters to verify that they are set properly.

## Example 2: Addressing Bits Within Registers

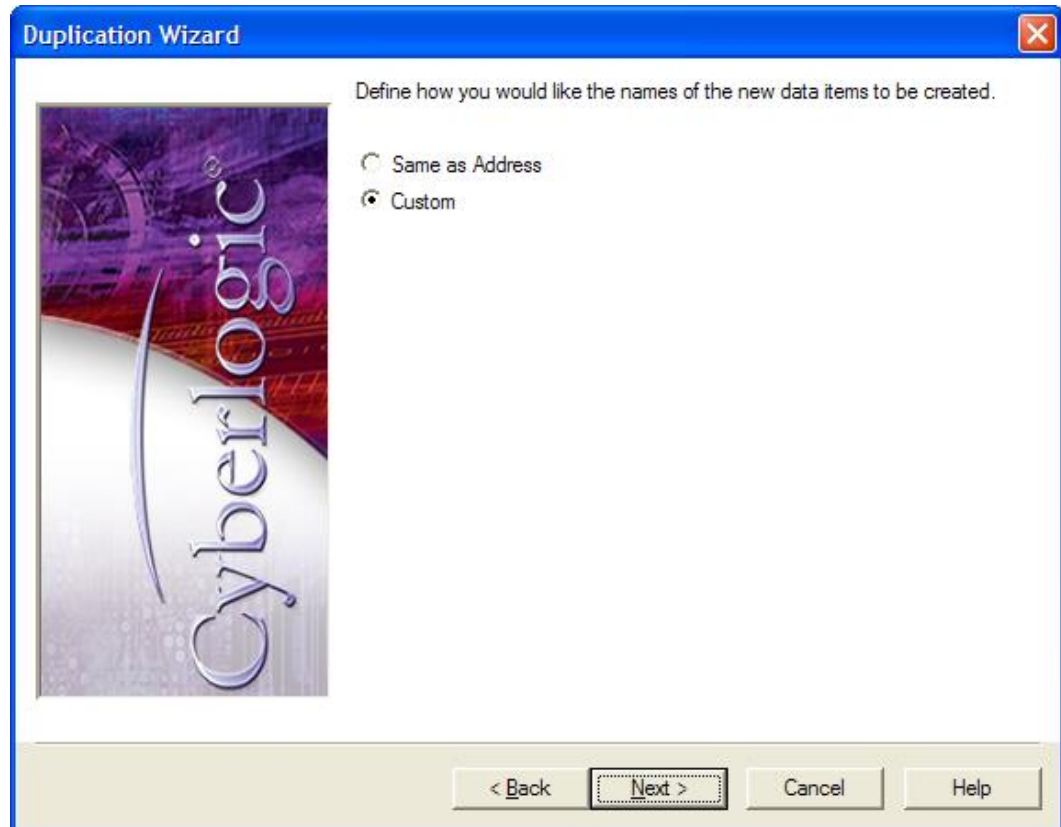
For this example, we selected a data item called *StatusItem3*. It is a bit within a word and is addressed as 400015/3,1. We want to address additional bits within this same word.



1. On the Welcome screen, click the **Next** button to continue.



2. Select **Bit** to indicate that you want to address additional bits within the same register, rather than additional registers.
3. Enter **4** for the **Starting Bit** value, to specify that the first duplicate should be 400015/4,1.
4. Enter **4** for the **Number of Items**, to create four duplicates.
5. Enter **1** for the **Increment**, so that you will access consecutive bits.
6. The lower window will show the new addresses that will be accessed. Verify that these are correct.
7. Click **Next** to continue.



8. Select **Custom** to indicate that you want to specify names for the new data items, rather than simply using each item's address as its name.
9. Click **Next** to continue.



**Duplication Wizard**

The wizard creates names consisting of a prefix, a numeric value and a suffix. Enter text for the prefix and suffix. For the number, select the starting value, the amount to increment for each item, and the number of digits to display.

Prefix

Starting Value      Increment      Numeric Places  
           

Suffix

Address	Name
400015/4,1	StatusItem4
400015/5,1	StatusItem5
400015/6,1	StatusItem6
400015/7,1	StatusItem7

Click Finish to create the new Data Items.

< Back    Next >    Cancel    Help

The name for each item will be built by concatenating a prefix, a number and a suffix.

10. Enter **StatusItem** as the **Prefix**.

The next three fields specify the number.

11. Enter **4** as the **Starting Value**.

12. Enter **1** as the **Increment**.

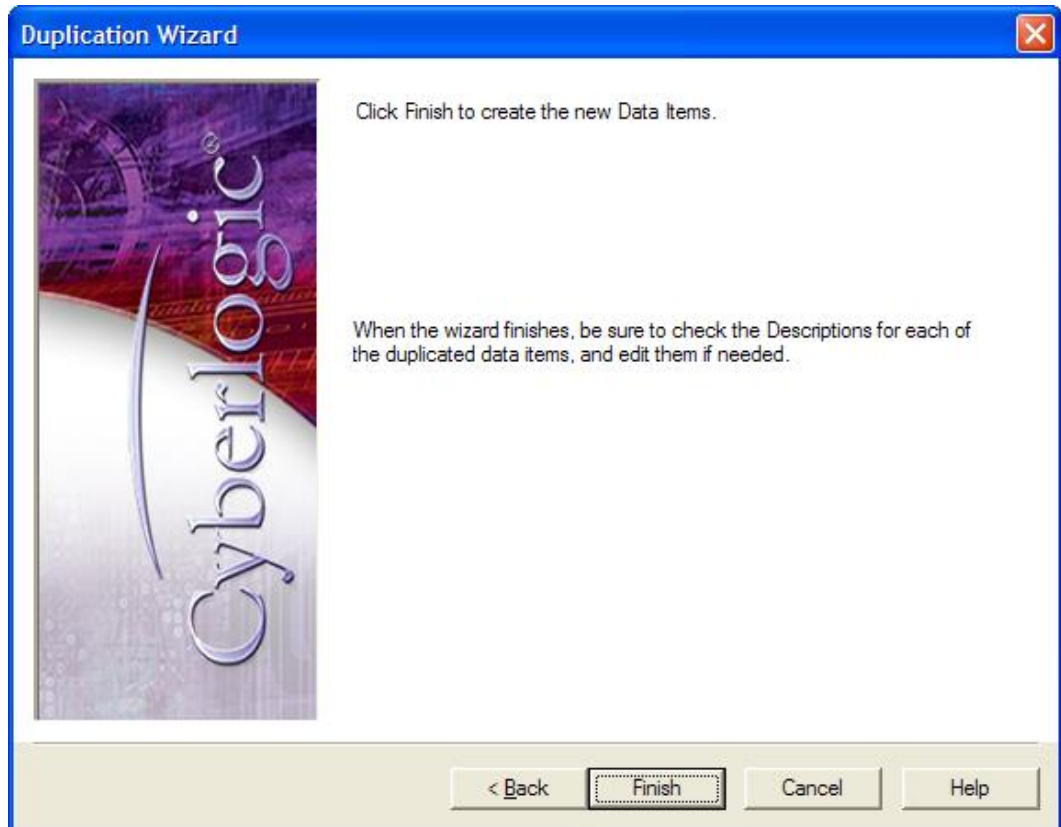
The data items will therefore be numbered consecutively between 4 and 7.

13. Select **1** for the **Numeric Places**, because there is no need to force the use of leading zeros in the name.

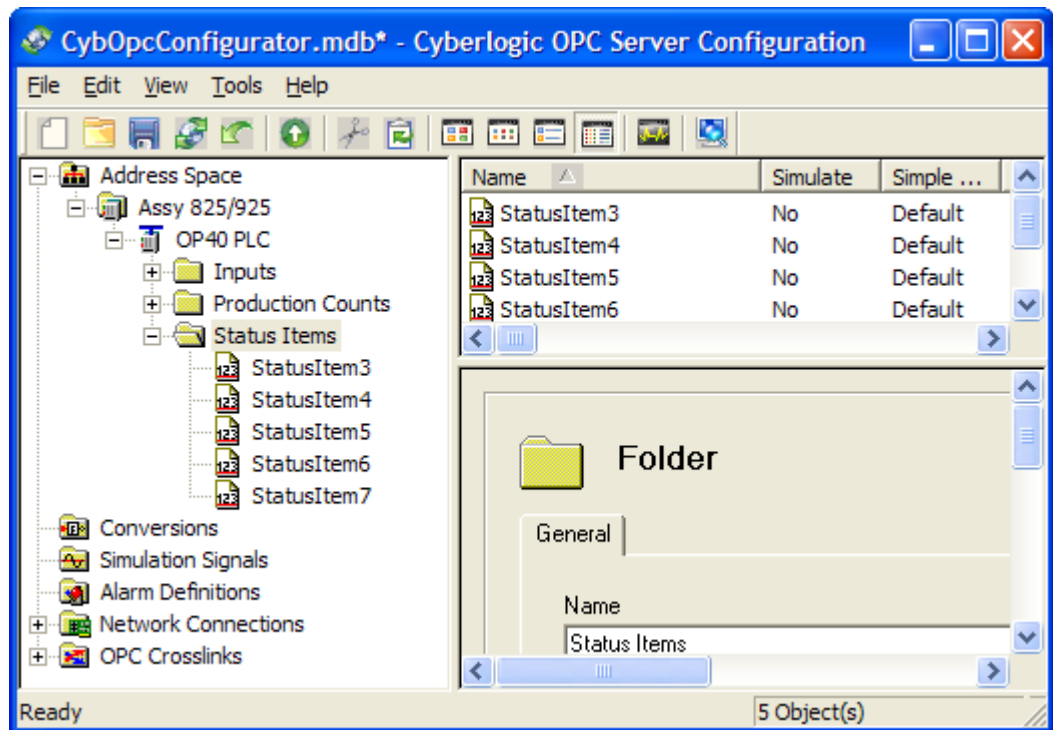
14. The **Suffix** field is optional and is not used in this example, so leave it blank.

15. The wizard will show you the names that will be assigned to each address. Verify that these are correct.

16. Click **Next** to continue.



17. Click **Finish** to create the duplicate data items and exit the wizard.



Your duplicates will be created. You will find, however, that the Description field for each duplicate is the same as the original. You may wish to edit these descriptions. You should also check the other parameters to verify that they are set properly.

## APPENDIX C: ADDRESS WIZARD

The Address Wizard will help you to define the correct address for the data you want to access. To activate the Address Wizard, click the ***Wizard...*** button on the Data tab of the data item dialog.

The screenshot shows the 'Data Item' dialog box with the 'Data' tab selected. The 'Address' field contains the value '400001'. Below it, the 'Data Type' is set to 'Default' and the 'String Length' field is empty. The 'Span Messages' checkbox is checked. In the 'Bit Order Override' section, the 'Bit Order Override' checkbox is unchecked, and the radio buttons for 'Dbl Word', 'Word', 'Byte', 'Nibble', and 'Bit' are also unchecked. The 'OPC Client' section has 'Canonical Data Type' set to 'Default' and the 'Use Conversion' checkbox is unchecked. The 'Conversion' dropdown menu is set to '<Not Assigned>'. At the bottom of the dialog, there are buttons for 'Apply', 'Cancel', 'Add New', and 'Help'.

This appendix provides two examples of the use of the Address Wizard.

[Example 1: Simple Modicon PLC Addressing](#) covers setting up a simple address for a single register.

[Example 2: Arrays of Registers](#) shows how to address registers as an array.

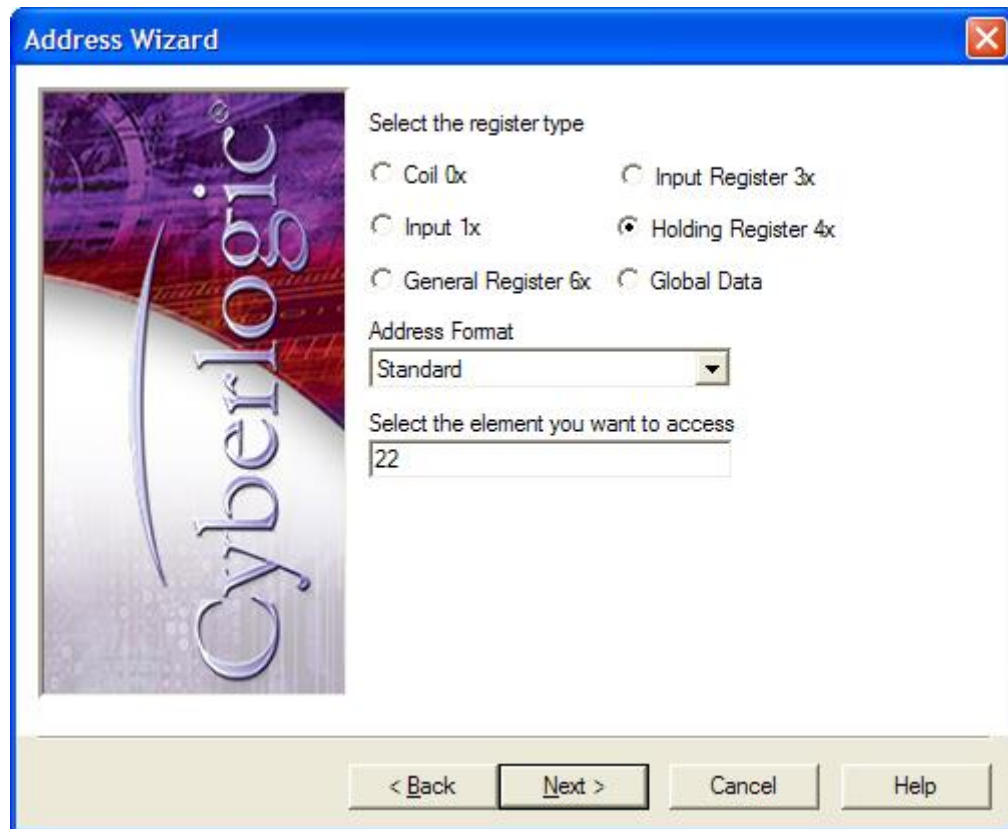
[Example 3: OMNI Flow Controller Bit Field](#) shows how to set up a bit field inside an OMNI Flow Controller register.

### Example 1: Simple Modicon PLC Addressing

This example shows how to set up the addressing for a single register. In this example, the Address Map field on the MBX Device associated with this data item is set to <Modicon PLC>.

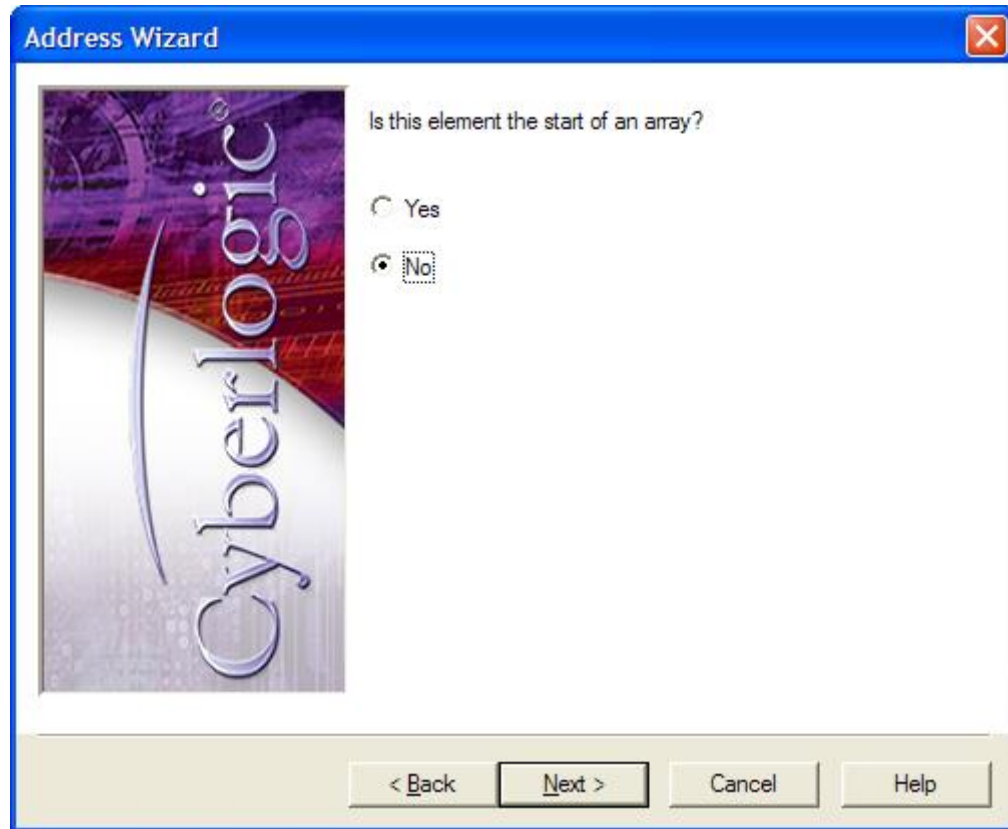


1. From the Welcome screen, click **Next** to continue.



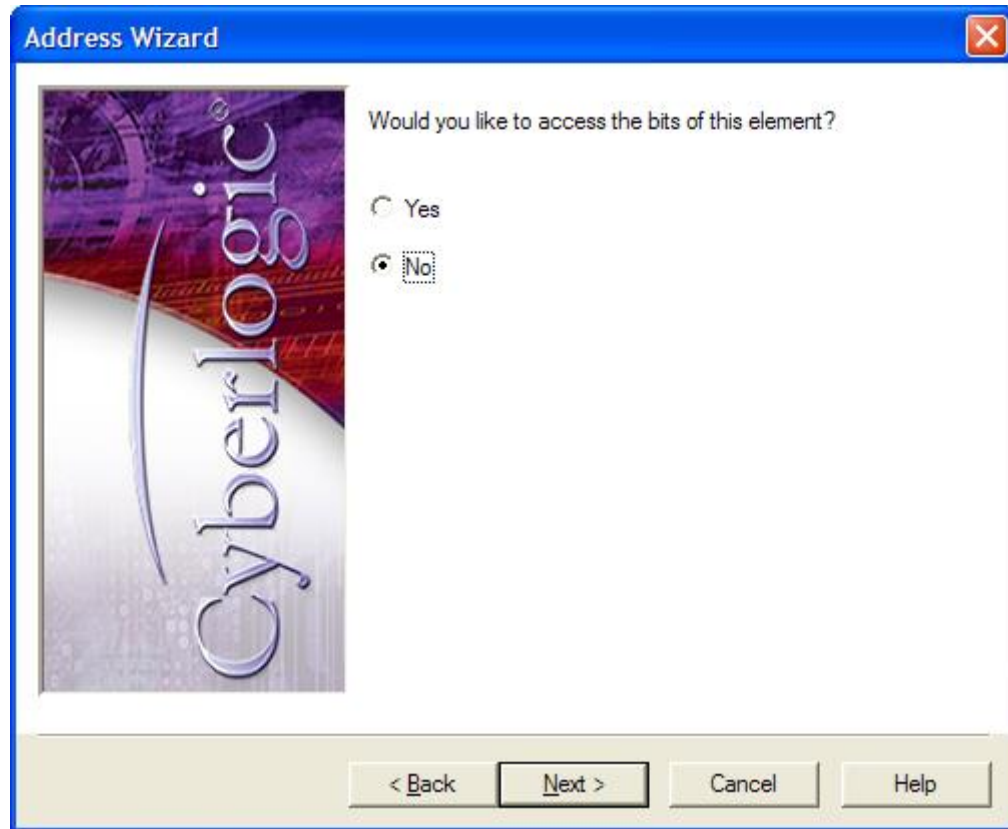
On this screen, you must select the specific register you want to access.

2. For this example, select **Holding Register 4x** as the type.
3. Select the **Standard** address format.
4. Enter **22** as the element number, to select register 40022.
5. Click **Next**.



For this example, we do not want to access the data as an array.

6. Select **No**.
7. Click **Next**.



It is possible to access individual bits or groups of bits within the register, but we want to access the register as a word.

8. Select **No**.
9. Click **Next**.





This screen shows the result of your selections. The register address to be created is 400022.

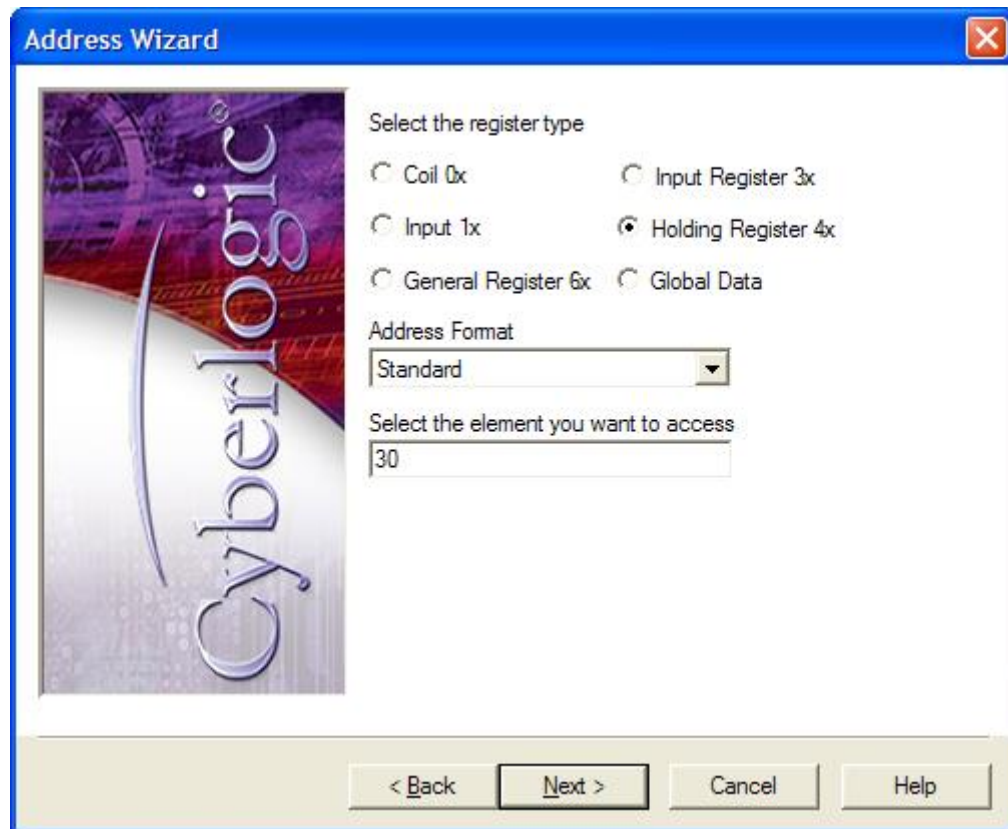
10. Click ***Finish***.

## Example 2: Arrays of Registers

This example shows how to set up the addressing to allow you to access several registers as an array. In this example, the Address Map field on the MBX Device associated with this data item is set to <Modicon PLC>.

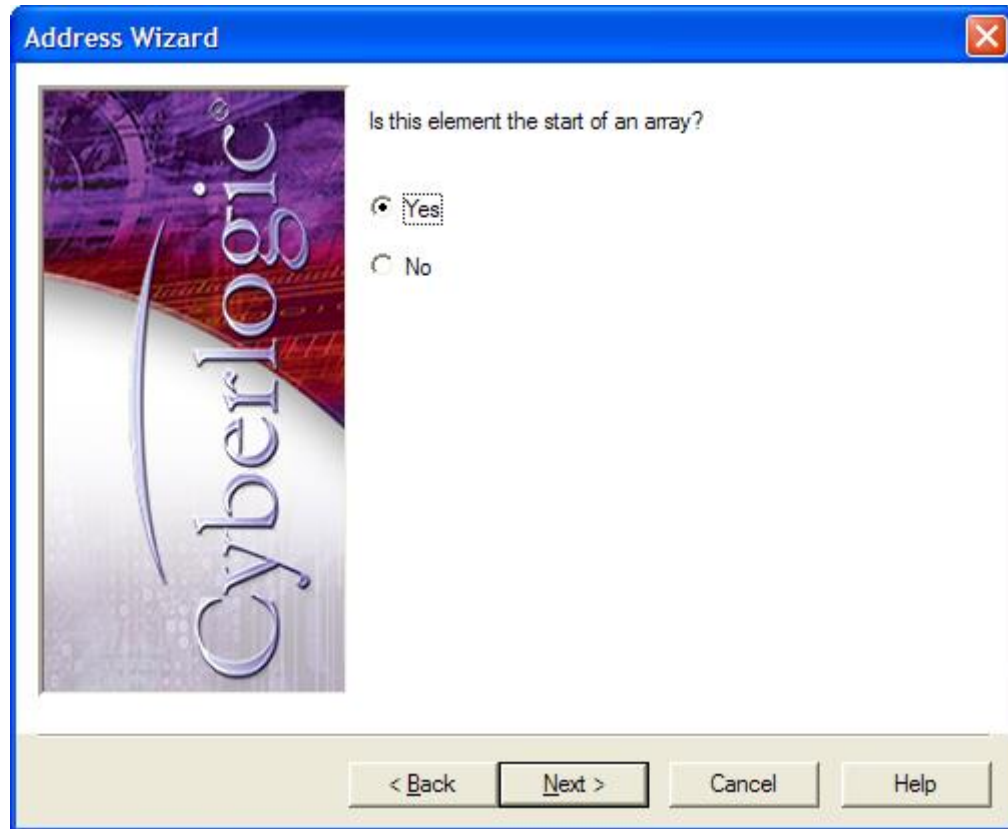


1. From the Welcome screen, click **Next** to continue.



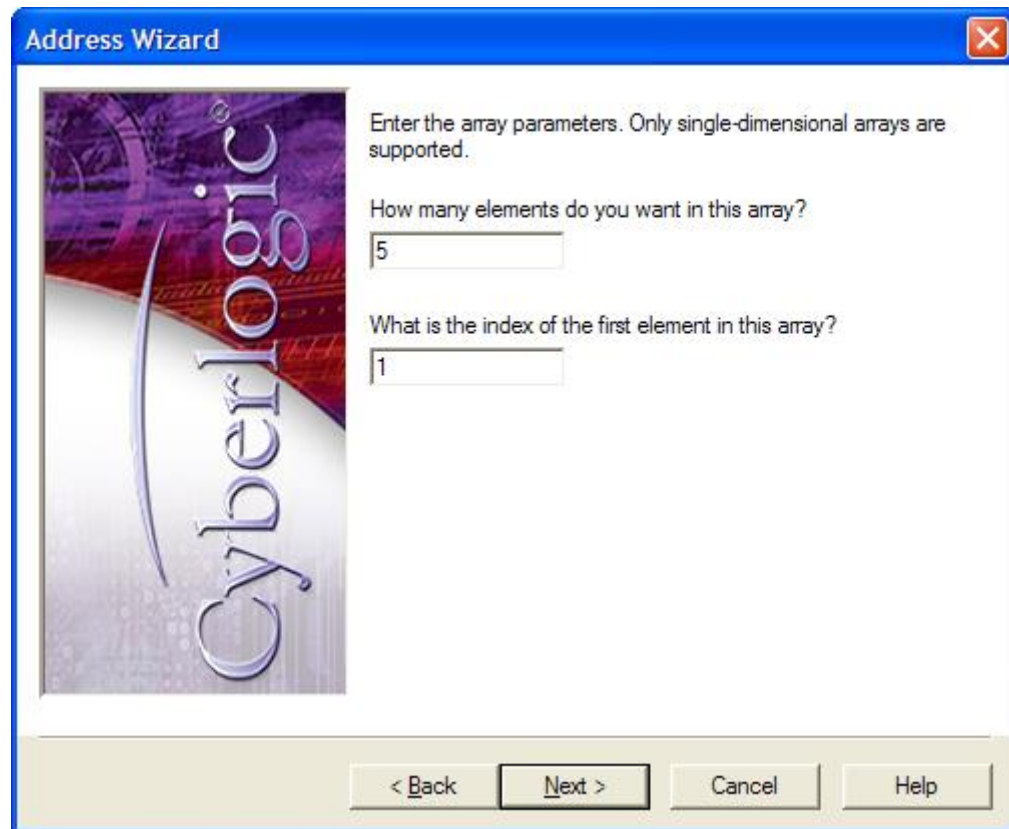
On this screen, you must select the first element of the array, that is, the first register you want to access.

2. For this example, select **Holding Register 4x** as the type.
3. Select the **Standard** address format.
4. Enter **30** as the element number, to select register400030.
5. Click **Next**.



For this example, we want to access the data as an array.

6. Select **Yes**.
7. Click **Next**.



Here you must specify the size of the array and how the elements will be referenced.

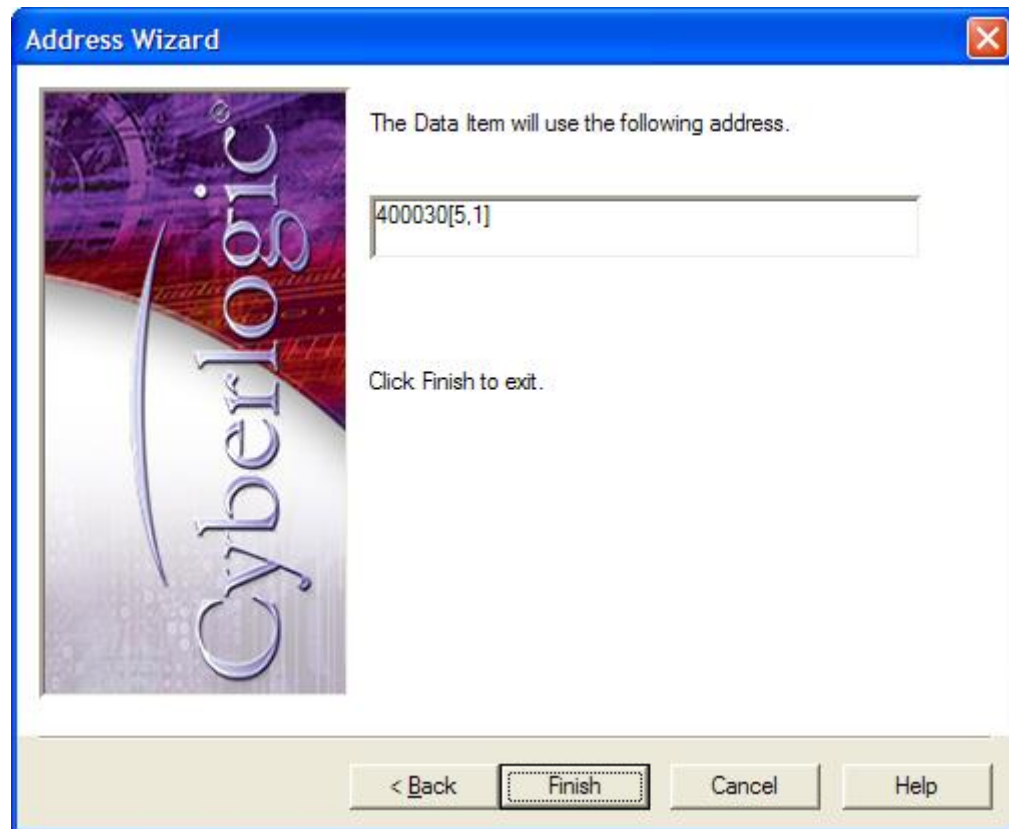
8. Enter **5** for the number of elements in the array.

For this example, the five elements in the array will be registers 400030 through 400034.

9. Enter **1** for the index of the first element.

This value specifies how the client will address the array. In this case, the array reference will start at 1, corresponding to 400030. The register 400031 would then be addressed using reference 2 and so on.

10. Click **Next** to continue.

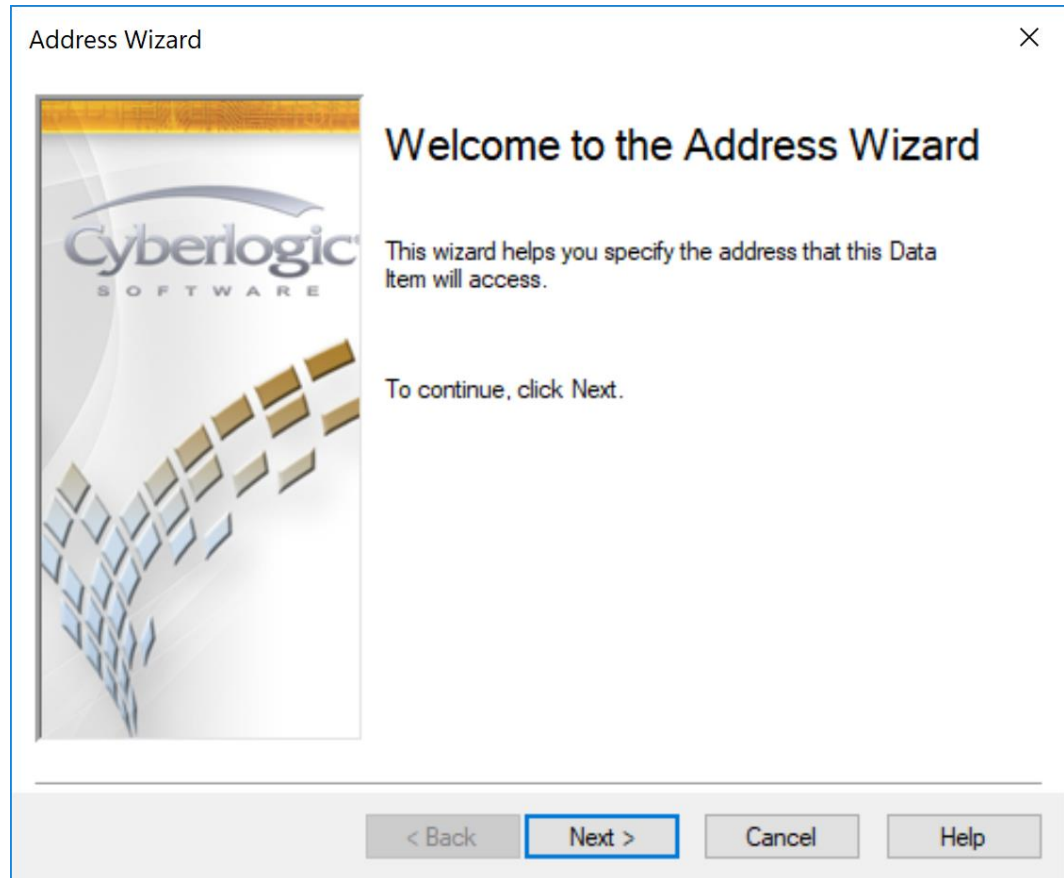


This screen shows the result of your selections. The bracketed values show that the array contains 5 elements and that the index for addressing starts at 1.

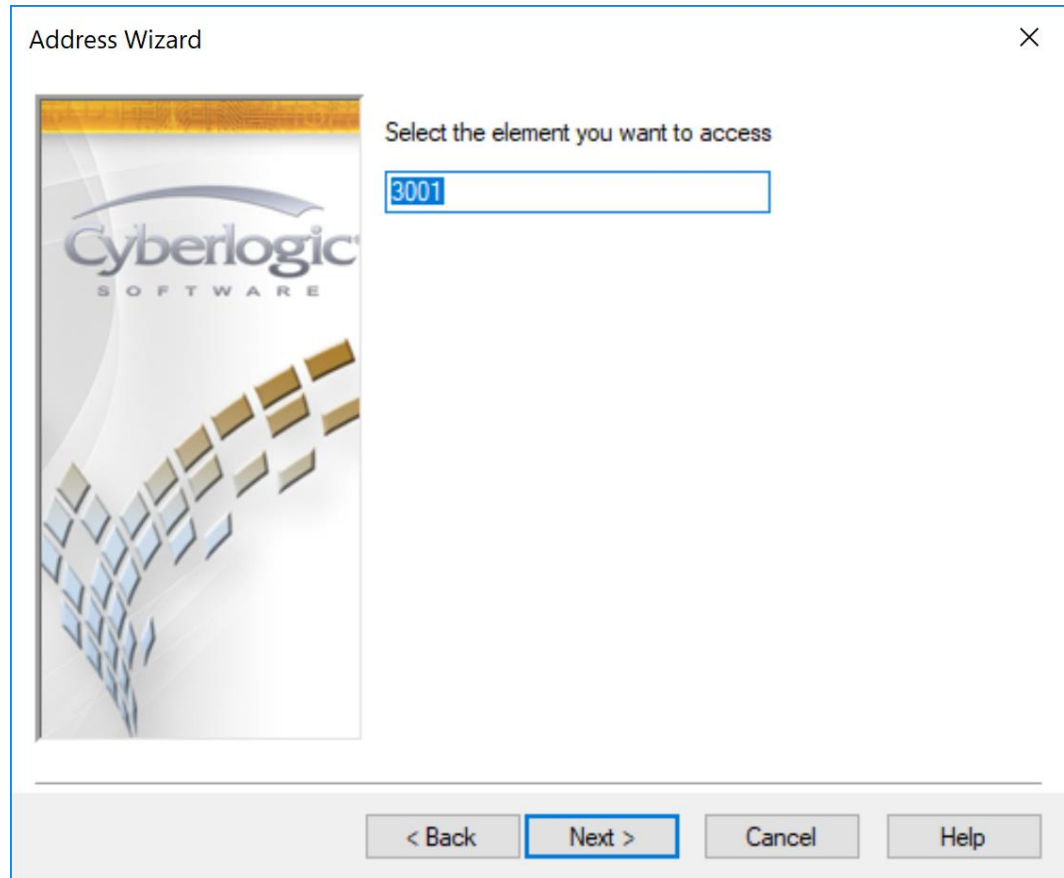
11. Click ***Finish***.

### **Example 3: OMNI Flow Controller Bit Field**

This example shows how to set up the addressing to allow you to access a few bits inside a register as a number (bit field). In this example, the Address Map field on the MBX Device associated with this data item is set to OMNI Flow Controller.



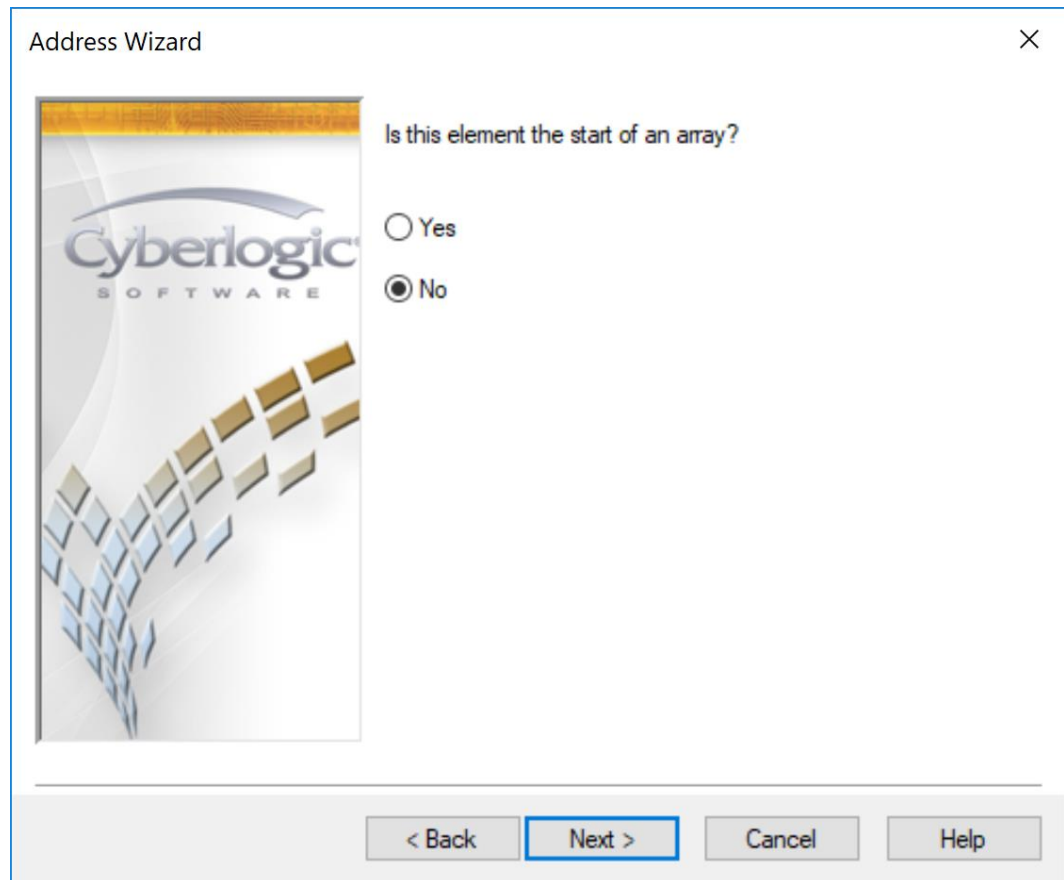
1. From the Welcome screen, click Next to continue.



On this screen, you must select the specific register you want to access.

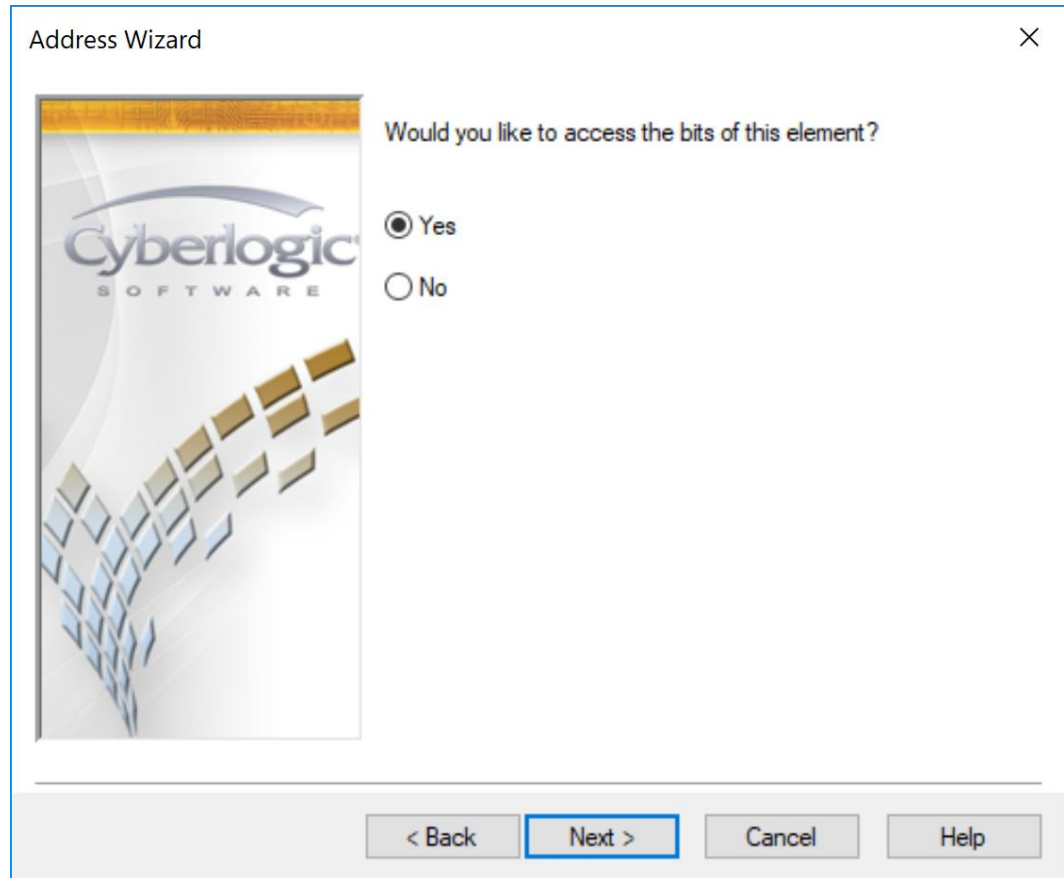
2. Enter 3001 as the element number, to select register 3001.
3. Click ***Next***.





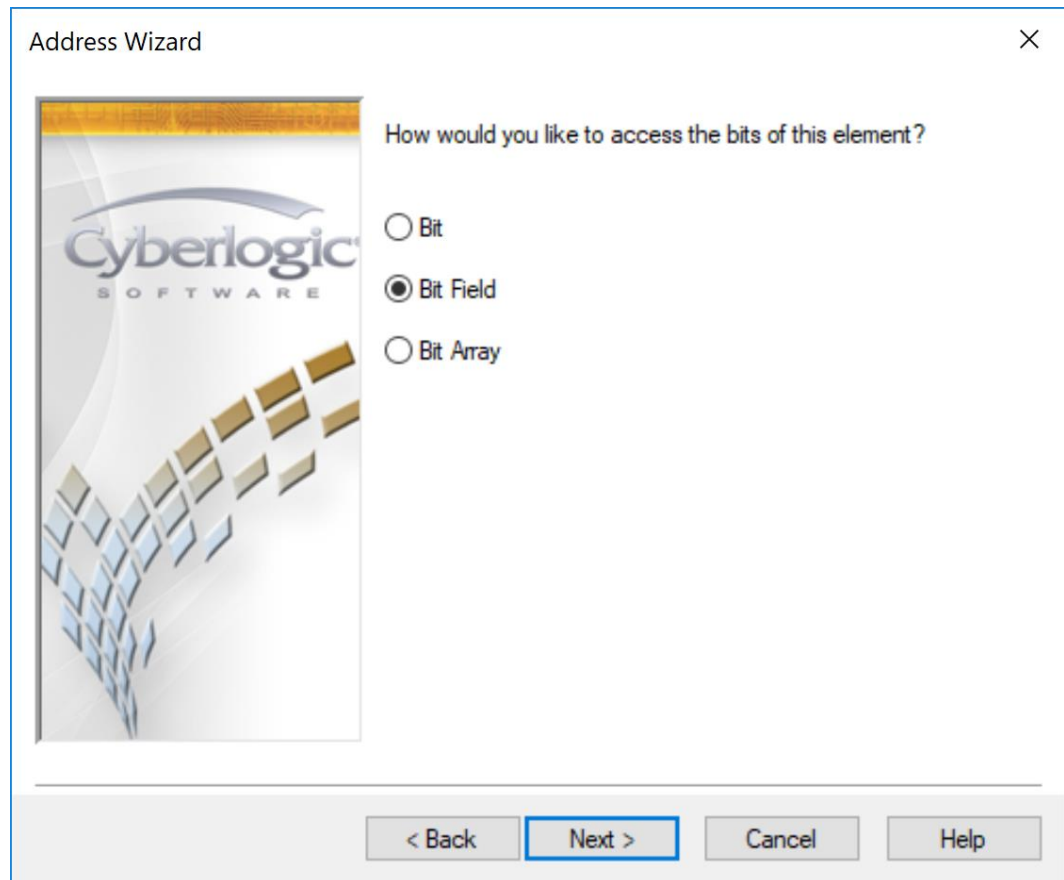
For this example, we do not want to access the data as an array.

4. Select **No**.
5. Click **Next**.



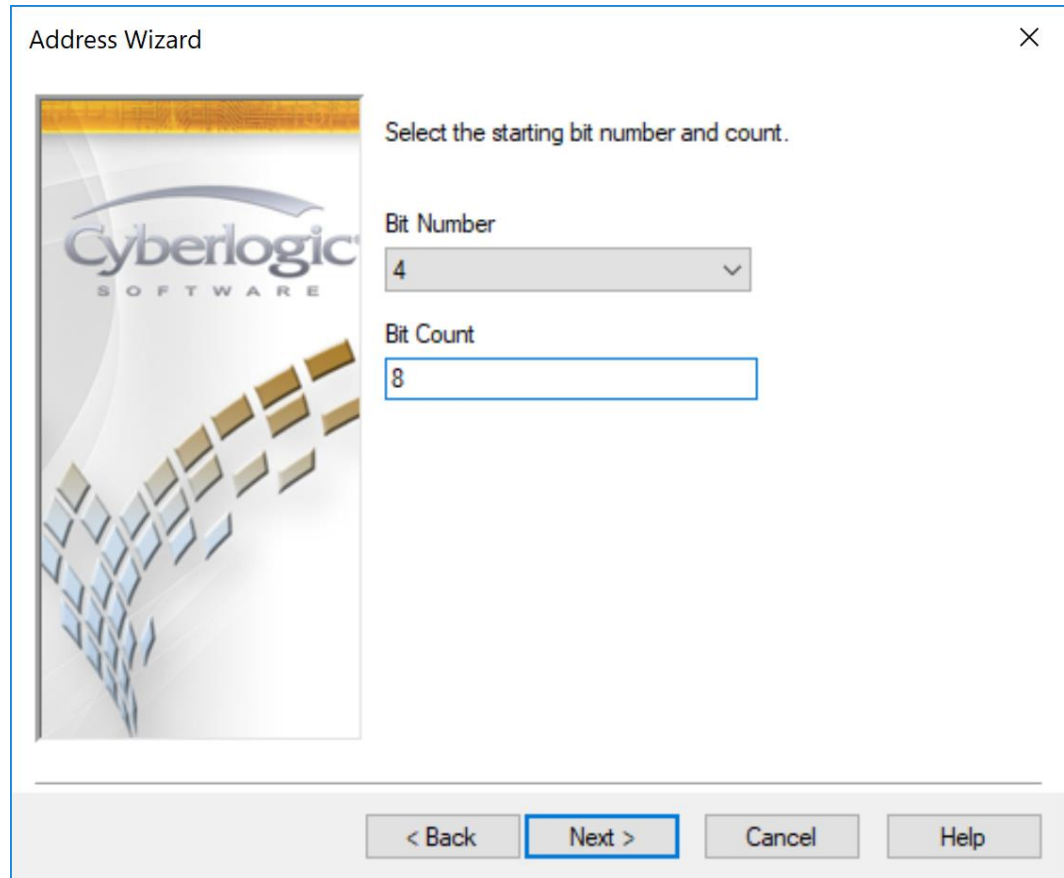
It is possible to access individual bits or groups of bits within the register.

6. Select **Yes**.
7. Click **Next**.



You can access an individual bit or a group of bits as either a bit field or an array of bits.

8. Select **Bit Field**.
9. Click **Next**.



Address Wizard

Select the starting bit number and count.

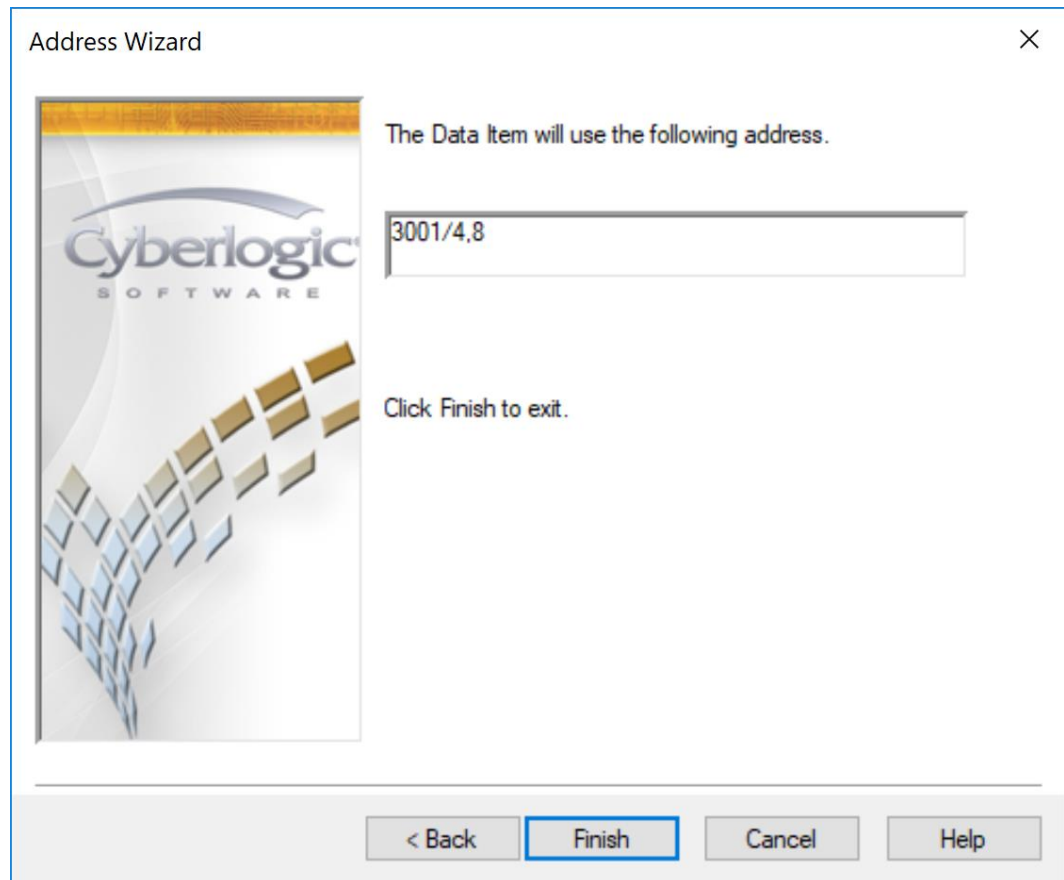
Bit Number  
4

Bit Count  
8

< Back   Next >   Cancel   Help

In this example, we will select 8 bits starting from bit 4.

10. Select 4 for the **Bit Number**.
11. Enter 8 for the **Bit Count**.
12. Click **Next**.



This screen shows the result of your selections. The register address to be created is 3001/4,8.

13. Click ***Finish***.

## APPENDIX D: UNSOLICITED MESSAGE PROGRAMMING

The MBX Driver Agent accepts unsolicited read and write messages. The read requests are used to read the content of the server status block. The write requests allow trusted sources to update data in selected data items. The user identifies the trusted sources by configuring the [Unsolicited Message Filters](#).

Programmable controllers can send unsolicited messages by activating MSTR block instructions in their logic program. For more information on how to send Modbus messages from various programmable controllers, refer to the appropriate Schneider documentation.

Custom applications written in C/C++ can use the [MBX SDK](#) to send communication requests over Modbus, Modbus Plus and Ethernet networks.

### ***Read Requests***

The read requests are used to read the content of the server status block.

The MBX Driver Agent accepts the Read Holding Registers (Fnc 3) and Read/Write 4X Registers (Fnc 23) query messages with the read register addresses in the range of 400001-400006.

### ***Write Requests***

Write requests update data in data items configured for an unsolicited update. A single message can update data in several data items. However, the message must contain data for the entire data item. Partial data updates are not allowed.

There are a number of query messages that can be used depending on the register address and the data type assigned to the data item. The following sections describe all supported Modbus functions.

#### *Force Single Coil (Fnc 5)*

This function can be used to update data in data items that represent a single coil.

#### *Force Multiple Coils (Fnc 15)*

This function can be used to update data in data items that represent either a single coil or multiple coils. The maximum number of coils that can be sent by this query message is 1968.

#### *Preset Single Register (Fnc 6)*

This function can be used to update data in data items that require no more than a single holding register. That includes bit-fields in the range of 1-16 bits.

*Preset Multiple Registers (Fnc 16)*

This function can be used to update data in data items that range in size from 1 bit to 120 sixteen-bit holding registers.

*Write General Reference (Fnc 21)*

This function can be used to update data in data items that range in size from 1 bit to 122 sixteen-bit general reference registers.

*Mask Write 4X registers (Fnc 22)*

This function can be used to update data in data items that require no more than a single holding register. Since individual bits within a register can be modified, this function is typically used for bit-fields that fit into a single register.

*Read/Write 4X Registers (Fnc 23)*

This function can be used to update data in data items that range in size from 1 bit to 118 sixteen-bit holding registers.

## **APPENDIX E: CONFIGURING MAXIMUM CONCURRENT REQUESTS**

The Maximum Network Requests and Maximum Node Requests settings limit the number of transactions that the network connection or network node will process simultaneously. Selecting the proper values for these limits can be crucial in obtaining peak performance from your OPC server. The server uses these limits to determine the best way to balance the network load while ensuring an optimum level of communication with each network node.

Although the default values will work well for most common network layouts, some cases may require a few adjustments. In this appendix, we will discuss how to decide on the proper settings to use, and will provide examples of some typical kinds of configurations.

### ***Two Levels of Limits***

The maximum request values are set at both the network connection level and at the network node level.

The maximum network request limit is entered on the [Settings Tab](#) of the network connection. This parameter limits the total number of simultaneous transactions for all network nodes on that network connection.

The maximum node request limit is entered on the [Optimizations Tab](#) of the network node. This parameter allows you to limit the number of simultaneous transactions for that individual network node. If Unlimited is selected, the number of transactions is limited only by the value set for the network connection.

These parameters interact, so each must be taken into consideration when setting the other. For example, the number of simultaneous transactions actually handled by a network node will never be greater than the lower of the two numbers. Furthermore, the number of simultaneous transactions handled by a network connection will never be greater than the total of the limits set on the nodes on that network.

In addition, you must remember that the resources available to process transactions through the network connection are shared among the network nodes. This further restricts the number of transactions that may be available for communication to a given network node. As the client applications request data from the various network nodes, the server arbitrates these competing requests, allowing them to be processed as resources become available. Thus, the number of transactions being processed by each network node will constantly be changing, but each network node will never process more than its limit, and all of them combined will never process more than the limit for the network connection.

### ***Resources and Performance***

The purpose of these parameter settings is to allow the user to allocate system resources and network bandwidth in a way that will yield the best performance. If the concurrent request limit is set too low, throughput will suffer even though there are unused resources available to the server.



However, choosing the maximum setting doesn't always improve performance, and may actually make things worse. There are a number of situations in which you should lower these limits.

- *Bridging to other networks.* If you have two or more different networks bridged together, and the messages must pass through more than one network, you should choose values consistent with the slowest network.
- *Slow PLCs.* Some older PLCs may be able to connect to fast networks, yet they run relatively slowly in their internal processing of messages. Even newer controllers may be bogged down by very large or complex programming. Network nodes like these may not be capable of handling the messages as fast as the network can pass them. A lower limit on simultaneous transactions would keep the PLCs from being overloaded with messages.
- *Other software using the network.* If you have applications other than the OPC server communicating on the network, you may find that the OPC server consumes so much bandwidth that the other applications cannot run satisfactorily. For example, with the OPC server running, your programming software may not be able to connect to the PLC, or uploads and downloads may take excessively long. You may wish to lower the OPC server's limits to prevent the network from saturating and permit the programming software to run.

The goal is to adjust these limits to a level that will allocate the system resources most effectively.

### ***Example 1: Network Node Set to Lower Limit***

A network connection is configured for a maximum of 16 network requests and one of its nodes is configured for a maximum of 4.

The limit for that node will be 4 simultaneous transactions, regardless of the fact that the network connection can handle more. This means that there will always be at least 12 concurrent requests available to other network nodes using that network connection.

### ***Example 2: Network Node Set to Unlimited***

A network connection is configured for a maximum of 8 network requests and one of its nodes is configured as Unlimited.

The maximum number for that node would be 8, that is, everything the network connection can handle. If the network connection limit is increased to 16, then the limit for that network node would increase as well.

### ***Example 3: Interaction Between Multiple Network Nodes***

The network connection is configured for a maximum of 20 network requests. It serves five network nodes, each with a limit of 8.

The total of the limits for the five network nodes is then 40, twice what the network connection can handle. This simply means that it will not be possible run the limit of 8 requests on all five network nodes at the same time.

In this configuration, no more than two network nodes could run at their limit of 8 simultaneous transactions, and occasionally one or two might reach that limit. However, the server will always try to spread the load equally among all nodes, so most of the time all nodes would be running at about 4 simultaneous transactions.

#### ***Example 4: Network Nodes with Varying Speeds***

The network connection is configured for a maximum of 10 network requests. It serves three network nodes, one of which is limited to a maximum of 8 node requests, and the other two limited to 2.

The benefit of this configuration is that it avoids the situation where the slower nodes are overwhelmed with transactions they cannot handle, while the faster node is deprived of transactions that it could handle. When the slower nodes are each handling two transactions, the remaining 6 that the network can handle will be available to the faster node.

#### ***Example 5: Bridging to a Slower Network for All PLCs***

The network connection provides connectivity to an Ethernet network, but all of the messages pass through a bridge to a Modbus Plus network before reaching the PLCs.

The recommended range for the maximum network requests on an Ethernet (Modbus TCP) network is 16-32, but the range for Modbus Plus is 8-16. In this situation, the slower network determines the throughput, so the correct range to use is the lower of the two. You should select a value between 8 and 16.

#### ***Example 6: Bridging to a Slower Network for Some PLCs***

The network connection provides connectivity to an Ethernet network, and some of the PLCs are directly connected to Ethernet. For others, the messages must pass through a bridge to reach the PLCs via a serial Modbus connection.

The recommended range for the maximum network requests on an Ethernet (Modbus TCP) network is 16-32, but the range for Modbus is 2-4. You can set up the network connection for the Ethernet range of 16-32, and do the same for the network nodes that are directly connected to the Ethernet, perhaps specifying them as Unlimited. For the network nodes that use Modbus, you must set the maximum node requests in the 2-4 range, to accommodate the slower connection they use.

## **APPENDIX F: WRITING TO PLC INPUTS**

The PLC inputs (1xxxx) and input registers (3xxxx) are normally updated only by the PLC itself, and therefore cannot be written to externally though the use of normal data type messages. However, PLC programming packages do have the capability of writing to inputs and input registers. They use special programming messages to do this.

The MBX OPC Driver Agent can write data to the inputs and input registers of Modicon controllers by acting as though it is a PLC programmer. This is not the usual mode of operation, and there are a number of restrictions. This appendix explains how to perform these write operations for those who want to use this capability.

### ***Preparation, Restrictions and Limitations***

The programming messages that the driver agent uses have limitations that impose certain requirements and restrictions on the use of Modicon input write operations.

- You must first use your PLC programming software to disable all 1xxxx inputs that you want to write to.
- You must then detach the programming software from the PLC. You cannot write to a PLC that has any programming software attached to it.
- You can write to input register bits (such as 30001/00) and bit fields (such as 30001/00,3) only when the PLC is in program mode. However, you can always write to full 3xxxx input registers.
- Due to the programming message restrictions, programming messages can be processed only one at a time. In comparison, multiple data type messages can be processed simultaneously. As a result, you will observe significantly lower performance when writing to inputs and input registers.

### ***Writing to Multiple Nodes***

For Ethernet and serial Modbus, there are no restrictions on the number of PLCs you can write to.

The Modbus Plus driver, however, uses a fixed number of Program Master (PM) paths to permit it to write to that number of PLCs simultaneously. The default value is six. You can initiate more than six writes, but only six will proceed at a time. The rest will be queued up for execution after earlier writes complete and their PM paths are released.

The PM path usage works like this:

1. The driver sends a login message and then sends the data.
2. The driver waits briefly to see if there is more data to send. This delay is configurable, and the default is one second.
3. The driver then sends a logout message.
4. The PM path is quarantined for eleven seconds. During this time, it can be used for another write to the same PLC, but is not available for writes to any other PLC.

5. The PM path is released and the driver can use it to write to any PLC.

Note that if you repeatedly write to the same PLC less than eleven seconds after the driver logs out from the previous message, the PM path will never be released. It will never be available for writes to any other PLC. Therefore, if you plan to write to more PLCs than the available number of PM paths, you must be sure to allow enough time for the PM paths to be released.

**Note**

The eleven-second quarantine is enforced in hardware by each Modbus Plus adapter. It cannot be shortened.

### ***Performance Considerations under Modbus Plus***

The Program Master paths work in a way that requires extra care in configuring Modbus Plus communication. Here are the issues you must consider to obtain the best performance from your system.

#### *Number of PM Paths*

There are a total of eight PM paths available, but the software defaults to using a maximum of six. This leaves two PM paths available for other applications, such as PLC programming or bridging software.

- You can increase this setting to allow more simultaneous write operations, but that may prevent other applications from communicating.
- You can decrease this setting to free more PM paths for other applications, but that will reduce the number of simultaneous writes that will be possible.

#### *PM Path Hold Time*

When the driver writes to a PLC, it first sends a message to log-in the PM path, then it sends one or more messages to write the data. After all data messages have been processed, it will hold onto the PM path for a time before it sends the logout message. If a write request arrives before this delay is complete, the driver will not logout, but will simply send more data.

- You can increase this delay to keep the PM path open and reduce the number of logout/login messages that the driver will send. If you expect to rapidly request writes to the same PLC, one after another, this can reduce overhead and improve performance.
- You can decrease this delay so that the driver will release the PM path more quickly after it writes the data. If you expect to request writes to many different PLCs, this can reduce the time each will have to wait for an available PM path.

### ***Configurable Settings***

As mentioned previously, there are two settings you can configure.

**Caution!**

This topic contains instructions for editing the Windows Registry. Editing the Registry incorrectly can cause serious, system-wide problems that may require you to re-install Windows to correct them. Edit the registry at your own risk. Always make a backup of the registry before you make any changes. If you do not feel comfortable editing the registry, do not attempt these instructions. Instead, seek the help of a trained computer specialist.

Maximum PM Paths

This setting specifies the maximum number of Program Master paths that the driver may use for writing input values. The possible range is 1 – 8, and the default value is 6.

The value is set in the Windows Registry. Open the Registry Editor and go to the specified key.

On 32-bit systems:

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Cyberlogic\MBX OPC Server*

On 64-bit systems:

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Cyberlogic\MBX OPC Server*

Create a registry entry called *MaxPmPaths* of type *REG\_DWORD*. You may then set *MaxPmPaths* to the desired value.

**Caution!**

After you edit *MaxPmPaths*, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the change you have made to take effect.

PM Path Hold Time

This setting specifies how long the driver will wait, after it writes the data, before sending the logout message. The delay is specified in milliseconds, and the default value is 1000. A value of 0 is permitted, but is not recommended. It will cause the driver to logout as soon as it writes the data, which may seriously degrade performance.

The value is set in the Windows Registry. Open the Registry Editor and go to the specified key.

On 32-bit systems:

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Cyberlogic\MBX OPC Server*

On 64-bit systems:

*HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Cyberlogic\MBX OPC Server*

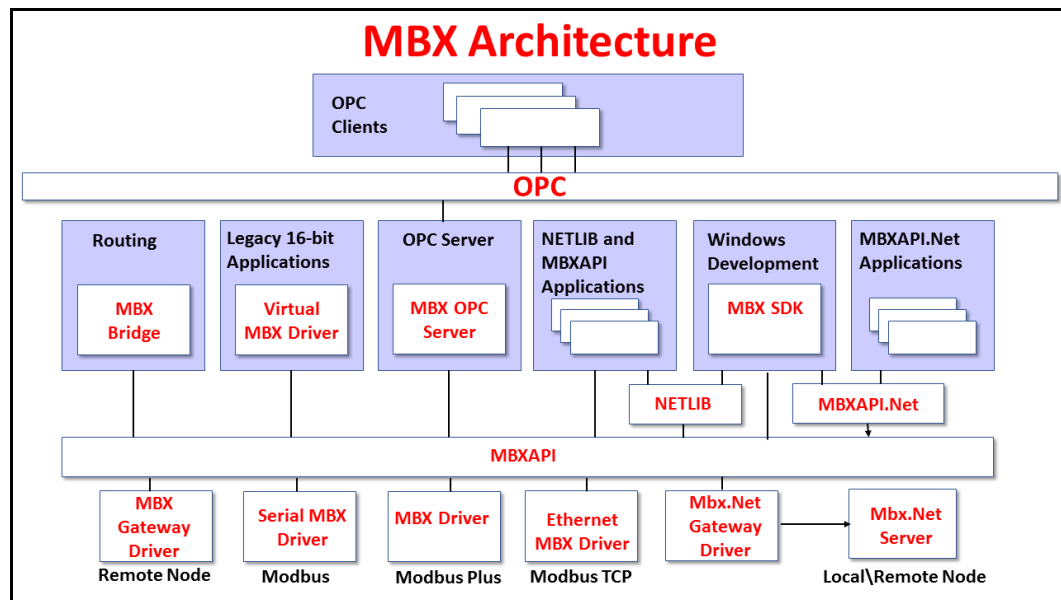
Create a registry entry called *PmPathHoldTime* of type *REG\_DWORD*. You may then set *PmPathHoldTime* to the desired value.

**Caution!** After you edit *PmPathHoldTime*, you must open the **File** menu and select **Save & Update Server**, or click the **Save & Update Server** toolbar button, for the change you have made to take effect.

## APPENDIX G: MBX ARCHITECTURE AND COMPANION PRODUCTS

The Ethernet MBX Driver is part of the Cyberlogic MBX family. This family consists of several well-integrated products that provide connectivity for Modbus, Modbus Plus and Modbus TCP (Ethernet) networks in distributed environments.

This section illustrates the layout of the MBX architecture. It includes a description of each MBX component along with suggested methods for employing them to support Modicon networks.



*The MBX architecture presents a consistent framework to address different connectivity needs.*

### MBX Driver

The MBX Driver provides connectivity between Modbus Plus interface adapters and Windows-based applications. It supports all Modbus Plus interface adapters for PCI Express (PCIe), PCI, USB and PCMCIA buses that are compatible with the supported operating systems. For a complete list of supported adapters, refer to the MBX Driver help file. Multiple interface cards can be installed at the same time, limited only by the number of available slots.

The kernel mode device driver of the MBX Driver is the highest-performance Modbus Plus driver in the industry. The driver operates in either interrupt or polled mode and fully implements all Modbus Plus features, providing support for Data Master/Slave, Program Master/Slave, Global Data and Peer Cop. The high-performance native API (MBXAPI) of the MBX Driver takes advantage of the event-driven, multitasking, multithreaded features of Windows operating systems.

The driver includes the MBX Gateway Server for remote access by the MBX Gateway Driver and is fully compatible with all other components of the MBX family.

The MBX Driver is included in the following products:

- MBX OPC Enterprise Suite
- MBX OPC Premier Suite
- MBX OPC Server Suite
- MBX Bridge Suite
- MBX Driver Suite

## **Ethernet MBX Driver**

The Cyberlogic Ethernet MBX Driver emulates Modbus Plus over the Modbus TCP protocol. This allows most Modbus Plus-compatible software to gain instant access to Modbus TCP-enabled devices without code modifications. It is compatible with all Ethernet cards supported by Windows.

The driver includes the MBX Gateway Server for remote access by the MBX Gateway Driver and is fully compatible with all other components of the MBX family.

The Ethernet MBX Driver is included in the following products:

- MBX OPC Enterprise Suite
- MBX OPC Premier Suite
- MBX OPC Server Suite
- MBX Bridge Suite
- MBX Driver Suite

## **Serial MBX Driver**

The Serial MBX Driver provides connectivity to Modbus-compatible devices through standard serial COM ports. It supports both master and slave node communications for Modbus ASCII and Modbus RTU protocols.

The driver includes the MBX Gateway Server for remote access by the MBX Gateway Driver and is fully compatible with all other components of the MBX family.

The Serial MBX Driver is included in the following products:

- MBX OPC Enterprise Suite
- MBX OPC Premier Suite
- MBX OPC Server Suite
- MBX Bridge Suite
- MBX Driver Suite (Some OEM versions do not include the Serial MBX Driver.)



## MBX Gateway Driver

The MBX Gateway Driver lets applications use MBX devices on remote MBX Gateway Server nodes as though they were on the local system. The client system running the MBX Gateway Driver must be a Windows node connected over a standard LAN to another system running the MBX Gateway Server. It can then access the Modbus, Modbus Plus and Modbus TCP networks that are connected to the server node.

For example, the MBX Gateway Driver provides complete MBX Driver functionality to the client node applications, including support for Data Master/Slave, Program Master/Slave, Global Data and Peer Cop. An interface adapter, such as a PCIe-85 card, is not required on the client node. MBX Gateway Driver nodes can communicate with multiple remote servers and all Windows-compatible TCP/IP networks are supported.

The MBX Gateway Driver is compatible with all other components of the MBX family.

The MBX Gateway Driver is included in the following products:

- MBX OPC Enterprise Suite
- MBX OPC Premier Suite
- MBX OPC Server Suite
- MBX Bridge Suite
- MBX Driver Suite

## Mbx.Net Gateway Driver

The Cyberlogic Mbx.Net Gateway Driver provides similar functionality to the Cyberlogic MBX Gateway Driver. However, the Mbx.Net Gateway Driver uses newer and more secure infrastructure, and allows for more communication protocols.

The driver connects existing applications that use the MBXAPI, MBXAPI.Net or NETLIB to the [Mbx.Net Server](#). The server can be on the local network or anywhere on the internet. The Mbx.Net Gateway node provides access to all MBX devices as though they were on the local system. The connection can be through Http, Net.Tcp or Net.Pipe protocols. A secure connection can be established and protected by a user name/password.

The Mbx.Net Gateway Driver is compatible with all other components of the MBX family.

The Mbx.Net Gateway Driver is included in the following products:

- Mbx.Net Suite
- Mbx.Net Premier Suite

## Virtual MBX Driver

The Virtual MBX Driver enables 16-bit NETLIB/NetBIOS-compatible applications, such as Modsoft and Concept, to run concurrently with 32-bit applications on the same computer.

It allows multiple 16-bit applications and multiple instances of a single 16-bit application to run under the latest 32-bit Windows operating systems.

If your computer uses a 64-bit edition of Windows, refer to Cyberlogic Knowledge Base article *KB2010-02 Running 16-Bit Applications* for important information on using the Virtual MBX Driver on your system.

The Virtual MBX Driver is fully compatible with all MBX components and requires at least one of these drivers to operate:

- MBX Driver
- Ethernet MBX Driver
- Serial MBX Driver
- MBX Gateway Driver
- Mbx.Net Gateway Driver

The Virtual MBX Driver is included in the following products:

- MBX OPC Enterprise Suite
- MBX OPC Premier Suite
- MBX OPC Server Suite
- MBX Bridge Suite
- MBX Driver Suite

## **MBX Bridge**

The MBX Bridge seamlessly routes messages between MBX-compatible devices. For example, the MBX Bridge can route messages between Ethernet and Modbus Plus networks, between Modbus and Modbus Plus networks or any other combination of the supported networks.

Depending on the user's needs, it requires one or more of the following drivers to operate:

- MBX Driver
- Ethernet MBX Driver
- Serial MBX Driver
- MBX Gateway Driver
- Mbx.Net Gateway Driver

The MBX Bridge is included in the MBX Bridge Suite.

## **MBX OPC Server**

The Cyberlogic MBX OPC Server connects OPC-compliant client applications to Modbus, Modbus Plus and Modbus TCP networks. It supports the latest OPC Data Access and OPC

Alarms and Events specifications and uses the MBX drivers for connectivity to Modicon networks.

The MBX OPC Server supports multiple, priority-based access paths for reliable, redundant communications. It also supports both solicited and unsolicited communications and uses an advanced transaction optimizer to guarantee minimum load on your networks. With only a couple of mouse clicks, the MBX OPC Server will automatically detect and configure the attached networks and node devices. Other noteworthy features include DirectAccess, Data Write Protection and Health Watchdog.

The MBX OPC Server is included in the following products:

- MBX OPC Enterprise Suite
- MBX OPC Premier Suite
- MBX OPC Server Suite

## **MbpStat.Net**

The MbpStat.Net provides the same functionality as the original DOS-based MBPSTAT application. It can monitor local or remote Modbus Plus by connecting to systems running the Mbx.Net Server. It can run on 32-bit and 64-bit Windows, and it does not require the Virtual MBX Driver.

The MbpStat.Net is included in the following products:

- Mbx.Net Suite
- Mbx.Net Premier Suite

## **Mbx.Net Server**

The Cyberlogic Mbx.Net Server connects applications that use the MBXAPI, MBXAPI.Net or NETLIB to Modbus, Modbus Plus and Modbus TCP networks. The client applications and server can be on the local network or connected through the Internet.

The server is a Windows Communication Foundation (WCF) server that supports Http, Net.Tcp and Net.pipe protocols. It allows connections to be secured with a user name/password. When used with an application like MbpStat.Net it allows remote monitoring of networks and devices. Existing applications can use the server along with the Mbx.Net Gateway Driver to securely connect to remote devices.

The Mbx.Net Server is included in the following products:

- Mbx.Net Suite
- Mbx.Net Premier Suite

## **MBX SDK**

Software developers can use the MBX Software Development Kit to provide connectivity to Modbus, Modbus Plus and Modbus TCP networks from their 32-bit and 64-bit C/C++/C# applications.

The SDK supports NETLIB, and Cyberlogic's high-performance MBXAPI and MBXAPI.Net interfaces. NETLIB is an excellent bridge for developers who would like to port their 16-bit applications to the latest Windows environments. Developers of new applications can use any of the three interfaces. For a complete reference of all NETLIB library functions, refer to *Modicon IBM Host Based Devices User's Guide*, available from Schneider Electric (Order #890 USE 102 00).

Since all MBX family drivers are built on the same MBX architecture, applications developed with the MBX SDK can be used with all MBX family drivers and can execute under all current Windows operating systems.