ProSoft® TECHNOLOGY

Where Automation Connects.



MVI56E-MNETC/MNETCXT

ControlLogix® Platform

Modbus TCP/IP Client Enhanced Communication Module - Client/Server

October 13, 2025

USER MANUAL

Your Feedback Please

We always want you to feel that you made the right decision to use our products. If you have suggestions, comments, compliments or complaints about our products, documentation, or support, please write or call us.

How to Contact Us

ProSoft Technology, Inc.
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
ps.support@belden.com

MVI56E-MNETC/MNETCXT User Manual
For Public Use.

October 13, 2025

ProSoft Technology®, is a registered copyright of ProSoft Technology, Inc. All other brand or product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

Content Disclaimer

This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither ProSoft Technology nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. Information in this document including illustrations, specifications and dimensions may contain technical inaccuracies or typographical errors. ProSoft Technology makes no warranty or representation as to its accuracy and assumes no liability for and reserves the right to correct such inaccuracies or errors at any time without notice. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of ProSoft Technology. All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components. When devices are used for applications with technical safety requirements, the relevant instructions must be followed. Failure to use ProSoft Technology software or approved software with our hardware products may result in injury, harm, or improper operating results. Failure to observe this information can result in injury or equipment damage.

Copyright © 2025 ProSoft Technology, Inc. All Rights Reserved.



For professional users in the European Union

If you wish to discard electrical and electronic equipment (EEE), please contact your dealer or supplier for further information.



Warning – Cancer and Reproductive Harm – www.P65Warnings.ca.gov

Agency Approvals & Certifications

Please visit our website: www.prosoft-technology.com

Open-Source Information

Open-Source Software used in the product

The product contains, among other things, Open-Source Software files, as defined below, developed by third parties and licensed under an Open-Source Software license. These Open-Source Software files are protected by copyright. Your right to use the Open-Source Software is governed by the relevant applicable Open-Source Software license conditions. Your compliance with those license conditions will entitle you to use the Open-Source Software as foreseen in the relevant license. In the event of conflicts between other ProSoft Technology, Inc. license conditions applicable to the product and the Open-Source Software license conditions, the Open-Source Software conditions shall prevail. The Open-Source Software is provided royalty-free (i.e. no fees are charged for exercising the licensed rights). Open-Source Software contained in this product and the respective Open-Source Software licenses are stated in the module webpage, in the link Open-Source.
If Open-Source Software contained in this product is licensed under GNU General Public License (GPL), GNU Lesser General Public License (LGPL), Mozilla Public License (MPL) or any other Open-Source Software license, which requires that source code is to be made available and such source code is not already delivered together with the product, you can order the corresponding source code of the Open-Source Software from ProSoft Technology, Inc. - against payment of the shipping and handling charges - for a period of at least 3 years since purchase of the product. Please send your specific request, within 3 years of the purchase date of this product, together with the name and serial number of the product found on the product label to:

ProSoft Technology, Inc.
Director of Engineering
9201 Camino Media, Suite 200
Bakersfield, CA 93311
USA

Warranty regarding further use of the Open-Source Software

ProSoft Technology, Inc. provides no warranty for the Open-Source Software contained in this product, if such Open-Source Software is used in any manner other than intended by ProSoft Technology, Inc. The licenses listed define the warranty, if any, from the authors or licensors of the Open-Source Software. ProSoft Technology, Inc. specifically disclaims any warranty for defects caused by altering any Open-Source Software or the product's configuration. Any warranty claims against ProSoft Technology, Inc. in the event that the Open-Source Software contained in this product infringes the intellectual property rights of a third party are excluded. The following disclaimer applies to the GPL and LGPL components in relation to the rights holders:
"This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License and the GNU Lesser General Public License for more details."
For the remaining Open-Source components, the liability exclusions of the rights holders in the respective license texts apply. Technical support, if any, will only be provided for unmodified software.

Contents

Your Feedback Please 2
How to Contact Us 2
Content Disclaimer 2
Agency Approvals & Certifications 2
Open-Source Information 3
Open-Source Software used in the product 3
Warranty regarding further use of the Open-Source Software 3

1 Start Here 7

1.1 System Requirements 7
1.2 Setting Jumpers 8
1.3 Installing the Module in the Rack 9
1.3.1 Before You Import the Add-On Instruction 10
1.3.2 Creating the Module 11
1.3.3 Importing the Add-On Instruction 14
1.3.4 Adding Multiple Modules (Optional) 17
1.4 Downloading the Sample Program to the Processor 23

2 MVI56E-MNETC/MNETCXT Configuration 24

2.1 Using ProSoft Configuration Builder Software 24
2.1.1 Installing ProSoft Configuration Builder 24
2.1.2 Upgrading from MVI56-MNETC in ProSoft Configuration Builder 25
2.1.3 Setting Up the Project 26
2.1.4 Setting Module Parameters 27
2.1.5 Module 28
2.1.6 MNET Servers 33
2.1.7 MNET Client x 35
2.1.8 MNET Client x Commands 37
2.1.9 Static ARP Table 44
2.1.10 Ethernet Configuration 45
2.2 Connecting Your PC to the Module 46
2.2.1 Using CIPconnect to Connect to the Module 46
2.2.2 Using RSWho to Connect to the Module 56
2.2.3 Connecting Your PC to the Module's Ethernet Port 57
2.3 Downloading the Project to the Module 60

3 Using Controller Tags 62

3.1 Controller Tags 62
3.1.1 MVI56E-MNETC Controller Tags 63
3.2 User-Defined Data Types (UDTs) 64
3.2.1 MVI56E-MNETC User-Defined Data Types 64
3.3 Controller Tag Overview 66
3.3.1 MNETC.DATA 66
3.3.2 MNETC.CONTROL 69
3.3.3 MNETC.STATUS 69
3.3.4 MNETC.UTIL 70

4 Diagnostics and Troubleshooting 71

4.1 Ethernet LED Indicators 71
4.1.1 Scrolling LED Status Indicators 72
4.1.2 Non-Scrolling LED Status Indicators 73
4.2 Clearing a Fault Condition 73
4.3 Troubleshooting the LEDs 74
4.4 Using the Diagnostics Menu in ProSoft Configuration Builder 75
4.4.1 The Diagnostics Menu 78
4.4.2 Monitoring Module Information 79
4.4.3 Monitoring Backplane Information 80
4.4.4 Monitoring Database Information 81
4.4.5 Monitoring MNETC Server Information 82
4.4.6 Monitoring MNET Client Information 82
4.5 Reading Status Data from the Module 83
4.5.1 Status Data Definition 84
4.5.2 Configuration Error Word 86
4.5.3 Client Command Errors 87
4.6 Connecting to the Module's Webpage 89

5 Reference 90

5.1 Product Specifications 90
5.1.1 General Specifications 90
5.1.2 Functional Specifications 91
5.1.3 Hardware Specifications 92
5.2 Functional Overview 92
5.2.1 Backplane Data Transfer 92
5.2.2 Normal Data Transfer Blocks 95
5.2.3 Special Function Blocks 99
5.2.4 Data Flow between MVI56E-MNETC/MNETCXT Module and Processor 117
5.3 Ethernet Cable Specifications 121
5.3.1 Ethernet Cable Configuration 121
5.3.2 Ethernet Performance 121
5.4 Modbus Protocol Specification 122
5.4.1 About the Modbus Protocol 122
5.4.2 Read Coil Status (Function Code 01) 123
5.4.3 Read Input Status (Function Code 02) 124
5.4.4 Read Holding Registers (Function Code 03) 125
5.4.5 Read Input Registers (Function Code 04) 126
5.4.6 Force Single Coil (Function Code 05) 127
5.4.7 Preset Single Register (Function Code 06) 128
5.4.8 Read Exception Status (Function Code 07) 128
5.4.9 Diagnostics (Function Code 08) 129
5.4.10 Force Multiple Coils (Function Code 15) 131
5.4.11 Preset Multiple Registers (Function Code 16) 132
5.4.12 Modbus Exception Responses 133
5.5 Using the Optional Add-On Instruction 135
5.5.1 Before You Begin 135
5.5.2 Overview 135
5.5.3 Importing the Optional Add-On Instruction 136
5.5.4 Reading the Ethernet Settings from the Module 141
5.5.5 Writing the Ethernet Settings to the Module 142
5.5.6 Reading the Clock Value from the Module 143
5.5.7 Writing the Clock Value to the Module 144

5.6 Adding the Module to an Existing Project 145
5.7 Using the Sample Program 148
5.7.1 Opening the Sample Program in RSLogix 148
5.7.2 Choosing the Controller Type 150
5.7.3 Selecting the Slot Number for the Module 151
5.7.4 Downloading the Sample Program to the Processor 152
5.7.5 Adding the Sample Ladder to an Existing Application 152

6 Support, Service & Warranty 153

6.1 Contacting Technical Support 153
6.2 Warranty Information 153

1 Start Here

To get the most benefit from this User Manual, you should have the following skills:

- **Rockwell Automation® RSLogix™ software:** launch the program, configure ladder logic, and transfer the ladder logic to the processor
- **Microsoft Windows®:** install and launch programs, execute menu commands, navigate dialog boxes, and enter data
- **Hardware installation and wiring:** install the module, and safely connect Modbus TCP/IP and ControlLogix devices to a power source and to the MVI56E-MNETC/MNETCXT module's application port(s)

1.1 System Requirements

The MVI56E-MNETC/MNETCXT module requires the following minimum hardware and software components:

- Rockwell Automation ControlLogix® processor (firmware version 10 or higher) with compatible limited voltage power supply and one free slot in the rack for the MVI56E-MNETC/MNETCXT module. The module requires 800mA of available 5 VDC and 3 mA of available 24 VDC power.
- Rockwell Automation RSLogix 5000 programming software
  - Version 16 or higher required for Add-On Instruction
  - Version 15 or lower must use Sample Ladder, available from www.prosoft-technology.com
- Rockwell Automation RSLinx® communication software version 2.51 or higher
- ProSoft Configuration Builder (PCB) (included)
- Pentium® II 450 MHz minimum. Pentium III 733 MHz (or better) recommended
- Supported operating systems:
  - Microsoft Windows 10
  - Microsoft Windows 7 Professional (32-or 64-bit)
  - Microsoft Windows 2000 Professional with Service Pack 1, 2, or 3
  - Microsoft Windows Server 2003
- 128 Mbytes of RAM minimum, 256 Mbytes of RAM recommended
- 100 Mbytes of free hard disk space (or more based on application requirements)

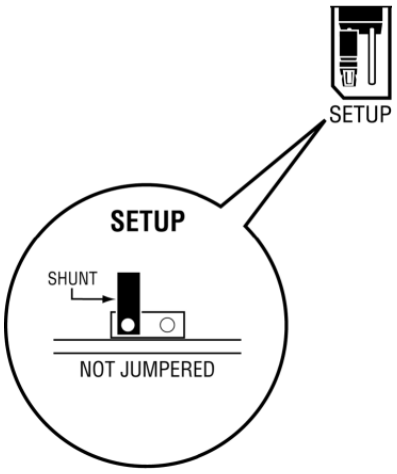
Note: The Hardware and Operating System requirements in this list are the minimum recommended to install and run software provided by ProSoft Technology®. Other third-party applications may have different minimum requirements. Refer to the documentation for any third-party applications for system requirements.

Note: You can install the module in a local or remote rack. For remote rack installation, the module requires EtherNet/IP or ControlNet communication with the processor.

1.2 Setting Jumpers

The Setup Jumper acts as "write protection" for the module's firmware. In "write protected" mode, the Setup pins are not connected, and the module's firmware cannot be overwritten. The module is shipped with the Setup jumper OFF. Do not jumper the Setup pins together unless you are directed to do so by ProSoft Technical Support (or you want to update the module firmware).

The following illustration shows the MVI56E-MNETC/MNETCXT jumper configuration with the Setup Jumper OFF.



Note: If you are installing the module in a remote rack, you may prefer to leave the Setup pins jumpered. You can update the module's firmware without requiring physical access to the module.

Security considerations:

Leaving the Setup pin jumpered leaves the module open to unexpected firmware updates.

Consider segmenting the data flow for security reasons. Per IEC 62443-1-1, you should align with IEC 62443 and implement segmentation of the control system. Relevant capabilities are firewalls, unidirectional communication, DMZ. Oil and Gas customers should also see DNVGL-RP-G108 for guidance on partitioning.

Practice security by design, per IEC 62443-4-1, including layers of security and detection. The module relies on overall network security design, as it is only one component of what should be a defined zone or subnet.

1.3 Installing the Module in the Rack

Make sure your ControlLogix processor and power supply are installed and configured, before installing the MVI56E-MNETC/MNETCXT module. Refer to your Rockwell Automation product documentation for installation instructions.

Warning: You must follow all safety instructions when installing this or any other electronic devices. Failure to follow safety procedures could result in damage to hardware or data, or even serious injury or death to personnel. Refer to the documentation for each device you plan to connect to verify that suitable safety procedures are in place before installing or servicing the device.

After you have checked the placement of the jumpers, insert the MVI56E-MNETC/MNETCXT into the ControlLogix chassis. Use the same technique recommended by Rockwell Automation to remove and install ControlLogix modules.

Warning: When you insert or remove the module while backplane power is on, an electrical arc can occur. An electrical arc can cause personal injury or property damage by sending an erroneous signal to the system's actuators. This can cause unintended machine motion or loss of process control. Electrical arcs may also cause an explosion when they happen in a hazardous environment. Verify that power is removed or the area is non-hazardous before proceeding.

Repeated electrical arcing causes excessive wear to contacts on both the module and its mating connector. Worn contacts may create electrical resistance that can affect module operation.

Note: If the module is improperly inserted, the system may stop working or may behave unpredictably.

Note: When using the XT version (if applicable), you must use the 1756-A5XT or 1756-A7LXT chassis to uphold the XT specifications. In these chassis, modules are spaced further apart than in standard ControlLogix chassis. Blank spacers are inserted between active modules.

1.3.1 Before You Import the Add-On Instruction

Note: This section only applies if your processor is using RSLogix 5000 version 16 or higher. If you have an earlier version, please see Using the Sample Program (page 148).

Two Add-On Instructions are provided for the MVI56E-MNETC/MNETCXT module. The first is required for setting up the module; the second is optional.

Download the files from www.prosoft-technology.com. Save them to a convenient location in your PC, such as *Desktop* or *My Documents*.

File Name	Description
MVI56EMNETC_AddOn_Rung_v1_x.L5X. A newer version may be available at: www.prosoft-technology.com	L5X file containing Add-On Instruction, user defined data types, controller tags and ladder logic required to configure the MVI56E-MNETC/MNETCXT module
MVI56EMNETC_Optional_AddOn_Rung_v1_x.L5X. A newer version may be available at: www.prosoft-technology.com	Optional L5X file containing additional Add-On Instruction with logic for changing Ethernet configuration and clock settings.

About the Optional Add-On Instruction

The Optional Add-On Instruction performs the following tasks:

- Read/Write Ethernet Configuration

  Allows the processor to read or write the module IP address, subnet mask, and network gateway IP address.

- Read/Write Module Clock Value

  Allows the processor to read and write the module clock settings. The module's free-running clock also stores the last time that the Ethernet configuration was changed or the last time the module was restarted or rebooted. The date and time of the last change or restart is displayed on the scrolling LED during module power-up/start-up sequence.

For more information, see Using the Optional Add-On Instruction (page 135).

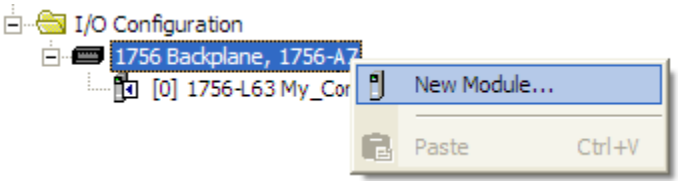
Note: You can also set the date and time from the module's web page. See Connecting to the Module's Webpage (page 88).

Important: The Optional Add-On Instruction supports only the two features listed above. You must use the regular MVI56E-MNETC/MNETCXT Add-On Instruction for all other features including backplane transfer and Modbus data communication.

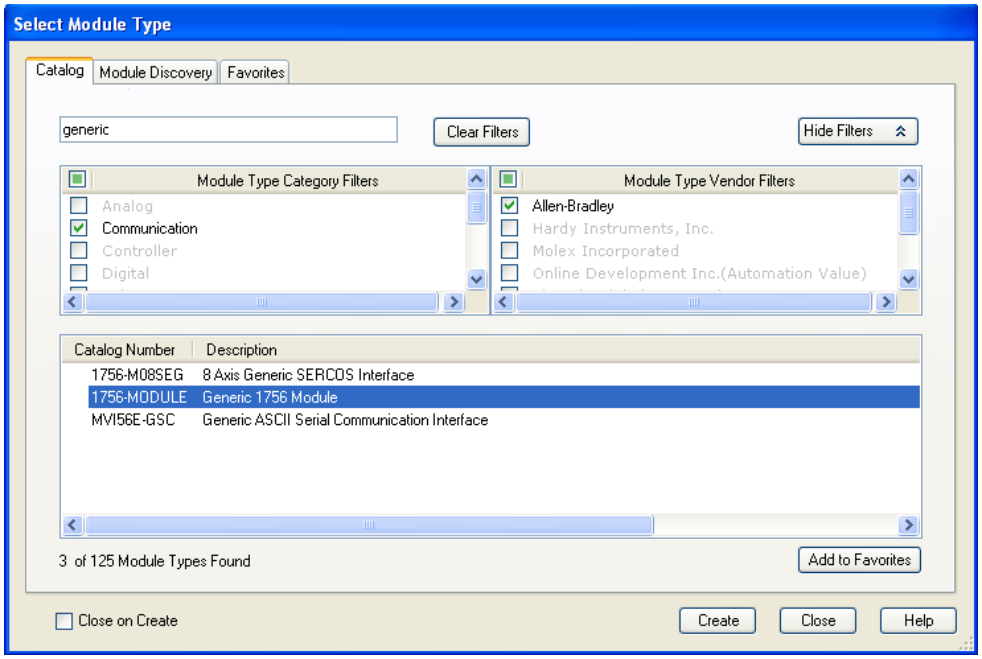
1.3.2 Creating the Module

1 Add the MVI56E-MNETC/MNETCXT module to the project.

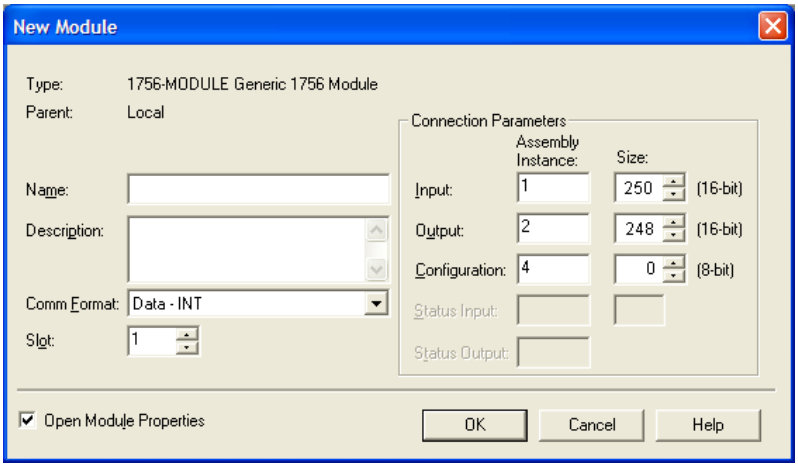
In the **CONTROLLER ORGANIZATION** window, select **I/O CONFIGURATION** and click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE...**



This action opens the **SELECT MODULE** dialog box. Enter *generic* in the text box and select the **GENERIC 1756 MODULE**. If you're using a controller revision of 16 or less, expand **OTHER** in the **SELECT MODULE** dialog box, and then select the **GENERIC 1756 MODULE**.



2 Click **CREATE.** This action opens the **NEW MODULE** dialog box.



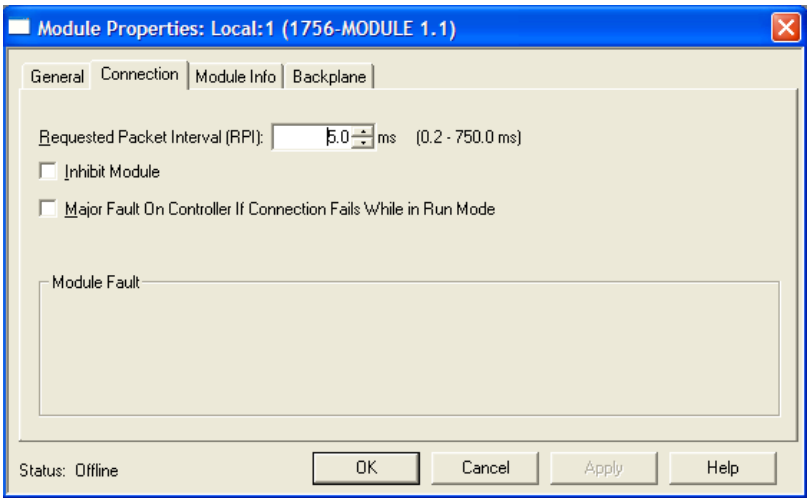
3 In the **NEW MODULE** dialog box, enter the following values.

Parameter	Value
NAME	Enter a module identification string. Example: MNETC
DESCRIPTION	Enter a description for the module. Example: Modbus TCP/IP Client Enhanced Communication Module - Client/Server
COMM FORMAT	Select **DATA-INT**
SLOT	Enter the slot number in the rack where the MVI56E-MNETC/MNETCXT module is located
INPUT ASSEMBLY INSTANCE	1
INPUT SIZE	250
OUTPUT ASSEMBLY INSTANCE	2
OUTPUT SIZE	248
CONFIGURATION ASSEMBLY INSTANCE	4
CONFIGURATION SIZE	0

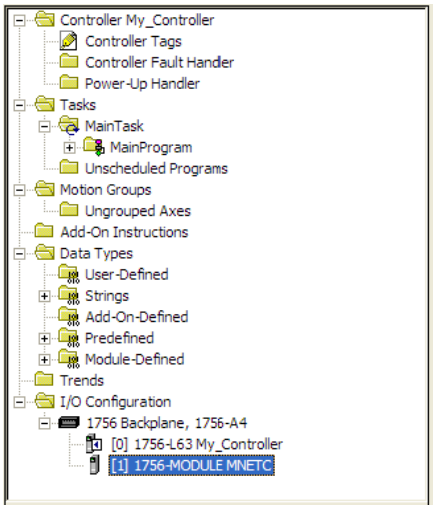
Important: You must select the **COMM FORMAT** as **DATA - INT** in the dialog box, otherwise the module will not communicate over the backplane of the ControlLogix rack.

4 Click **OK** to continue.

5 Edit the Module Properties. Select the **REQUESTED PACKET INTERVAL** value for scanning the I/O on the module. This value represents the minimum frequency at which the module will handle scheduled events. This value should not be set to less than 1 millisecond. The default value is 5 milliseconds. Values between 1 and 10 milliseconds should work with most applications.

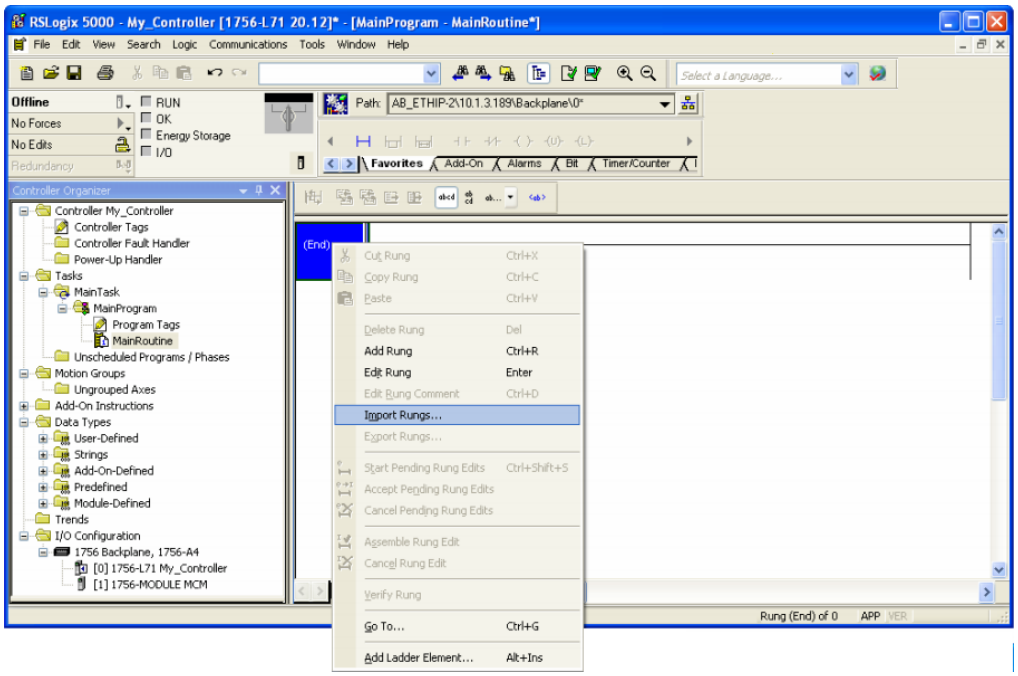


6 Click **OK** to save the module and close the dialog box. Notice that the module now appears in the **CONTROLLER ORGANIZATION** window.

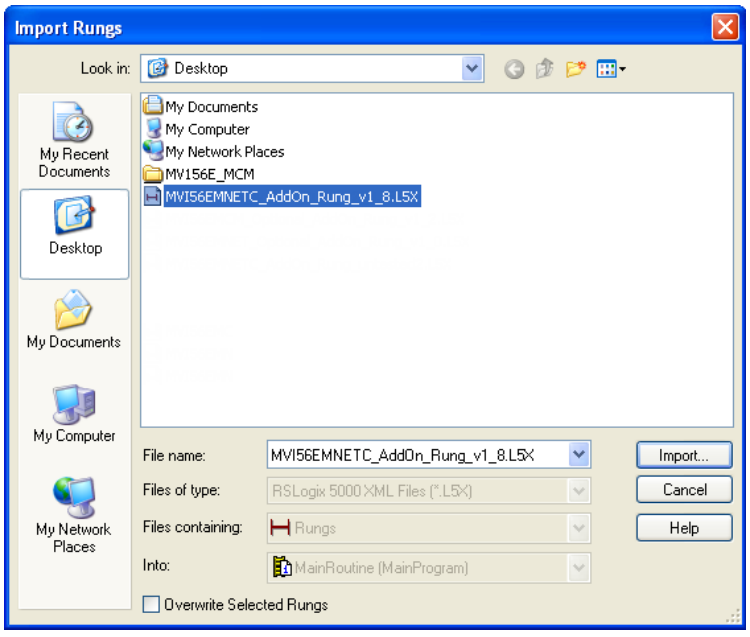


1.3.3 Importing the Add-On Instruction

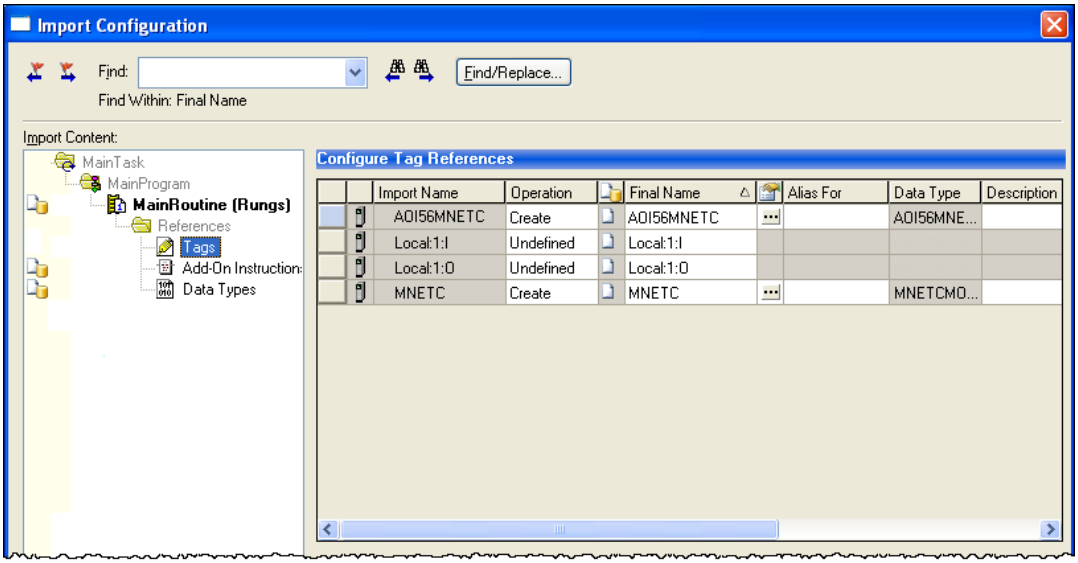
1. In the **CONTROLLER ORGANIZATION** window, expand the **TASKS** folder and subfolders until you reach the **MAINPROGRAM** folder.
2. In the **MAINPROGRAM** folder, double-click to open the **MAINROUTINE** ladder.
3. Select an empty rung in the routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNGS…**



4. Navigate to the location on your PC where you saved the Add-On Instruction (for example, *My Documents* or *Desktop*). Select the .L5X file.

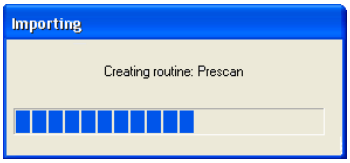


This action opens the **IMPORT CONFIGURATION** dialog box. Click **TAGS** under **MAINROUTINE** to show the controller tags that will be created. Note that if you are using a controller revision number of 16 or less, the **IMPORT CONFIGURATION** dialog box does not show the **IMPORT CONTENT** tree.

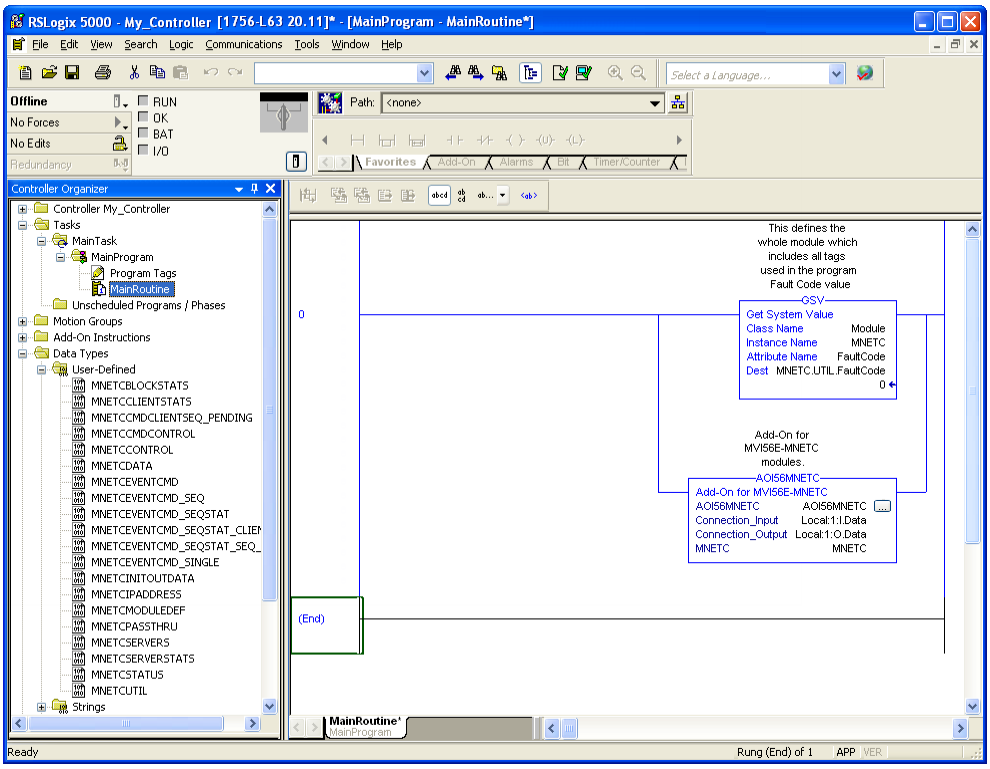


5 If you are using the module in a different slot (or remote rack), edit the connection input and output variables that define the path to the module. Edit the text in the **FINAL NAME** column **(NAME** column for controller revision 16 or less). For example, if your module is located in slot 3, change Local:1:I in the above picture to Local:3:I. Do the same for Local:1:O. If your module is located in Slot 1 of the local rack, this step is not required.

6 Click **OK** to confirm the import.



When the import is completed, the new rung with the Add-On Instruction appears as shown in the following illustration.



The procedure also imports the new User Defined Data Types, Controller Tags, and the Add-On instruction for your project.

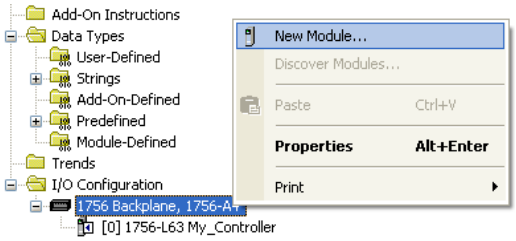


7 Save the application and then download the sample ladder logic to the processor.

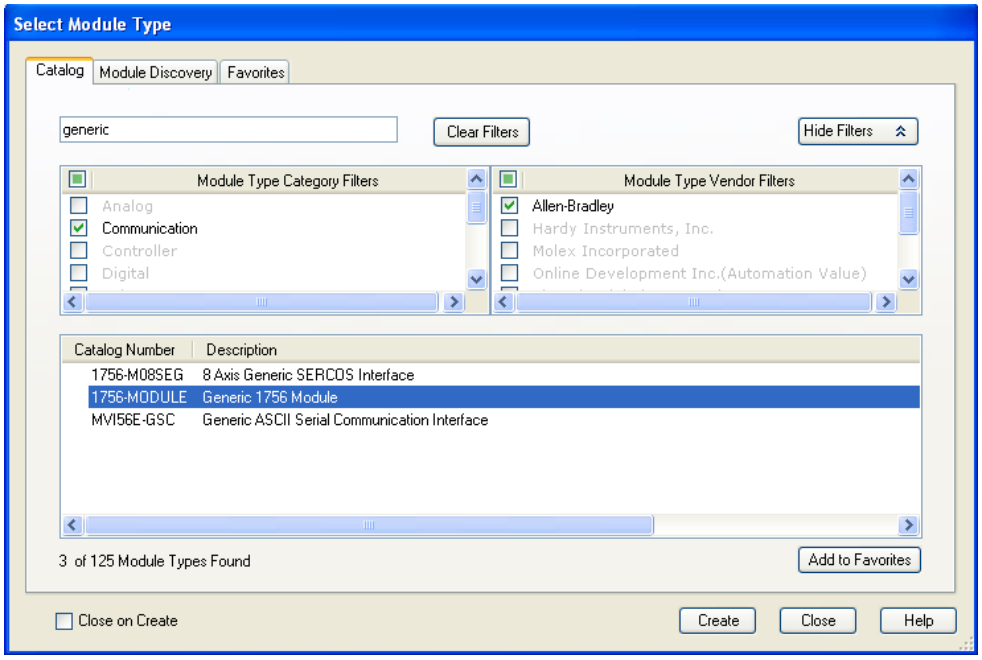
1.3.4 Adding Multiple Modules (Optional)

Important: If your application requires more than one MVI56E-MNETC/MNETCXT module in the same project, follow the steps below.

1 In the **I/O CONFIGURATION** folder, click the right mouse button to open a shortcut menu, and then choose **NEW MODULE**.



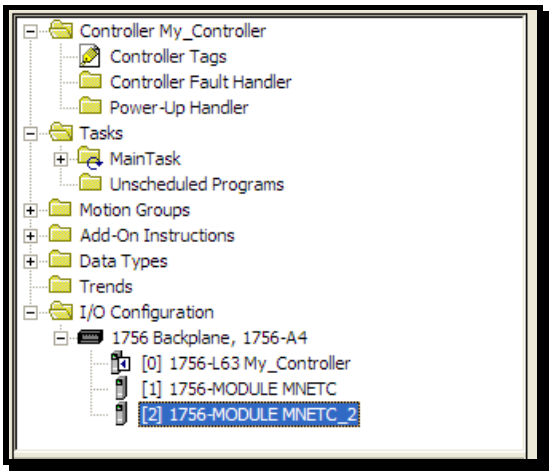
2 Select **1756-MODULE**. If you're using a controller revision of 16 or less, expand **OTHER** in the **SELECT MODULE** dialog box, and then select the **1756-MODULE**.



3 Fill the module properties as follows:

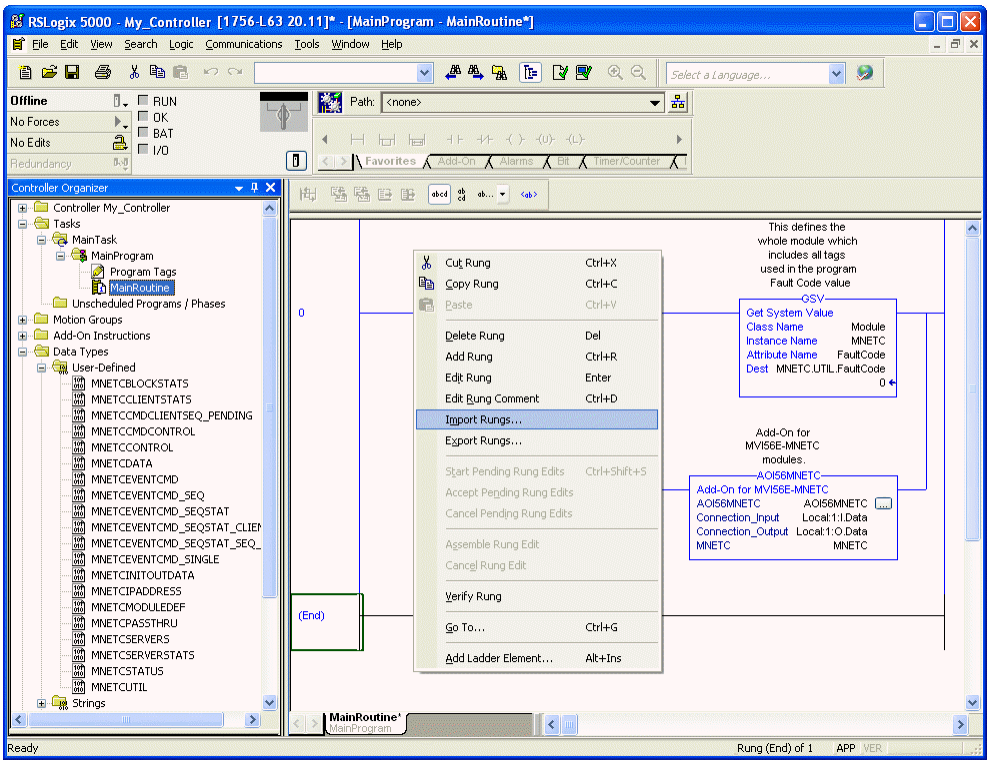
Parameter	Value
NAME	Enter a module identification string. Example: MNETC_2.
DESCRIPTION	Enter a description for the module. Example: ProSoft Modbus TCP/IP Enhanced Communication Module.
COMM FORMAT	Select **DATA-INT.**
SLOT	Enter the slot number in the rack where the MVI56E-MNETC/MNETCXT module is located.
INPUT ASSEMBLY INSTANCE	1
INPUT SIZE	250
OUTPUT ASSEMBLY INSTANCE	2
OUTPUT SIZE	248
CONFIGURATION ASSEMBLY INSTANCE	4
CONFIGURATION SIZE	0

4 Click **OK** to confirm. The new module is now visible:

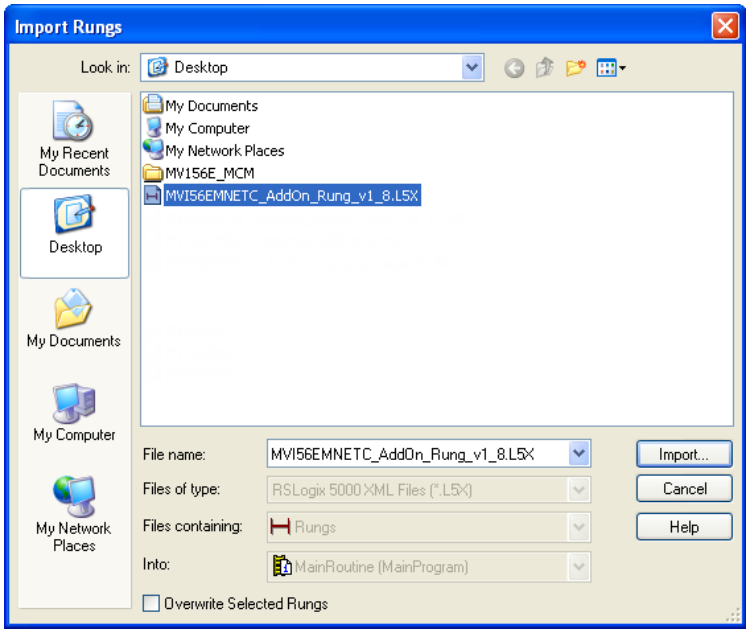


5 Expand the **TASKS** folder, and then expand the **MAINTASK** folder.
6 In the **MAINPROGRAM** folder, double-click to open the **MAINROUTINE** ladder.

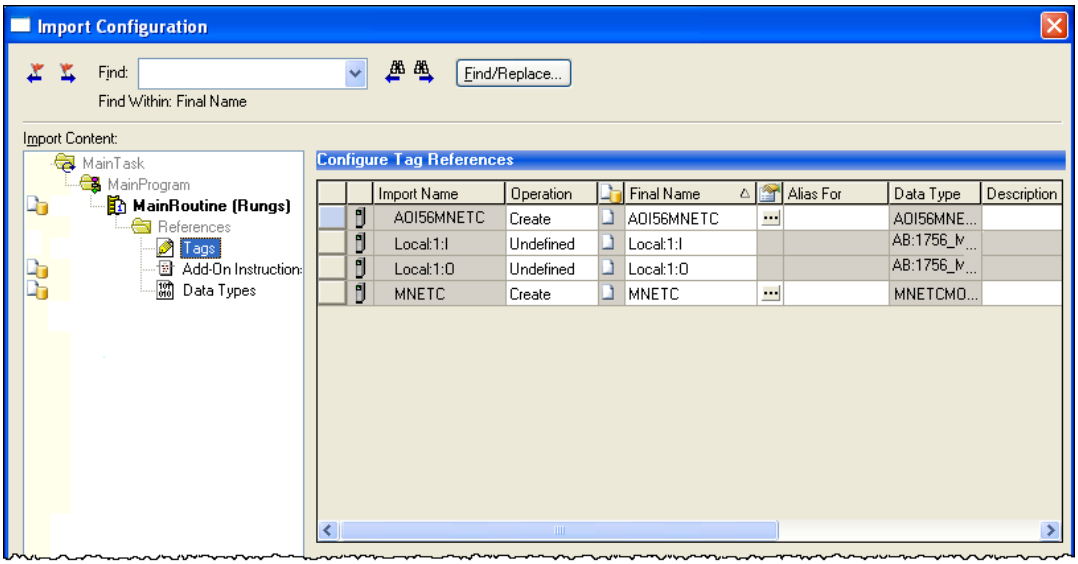
7 Select an empty rung in the routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNGS…**



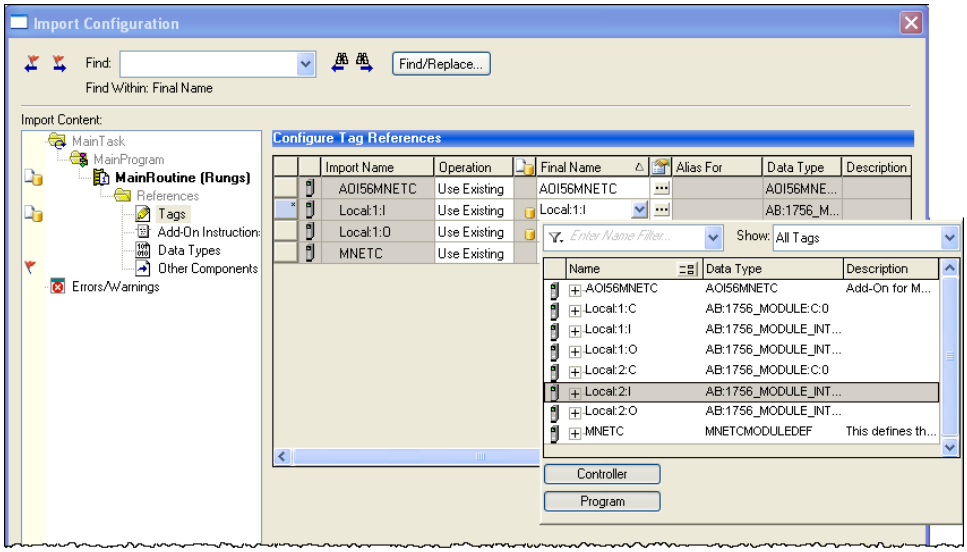
8 Select the .L5X file, and then click **IMPORT.**



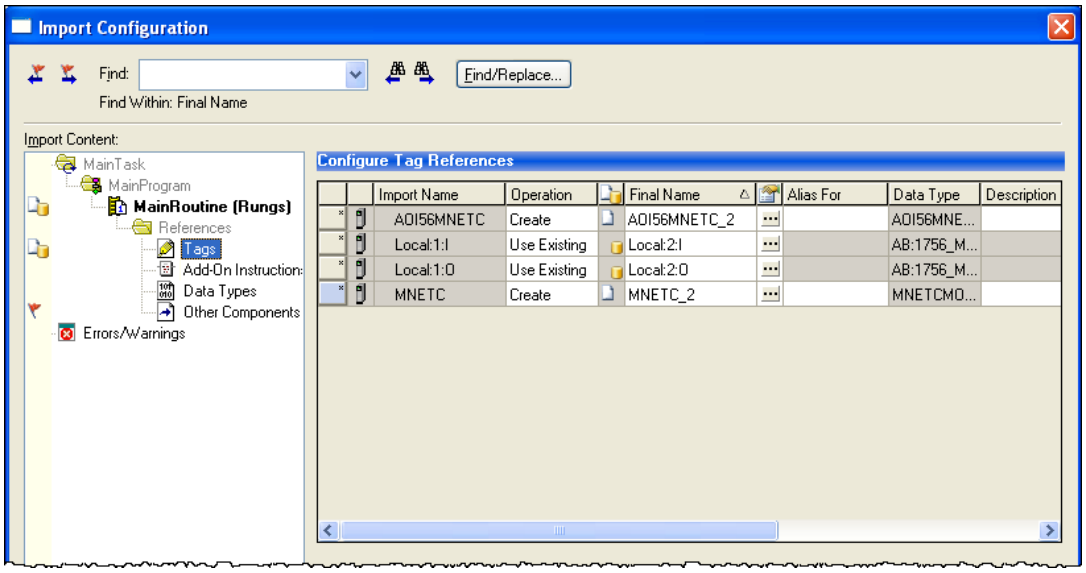
9 This action opens the **IMPORT CONFIGURATION** window. Click **TAGS** under **MAINROUTINE** to show the tags that will be imported.



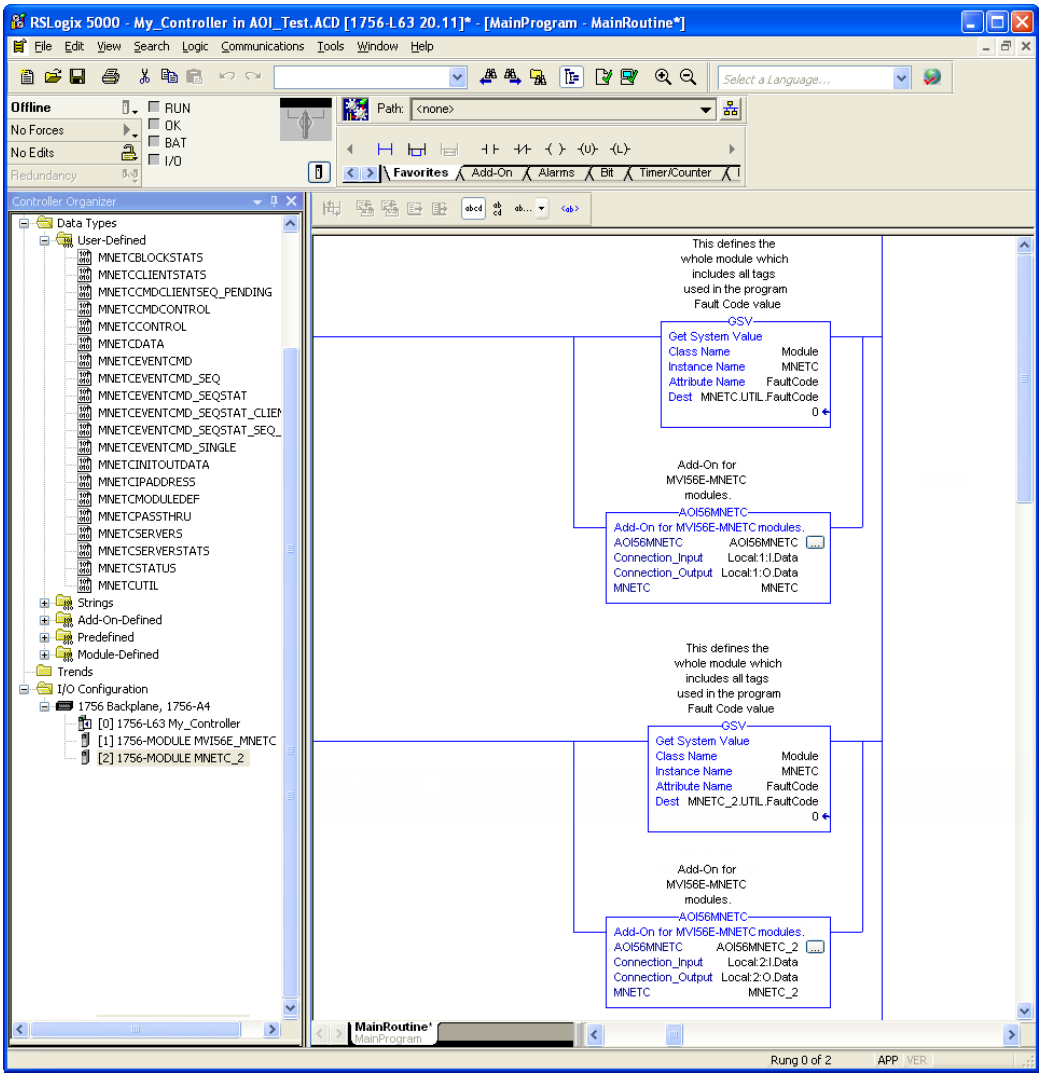
10 Associate the I/O connection variables to the correct module. The default values are Local:1:I and Local:1:O so you may have to edit the **FINAL NAME** field to change the values. You can also click the drop-down arrow to select the correct name.



11 Change the default tags **MNETC** and **AOI56MNETC** to avoid conflict with existing tags. In this step, append the string "_2", as shown in the following illustration.



12 Click **OK** to confirm.

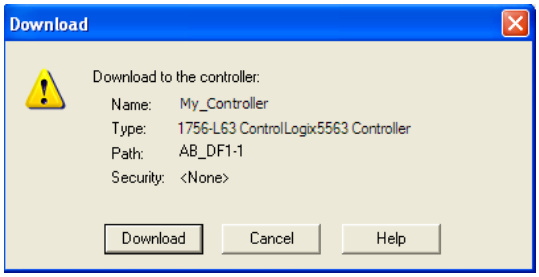


The setup procedure is now complete. Save the project and download the application to your ControlLogix processor.

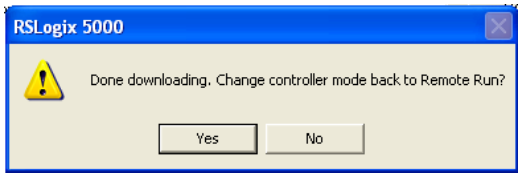
1.4 Downloading the Sample Program to the Processor

Note: The key switch on the front of the ControlLogix processor must be in the REM or PROG position.

1. If you are not already online with the processor, in RSLogix 5000 open the *Communications* menu, and then choose **DOWNLOAD**. RSLogix 5000 will establish communication with the processor. You do not have to download through the processor's serial port, as shown here. You may download through any available network connection.
2. When communication is established, RSLogix 5000 will open a confirmation dialog box. Click the **DOWNLOAD** button to transfer the sample program to the processor.



3. RSLogix 5000 will compile the program and transfer it to the processor. This process may take a few minutes.
4. When the download is complete, RSLogix 5000 will open another confirmation dialog box. If the key switch is in the REM position, click **OK** to switch the processor from PROGRAM mode to RUN mode.



Note: If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

2 MVI56E-MNETC/MNETCXT Configuration

2.1 Using ProSoft Configuration Builder Software

ProSoft Configuration Builder (PCB) provides a quick and easy way to manage module configuration files customized to meet your application needs. PCB is not only a powerful solution for new configuration files, but also allows you to import information from previously installed (known working) configurations to new projects.

Note: During startup and initialization, the MVI56E-MNETC/MNETCXT module receives its protocol and backplane configuration information from the installed Personality Module (Compact Flash). Use *ProSoft Configuration Builder* to configure module settings and to download changes to the Personality Module.

2.1.1 Installing ProSoft Configuration Builder

Use the ProSoft Configuration Builder (PCB) software to configure the module. You can find the latest version of the ProSoft Configuration Builder (PCB) on our web site: www.prosoft-technology.com. The installation filename contains the PCB version number. For example, **PCB_4.3.4.5.0238.EXE**.

If you are installing PCB from the ProSoft website:

1. Open a browser window and navigate to www.prosoft-technology.com.
2. Perform a search for 'pcb' in the Search bar. Click on the ProSoft Configuration Builder search result.
3. On the PCB page, click the download link for ProSoft Configuration Builder, and save the file to your Windows desktop.
4. After the download completes, double-click the file to install. If you are using Windows 7, right-click the PCB installation file and then choose **RUN AS ADMINISTRATOR**. Follow the instructions that appear on the screen.
5. If you want to find additional software specific to your MVI56E-MNETC/MNETCXT, enter the model number into the ProSoft website search box and press the **ENTER** key.

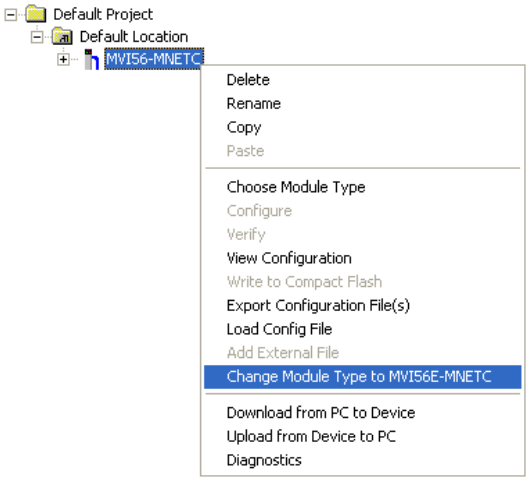
2.1.2 Upgrading from MVI56-MNETC in ProSoft Configuration Builder

MVI56E-MNETC/MNETCXT modules are fully backward-compatible with MVI56-MNETC modules. However, you will need to convert your MVI56-MNETC configuration in *ProSoft Configuration Builder* to a form that your new MVI56E-MNETC/MNETCXT module will accept when you download it.

ProSoft Configuration Builder version 2.2.2 or later has an upgrade option that easily performs this conversion, while preserving all your configuration settings and any name you may have given your module.

Important: For this procedure, *ProSoft Configuration Builder* version 2.2.2 or later must be installed on your PC. You can download the latest version from www.prosoft-technology.com.

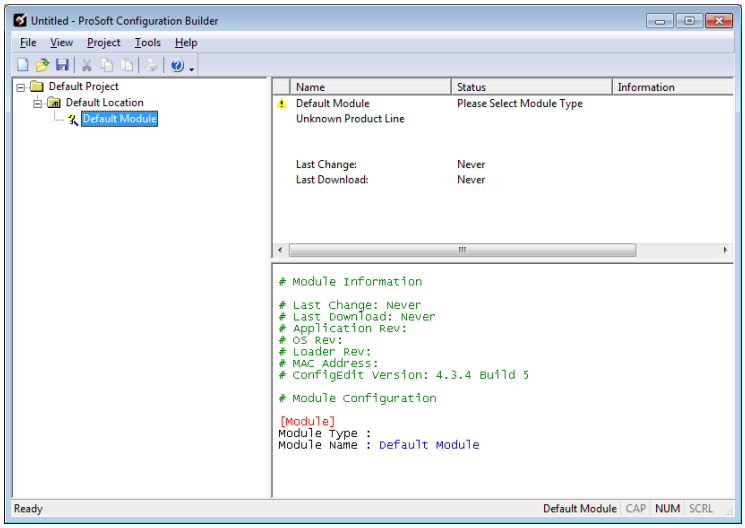
1 In *ProSoft Configuration Builder's* tree view, click the **MODULE** icon and right-click to open a shortcut menu.



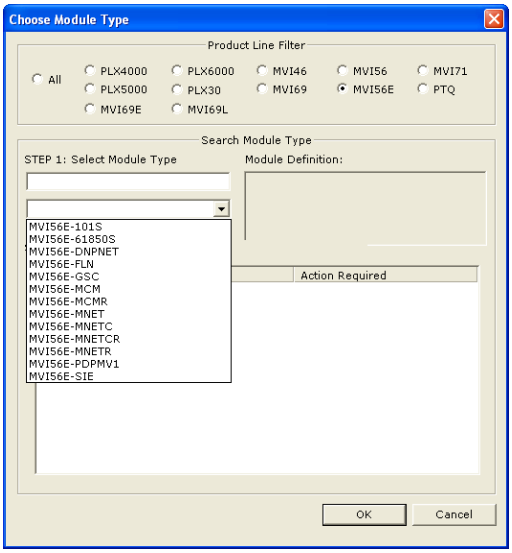
2 On the shortcut menu, select **CHANGE MODULE TYPE TO MVI56E-MNETC**.

2.1.3 Setting Up the Project

If you have used other Windows configuration tools before, you will find the screen layout familiar. *PCB*'s window consists of a tree view on the left, and an information pane and a configuration pane on the right side of the window. When you first start *PCB*, the tree view consists of folders for *Default Project* and *Default Location*, with a *Default Module* in the *Default Location* folder. The following illustration shows the *PCB* window with a new project.



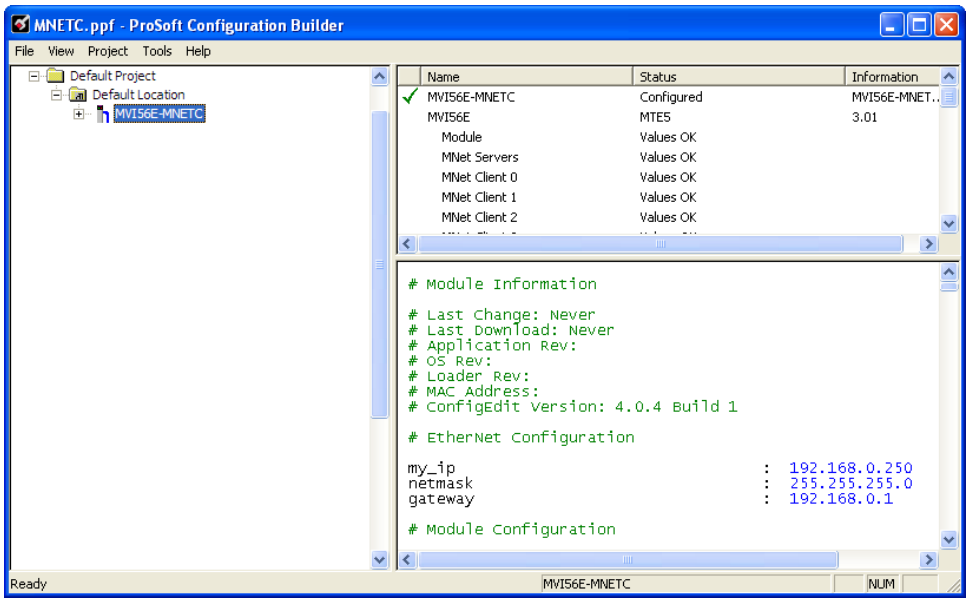
1 In PCB, right-click **DEFAULT MODULE** in the tree view and then choose **CHOOSE MODULE TYPE**. This opens the *Choose Module Type* dialog box.



2 In the *Product Line Filter* area of the dialog box, select **MVI56E**. In the *Select Module Type* dropdown list, select **MVI56E-MNETC/MNETCXT**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window.

2.1.4 Setting Module Parameters

Notice that the contents of the information pane and the configuration pane changed when you added the MVI56E-MNETC/MNETCXT module to the project.



At this time, you may wish to rename the *Default Project* and *Default Location* folders in the tree view.

Renaming an Object

1. Right-click the object and then choose **RENAME.**
2. Type the name to assign to the object.
3. Click away from the object to save the new name.

Configuring Module Parameters

1. Click the **[+]** sign next to the module icon to expand module information.
2. Click the **[+]** sign next to any icon to view module information and configuration options.
3. Double-click any icon to open an *Edit* dialog box.
4. To edit a parameter, select the parameter in the left pane and make your changes in the right pane.
5. Click **OK** to save your changes.

Printing a Configuration File

1. In the main PCB window, right-click the **MVI56E-MNETC/MNETCXT MODULE** icon and then choose **VIEW CONFIGURATION**.
2. In the *View Configuration* dialog box, click the **FILE** menu and then click **PRINT**.
3. In the *Print* dialog box, choose the printer to use from the drop-down list, select the printing options, and then click **OK**.

2.1.5 Module

This section of the configuration describes the database setup and module-level parameters.

Adjust the Input and Output Array Sizes (Optional)

Tip: If you have not installed ProSoft Configuration Builder, see Installing ProSoft Configuration Builder (page 24).

The module internal database is divided into two user-configurable areas:

- Read Data
- Write Data.

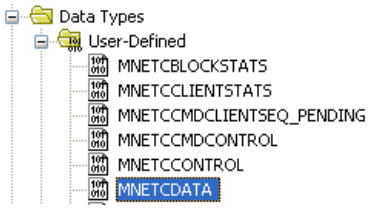
The Read Data area is moved from the module to the processor, while the Write Data area is moved from the processor to the module. You can configure the start register and size of each area. The size of each area you configure must match the Add-On instruction controller tag array sizes for the **READDATA** and **WRITEDATA** arrays.

The MVI56E-MNETC/MNETCXT sample program is configured for 600 registers of **READDATA** and 600 registers of **WRITEDATA,** which is sufficient for most application. This topic describes how to configure user data for applications requiring more than 600 registers of ReadData and WriteData. In this example, we will expand both the Read and Write Data sizes to 1000.

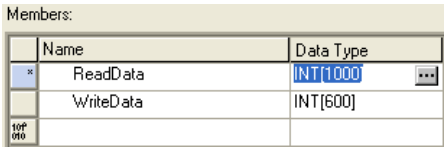
Important: Because the module pages data in blocks of 200 registers at a time, you must configure your user data in multiples of 200 registers.

Caution: When you change the array size, RSLogix may reset the MNETC tag values to zero. To avoid data loss, be sure to save your settings before continuing.

1 In the **CONTROLLER ORGANIZATION** window, expand the **DATA TYPES** and **USER-DEFINED** folders, and then double-click **MNETCDATA.** This action opens an edit window for the MNETCDATA data type.



2 In the edit window, change the value of the **READDATA** array from **INT[600]** to **INT[1000]** as shown, and then click **APPLY.**



Note: If RSLogix resets your data values, refer to the backup copy of your program to re-enter your configuration parameters.

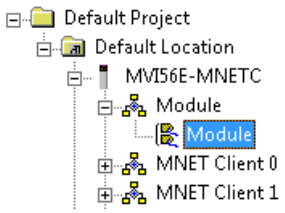
3 Next, navigate to **CONTROLLER TAGS** and double click to open an edit window. Click the **MONITOR TAGS** tab at the bottom of the edit window.

You execute the rest of the steps in ProSoft Configuration Builder.

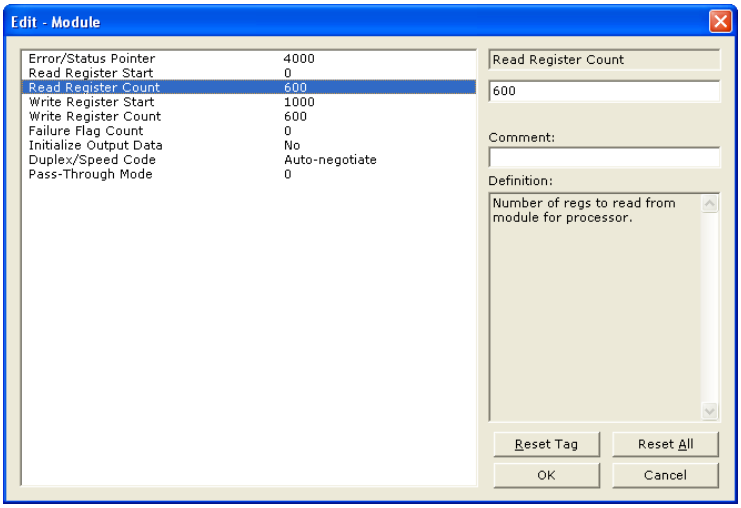
4 Start ProSoft Configuration Builder.

5 Double-click the **MVI56E-MNETC** icon to expand the menu for the module in PCB.

6 Click the **[+]** icon next to the **MODULE** icon to expand the Module menu.



7 Double-click the **MODULE** icon to open up the *Edit - Module* dialog box.



8 Click **READ REGISTER COUNT** and change the value from 600 to 1000.

9 Click **OK** to close the dialog box.

10 Save and Downloading the Project to the Module (page 60) and reboot.

To modify the **WRITEDATA** array, follow the above steps, but substitute:

- **WRITEDATA** for ReadData in RSLogix.
- **WRITE REGISTER COUNT** for Read Register Count in PCB.

Note: make sure that the **READDATA** and **WRITEDATA** arrays do not overlap in the module memory. For example, if your application requires 2000 words of WriteData starting at register 0, then your **WRITE REGISTER START** in PCB must be set to a value of 2000 or greater.

Backplane Error/Status Pointer

-1 to **9955**

This parameter sets the address in the internal database where the backplane error/status data will be placed. If you want the error/status data to be moved to the processor and placed into the *ReadData* array, the value entered should be a module memory address in the Read Data area. If the value is set to **-1**, the error/status data will not be stored in the module's internal database and will not be transferred to the processor's *ReadData* array.

Enabling the *Error/Status Pointer* is optional. The error/status data is routinely returned as part of the input image, which is continually being transferred from the module to the processor. For more information, see Normal Data Transfer Blocks (page 95).

Read Register Start

0 to **9999**

The *Read Register Start* parameter specifies the start of the Read Data area in module memory. Data in this area will be transferred from the module to the processor.

The total user database memory space is limited to the first 10,000 registers of module memory, addresses 0 through 9999. Therefore, the practical limit for this parameter is 9999 minus the value entered for Read Register Count, so that the Read Data Area does not try to extend above address 9999. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

Note: To take advantage of the new features described above, your MVI56E-MNETC/MNETCXT module needs to have firmware version 3.01 or higher, and your MVI56E-MNETC/MNETCXT Add-On Instruction needs to be version 1.8 or higher. Earlier versions have no server capabilities and support only up to 5000 user database registers.

Read Register Count

0 to **10000**

The *Read Register Count* parameter specifies the size of the Read Data area of module memory and the number of registers to transfer from this area to the processor, up to a maximum of 10,000 words.

Note: Total *Read Register Count* and *Write Register Count* cannot exceed 10,000 total registers. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

Write Register Start

0 to **9999**

The *Write Register Start* parameter specifies the start of the Write Data area in module memory. Data in this area will be transferred in from the processor.

Note: Total user database memory space is limited to the first 10,000 registers of module memory, addresses 0 through 9999. Therefore, the practical limit for this parameter is 9999 minus the value entered for *Write Register Count*, so that the Write Data Area does not try to extend above address 9999. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

Write Register Count

0 to **10000**

The *Write Register Count* parameter specifies the size of the Write Data area of module memory and the number of registers to transfer from the processor to this memory area, up to a maximum value of 10,000 words.

Note: Total *Read Register Count* and *Write Register Count* cannot exceed 10,000 total registers. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

Failure Flag Count

If this value is greater than zero the protocol communication will be interrupted once a backplane failure is detected, or communication with the processor fails. A value of zero will disable this feature.

Initialize Output Data

0 = No, **1** = Yes

This parameter is used to determine if the output data for the module should be initialized with values from the processor. If the value is set to **0**, the output data will be initialized to 0. If the value is set to **1**, the data will be initialized with data from the processor. Use of this option requires associated ladder logic to pass the data from the processor to the module.

Pass-Through Mode

0, **1**, **2** or **3**

This parameter specifies the pass-through mode for write messages received by the MNET and MBAP server ports.

- If the parameter is set to **0**, all write messages will be placed in the module's virtual database.
- If a value of **1** is entered, write messages received will be sent to the processor as unformatted messages.
- If a value of **2** is entered, write messages received will be sent to the processor with the bytes swapped in a formatted message.
- If a value of **3** is entered, write messages received will be sent to the processor as formatted messages.

Note: To take advantage of the new features described above, your MVI56E-MNETC/MNETCXT module needs to have firmware version 3.01 or higher, and your MVI56E-MNETC/MNETCXT Add-On Instruction needs to be version 1.8 or higher. Earlier versions have no server capabilities and support only up to 5000 user database registers.

Duplex/Speed Code

0, **1**, **2**, **3** or **4**

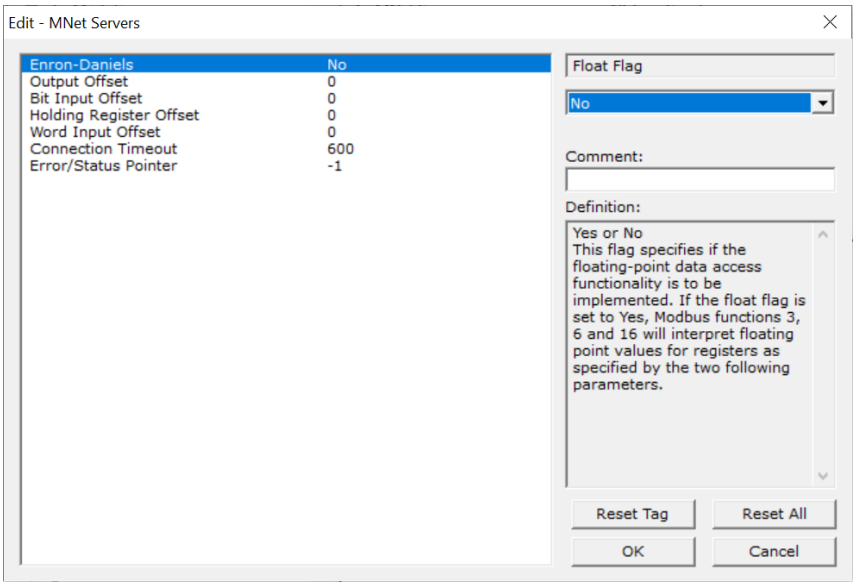
This parameter allows you to cause the module to use a specific duplex and speed setting.

- Value = **1**: Half duplex, 10 MB speed
- Value = **2**: Full duplex, 10 MB speed
- Value = **3**: Half duplex, 100 MB speed
- Value = **4**: Full duplex, 100 MB speed
- Value = **0**: Auto-negotiate

Auto-negotiate is the default value for backward compatibility. This feature is not implemented in older software revisions.

2.1.6 MNET Servers

This section contains database offset information used by the server when accessed by external Clients. These offsets can be utilized to segment the database by data type.



Note: To take advantage of the new features described above, your MVI56E-MNETC/MNETCXT module needs to have firmware version 3.01 or higher, and your MVI56E-MNETC/MNETCXT Add-On Instruction must be version 1.8 or higher. Earlier versions have no server capabilities and support only up to 5000 user database registers.

Enron-Daniels

YES or **NO**

This flag specifies how the server driver will respond to Function Code 3, 6, and 16 commands (read and write Holding Registers) from a remote master when it is moving 32-bit floating-point data.

If the remote client expects to receive or will send one complete 32-bit floating-point value for each count of one (1), then set this parameter to **YES**. When set to **YES**, the server driver will return values from two consecutive 16-bit internal memory registers (32 total bits) for each count in the read command, or receive 32-bits per count from the client for write commands. Example: Count = **10**, server driver will send 20 16-bit registers for 10 total 32-bit floating-point values.

If, however, the remote client sends a count of two (2) for each 32-bit floating-point value it expects to receive or send, or, if you do not plan to use floating-point data in your application, then set this parameter to **NO**.

Important: Depending on the type of Modbus function code / addressing being used, you will also need to set the appropriate *Output Offset, Bit Input Offset, Holding Register Offset, and/or Word Input Offset* parameter values.

Output Offset

0 to **9999**

This parameter defines the start register for the Modbus command data in the internal database. This parameter is enabled when a value greater than **0** is set. For example, if the *Output Offset* value is set to **3000**, data requests for Modbus Coil Register address 00001 will use the internal database register 3000, bit 0. If the *Output Offset* value is set to **3000**, data requests for Modbus Coil register address 00016 will use the internal database register 3000, bit 15. Function codes affected are 1, 5, and 15.

Bit Input Offset

0 to **9999**

This parameter defines the start register for Modbus command data in the internal database. This parameter is enabled when a value greater than **0** is set. For example, if the *Bit Input Offset* value is set to **3000**, data requests for Modbus Input Register address 10001 will use the internal database register 3000, bit 0. If the *Bit Input Offset* is set to **3000**, data requests for Modbus Coil register address 10016 will use the internal database register 3000, bit 15. Function code 2 is affected.

Holding Register Offset

0 to **9999**

This parameter defines the start register for the Modbus Command data in the internal database. This parameter is enabled when a value greater than **0** is set. For example, if the *Holding Register Offset* value is set to **4000**, data requests for Modbus Word register 40001 will use the internal database register 4000. Function codes affected are 3, 6, 16, & 23.

Word Input Offset

0 to **9999**

This parameter defines the start register for Modbus Command data in the internal database. This parameter is enabled when a value greater than **0** is set. For example, if the *Word Input Offset* value is set to **4000**, data requests for Modbus Word register address 30001 will use the internal database register 4000. Function code 4 is affected.

Connection Timeout

0 to **1200** seconds

Thhe number of seconds the server will wait to receive new data. If the server does not receive any new data during this time, it will close the connection.

Error/Status Pointer

-1 to **9980** seconds

This is the database offset for MBAP and MNET server status data.

2.1.7 MNET Client x

This section defines general configuration for the MNET Client (Master).

Error/Status Pointer

-1 to **9990**

This parameter sets the address in the internal database where the Client error/status data will be placed. If you want the error/status data to be moved to the processor and placed into the *ReadData* array, the value entered should be a module memory address in the Read Data area. If the value is set to **-1**, the error/status data will not be stored in the module's internal database and will not be transferred to the processor's *ReadData* array.

Enabling the *Error/Status Pointer* is optional. Alternatively, the error/status data for a specific Client can be requested by the processor and returned in a special Client Status block. For more information, see Client Status Blocks (3000 to 3029) (page 103).

Command Error Pointer

-1 to **9984**

This parameter sets the address in the internal database where the Command Error List data will be placed. If you want the Command Error List data to be moved to the processor and placed into the *ReadData* array, the value entered should be a module memory address in the Read Data area. If the value is set to **-1**, the Command Error List data will not be stored in the module's internal database and will not be transferred to the processor's *ReadData* array.

Enabling the *Command Error Pointer* is optional. Alternatively, the Command Error List data for a specific Client can be requested by the processor and returned in a special Client Status block. For more information, see Client Status Blocks (3000 to 3029) (page 103).

Minimum Command Delay

0 to **65535** milliseconds

This parameter specifies the number of milliseconds to wait between the initial issuances of a command. This parameter can be used to delay all commands sent to servers to avoid "flooding" commands on the network. This parameter does not affect retries of a command as they will be issued when failure is recognized.

Response Timeout

0 to **65535** milliseconds

This is the time in milliseconds that a Client will wait before re-transmitting a command if no response is received from the addressed server. The value to use depends on the type of communication network used, and the expected response time of the slowest device on the network.

Retry Count

0 to **10**

This parameter specifies the number of times a command will be retried if it fails.

Enron Daniels

YES or **NO**

This flag specifies how the Client driver will issue Function Code 3, 6, and 16 commands (read and write Holding Registers) to a remote server when it is moving 32-bit floating-point data.

If the remote server expects to receive or will send one complete 32-bit floating-point value for each count of one (1), then set this parameter to **YES**. When set to **YES**, the Client driver will send values from two consecutive 16-bit internal memory registers (32 total bits) for each count in a write command, or receive 32 bits per count from the server for read commands. Example: Count = **10**, Client driver will send 20 16-bit registers for 10 total 32-bit floating-point values.

If, however, the remote server expects to use a count of two (2) for each 32-bit floating-point value it sends or receives, or if you do not plan to use floating-point data in your application, then set this parameter to **NO**, which is the default setting.

ARP Timeout

1 to **60**

This parameter specifies the number of seconds to wait for an ARP reply after a request is issued.

Command Error Delay

0 to **300**

This parameter specifies the number of 100 millisecond intervals to turn off a command in the error list after an error is recognized for the command. If this parameter is set to **0**, there will be no delay.

MBAP Port Override

YES or **NO**

If this parameter is set to **YES**, all messages generated by the Client driver will be MBAP format messages to all Service Port values.

If this parameter is set to **NO** (default value), or is omitted from the configuration file, all messages sent to Service Port 502 will be MBAP format messages, and all other Service Ports values will use the encapsulated Modbus message format (MNET).

Each Client is configured independently in the configuration file.

This parameter applies to firmware version 1.05 and above. For downward compatibility, you may omit this parameter from the Client's configuration.

2.1.8 MNET Client x Commands

The *MNET Client x Commands* section of the configuration sets the Modbus TCP/IP Client command list. This command list polls Modbus TCP/IP server devices attached to the Modbus TCP/IP Client port. The module supports numerous commands. This permits the module to interface with a wide variety of Modbus TCP/IP protocol devices.

The function codes used for each command are those specified in the Modbus protocol. Each command list record has the same format. The first part of the record contains the information relating to the MVI56E-MNETC/MNETCXT communication module, and the second part contains information required to interface to the Modbus TCP/IP server device.

Command List Overview

In order to interface the module with Modbus TCP/IP server devices, you must construct a command list. The commands in the list specify the server device to be addressed, the function to be performed (read or write), the data area in the device to interface with, and the registers in the internal database to be associated with the device data. The Client command list supports up to 16 commands.

The command list is processed from top (command #1) to bottom. A poll interval parameter is associated with each command to specify a minimum delay time in tenths of a second between the issuances of a command. If the user specifies a value of **10** for the parameter, the command will be executed no more frequently than every 1 second.

Note: If you are using only Event Commands or issuing commands from the Command List using Command Control from ladder logic, it is likely that the module will not leave any inactive TCP/IP socket connections open for more than 60-seconds. To maintain an open socket connection, your configuration or application must be designed so that at least one command is issued to each server connection at less than 60-second intervals. The 60-second connection timeout is not user-configurable and was put in place to prevent long delays between commands.

Commands Supported by the Module

The format of each command in the list depends on the Modbus Function Code being executed.

The following table lists the functions supported by the module.

Function Code	Definition	Supported in Client	Supported in Server
1	Read Coil Status	X	X
2	Read Input Status	X	X
3	Read Holding Registers	X	X
4	Read Input Registers	X	X
5	Force (Write) Single Coil	X	X
6	Preset (Write) Single Register	X	X
7	Read Exception Status	X	X
8	Diagnostics		X
15	Force (Write) Multiple Coils	X	X
16	Preset (Write) Multiple Registers	X	X
22	Mask Write 4X		X
23	Read/Write		X

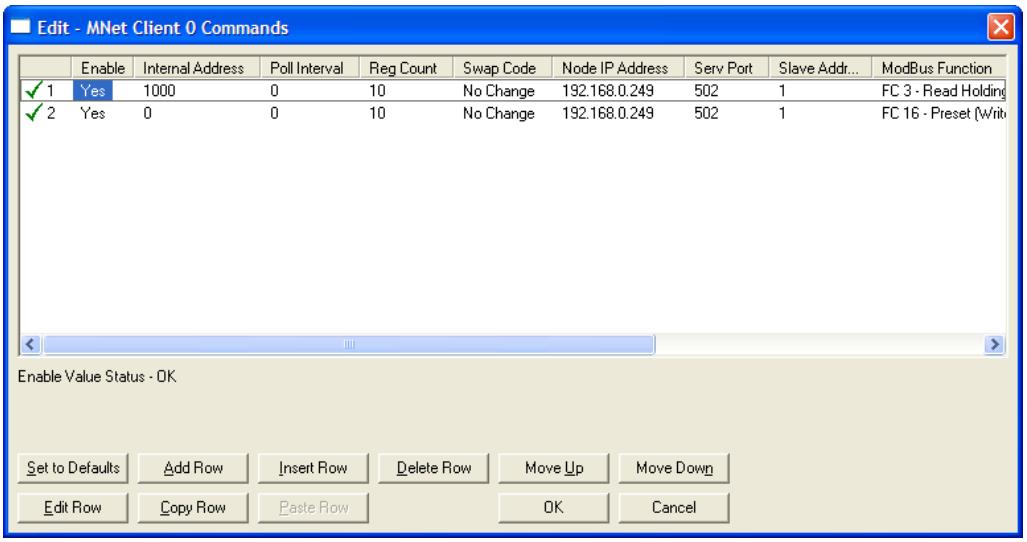
Each command list record has the same general format. The first part of the record contains the information relating to the communication module and the second part contains information required to interface to the Modbus TCP/IP server device.

Command Entry Formats

The following table shows the structure of the configuration data necessary for each of the supported commands.

1	2	3	4	5	6	7	8	9	10
Enable Code	Internal Address	Poll Interval Time	Count	Swap Code	IP Address	Serv Port	Slave Node	Function Code	Device Modbus Address
Code	Register (bit)	1/10th Seconds	Bit Count	0	IP Address	Port #	Address	Read Coil (0x)	Register
Code	Register (bit)	1/10th Seconds	Bit Count	0	IP Address	Port #	Address	Read Input (1x)	Register
Code	Register	1/10th Seconds	Word Count	Code	IP Address	Port #	Address	Read Holding Registers (4x)	Register
Code	Register	1/10th Seconds	Word Count	0	IP Address	Port #	Address	Read Input Registers (3x)	Register
Code	1 bit	1/10th Seconds	Bit Count	0	IP Address	Port #	Address	Force (Write) Single Coil (0x)	Register
Code	1 bit	1/10th Seconds	Word Count	0	IP Address	Port #	Address	Preset (Write) Single Register (4x)	Register
Code	Register (bit)	1/10th Seconds	Bit Count	0	IP Address	Port #	Address	Force (Write) Multiple Coil (0x)	Register
Code	Register	1/10th Seconds	Word Count	0	IP Address	Port #	Address	Preset (Write) Multiple Register (4x)	Register

Command list example:



Enable

NO (0) or **YES** (1)

This field defines whether or not the command is to be executed.

Value	Description
NO (0)	The command is disabled and will not be executed in the normal polling sequence.
YES (1)	The command is executed each scan of the command list if the Poll Interval Time is set to zero (**0**). If the Poll Interval time is set, the command will be executed when the interval timer expires.
CONDITIONAL (2)	For function codes 5, 15, 6, or 16; data will be sent to the target device only when the data to be written has been changed. This applies only to write commands.

Important: The commands must also be enabled in the ladder logic in order for them to be executed. The *MNETC.CONTROL.CmdControl.WriteCmdBits[x]* controller tag array holds 16-command bit arrays for each Client. If a bit for a specific command is set to zero (**0**) in the *WriteCmdBits[x]* controller tag, the command will not be executed, regardless of its enabled or disabled state in the configuration. For more information, see *Command Control Blocks* (page 109).

Internal Address

0 to **159,999** (for bit-level addressing)
or
0 to **9999** (for word-level addressing)

Note: The bit address range above is available with the following minimum requirements:

*MVI56E-MNETC firmware v3.04.007 or later.
*ProSoft Configuration Builder (PCB) v4.6.0.002 or later.

Previous versions have a bit address range of **0** to **65535**.

This field specifies the database address in the module's internal database to use as the destination for data brought in by a read command or as the source for data to be sent out by a write command. The database address is interpreted as a bit address or a 16-bit word (register) address, depending on the Modbus Function Code used in the command.

- For Modbus functions 1, 2, 5, and 15, this parameter is interpreted as a bit-level address.
- For Modbus functions 3, 4, 6, and 16, this parameter is interpreted as a word-level or register-level address.

Poll Interval

0 to **65535**

This parameter specifies the minimum interval between issuances of a command during continuous command execution (*Enable* code of **1**). The parameter is entered in tenths of a second. Therefore, if a value of **100** is entered for a command, the command executes no more frequently than every 10 seconds.

Reg Count

Registers: **1** to **125**

Coils: **1** to **800**

This parameter specifies the number of 16-bit registers or binary bits to be transferred by the command.

- Functions 5 and 6 ignore this field as they apply only to a single data point.
- For functions 1, 2, and 15, this parameter sets the number of bits (inputs or coils) to be transferred by the command.
- For functions 3, 4, and 16, this parameter sets the number of registers to be transferred by the command.

Swap Code

NONE

SWAP WORDS

SWAP WORDS & BYTES

SWAP BYTES

This parameter defines if and how the order of bytes in data received or sent is to be rearranged. This option exists to allow for the fact that different manufacturers store and transmit multi-byte data in different combinations. This parameter is helpful when dealing with floating-point or other multi-byte values, as there is no one standard method of storing these data types. The parameter can be set to rearrange the byte order of data received or sent into an order more useful or convenient for other applications. The following table defines the valid *Swap Code* values and the effect they have on the byte-order of the data.

Swap Code	Description
NONE	No change is made in the byte ordering (1234 = 1234)
SWAP WORDS	The words are swapped (1234=3412)
SWAP WORDS & BYTES	The words are swapped, then the bytes in each word are swapped (1234=4321)
SWAP BYTES	The bytes in each word are swapped (1234=2143)

These swap operations affect 4-byte (or 2-word) groups of data. Therefore, data swapping using these *Swap Codes* should be done only when using an even number of words, such as when 32-bit integer or floating-point data is involved.

Node IP Address

XXX.XXX.XXX.XXX

The IP address of the device being addressed by the command.

Service Port

502 or other port numbers supported on a server

Use a value of **502** when addressing Modbus TCP/IP servers that are compatible with the Schneider Electric MBAP specifications (this will be most devices). All other service port values will generate a Modbus command message encapsulated in a TCP/IP packet.

Slave Address

0 - Broadcast to all nodes

1 to **255**

Use this parameter to specify the slave address of a remote Modbus Serial device through a Modbus Ethernet to Serial converter.

Note: Use the *Node IP Address* parameter to address commands to a remote Modbus TCP/IP device. See Node IP Address (page 41).

Note: Most Modbus devices accept an address in the range of only 1 to 247, so check with the slave device manufacturer to see if a particular slave can use addresses 248 to 255.

If the value is set to zero, the command will be a broadcast message on the network. The Modbus protocol permits broadcast commands for **write** operations. **Do not** use node address 0 for **read** operations.

Modbus Function

1, **2**, **3**, **4**, **5**, **6**, **7**, **15**, or **16**

This parameter specifies the Modbus Function Code to be executed by the command. These function codes are defined in the Modbus protocol. The following table lists the purpose of each function supported by the module. More information on the protocol is available from www.modbus.org.

Modbus Function Code	Description
1	Read Coil Status
2	Read Input Status
3	Read Holding Registers
4	Read Input Registers
5	Force (Write) Single Coil
6	Preset (Write) Single Register
7	Read Exception Status
15	Force Multiple Coils
16	Preset Multiple Registers

MB Address in Device

This parameter specifies the starting Modbus register or bit address in the server to be used by the command. Refer to the documentation of each Modbus server device for the register and bit address assignments valid for that device.

The Modbus Function Code determines whether the address will be a register-level or bit-level OFFSET address into a given data type range. The offset will be the target data address in the server minus the base address for that data type. Base addresses for the different data types are:

- 00001 or 000001 (0x0001) for bit-level Coil data (Function Codes 1, 5, and 15).
- 10001 or 100001 (1x0001) for bit-level Input Status data (Function Code 2)
- 30001 or 300001 (3x0001) for Input Register data (Function Code 4)
- 40001 or 400001 (4x0001) for Holding Register data (Function Codes 3, 6, and 16).

Address calculation examples:

- For bit-level Coil commands (FC 1, 5, or 15) to read or write a Coil 0X address 00001, specify a value of 0 (00001 - 00001 = 0).
- For Coil address 00115, specify 114
  (00115 - 00001 = 114)
- For register read or write commands (FC 3, 6, or 16) 4X range, for 40001, specify a value of 0
  (40001 - 40001 = 0).
- For 01101, 11101, 31101 or 41101, specify a value of 1100.
  (01101 - 00001 = 1100)
  (11101 -10001 = 1100)
  (31101 - 30001 = 1100)
  (41101 - 40001 = 1100)

Note: If the documentation for a particular Modbus server device lists data addresses in hexadecimal (base16) notation, you will need to convert the hexadecimal value to a decimal value to enter in this parameter. In such cases, it is not usually necessary to subtract 1 from the converted decimal number, as this addressing scheme typically uses the exact offset address expressed as a hexadecimal number.

Comment

0 to 32 alphanumeric characters

2.1.9 Static ARP Table

The Static ARP Table defines a list of static IP addresses that the module will use when an ARP (Address Resolution Protocol) is required. The module will accept up to 40 static IP/MAC address data sets.

Use the Static ARP table to reduce the amount of network traffic by specifying IP addresses and their associated MAC (hardware) addresses that the MVI56E-MNETC/MNETCXT module will be communicating with regularly.

Important: If the device in the field is changed, this table must be updated to contain the new MAC address for the device and downloaded to the module. If the MAC is not changed, no communications with the module will be provided.

IP Address

Dotted notation

This table contains a list of static IP addresses that the module will use when an ARP is required. The module will accept up to 40 static IP/MAC address data sets.

Important: If the device in the field is changed, this table must be updated to contain the new MAC address for the device and downloaded to the module. If the MAC is not changed, no communications with the module will occur.

Hardware MAC Address

Hex value

This table contains a list of static MAC addresses that the module will use when an ARP is required. The module will accept up to 40 static IP/MAC address data sets.

Important: If the device in the field is changed, this table must be updated to contain the new MAC address for the device and downloaded to the module. If the MAC is not changed, no communications with the module will occur.

2.1.10 Ethernet Configuration

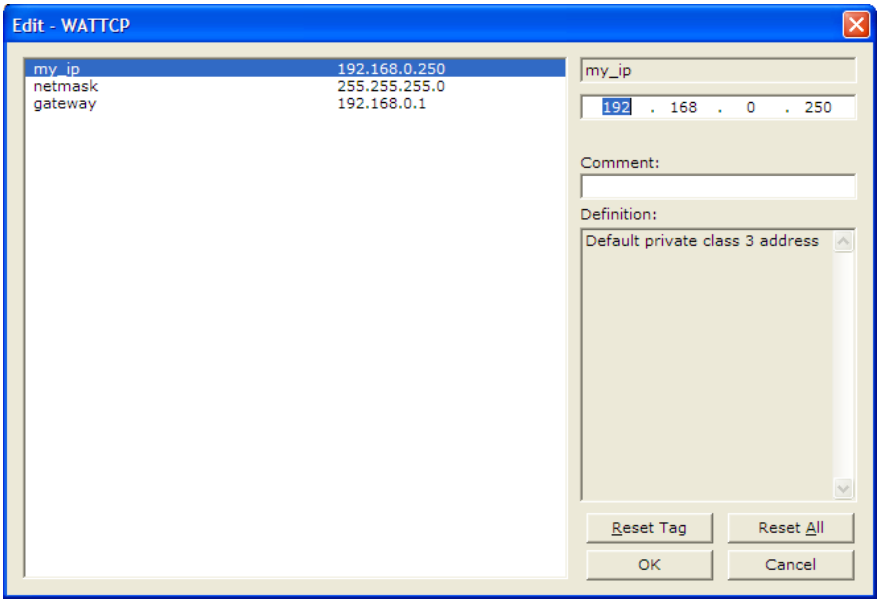
Use this procedure to configure the Ethernet settings for your module. You must assign an IP address, subnet mask and gateway address. After you complete this step, you can connect to the module with an Ethernet cable.

1 Determine the network settings for your module, with the help of your network administrator if necessary. You will need the following information:

- IP address (fixed IP required) _____ . _____ . _____ . _____
- Subnet mask _____ . _____ . _____ . _____
- Gateway address _____ . _____ . _____ . _____

Note: The gateway address is optional, and is not required for networks that do not use a default gateway.

2 Double-click the **ETHERNET CONFIGURATION** icon. This action opens the *Edit* dialog box.



3 Edit the values for *my_ip*, *netmask* (subnet mask) and *gateway* (default gateway).

4 When you are finished editing, click **OK** to save your changes and return to the *ProSoft Configuration Builder* window.

2.2 Connecting Your PC to the Module

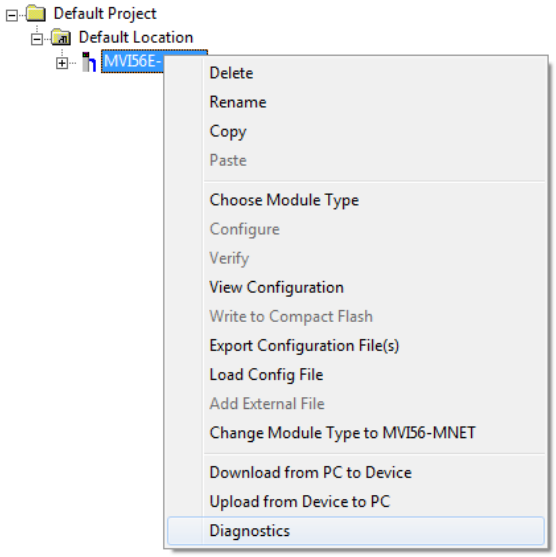
2.2.1 Using CIPconnect to Connect to the Module

You can use CIPconnect® to connect a PC to the MVI56E-MNETC/MNETCXT module over Ethernet using Rockwell Automation's 1756-ENBT EtherNet/IP® module. This allows you to configure the MVI56E-MNETC/MNETCXT module and network, upload and download files, and view network and module diagnostics from a PC. RSLinx is not required when you use CIPconnect. All you need are:

- The IP addresses and slot numbers of any 1756-ENBT modules in the path
- The ControlNet node numbers and slot numbers of any 1756-CNBx ControlNet Bridge modules in the path
- The slot number of the MVI56E-MNETC/MNETCXT in the destination ControlLogix chassis (the last ENBT/CNBx and chassis in the path).

To use CIPconnect, follow these steps.

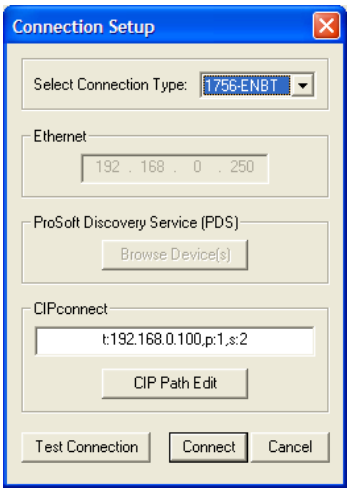
1 In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** icon and then choose **DIAGNOSTICS.**



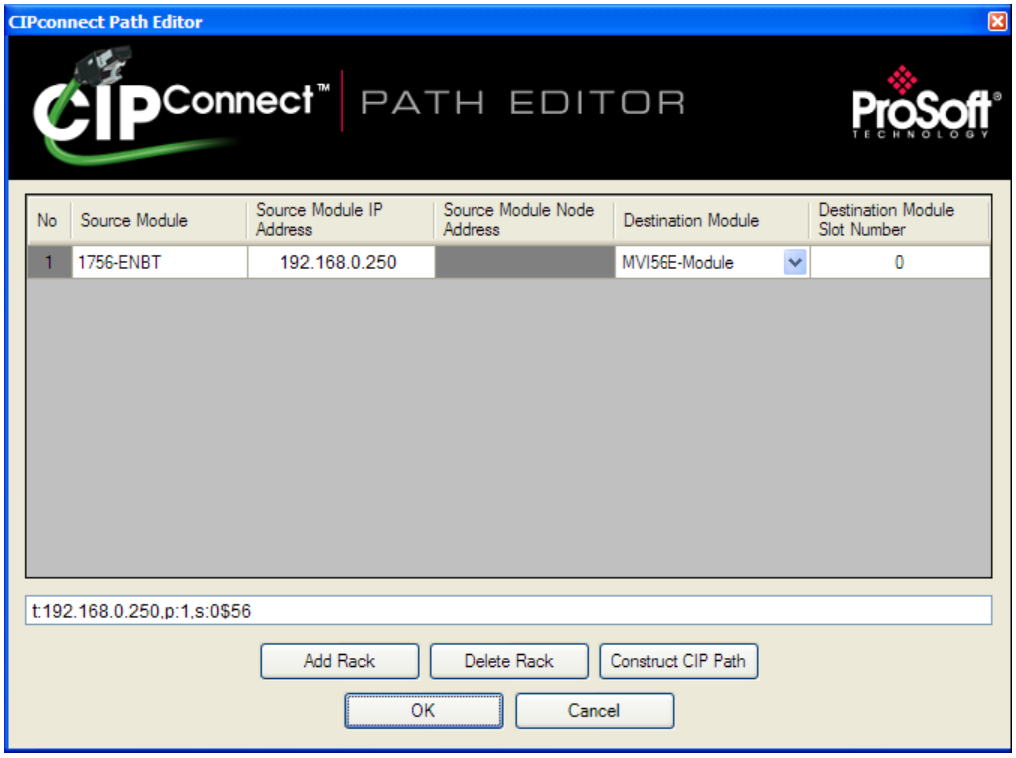
2 In the *Diagnostics* window, click the **SET UP CONNECTION** button.



3 In the *Select Connection Type* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.



4 Click **CIP PATH EDIT** to open the *CIPconnect Path Editor* dialog box.



The *CIPconnect Path Editor* allows you to define the path between the PC and the MVI56E-MNETC/MNETCXT module. The first connection from the PC is always a 1756-ENBT (Ethernet/IP) module.

Each row corresponds to a physical rack in the CIP path.

- If the MVI56E-MNETC/MNETCXT module is located in the same rack as the first 1756-ENBT module, select **RACK NO. 1** and configure the associated parameters.
- If the MVI56E-MNETC/MNETCXT is available in a remote rack (accessible through ControlNet or Ethernet/IP), include all racks (by using the **ADD RACK** button).

Parameter	Description
Source Module	Source module type. This field is automatically selected depending on the destination module of the last rack (1756-CNB or 1756-ENBT).
Source Module IP Address	IP address of the source module (only applicable for 1756-ENBT)
Source Module Node Address	Node address of the source module (only applicable for 1756-CNB)
Destination Module	Select the destination module associated to the source module in the rack. The connection between the source and destination modules is performed through the backplane.
Destination Module Slot Number	The slot number where the destination MVI56E-MNETC/MNETCXT module is located.

To use the CIPconnect Path Editor, follow these steps.

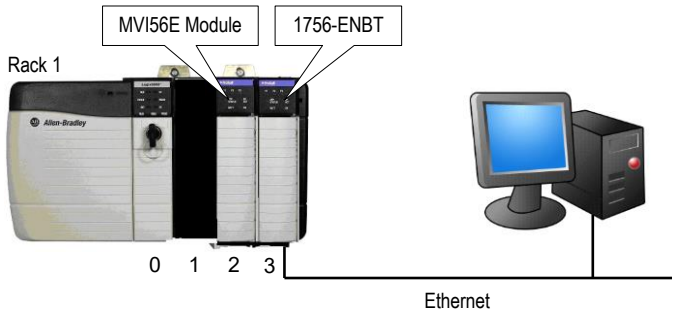
1 Configure the path between the 1756-ENBT connected to your PC and the MVI56E-MNETC/MNETCXT module.
   - If the module is located in a remote rack, add more racks to configure the full path.
   - The path can only contain ControlNet or Ethernet/IP networks.
   - The maximum number of supported racks is six.

2 Click **CONSTRUCT CIP PATH** to build the path in text format.

3 Click **OK** to confirm the configured path.

Example 1: Local Rack Application

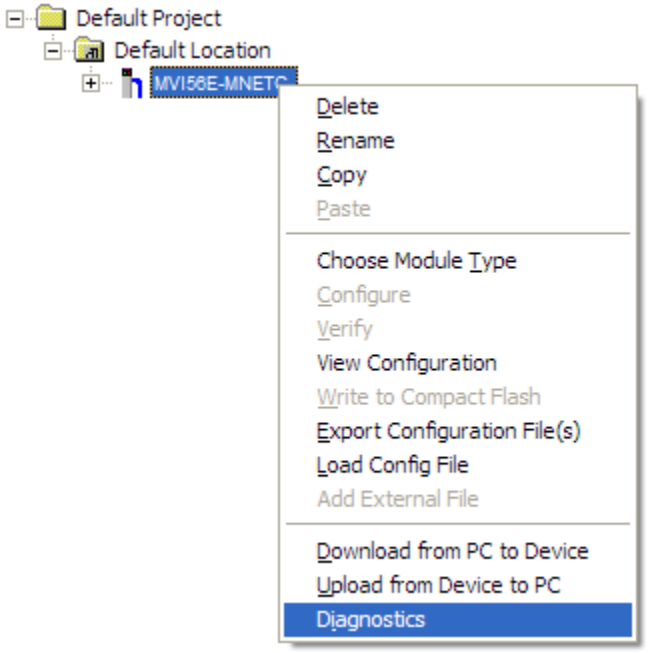
For this example, the MVI56E-MNETC/MNETCXT module is located in the same rack as the 1756-ENBT that is connected to the PC.



Rack 1

Slot	Module	Network Address
0	ControlLogix Processor	-
1	Any	-
2	MVI56E-MNETC/MNETCXT	-
3	1756-ENBT	IP=192.168.0.100

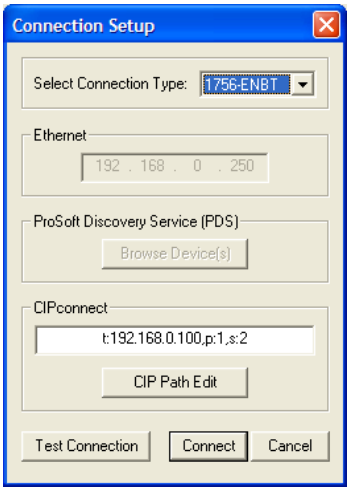
1 In *ProSoft Configuration Builder*, right-click the MVI56E-MNETC/MNETCXT icon and then choose **DIAGNOSTICS.**



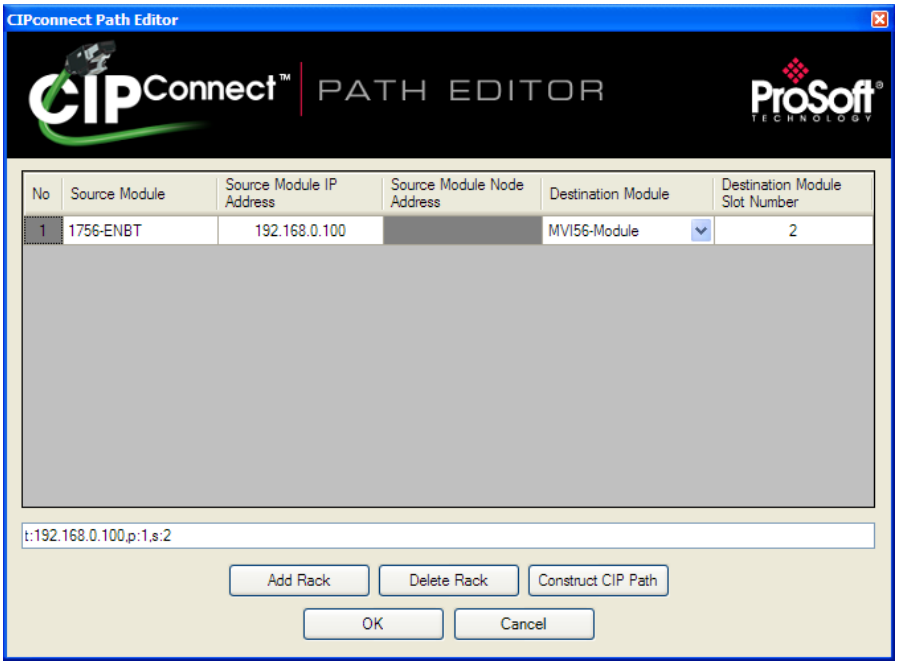
2 In the *Diagnostics* window, click the **SET UP CONNECTION** button.



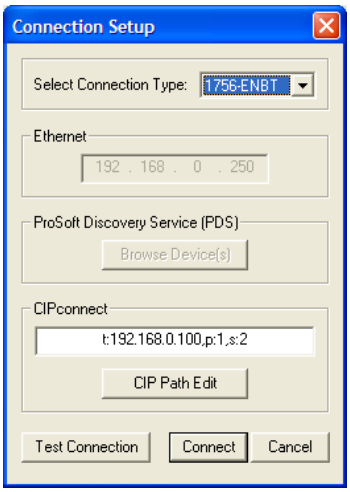
3 In the *Select Connection Type* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.



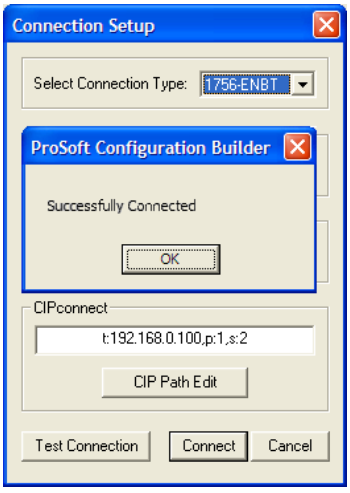
4 Configure the path as shown in the following illustration, and click **CONSTRUCT CIP PATH** to build the path in text format.



5 Click **OK** to close the *CIPconnect Path Editor* and return to the *Connection Setup* dialog box.
6 Check the new path in the *Connection Setup* dialog box.



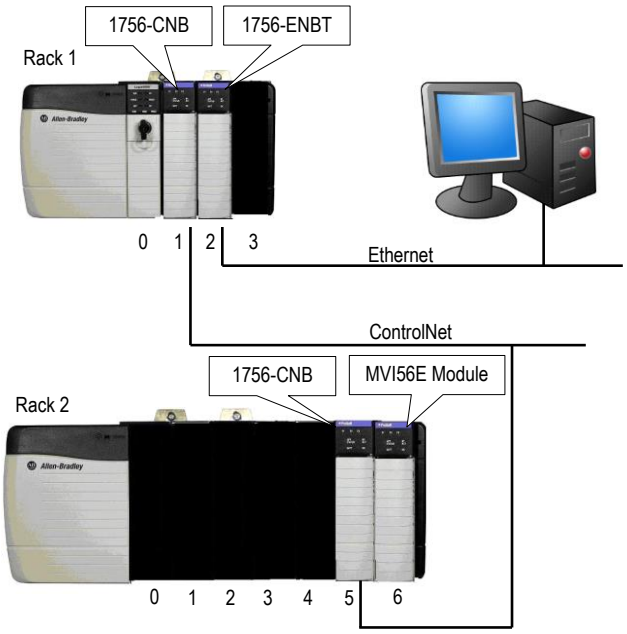
7 Click **TEST CONNECTION** to verify that the physical path is available. The following message should be displayed upon success.



8 Click **OK** to close the *Test Connection* pop-up and then click **CONNECT** to close the *Connection Set* up dialog box. The Diagnostics menu is now connected through CIPconnect.

Example 2: Remote Rack Application

For this example, the MVI56E-MNETC/MNETCXT module is located in a remote rack accessible through ControlNet, as shown in the following illustration.



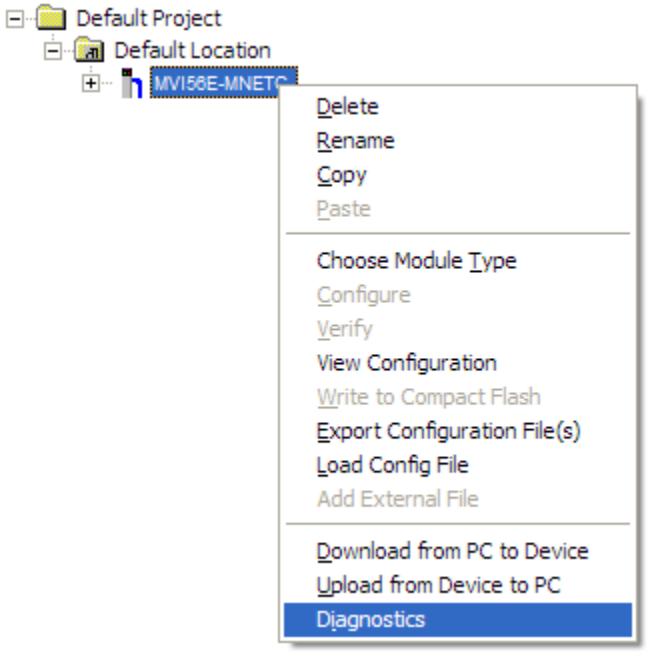
Rack 1

Slot	Module	Network Address
0	ControlLogix Processor	-
1	1756-CNB	Node = 1
2	1756-ENBT	IP=192.168.0.100
3	Any	-

Rack 2

Slot	Module	Network Address
0	Any	-
1	Any	-
2	Any	-
3	Any	-
4	Any	-
5	1756-CNB	Node = 2
6	MVI56E-MNETC/MNETCXT	-

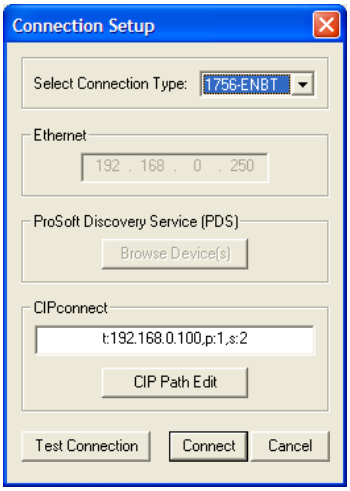
1 In *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** icon and then choose **DIAGNOSTICS.**



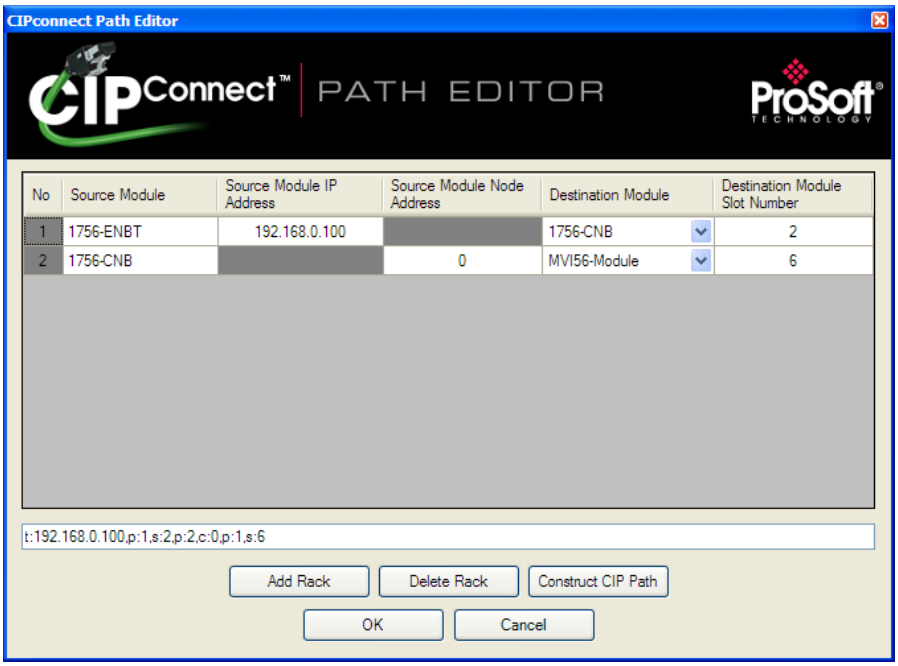
2 In the *Diagnostics* window, click the **SET UP CONNECTION** button.



3 In the *Select Connection Type* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.

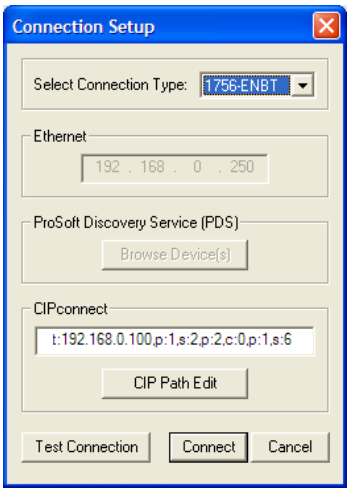


4 Configure the path as shown in the following illustration, and click **CONSTRUCT CIP PATH** to build the path in text format.

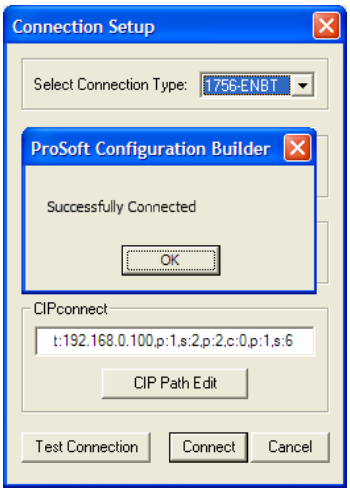


5 Click **OK** to close the *CIPconnect Path Editor* and return to the *Connection Setup* dialog box.

6 Check the new path in the *Connection Setup* dialog box.



7 Click **TEST CONNECTION** to verify that the physical path is available. The following message should be displayed upon success.

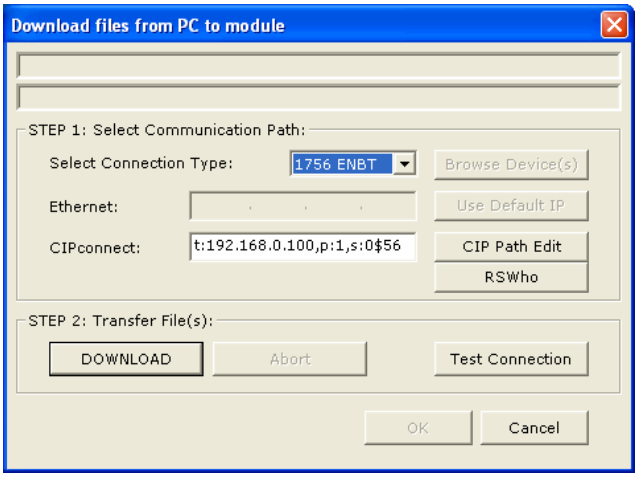


8 Click **OK** to close the *Test Connection* pop-up and then click **CONNECT** to close the *Connection Set* up dialog box. The Diagnostics menu is now connected through CIPconnect.

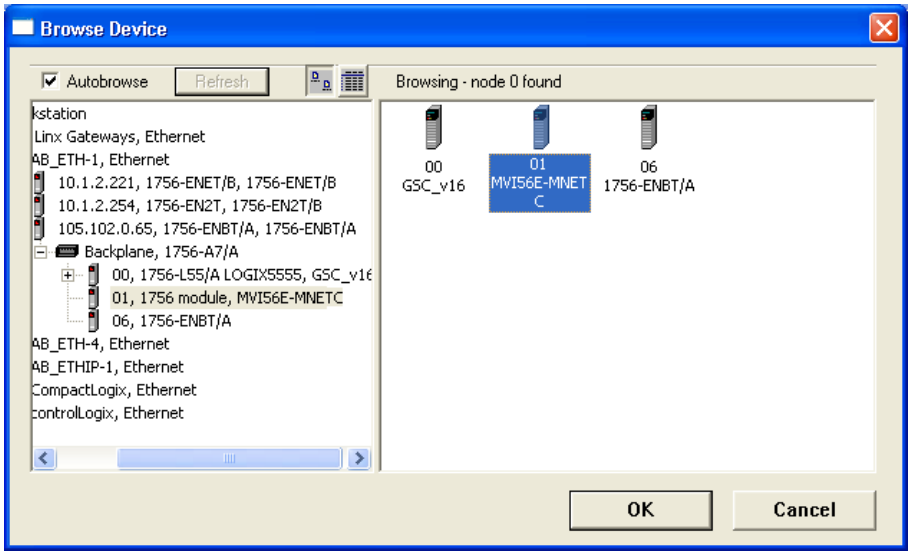
2.2.2 Using RSWho to Connect to the Module

You need to have RSLinx installed on your PC to use this feature. You also need an ENBT module set up in the rack. For information on setting up the ENBT module, see Using CIPconnect to Connect to the Module (page 46).

1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** module.
2. From the shortcut menu, choose **DOWNLOAD FROM PC TO DEVICE**.
3. In the *Download* dialog box, choose **1756 ENBT** from the *Select Connection Type* dropdown box.



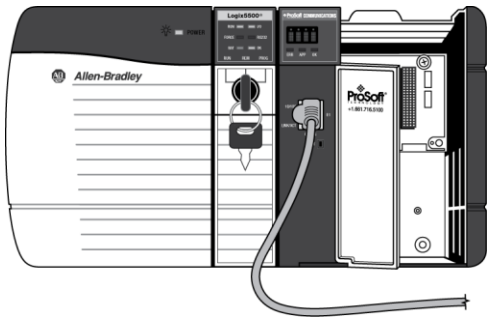
4. Click **RSWHO** to display modules on the network. The MVI56E-MNETC/MNETCXT module will automatically be identified on the network.



5. Select the module, and then click **OK**.

2.2.3 Connecting Your PC to the Module's Ethernet Port

With the module securely mounted, connect one end of the Ethernet cable to the *Config (E1)* Port, and the other end to an Ethernet hub or switch accessible from the same network as your PC. You can also connect directly from the Ethernet Port on your PC to the *Config (E1)* Port on the module by using an Ethernet crossover cable (not included).

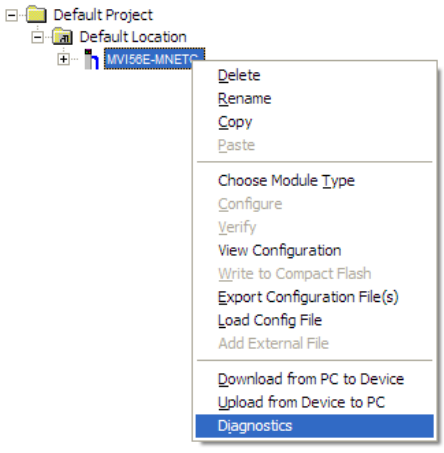


Setting Up a Temporary IP Address

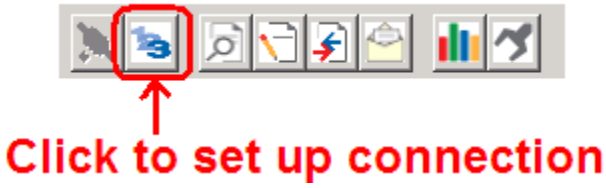
Important: *ProSoft Configuration Builder* locates MVI56E-MNETC/MNETCXT modules through UDP broadcast messages. These messages may be blocked by routers or layer 3 switches. In that case, *ProSoft Discovery Service* will be unable to locate the modules.

To use *ProSoft Configuration Builder*, arrange the Ethernet connection so that there is no router/ layer 3 switch between the computer and the module OR reconfigure the router/ layer 3 switch to allow routing of the UDP broadcast messages.

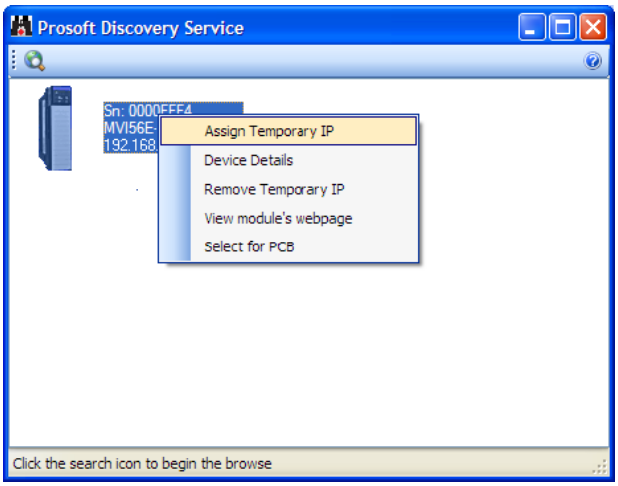
1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** icon to open a shortcut menu.
2. On the shortcut menu, choose **DIAGNOSTICS.**



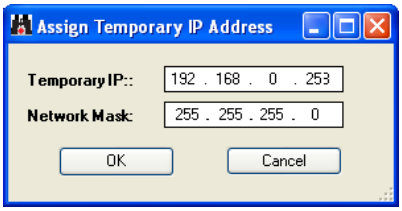
3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.



4 In the *Connection Setup* dialog box, click the **BROWSE DEVICE(S)** button to open the *ProSoft Discovery Service*. Select the module, then right-click and choose **ASSIGN TEMPORARY IP**.

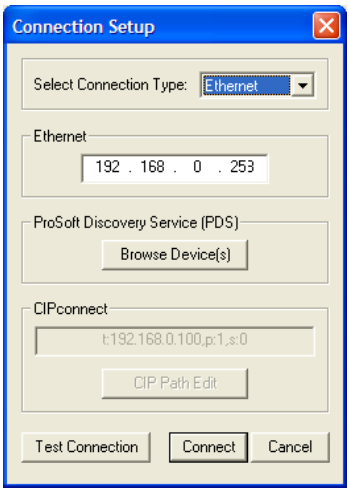


5 The module's default IP address is 192.168.0.250. Choose an unused IP within your subnet, and then click **OK**.

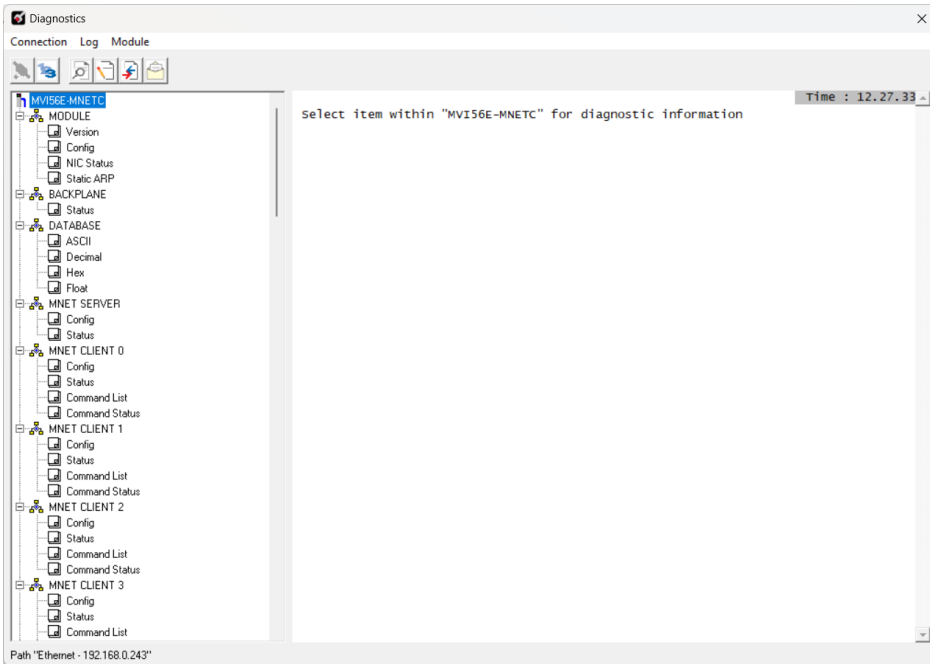


Important: The temporary IP address is only valid until the next time the module is initialized. For information on how to set the module's permanent IP address, see Ethernet Configuration (page 45).

6 Close the *ProSoft Discovery Service* window. Enter the temporary IP in the Ethernet address field of the *Connection Setup* dialog box, then click the **TEST CONNECTION** button to verify that the module is accessible with the current settings.



7 If the *Test Connection* is successful, click **CONNECT**. The *Diagnostics* menu will display in the *Diagnostics* window.



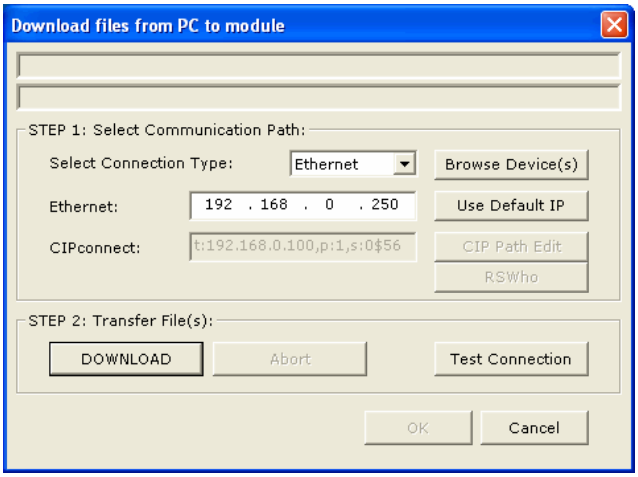
2.3 Downloading the Project to the Module

Note: For alternative methods of connecting to the module with your PC, see Connecting Your PC to the Module (page 45).

In order for the module to use the settings you configured, you must download the updated Project file from your PC to the module.

1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** icon and then choose **DOWNLOAD FROM PC TO DEVICE.** This opens the *Download* dialog box.
2. In the *Download* dialog box, choose the connection type in the *Select Connection Type* dropdown box:
   - Choose **ETHERNET** if you are connecting to the module through the Ethernet cable.
   - Choose **1756 ENBT** if you are connecting to the module through CIPconnect or RSWho.
     See Connecting Your PC to the Module (page 45) for more information.

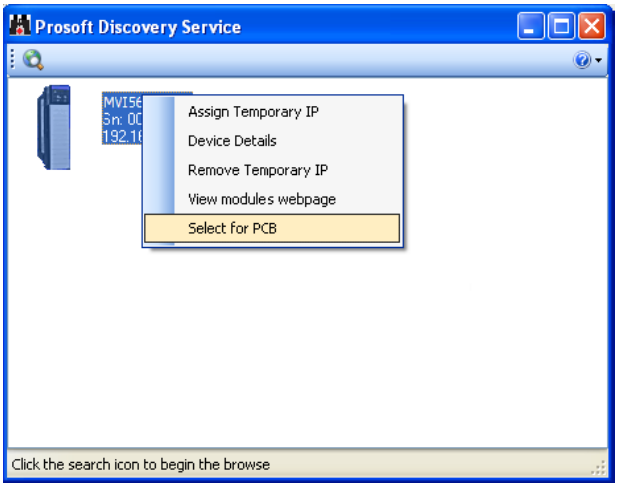
Note: If you are connected to the module using an Ethernet cable and set a temporary IP address, the Ethernet address field contains that temporary IP address. ProSoft Configuration Builder uses this temporary IP address to connect to the module.



3. Click **TEST CONNECTION** to verify that the IP address allows access to the module.
4. If the connection succeeds, click **DOWNLOAD** to transfer the Ethernet configuration to the module.

If the *Test Connection* procedure fails, you will see an error message. To correct the error, follow these steps.

1. Click **OK** to dismiss the error message.
2. In the *Download* dialog box, click **BROWSE DEVICE(S)** to open *ProSoft Discovery Service*.



3. Right-click the module and then choose **SELECT FOR PCB**.
4. Close *ProSoft Discovery Service*.
5. Click **DOWNLOAD** to transfer the configuration to the module.

3 Using Controller Tags

Controller tags are a feature of the RSLogix software and are part of the MVI56E-MNETC/MNETCXT Add-On Instruction. Refer to the section *Adding the Module to an Existing Project* (page 145) for information on importing the Add-On Instruction into RSLogix.

3.1 Controller Tags

Data related to the MVI56E-MNETC/MNETCXT is stored in the ladder logic in variables called controller tags. You use controller tags to manage communication between the MVI56E-MNETC/MNETCXT module and the ControlLogix processor:

- View the read and write data being transferred between the module and the processor.
- View status data for the module.
- Set up and trigger special functions.
- Initiate module restarts (Warm Boot or Cold Boot).

Individual controller tags can be grouped into collections of controller tags called controller tag structures. A controller tag structure can contain any combination of:

- Individual controller tags
- Controller tag arrays
- Lower-level controller tag structures

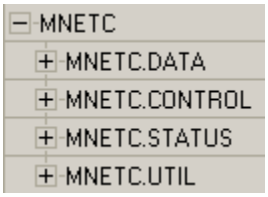
The controller tags are included in the MVI56E-MNETC/MNETCXT Add-On Instruction ladder logic. After you import the Add-On Instruction, you can find the controller tags in the *Controller Tags* subfolder, located in the *Controller* folder in the *Controller Organizer* pane of the main RSLogix 5000 window. This controller tag structure is arranged as a tree structure. Individual controller tags are found at the lowest level of the tree structure. Each individual controller tag is defined to hold data of a specific type, such as integer or floating-point data.

The Add-On Instruction also includes user-defined data types (UDTs). UDTs are collections of data types and declares the data types for the controller tag structures.

The MVI56E-MNETC/MNETCXT Add-On Instruction is extensively commented to provide information on the purpose and function of each user-defined data type and controller tag. For most applications, the Add-On Instruction works without needing any modification.

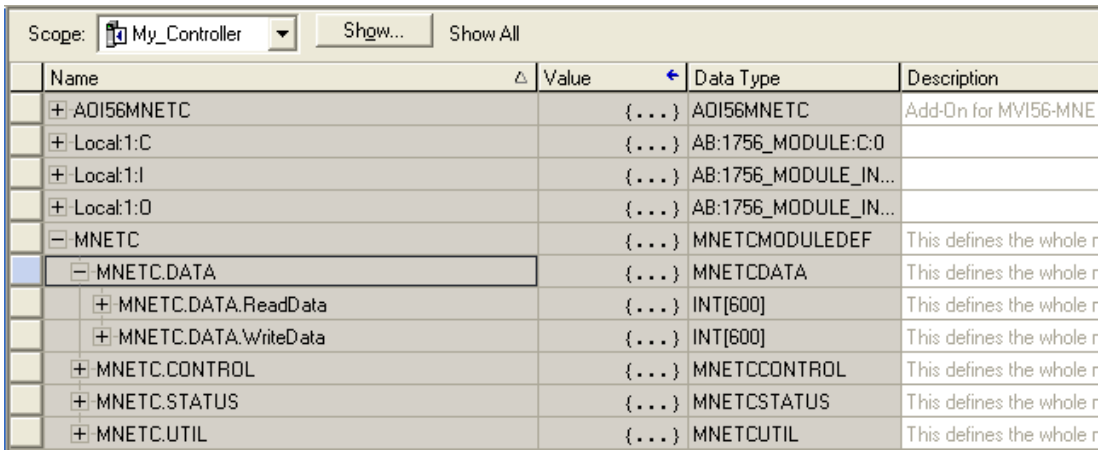
3.1.1 MVI56E-MNETC Controller Tags

The main controller tag structure, *MNETC*, is broken down into four lower-level controller tag structures.



The four lower-level controller tag structures contain other controller tags and controller tag structures. Click the **[+]** sign next to any controller tag structure to expand it and view the next level in the structure.

For example, if you expand the *MNETC.DATA* controller tag structure, you will see that it contains two controller tag arrays, *MNETC.DATA.ReadData* and *MNETC.DATA.WriteData*, which are 600-element integer arrays by default.



Each controller tag in the Add-On Instruction is commented in the *Description* column.

Notice that the *Data Type* column displays the data types used to declare each controller tag, controller tag array or controller tag structure. Individual controller tags are declared with basic data types, such as INT and BOOL. Controller tag arrays are declared with arrays of basic data types. Controller tag structures are declared with user-defined data types (UDTs).

3.2 User-Defined Data Types (UDTs)

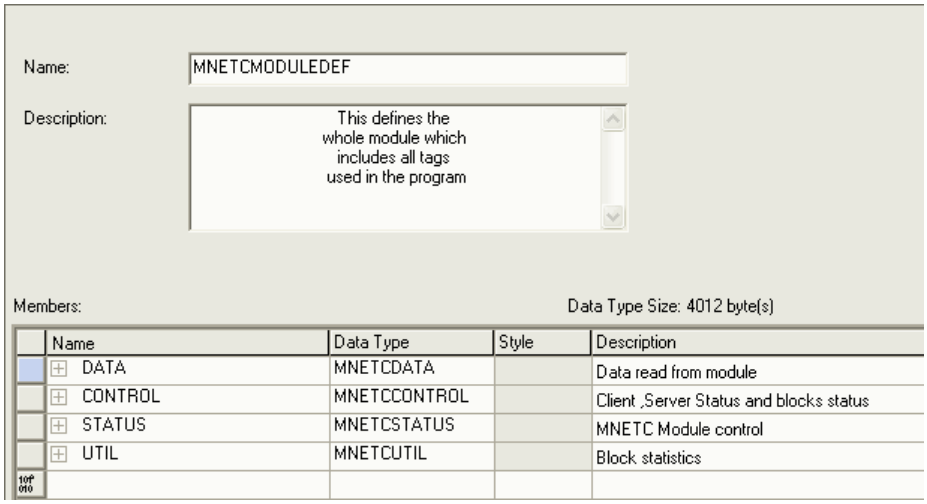
User-defined data types (UDTs) allow you to organize collections of data types into groupings. You can use these groupings, or data type structures, to declare the data types for controller tag structures. Another advantage of defining a UDT is that you may reuse it in other controller tag structures that use the same data types.

The Add-On Instruction for the MVI56E-MNETC/MNETCXT module has pre-defined UDTs. You can find them in the *User-Defined* subfolder, located in the *Data Types* folder in the *Controller Organizer* pane of the main RSLogix window. Like the controller tags, the UDTs are organized in a multiple-level tree structure.

3.2.1 MVI56E-MNETC User-Defined Data Types

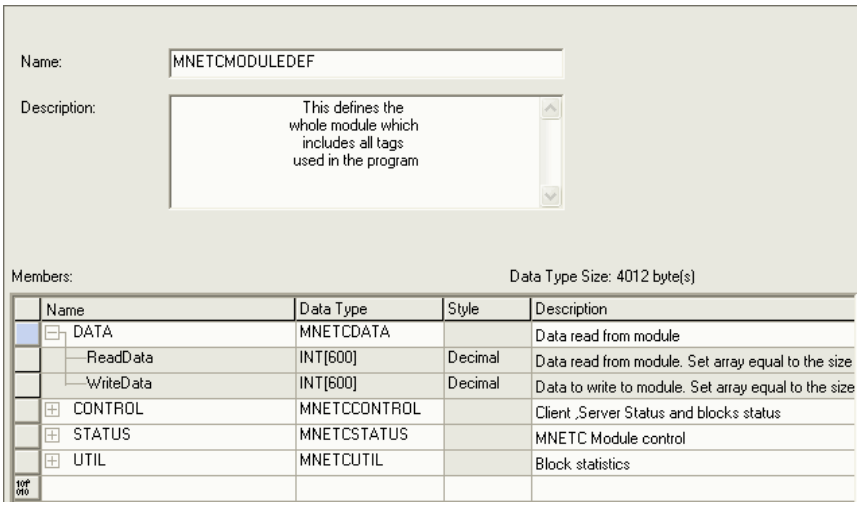
Eleven different UDTs are defined for the MVI56E-MNETC Add-On Instruction.

The main UDT, *MNETCMODULEDEF*, contains all the data types for the module and was used to create the main controller tag structure, *MNETC*. There are four UDTs one level below *MNETCMODULEDEF*. These lower-level UDTs were used to create the *MNETC.DATA*, *MNETC.CONTROL*, *MNETC.STATUS*, and *MNETC.UTIL* controller tag structures.



Click the **[+]** signs to expand the UDT structures and view lower-level UDTs.

For example, if you expand *MNETC.DATA*, you will see that it contains two UDTs, *ReadData* and *WriteData*. Both of these are 600-element integer arrays by default.



Name	Data Type	Style	Description
DATA	MNETCDATA		Data read from module
ReadData	INT[600]	Decimal	Data read from module. Set array equal to the size
WriteData	INT[600]	Decimal	Data to write to module. Set array equal to the size
CONTROL	MNETCCONTROL		Client ,Server Status and blocks status
STATUS	MNETCSTATUS		MNETC Module control
UTIL	MNETCUTIL		Block statistics

Notice that these UDTs are the data types used to declare the *MNETC.DATA.ReadData* and *MNETC.DATA.WriteData* controller tag arrays.

Each UDT is commented in the *Description* column.

3.3 Controller Tag Overview

This and the following sections describe the MNETC controller tags in detail.

Controller Tag	Description
MNETC.DATA	MNET input and output data transferred between the processor and the module
MNETC.CONTROL	Governs the data movement between the PLC rack and the module
MNETC.STATUS	Status information
MNETC.UTIL	Block statistics and generic tags used for internal ladder processing (DO NOT MODIFY)

3.3.1 MNETC.DATA

The controller tags in *MNETC.DATA* hold data to be transferred between the processor and the MVI56E-MNETC/MNETCXT module. This read and write data is transferred between the processor and module as "pages," or blocks, of data up to 200 words long.

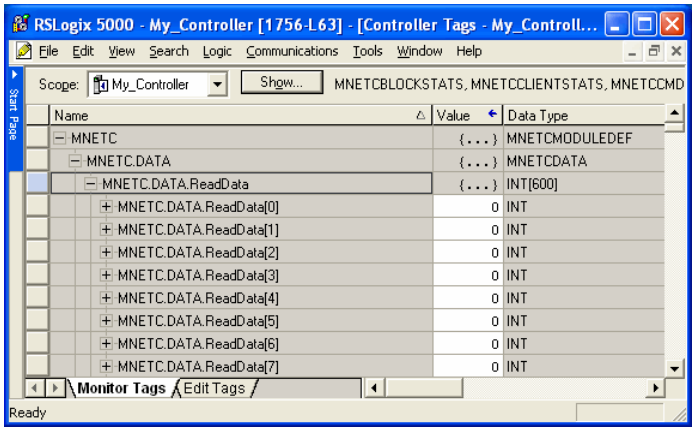
The data types for the *MNETC.DATA.ReadData* and *MNETC.DATA.WriteData* controller tag arrays are integer arrays containing variable numbers of elements.

Controller Tag	Data Type	Description
ReadData	INT[x]	Data read from module. Array size is equal to the size set in the configuration.
WriteData	INT[x]	Data to write to module. Array size is equal to the size set in the configuration.

MNETC.DATA.ReadData

The *ReadData* controller tag array should accommodate the value entered in the *Read Register Count* (page 30) parameter of the MVI56E-MNETC/MNETCXT configuration file in *Prosoft Configuration Builder*. The default length of this array is 600. If more than 600 registers are needed, please see Adjust the Input and Output Array Sizes.

For ease of use, this array should be dimensioned as a multiple of 200 words. This data is paged up to 200 words at a time from the module to the processor. The ladder logic places the data received into the proper position in the *ReadData* array. This data is used for status and control in the processor ladder logic.



The *ReadData* array is related to the contents of the Read Data area of the module's internal database. To view the actual registers in the module's internal database, access the database display from *ProSoft Configuration Builder's Diagnostics* menu. For more information, see the section on *PCB* The Diagnostics Menu (page 78).

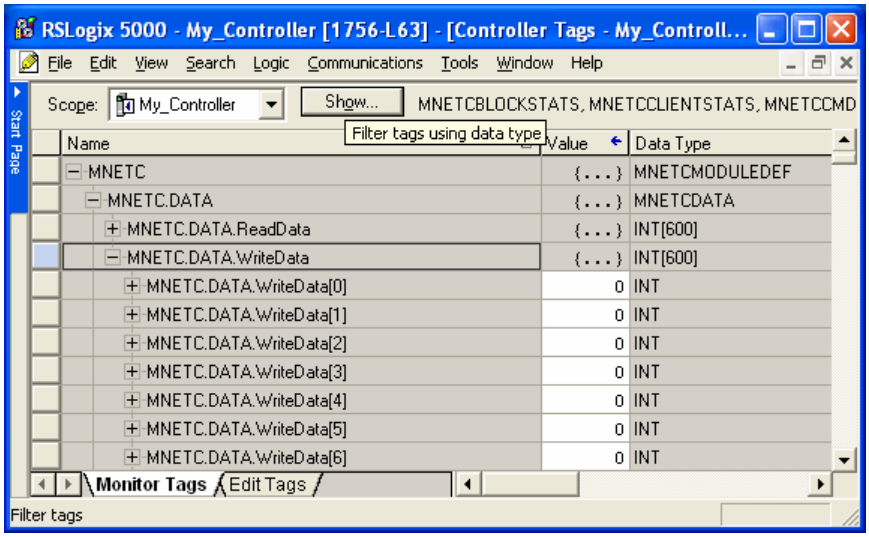
```
DATABASE DISPLAY 0 TO 99 (DECIMAL)

   6666    7777    8888    9999    1010       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
      0       0       0       0       0       0       0       0
```

MNETC.DATA.WriteData

The *WriteData* controller tag array should accommodate the value entered in the *Read Register Count* (page 30) parameter of the MVI56E-MNETC/MNETCXT configuration file in *Prosoft Configuration Builder*. The default length of this array is 600. If more than 600 registers are needed, please see Adjust the Input and Output Array Sizes.

For ease of use, this array should be dimensioned as a multiple of 200 words. This data is paged up to 200 words at a time from the processor to the module. The ladder logic places the write data into the output image for transfer to the module. This data is passed from the processor to the module for status and control information for use in other nodes on the network.



The *WriteData* array is related to the contents of the Write Data area of the module's internal database. To view the actual registers in the module's internal database, access the database display from *ProSoft Configuration Builder's Diagnostics* menu. For more information, see the section on *PCB* The Diagnostics Menu (page 78).

```
DATABASE DISPLAY 1000 TO 1099 (DECIMAL)

  1111   2222   3333   4444   5555      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
     0      0      0      0      0      0      0      0      0      0
```

3.3.2 MNETC.CONTROL

This controller tag structure is used to request special tasks from the module. For more information, see Special Function Blocks (page 99).

Controller Tag	Data Type	Description
BootTimer	TIMER	Timer used to clear both cold and warm boot requests
ColdBoot	BOOL	Hardware reset of the module
WarmBoot	BOOL	Configuration data reset in the module
ResetStatus	BOOL	Reset status values
CmdID	INT	Command ID from 1 to 16
CmdControl	MNETCCMDCONTROL	Holds Command Control statistics
CmdControlPending	BOOL	Halts rung until module is ready
CmdControlTrigger	BOOL	Command Control Trigger
IPAddress	MNETCIPADDRESS	Getting and setting IP address to and from module
EventCmd	MNETCEVENTCMD	Holds Event Command configuration
EventSeqCmd	MNETCEVENTCMD_SEQ	Event sequence commands

3.3.3 MNETC.STATUS

This controller tag structure contains module and Client status data. For a more complete description of the *MNETC.STATUS* controller tag structure, refer to the Status Data Definition (page 83).

Name	Data Type	Description
PassCnt	INT	Program cycle counter
ProductVersion	INT	This is used to pass the product version to the processor
ProductCode	INT[2]	This is used to pass the product code to the processor
BlockStats	MNETCBLOCKSTATS	Block transfer statistics
CmdBits	INT[30]	Command bits array to be used for 30 Clients
ClientStatsTrigger	BOOL	Get Client status
ClientID	INT	Client ID to get status from
ClientStatus	MNETCCLIENTSTATS[30]	Client status data
CmdErrorList	INT[16]	Command Error List
EventSeqCmdPending	MNETCCMDCLIENTSEQ_PENDING	Number of sequence events pending in the module
EventSeqCmd	MNETCEVENTCMD_SEQSTAT	Event sequence event commands pending
EventSeqReady	DINT	Status data (bit=1 data ready, bit=0 no event data)
ServerStatus	MNETCSERVERS	Server status data
ServerStatsTrigger	BOOL	Trigger the Server Status block request

3.3.4 MNETC.UTIL

This controller tag structure stores the variables required for the data transfer between the processor and the MVI56E-MNETC/MNETCXT module.

Name	Data Type	Description
LastRead	INT	Index of last read block
LastWrite	INT	Index of last write block
BlockIndex	INT	Computed block offset for data table
StatusIndex	INT	Computed block offset for status data
ReadDataSizeGet	INT	Gets *ReadData* array length
WriteDataSizeGet	INT	Gets *WriteData* array length
ReadDataBlkCount	INT	Holds the value of the block counts of the *ReadData* array
WriteDataBlkCount	INT	Holds the value of the block counts of the *WriteData* array
RBTSremainder	INT	Holds remainder calculation value from the read array
WBTSremainder	INT	Holds remainder calculation value from the write array
PassThru	MNETCPASSTHRU	Modbus pass-through commands
IPsetPending	BOOL	Allows setting module IP address
IPgetPending	BOOL	Allows getting module IP address
InitOutputData	MNETCINITOUTDATA	Used to bring the module into a known state after a restart operation
FaultCode	INT	Fault Code value
CheckInitialization	BOOL	Check initialization trigger
StatusSeqIndex	INT	Computed block offset for status data

The *LastRead* tag stores the latest Read Block ID received from the module. The *LastWrite* tag stores the latest Write Block ID to be sent to the module. The *BlockIndex* tag is an intermediate variable used during the block calculation.

4 Diagnostics and Troubleshooting

The module provides information on diagnostics and troubleshooting in the following forms:

- LED status indicators on the front of the module provide information on the module's status.
- Status data contained in the module can be viewed in *ProSoft Configuration Builder* through the Ethernet port.
- Status data values are transferred from the module to the processor.

4.1 Ethernet LED Indicators

The Ethernet LEDs indicate the module's Ethernet port status as follows:

LED	State	Description
Data	OFF	Ethernet connected at 10Mbps duplex speed
	AMBER Solid	Ethernet connected at 100Mbps duplex speed
Link	OFF	No physical network connection is detected. No Ethernet communication is possible. Check wiring and cables.
	GREEN Solid or Blinking	Physical network connection detected. This LED must be ON solid for Ethernet communication to be possible.

4.1.1 Scrolling LED Status Indicators

The scrolling LED display indicates the module's operating status as follows:

Initialization Messages

Code	Message
Boot / DDOK	Module is initializing
Ladd	Module is waiting for required module configuration data from ladder logic to configure the application port(s)
Waiting for Processor Connection	Module did not connect to processor during initialization ▪ Sample ladder logic or AOI is not loaded on processor ▪ Module is located in a different slot than the one configured in the ladder logic/AOI ▪ Processor is not in RUN or REM RUN mode
Last config: <date>	Indicates the last date when the module changed its IP address. You can update the module date and time through the module's web page, or with the Optional MVI56E Add-On Instruction.
	After power up and every reconfiguration, the module will display the configuration of the application port(s). The information consists of: **Client** ▪ C0 C2 C3 C4 ….. C29

Operation Messages

After the initialization step, the following message pattern will be repeated.

```
<Backplane Status> <IP Address> <Backplane Status> <Port Status>
```

Code	Message
<Backplane Status>	OK: Module is communicating with processor ERR: Module is unable to communicate with processor. For this scenario, the <Port Status> message above is replaced with "Processor faulted or is in program mode".
<IP Address>	Module IP address
<C0>	OK: Port is communicating without error Communication Errors: port is having communication errors. Refer to Diagnostics and Troubleshooting (page 71) for further information about the error.

4.1.2 Non-Scrolling LED Status Indicators

The non-scrolling LEDs indicate the module's operating status as follows:

LED Label	Status	Indication
APP	OFF	The module is not receiving adequate power or is not securely plugged into the rack. May also be OFF during configuration download.
	GREEN	The MVI56E-MNETC/MNETCXT is working normally.
	RED	The most common cause is that the module has detected a communication error during operation of an application port. The following conditions may also cause a RED LED: ▪ The firmware is initializing during startup ▪ The firmware detects an on-board hardware problem during startup ▪ Failure of application port hardware during startup ▪ The module is shutting down ▪ The module is rebooting due to a ColdBoot or WarmBoot request from the ladder logic or Debug Menu
OK	OFF	The module is not receiving adequate power or is not securely plugged into the rack.
	GREEN	The module is operating normally.
	RED	The module has detected an internal error or is being initialized. If the LED remains RED for over 10 seconds, the module is not working. Remove it from the rack and re-insert it to restart its internal program.
ERR	RED	Not used.

4.2 Clearing a Fault Condition

Typically, if the OK LED on the front of the module remains RED for more than ten seconds, a hardware problem has been detected or the program has exited.

To clear the condition, follow these steps:

1. Turn off power to the rack.
2. Remove the card from the rack.
3. Verify that all jumpers are set correctly.
4. If the module requires a Compact Flash card, verify it is installed correctly.
5. Re-insert the card in the rack and turn the power back on.
6. Verify correct configuration data is being transferred to the module from the ControlLogix controller.

If the module's OK LED does not turn GREEN, verify that the module is inserted completely into the rack. If this does not cure the problem, contact ProSoft Technology Technical Support.

4.3 Troubleshooting the LEDs

Use the following troubleshooting steps if problems occur when the module is powered up. If these steps do not resolve the problem, please contact ProSoft Technology Technical Support.

Processor Errors

Problem Description	Steps to take
Processor Fault	Verify the module is securely plugged into the slot that has been configured for the module in the I/O Configuration of RSLogix. Verify the slot location in the rack has been configured correctly in the ladder logic.
Processor I/O LED flashes	This indicates a problem with backplane communications. A problem could exist between the processor and any installed I/O module, not just the MVI56E-MNETC/MNETCXT. Verify all modules in the rack are configured correctly.

Module Errors

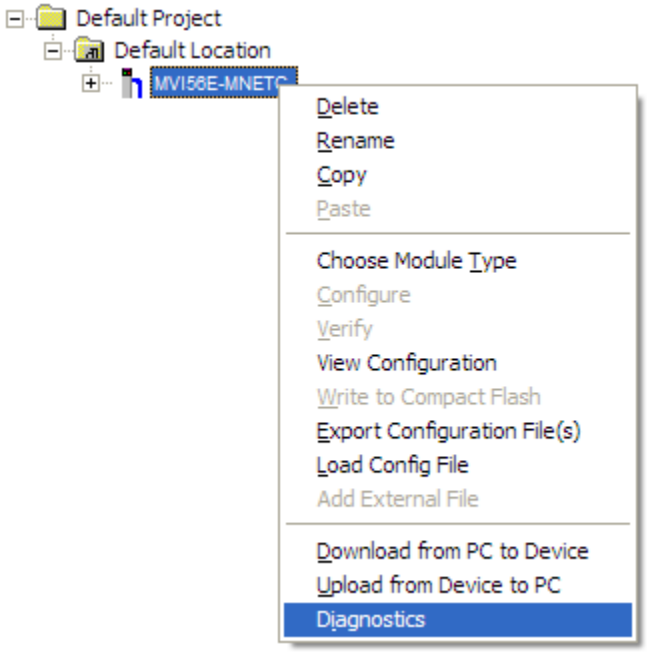
Problem Description	Steps to take
Module Scrolling LED display: <Backplane Status> condition reads ERR	This indicates that backplane transfer operations are failing. Connect to the module's Configuration/Debug port to check this. To establish backplane communications, verify the following items: ▪ The processor is in RUN or REM RUN mode. ▪ The backplane driver is loaded in the module. ▪ The module is configured for read and write data block transfer. ▪ The ladder logic handles all read and write block situations. ▪ The module is properly configured in the processor I/O configuration and ladder logic.
OK LED remains RED	The program has halted or a critical error has occurred. Connect to the communication port to see if the module is running. If the program has halted, turn off power to the rack, remove the card from the rack and re-insert the card in the rack, and then restore power to the rack.

4.4 Using the Diagnostics Menu in ProSoft Configuration Builder

Tip: You can have a ProSoft Configuration Builder *Diagnostics* window open for more than one module at a time.

To connect to the module's Configuration/Debug Ethernet port, refer to Connecting Your PC to the ControlLogix Processor (page 22).

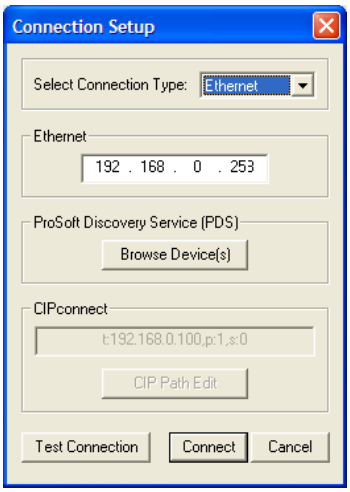
1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** icon to open a shortcut menu.
2. On the shortcut menu, choose **DIAGNOSTICS.**



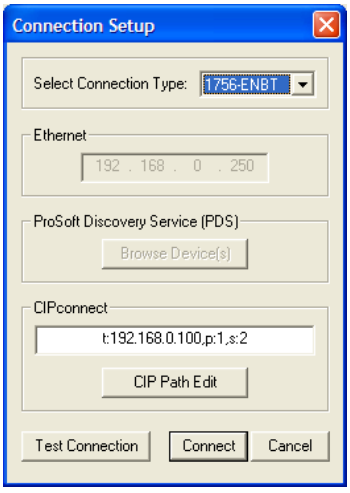
3. In the *Diagnostics* window, click the **SET UP CONNECTION** button to browse for the module's IP address.



4 In the *Connection Setup* dialog box, click the **TEST CONNECTION** button to verify that the module is accessible with the current settings.



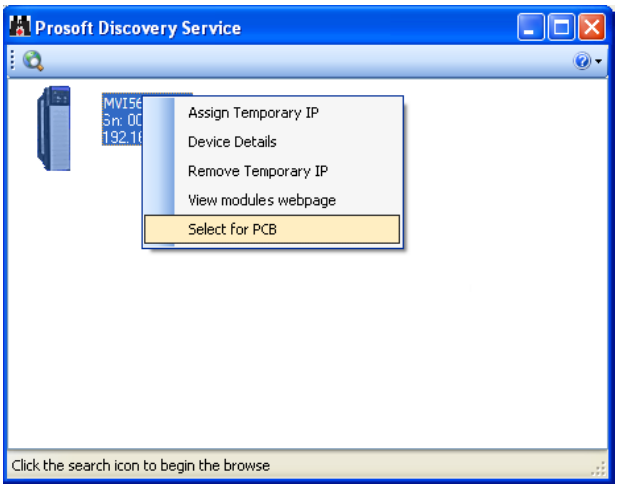
You can also use CIPconnect® to connect to the module through a 1756-ENBT card. Refer to Using CIPconnect to Connect to the Module (page 46) for information on how to construct a CIP path.



5 If the *Test Connection* is successful, click **CONNECT**.

If *PCB* is unable to connect to the module:

1 Click the **BROWSE DEVICE(S)** button to open the *ProSoft Discovery Service*. Select the module, then right-click and choose **SELECT FOR PCB**.

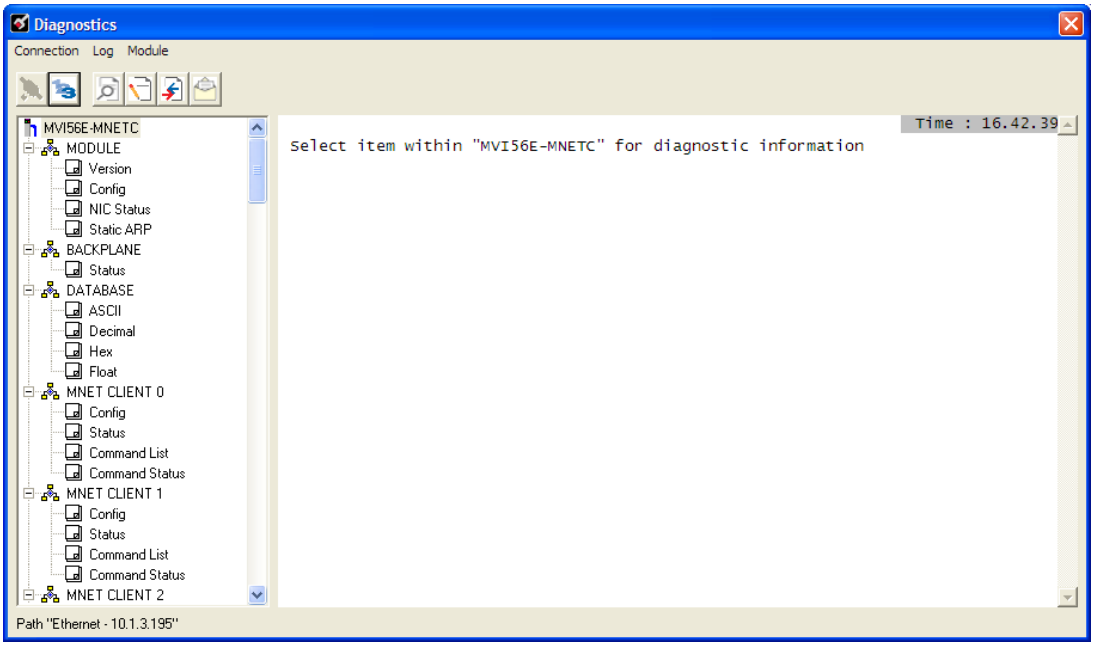


2 Close *ProSoft Discovery Service*, and click the **CONNECT** button again.
3 If these troubleshooting steps fail, verify that the Ethernet cable is connected properly between your computer and the module, either through a hub or switch (using the grey cable) or directly between your computer and the module (using the red cable).

If you are still not able to establish a connection, contact ProSoft Technology for assistance.

4.4.1 The Diagnostics Menu

The Diagnostics menu for this module is arranged as a tree structure, with the Main Menu at the top of the tree, and one or more sub-menus for each menu command. The first menu you see when you connect to the module is the Main menu.



4.4.2 Monitoring Module Information

Use the *MODULE* menu to view configuration and hardware information for the MVI56E-MNETC/MNETCXT module's backplane and Ethernet application port.

Version

Use the *Version* menu to view module hardware and firmware information.

```
MVI56E-MNETC > MODULE > Version :

PRODUCT NAME CODE              :MTE5
SOFTWARE REVISION LEVEL        :2.04
OPERATING SYSTEM REVISION      :1109
RUN NUMBER                     :1201
PROGRAM SCAN COUNTER           :15229
IP ADDRESS                     :10.1.3.195
ETHERNET ADDRESS (MAC)         :00:0d:8d:00:3a:9d
BACKPLANE DRIVER VERSION       :2.06
BACKPLANE API VERSION          :1.01
MODULE NAME                    :MVI56E-MNETC
VENDOR ID                      :309
DEVICE TYPE                    :12
PRODUCT CODE                   :5011
SERIAL NUMBER                  :000003EC
REVISION                       :2.04
SLOT                           :1
```

Config

Use the *Configuration* menu to view backplane configuration settings for the MVI56E-MNETC/MNETCXT module.

The information on this menu corresponds with the configuration information in the *Module* settings in *ProSoft Configuration Builder*.

NIC Status

Use the *NIC Status* (Network Interface Card) menu to view configuration and status information for the MVI56E-MNETC/MNETCXT module's Ethernet application port.

The information on this menu is useful for troubleshooting Ethernet network connectivity problems.

Static ARP

Use the *Static ARP* menu to view the list of IP and MAC addresses that are configured not to receive ARP (Address Resolution Protocol) messages from the module.

The Static ARP Table (page 44) defines a list of static IP addresses that the module will use when an ARP is required.

4.4.3 Monitoring Backplane Information

Use the *BACKPLANE* menu to view the backplane status information for the MVI56E-MNETC/MNETCXT module.

Backplane Status

Use the *Status* menu to view current backplane status, including

- Number of retries
- Backplane status
- Fail count
- Number of words read
- Number of words written
- Number of words parsed
- Error count
- Event count
- Command count

During normal operation, the read, write, and parsing values should increment continuously, while the error value should not increment.

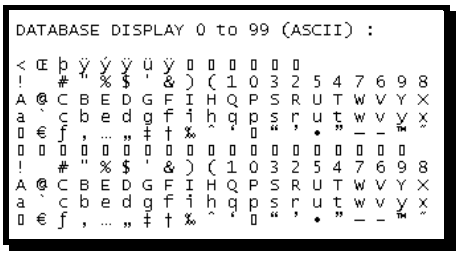
The status values on this menu correspond with members of the Status Data Definition. See Status Data in Read Block (page 96).

4.4.4 Monitoring Database Information

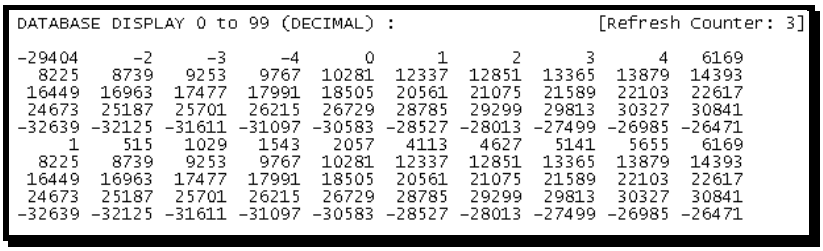
Use the *DATABASE* menu to view the contents of the MVI56E-MNETC/MNETCXT module's internal database.

You can view data in the following formats:

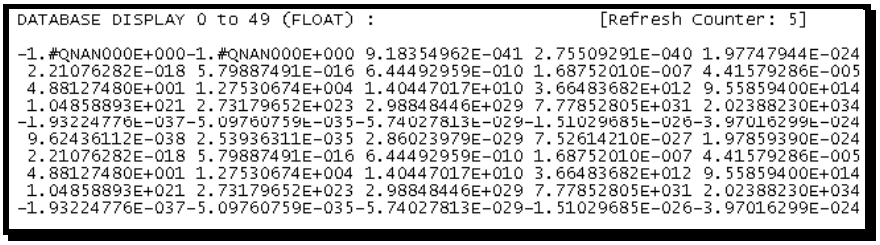
ASCII



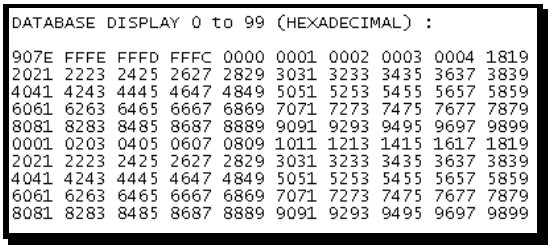
Decimal



Float



Hexadecimal



Use the scroll bar on the right edge of the window to view each page (100 words) of data.

4.4.5 Monitoring MNETC Server Information

The MNETC Server menu contains MNETC server configuration and status information.

Note: To take advantage of the new features described above, your MVI56E-MNETC/MNETCXT module needs to have firmware version 3.01 or higher, and your MVI56E-MNETC/MNETCXT Add-On Instruction needs to be version 1.8 or higher. Earlier versions have no server capabilities and support only up to 5000 user database registers.

Config

Use the *Configuration* menu to view configuration settings for MNET servers connected to the MNET Client.

The information on this menu corresponds with the configuration information in the *MNET Servers* settings in *ProSoft Configuration Builder MNET Servers (page 33) dialog box*. See MNET Servers (page 33).

Status

The *Status* menu contains the status of each MNET servers connected to clients. During normal operation, the number of requests and responses should increment. It will also display the number of connected clients when applicable.

4.4.6 Monitoring MNET Client Information

The MNETC Client x menu contains MNETC client configuration and status information.

Config

The *Configuration* menu contains configuration settings for MNET Client x.

The information on this menu corresponds with the configuration information in the *MNET Client x* settings in *ProSoft Configuration Builder*.

Status

The *Status* menu contains MNET Client x status information. During normal operation, the number of requests and responses should increment.

Command List

The *Command List* menu contains MNET Client x command list settings. The information on this menu corresponds with the *MNET Client x Commands* settings in *ProSoft Configuration Builder*.

Use the scroll bar on the right edge of the window to view each MNET Client command.

Command Status

The *Command Status* menu contains MNET Client x Command status.

A zero indicates no error.

A non-zero value indicates an error. For an explanation of each value, see Client Command Errors (page 86).

4.5 Reading Status Data from the Module

Module status information is useful for troubleshooting and can be accessed in several different ways.

In the ladder logic's *MNETC.STATUS* controller tag structure.

The MVI56E-MNETC/MNETCXT module returns status data in the input image that can be used to determine the module's operating status. This data is transferred from the module to the ControlLogix processor continuously as part of the Normal Data Transfer Blocks (page 95). You can view this data in the *MNETC.STATUS* controller tag structure in the ladder logic.

Client status data can also be requested and returned in a special Client Status Blocks (3000 to 3029) (page 103), outside of the normal data transfer block sequence. The status data contained in the Client Status block is different from the status data in the normal data transfer blocks. It can also be viewed in the *MNETC.STATUS* controller tag structure.

For more information about status data in *MNETC.STATUS*, see the Status Data Definition (page 83).

In *ProSoft Configuration Builder's Diagnostics* screens.

For more information, see the section on *The Diagnostics Menu* (page 78).

In database locations specified by *Error/Status Pointers* (optional).

If optional *Error/Status Pointers* are enabled, status data can also be found in the Read Data area of the module's database at the locations specified by the pointer configuration parameters. For more information, see Backplane Error/Status Pointer (page 30), Client Error/Status Pointer (page 35) and Command Error Pointer (page 35).

4.5.1 Status Data Definition

This section contains a description of the controller tags in the *MNETC.STATUS* controller tag structure, which contains module and Client status data.

- The first ten controller tags contain status data routinely transferred from the module to the processor in the Normal Data Transfer Blocks (page 95).
- The next controller tags are used to request and receive Client status data via the Client Status Blocks (3000 to 3029) (page 103).
- The next controller tags are the Event Command Blocks with Sequence Number (4000 to 4029, 4100 to 4129, 4200) (page 105).
- The remaining controller tags are used to request and receive server status data via the Server Client Status Blocks (3000 to 3029) (page 103).

Note: To access up-to-date status data from these remaining controller tags, you must ensure that a Client Status block or Server Status block was recently received from the module. Client Status blocks and Server Status blocks are not routinely sent from the module; they are returned on a once-per-request basis as a response to a Client Status block request or Server Status block request from the processor

Note: To take advantage of the new features described above, your MVI56E-MNETC/MNETCXT module needs to have firmware version 3.01 or higher, and your MVI56E-MNETC/MNETCXT Add-On Instruction needs to be version 1.8 or higher. Earlier versions have no server capabilities and support only up to 5000 user database registers.

Controller Tag	Data Type	Description
PassCnt	INT	This value is incremented each time a complete program cycle occurs in the module.
ProductVersion	INT	Product version
ProductCode	INT[2]	Product code
BlockStats.Read	INT	Total number of read blocks transferred from the module to the processor
BlockStats.Write	INT	Total number of write blocks transferred from the processor to the module
BlockStats.Parse	INT	Total number of blocks successfully parsed that were received from the processor
BlockStats.Event	INT	Total number of Event Command blocks received from the processor
BlockStats.Cmd	INT	Total number of Command Control blocks received from the processor
BlockStats.Err	INT	Total number of block errors recognized by the module
CmdBits[x]	INT	Displays enabled or disabled status of all 16 commands in the *Client x Command List* for each Client
ClientStatsTrigger	BOOL	Initiates request for Client Status block from module when set to **1**
ClientID	INT	Specifies Client (**0** to **29**) to request status data from
ClientStatus[x].CmdReq	INT	Total number of command list requests sent from Client
ClientStatus[x].CmdResp	INT	Total number of command list responses received by Client

Controller Tag	Data Type	Description
ClientStatus[x].CmdErr	INT	This value is incremented each time an error message is received from a remote unit or a local error is generated for a command.
ClientStatus[x].Requests	INT	Not used
ClientStatus[x].Responses	INT	Not used
ClientStatus[x].ErrSent	INT	Not used
ClientStatus[x].ErrRec	INT	Not used
ClientStatus[x].CfgErrWord	INT	Configuration Error Word - This word contains a bitmap that indicates general module configuration errors.
ClientStatus[x].CurErr	INT	Most recent error code recorded for the Client
ClientStatus[x].LastErr	INT	Previous most recent error code recorded for the Client
CmdErrorList[x]	INT	Command error code for each command (0-15) on the specified Client's command list
EventSeqCmdPending. Trigger	BOOL	Set to **1** to trigger the command queue status data request.
EventSeqCmdPending. Client[x]_QueueCount	SINT	Number of Event sequence commands for which status has not yet been retrieved
EventSeqCmdPending. Client[x]_WaitingMsgs	SINT	Total number of commands waiting in the command queue
EventSeqCmd.Trigger	BOOL	Set the value of this tag to **1** to trigger the Event Sequence Command block request.
EventSeqCmd.ClientID	INT	Specifies Client (**0** to **29**) to request status data from.
EventSeqCmd.Client[x]. Count	INT	Number of Event Sequence Commands executed by the specified Client for which status has not yet been retrieved
EventSeqCmd.Client[x]. Cmd[x].Sequence	INT	Sequence number for each Event Sequence command
EventSeqCmd.Client[x]. Cmd[x].Error	INT	Error code for each Event Sequence command: **0**=success, **1**=failure
EventSeqReady	DINT	One bit for each Client, indicating which Clients have Event Sequence commands with unretrieved status data: **0**=no status data available, **1**=status data available for retrieval
ServerStatusTrigger	BOOL	Set to **1** to trigger the Server Status block request.
ServerStatus.MNET. Requests	INT	This counter increments each time an MNet (port 2000) request is received.
ServerStatus.MNET. Responses	INT	This counter increments each time an MNet (port 2000) response message is sent.
ServerStatus.MNET. ErrSent	INT	This counter increments each time an MNet (port 2000) sends an exception response to Client. Example: Client sent illegal Modbus Data location address.
ServerStatus.MNET. ErrRec	INT	This counter increments each time an MNet (port 2000) receives a bad command. Example: Client sent illegal function command.
ServerStatus.MBAP. Requests	INT	This counter increments each time an MBAP (port 502) request is received.
ServerStatus.MBAP. Responses	INT	This counter increments each time an MBAP (port 502) response message is sent.
ServerStatus.MBAP. ErrSent	INT	This counter increments each time an MBAP (port 502) sends an exception response to Client. Example: Client sent illegal Modbus Data location address.

Controller Tag	Data Type	Description
ServerStatus.MBAP.ErrRec	INT	This counter increments each time an MBAP (port 502) receives a bad command. Example: Client sent illegal function command.

4.5.2 Configuration Error Word

The *Configuration Error Word* contains Client configuration error indications, in a bit-mapped format. Specific bits in the module's *Configuration Error Word* are turned on (set to 1) to indicate various configuration errors. The *Configuration Error Word* appears in the *MNETC.STATUS.ClientStatus[x]* controller tag array.

Bits set to 1 in the *Configuration Error Word* indicate the following errors.

Bit	Description	Hex Value
0	Reserved - not currently used	0001h
1	Reserved - not currently used	0002h
2	Reserved - not currently used	0004h
3	Reserved - not currently used	0008h
4	Invalid retry count parameter	0010h
5	The float flag parameter is not valid.	0020h
6	The float start parameter is not valid.	0040h
7	The float offset parameter is not valid.	0080h
8	The ARP Timeout is not in range (ARP Timeout parameter 0 or greater than 60000 milliseconds) and will default to 5000 milliseconds.	0100h
9	The Command Error Delay is > 300 and will default to 300.	0200h
10	Reserved - not currently used	0400h
11	Reserved - not currently used	0800h
12	Reserved - not currently used	1000h
13	Reserved - not currently used	2000h
14	Reserved - not currently used	4000h
15	Reserved - not currently used	8000h

Combinations of errors will result in more than one bit being set in the error word. Correct any invalid data in the configuration for proper module operation. A value of zero (0) in this word indicates all bits are clear, which means that all module configuration parameters contain valid values. However, this does not mean that the configuration is valid for the user application. Make sure each parameter is set correctly for the intended application.

4.5.3 Client Command Errors

There are several different ways to view Client Command Errors.

- In the *MNETC.STATUS.CmdErrorList* controller tag array
- On the Client status data screens in the *ProSoft Configuration Builder Diagnostics*
- At a module database location specified by the configuration's *MNET Client x Command Error Pointer*, if the *Command Error Pointer* is enabled. This means that the first register refers to command 1 and so on.

Word Offset	Description
0	Command 0 Error
1	Command 1 Error
2	Command 2 Error
3	Command 3 Error
…	….
…	…
15	Command 15 Error
16	Command 16 Error

For every command that has an error, the module automatically sets the *Poll Delay* parameter to the configured value in the *Command Error Delay* (in seconds). This instructs the module to wait for X seconds until it attempts to issue the command again. If set to 0, the module does not use the *Command Error Delay* and polls based on the configured *Poll Delay* in the Client Command list.

As the commands in the Client Command List are polled and executed, an error value is maintained in the module for each command. This error list can be transferred to the processor.

Standard Modbus Exception Code Errors

Code	Description
1	Illegal function
2	Illegal data address
3	Illegal data value
4	Failure in associated device
5	Acknowledge
6	Busy; message was rejected

Module Communication Error Codes

Code	Description
-2	Timeout while transmitting message
-11	Timeout waiting for response after request (same as -36)
253	Incorrect slave/server address in response
254	Incorrect function code in response
255	Invalid CRC/LRC value in response

MNET Client Specific Errors

Code	Description
-33	Failed to connect to server specified in command
-35	Invalid length of response message
-36	MNET command response timeout
-37	TCP/IP connection ended before session finished

Command List Entry Errors

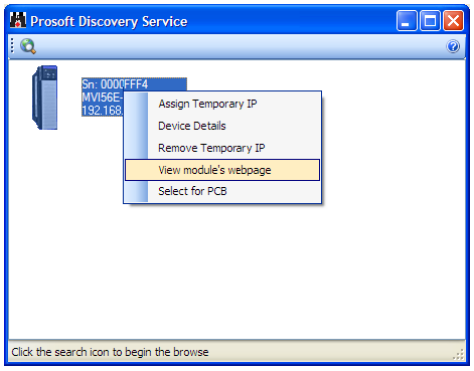
Code	Description
-40	Too few parameters
-41	Invalid enable code
-42	Internal address > maximum address
-43	Invalid node address (<0 or >255)
-44	Count parameter set to 0
-45	Invalid function code
-46	Invalid swap code
-47	Could not establish a connection. ARP could not resolve MAC from IP (Bad IP address, not part of network, invalid parameter to ARP routine).
-48	Error during ARP operation: the response to the ARP request did not arrive to the module after a 5 second timeout.

Note: When the Client gets error -47 or -48, it uses the adjustable ARP Timeout parameter in the configuration file to set an amount of time to wait before trying again to connect to this non-existent server. This feature allows the Client to continue sending commands and polling other existing servers, while waiting for the non-existent server to appear on the network.

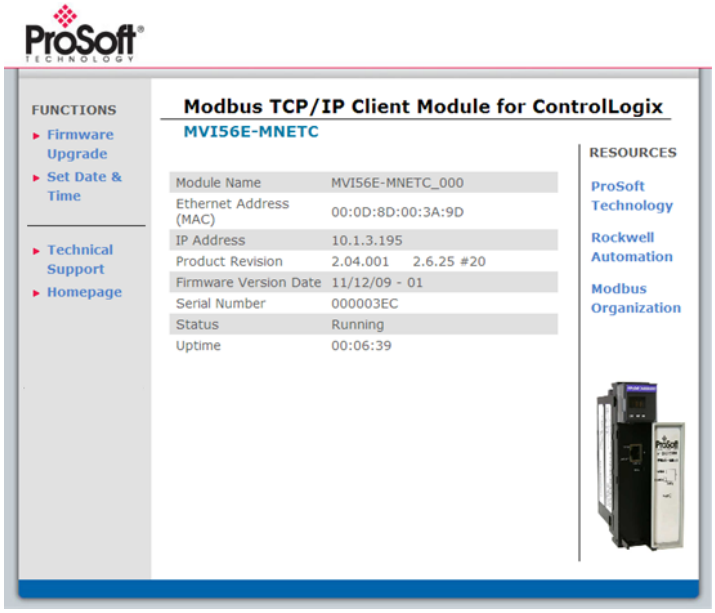
4.6 Connecting to the Module's Webpage

The module's internal webserver provides access to general product information, firmware download link, and links to the ProSoft Technology's website.

1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETC/MNETCXT** icon and then choose **DOWNLOAD FROM PC TO DEVICE**.
2. In the *Download* dialog box, choose the connection type in the *Select Connection Type* dropdown box:
   - Choose **ETHERNET** if you are connecting to the module through the Ethernet cable.
   - Choose **1756 ENBT** if you are connecting to the module through CIPconnect or RSWho.
     See Connecting Your PC to the Module (page 45) for more information.
3. In the *Download files from PC to module* dialog box, click **BROWSE DEVICE(S)**.
4. In *ProSoft Discovery Service*, right-click the **MVI56E-MNETC/MNETCXT** icon and then choose **VIEW MODULE'S WEBPAGE**.



This displays the module webpage.



5 Reference

5.1 Product Specifications

The MVI56E-MNET/MNETCXT Modbus TCP/IP Client Enhanced Communication Module - Client/Server allows Rockwell Automation® ControlLogix® Programmable Automation Controllers (PACs) to interface easily with Modicon Programmable Automation Controller (PACs) , as well as multiple Modbus TCP/IP server-compatible instruments and devices. The multi-Client module improves performance when controlling multiple servers on a Modbus TCP/IP network, by supporting up to 30 Clients. It also supports up to 40 server connections (10 Encapsulated, 30 MBAP).

MVI56E enhancements include configuration and management through the module's Ethernet port, and CIPconnect® technology for bridging though ControlNet™ and EtherNet/IP™ networks.

5.1.1 General Specifications

- Backward compatible with previous MVI56-MNETC versions
- Single-slot 1756 ControlLogix backplane compatible
- 10/100 Mbps auto crossover detection Ethernet configuration and application port
- User-definable module data memory mapping of up to 10,000 16-bit registers
- CIPconnect-enabled network configuration and diagnostics monitoring using ControlLogix 1756-ENxT and 1756-CNB modules and EtherNet/IP pass-through communication
- ProSoft Configuration Builder (PCB) software supported, a Windows-based graphical user interface providing simple product and network configuration
- Sample ladder logic and Add-On Instructions (AOI) are used for data transfer between module and processor
- 4-character, alpha-numeric, scrolling LED display of status and diagnostics data in plain English – no cryptic error or alarm codes to decipher
- ProSoft Discovery Service (PDS) software used to locate the module on the network and assign temporary IP address
- Personality Module - a non-volatile industrial-grade Compact Flash (CF) card used to store network and module configuration for easy disaster recovery, allowing quick in-the-field product replacement by transferring the CF card

Modbus TCP/IP Specifications

ProSoft Technology's Modbus TCP/IP implementation (MNET) includes both Client (Master) and server (slave) capabilities.

Modbus TCP/IP Server (Slave)

- Supports 30 independent server connections for Service Port 502 (MBAP)
- Supports 10 independent server connections for Service Port 2000 (Encapsulated)
- Accepts Modbus Function Codes 1, 2, 3, 4, 5, 6, 7, 8, 15, 16, 17, 22 and 23
- Module data can be derived from other Modbus server devices on the network through the Client or from the ControlLogix processor

Modbus TCP/IP Client (Master)

- Offers 30 Client connections with up to 16 commands each to talk to multiple servers
- Actively reads data from and writes data to Modbus TCP/IP devices, using MBAP or Encapsulated Modbus message formats
- Transmits Modbus Function Codes 1, 2, 3, 4, 5, 6, 7, 15, and 16
- ControlLogix processor can be programmed to use special functions to control the activity on the Client by actively selecting commands to execute from the command list (Command Control) or by issuing commands directly from the ladder logic (Event Commands)

5.1.2 Functional Specifications

- Modbus data types overlap in the module's memory database, so the same data can be conveniently read or written as bit-level or register-level data.
- Configurable floating-point data movement is supported, including support for Enron or Daniel® floating-point formats
- Special functions (Event Commands, Command Control, status, etc.) are supported by message transfer (unscheduled) using the MSG instruction
- Configurable parameters for the Client including a minimum response delay of 0 to 65535 ms and floating-point support
- Supports up to 30 Clients with up to 16 commands for each Client
- Error codes, counters, and module status available from module memory through the servers, through the Clients, or through the ladder logic and controller tags in RSLogix 5000

5.1.3 Hardware Specifications

Specification	Description
Dimensions	Standard 1756 ControlLogix® single-slot module
Backplane current load	800 mA @ 5 VDC 3 mA @ 24 VDC
Operating temperature	0°C to 60°C (32°F to 140°F) -25°C to 70°C (-13°F to 140°F) - MVI56E-MNETCXT
Storage temperature	-40°C to 85°C (-40°F to 185°F)
Extreme/Harsh Environment	MVI56E-MNETXT comes with conformal coating
Shock	30 g operational 50 g non-operational
Vibration	5 g from 10 to 150 Hz
Relative humidity	5% to 95% (with no condensation)
LED indicators	Application Status (APP) Module Status (OK)
4-character, scrolling, alphanumeric LED display	Shows module, version, IP, application port setting, port status, and error information
Ethernet port	10/100 Base-T, RJ45 Connector, for CAT5 cable Link and Activity LED indicators Auto-crossover cable detection

5.2 Functional Overview

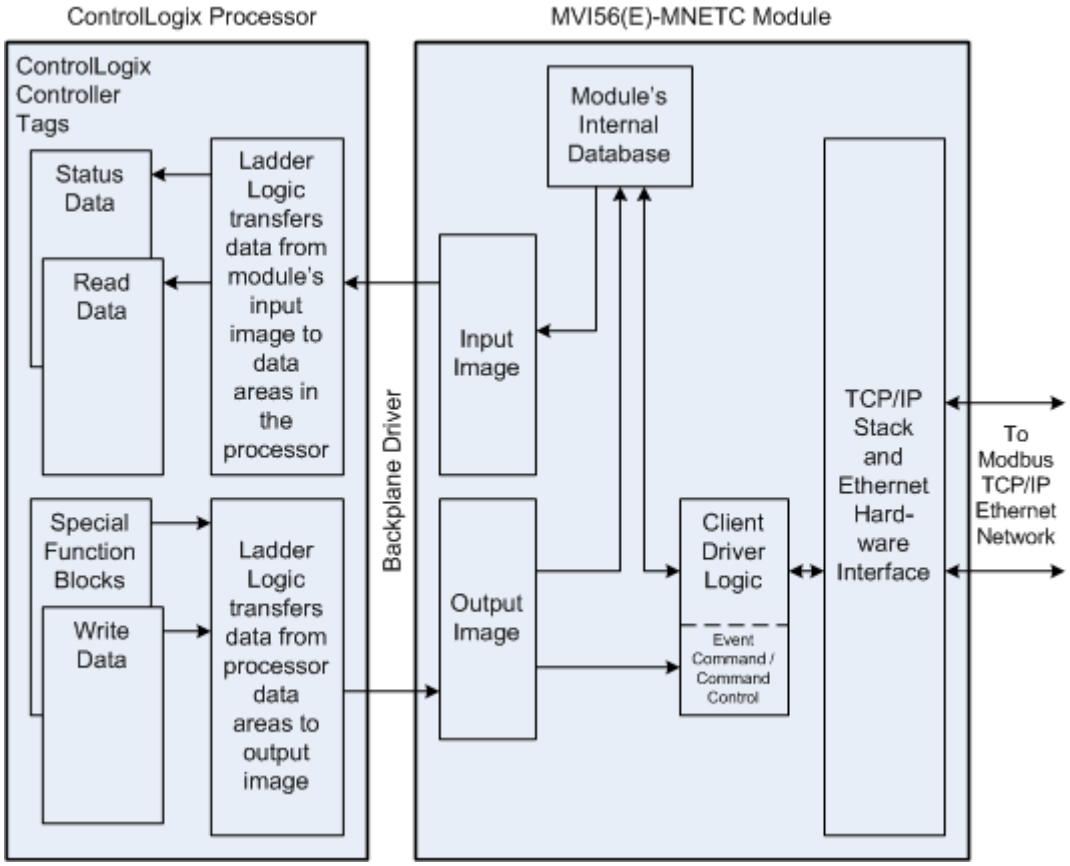
5.2.1 Backplane Data Transfer

The MVI56E-MNETC/MNETCXT module communicates directly over the ControlLogix backplane. Data is paged between the module and the ControlLogix processor across the backplane using the module's input and output images. The update frequency of the images is determined by the scheduled scan rate defined by the user for the module and the communication load on the module. Typical update times range from 1 to 10 milliseconds.

This bi-directional transfer of data is accomplished by the module putting data in the input image to send to the processor. Data in the input image is placed in the processor's controller tags by ladder logic. The input image is set to 250 words.

Processor logic inserts data to the output image to be transferred to the module. The module's firmware program extracts the data and places it in the module's internal database. The output image is set to 248 words.

The following illustration shows the data transfer method used to move data between the ControlLogix processor, the MVI56E-MNETC/MNETCXT module and the Modbus TCP/IP Network.



All data transferred between the module and the processor over the backplane is through the input and output images. Ladder logic must be written in the ControlLogix processor to interface the input and output image data with data defined in the controller tags. All data used by the module is stored in its internal database. This database is defined as a virtual Modbus data table with addresses from 0 (40001 Modbus) to 9999 (50000 Modbus).

Module's Internal Database Structure

10,000 registers for user data



Data contained in this database is transferred in blocks, or pages, using the input and output images. ControlLogix ladder logic and the MVI56E-MNETC/MNETCXT module's program work together to coordinate these block transfers. Up to 200 words of data can be transferred from the module to the processor (read block - input image) or from the processor to the module (write block - output image) in each block transfer. The block structure of each block type depends on the data content and function of the block. The module uses the following block identification numbers:

Block ID Range	Descriptions
-1	Null block
0	For firmware versions earlier than 2.05, this is a null block. For firmware versions 2.05 and newer, block 0 contains the same data as block 1. This feature enhances performance, especially when using less than 200 words of read/write data: ▪ If Read Register Count in the module configuration file is set > 200 words, Block ID 0 is not used. ▪ If Read Register Count in the module configuration file is set >0 and <= 200 words, Block ID contains the same data as block 1 (both read data and status data).
1 to 50	Read or Write blocks
1000 to 1049	Initialize Output Data blocks
2000 to 2029	Event Command blocks
3000 to 3029	Client Status blocks
5001 to 5016	Command Control blocks
9990	Set Module IP Address block
9991	Get Module IP Address block
9998	Warm-boot block
9999	Cold-boot block
3100	Server Status block
4000 to 4029	Event Sequence Command blocks
4100 to 4129	Event Sequence Command Error Report blocks
4200 to 4229	Event Sequence Command Count Status blocks
5001 to 5016	Command Control blocks
9956	Formatted Pass-through block from function 6 or 16 with word data
9957	Formatted Pass-through block from function 6 or 16 with floating-point data
9958	Formatted Pass-through block from function 5
9959	Formatted Pass-through block from function 15
9960	Formatted Pass-through block from function 22
9961	Formatted Pass-through block from function 23
9970	Function 99 indication block
9990	Set Module IP Address block
9991	Get Module IP Address block
9996	Unformatted Pass-through block with raw Modbus message
9997	Reset Module Status Data block
9998	Warm-boot block
9999	Cold-boot block

These block identification codes can be broken down into two groups:

Normal data transfer blocks

- Read and Write blocks (-1 to 50)

Special function blocks

- Initialize Output Data blocks (1000 to 1049)
- Event Command blocks (2000 to 2029)
- Client Status blocks (3000 to 3029)
- Server Status block (3100)
- Event Sequence Command blocks (4000 to 4029)
- Event Sequence Command Status blocks (4100 to 4129 and 4200)
- Command Control blocks (5001 to 5016)
- Pass-through Request blocks (9956 to 9961, 9970, and 9996)
- Module IP Address blocks (9990 and 9991)
- Reset Module Status block (9997)
- Warm-boot and Cold-boot blocks (9998 and 9999)

5.2.2 Normal Data Transfer Blocks

Normal data transfer includes the paging of user data from the module's internal database (registers 0 to 9999), as well as paging of status data. These data are transferred through read (input image) and write (output image) blocks.

The following topics describe the function and structure of each block.

Read Block

These blocks of data transfer information from the module to the ControlLogix processor.

The following table describes the structure of the input image.

Read Block from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Read Block ID: -1 to 50	1
2 to 201	Read Data	200
202	Program Scan Counter	1
203 to 208	Block Transfer Status	6
209 to 238	Command bit data for Clients	30
239 to 240	Product Code	2
241	Product Version	1
242 to 248	Reserved	7
249	Read Block ID	1

The Read Block ID is an index value used to determine where the 200 words of data from module memory will be placed in the *ReadData[x]* controller tag array of the ControlLogix processor. Each transfer can move up to 200 words (block offsets 2 to 201) of data. In addition to moving user data, the block also contains status data for the module. The Write Block ID associated with the block requests data from the ControlLogix processor.

During normal program operation, the module sequentially sends read blocks and requests write blocks.

For example, if the application uses three read and two write blocks, the sequence will be as follows:

R1W1→R2W2→R3W1→R1W2→R2W1→R3W2→R1W1→

This sequence will continue until interrupted by other write block numbers sent by the controller or by a command request from a node on the Modbus network or operator control through the module's Configuration/Debug port.

Status Data in Read Block

The following table describes in more detail the status information found in the Read Block.

Word Offset	Content	Description
202	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
203	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
204	Write Block Count	This field contains the total number of write blocks transferred from the processor to the module.
205	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
206	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
207	Command Block Count	This field contains the total number of command blocks received from the processor.
208	Error Block Count	This field contains the total number of block errors recognized by the module.
209	Client 0 command execution word	Each bit in this word enables/disable the commands for Client 0. If the bit is set, the command will execute. If the bit is clear, the command will be disabled. This data is set in the output image (WriteBlock) from the ladder logic.
210 to 238	Client 1 to Client 29 command execution words	These 29 words are used for each of the other 29 Clients in the module. This data is set in the output image (WriteBlock) from the ladder logic.
239 to 240	Product Code	The product ID code for the module
241	Product Version	The firmware version number for the module

Status information transferred in the Read block can be viewed in the *MNETC.STATUS* controller tag in the ladder logic. For more information, see the Status Data Definition (page 83).

Write Block

These blocks of data transfer information from the ControlLogix processor to the module.

The following table describes the structure of the output image.

Write Block from Processor to Module

Word Offset	Description	Length
0	Write Block ID: -1 to 50	1
1 to 200	Write Data	200
201 to 230	Command bit data for Clients (set)	30
231 to 246	Spare	16
247	Select Priority Read	1

The Write Block ID is an index value used to determine the location in the module's database where the data will be placed. Each transfer can move up to 200 words (block offsets 1 to 200) of data.

Select Priority Read Block (Write Block Offset 247)

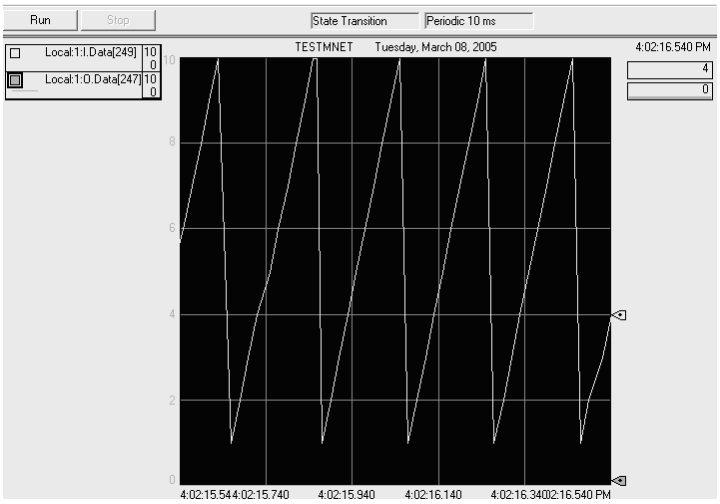
Note: The Select Priority Read Block feature is only available for firmware versions 1.36.000 and higher.

This register allows the processor to select which read blocks will be returned from the module. If this register equals zero, the module will return all read blocks in sequential order.

If this register has a non-zero value, the module will return the read block selected, and the following one.

This feature can be used for applications that require some read blocks to be updated more frequently than other blocks.

The following illustrations show the effect of changing the value of the Select Priority Read Block register (Write Block offset 247). In the following histogram curve, the Select Priority Read Block is equal to 0.



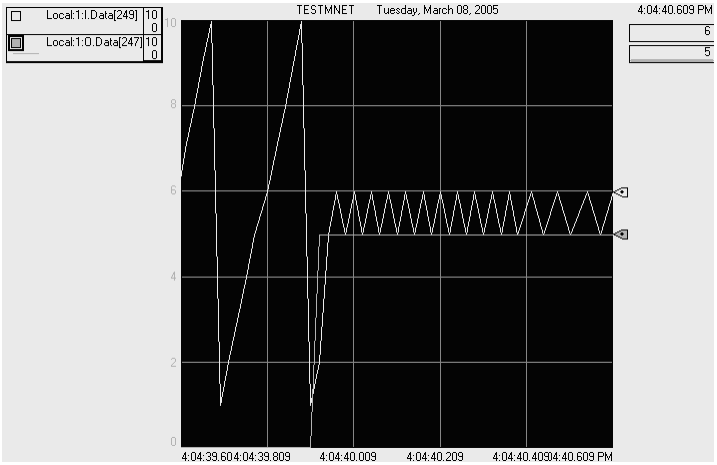
- Local:1.O.Data[247] = Select Priority Read Block.
- Local:1.I.Data[249] = Read Block ID.

In the example above, all read blocks (1 to 10) are returned in sequential order.

Select Priority Read Block = 5

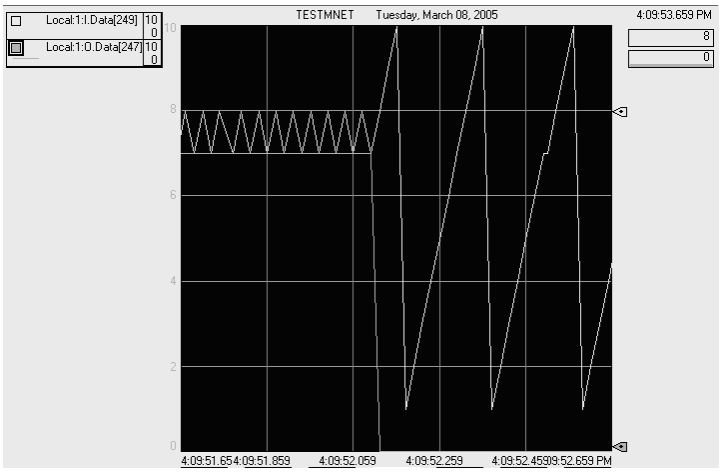
If the ladder logic changes the value of Local:1:O.Data[247] from 0 to 5, note that the Local:1:I.Data[249] value begins to alternate between Block IDs 5 and 6 as long as Local:1:I.Data[247] stays set to 5.

5-6-5-6-5-6-5-6-5-6-…



Select Priority Read Block = 0

After the ladder logic changes the value of Local:1:O.Data[247] from 5 to 0, then the Local:1:I.Data[249] value is updated as before, by returning all blocks 1 through 10 in a repeating sequence.

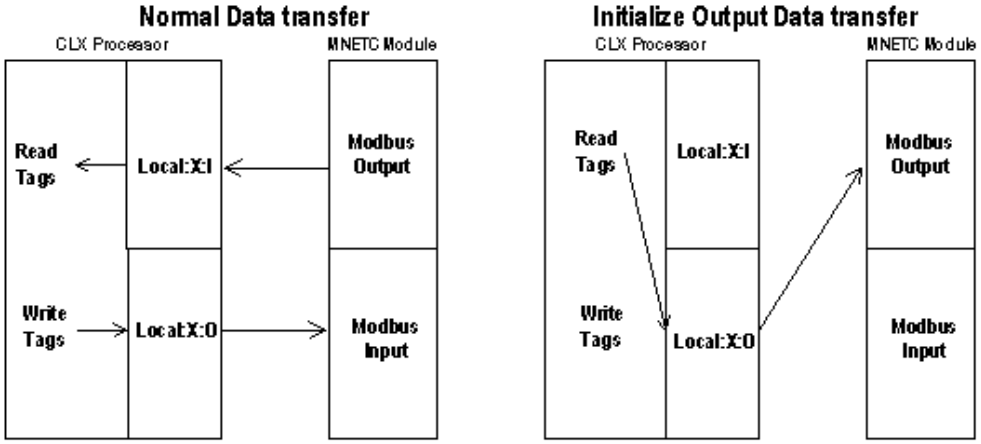


5.2.3 Special Function Blocks

Special function blocks are optional blocks used to request special tasks from the module.

Initialize Output Data Blocks (1000 to 1049)

Use the *Initialize Output Data* parameter in the configuration to bring the module to a known state after a restart operation. If the *Initialize Output Data* parameter is enabled, when the module performs a restart operation, it will request blocks of output data from the *ReadData* array in the processor to initialize the Read Data area of the module's internal database.



Block Request from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	1000 to 1049	1
2 to 248	Spare	247
249	1000 to 1049	1

Ladder logic subtracts 1000 from the value contained in word 249 to determine a block index. This bock index determines which 200-word block of data will be taken from the *ReadData* array and placed in the output image to be returned to the module.

Block Response from Processor to Module

Word Offset	Description	Length
0	1000 to 1049	1
1 to 200	Output data to preset in module.	200
201 to 247	Spare	47

Event Command Blocks (2000 to 2029)

Note: To use the Event Command Block for polling operations, you must configure at least one command in the Client even if the command is disabled.

During routine operation, the module continuously cycles through the user-defined *MNET Client x Commands* (page 37) for each Client, examining commands in the order they are listed, and sending enabled commands on the network. However, the module also has a special command priority queue, which is an internal buffer that holds commands from special function blocks until they can be sent on the network.

When one or more commands appear in the command priority queue:

1. The routine polling process is temporarily interrupted.
2. The commands in the command priority queue are executed until the queue is empty.
3. Then the module goes back to where it left off on the *MNET Client x Command List* and continues routine polling.

Event Command blocks send Modbus TCP/IP commands directly from controller tags by ladder logic to the Client command priority queue on the module. Event Commands are not placed in the module's internal database and are not part of the *MNET Client x Command List*.

Block Request from Processor to Module

Word Offset	Description	Length
0	2000 to 2029 (last digits indicate which Client to utilize)	1
1 to 4	IP Address	4
5	Service Port	1
6	Slave Address	1
7	Internal DB Address	1
8	Point Count	1
9	Swap Code	1
10	Modbus Function Code	1
11	Device Database Address	1
12 to 247	Spare	236

The module will use the parameters passed in this block to construct the command.

- The IP Address for the destination server to reach on the network is entered in four registers (1 to 4). Each octet value (0 to 255) of the destination server's IP address is placed in one of the four registers For example, to interface with node 192.168.0.100, enter the values 192, 168, 0 and 100 in registers 1 to 4.
- The *Service Port* field selects the TCP service port on the server to connect. If the parameter is set to 502, a standard MBAP (Modbus API for network communications) message will be generated. All other service port values will generate a Modbus command message encapsulated in a TCP/IP packet.
- The *Slave Address* is the Modbus node address for the message.
- The *Internal DB Address* parameter specifies the module's database location to associate with the command.
- The *Point Count* parameter defines the number of points or registers for the command.
- The *Swap Code* is used with Modbus functions 3 and 4 requests to change the word or byte order.
- The *Modbus Function Code* has one of the following values 1, 2, 3, 4, 5, 6, 7, 15, or 16.
- The *Device Database Address* is the Modbus register or point in the remote server device to be associated with the command.

The module then places the command in the command priority queue (if the queue is not already full; maximum capacity is 16 commands), and returns a response block to tell the ladder logic whether or not the command has been successfully added to the queue.

Block Response from Module to Processor

Word Offset	Description
0	Reserved
1	This word contains the next read request block identification code.
2	This word contains the result of the event request. If a value of one is present, the command was issued. If a value of zero is present, no room was found in the command priority queue.
3 to 248	Spare
249	This word contains the block identification code 2000 to 2029 requested by the processor.

Word 2 of the block can be used by the ladder logic to determine if the command was successfully added to the command priority queue. The command will fail if the queue for the Client is already full at the time when the Event Command block is received by the module.

Controller Tags

The elements of the *MNETC.CONTROL.EventCmd.Cmd[x]* controller tag array contain all the values needed to build one Modbus TCP/IP command, have it sent to a specific Client on the module, and control the processing of the returned response block.

Controller Tag	Description
IP0	Enter the first octet of the IP address of the target Modbus server.
IP1	Enter the second octet of the IP address of the target Modbus server.
IP2	Enter the third octet of the IP address of the target Modbus server.
IP3	Enter the fourth octet of the IP address of the target Modbus server.
ServPort	Enter **502** for a MBAP message or **2000** for a MNET message.
Node	Enter the Modbus slave node address. Enter **1** to **247**. Enter **0** if not needed.
DBAddress	Enter the module internal database address to associate with the command.
Count	Enter the number of words or bits to be transferred by the Client.
Swap	Enter the swap code for the data. This function is only valid for function codes 3 and 4.
Function	Enter the Modbus function code for the command.
Address	Enter the database address for the server.

When these values have been entered, set the bit in *MNETC.CONTROL.EventCmd.Trigger* to one (**1**) to trigger the execution of the Event Command.

Each of the EventCmd.Cmd[x] represents each individual client (one event command per client).

MNETC Client x	EventCmd.Cmd[x]
Client 0	Cmd[0]
Client 1	Cmd[1]
…	…
Client 29	Cmd[29]

Client Status Blocks (3000 to 3029)

Client status data for a specific Client can be requested and returned in a special Client Status block. The status data contained in the Client Status block is different from the status data contained in the normal data transfer blocks.

Block Request from Processor to Module

Word Offset	Description	Length
0	3000 to 3029 (last digits indicate which Client to consider)	1
1 to 247	Spare	247

Block Response from Module to Processor

Word Offset	Description	Length
0	0	1
1	Write Block ID	1
2	3000 to 3029 number requested	1
3 to 12	Client status data	10
13 to 28	Command error list data for Client	16
29 to 248	Reserved	220
249	3000 to 3029	1

Client Status Data

Word Offset	Client Status
3	Total number of command list requests
4	Total number of command list responses
5	Total number of command list errors
6	Not used
7	Not used
8	Not used
9	Not used
10	Configuration Error Word
11	Current Error
12	Last Error

Status information transferred in the Client Status block can be viewed in the *MNETC.STATUS* controller tag in the ladder logic. For more information, see Status Data Definition (page 83).

Controller Tags

To issue a Client Status block request, enter the appropriate values in the following members of the *MNETC.STATUS* controller tag in the ladder logic.

Controller Tag	Data Type	Description
ClientID	INT	Enter the Client (**0**-**29**) to request status data for.
ClientStatsTrigger	BOOL	Triggers the Client Status block request.

Server Status Blocks (3100)

Server status data for a specific server can be requested and returned in a special Server Status block. The status data contained in the Server Status block is different from the status data contained in the normal data transfer blocks.

Block Request from Processor to Module

Word Offset	Description	Length
0	3100	1
1 to 247	Spare	247

Block Response from Module to Processor

Word Offset	Description	Length
0	0	1
1	Write Block ID	1
2	3100	1
3 to 5	Spare	3
6 to 9	MNET server status data	4
10 to 15	Spare	6
16 to19	MBAP server status data	4
20 to 22	Spare	3
23 to 248	Reserved	226
249	3100	1

Server Status Data

Word Offset	Server Status
6	Total number of MNET command list requests
7	Total number of MNET command list responses
8	Total number of MNET command list errors sent
9	Total number of MNET command list errors received
10 to 15	Not used
16	Total number of MBAP command list requests
17	Total number of MBAP command list responses
18	Total number of MBAP command list errors sent
19	Total number of MBAP command list errors received

Status information transferred in the Server Status block can be viewed in the *MNETC.STATUS* controller tag in the ladder logic. For more information, see Status Data Definition (page 83).

Controller Tags

To issue a Server Status block request, enter the appropriate values in the following members of the *MNETC.STATUS* controller tag in the ladder logic.

Controller Tag	Data Type	Description
ServerStatsTrigger	BOOL	Triggers the Server Status block request.

Event Command Blocks with Sequence Number (4000 to 4029, 4100 to 4129, 4200)

Event Command Block with Sequence Number (4000 to 4029)

Note: To use the Event Command Blocks with Sequence Number for polling operations, you must configure at least one command in the Client even if the command is disabled.

During routine operation, the module continuously cycles through the user-defined *MNET Client x Commands* (page 37) for each Client, examining commands in the order they are listed, and sending enabled commands on the network. However, the module also has a special command priority queue, which is an internal buffer that holds commands from special function blocks until they can be sent on the network.

When one or more commands appear in the command priority queue:

1. The routine polling process is temporarily interrupted.
2. The commands in the command priority queue are executed until the queue is empty.
3. Then the module goes back to where it left off on the *MNET Client x Command List* and continues routine polling.

Event Command blocks send Modbus TCP/IP commands directly from controller tags by ladder logic to the Client command priority queue on the module. Event Commands are not placed in the module's internal database and are not part of the *MNET Client x Command List*.

Block 4000 is functionally identical to Block 2000 with the addition of the Event Command Sequence Number parameter.

Block Request from Processor to Module

Word Offset	Description	Length
0	4000 to 4029 (last digits indicate which Client to utilize)	1
1 to 4	IP Address	4
5	Service Port	1
6	Slave Address	1
7	Internal DB Address	1
8	Point Count	1
9	Swap Code	1
10	Modbus Function Code	1
11	Device Database Address	1
12	Event Command Sequence Number	1
13 to 247	Spare	235

The module will use the parameters passed in this block to construct the command.

- The *IP Address* for the destination server to reach on the network is entered in four registers (1 to 4). Each octet value (**0 TO 255**) of the destination server's IP address is placed in one of the four registers For example, to interface with node 192.168.0.100, enter the values 192, 168, 0 and 100 in registers 1 to 4.
- The *Service Port* field selects the TCP service port on the server to connect. If the parameter is set to 502, a standard MBAP (Modbus API for network communications) message will be generated. All other service port values will generate a Modbus command message encapsulated in a TCP/IP packet.
- The *Slave Address* is the Modbus node address for the message.
- The *Internal DB Address* parameter specifies the module's database location to associate with the command.
- The *Point Count* parameter defines the number of points or registers for the command.
- The *Swap Code* is used with Modbus functions 3 and 4 requests to change the word or byte order.
- The *Modbus Function Code* has one of the following values 1, 2, 3, 4, 5, 6, 7, 15 or 16.
- The *Device Database Address* is the Modbus register or point in the remote server device to be associated with the command.
- The Event Command Sequence Number is the identifier for the command.

The module then places the command in the command priority queue (if the queue is not already full; maximum capacity is 16 commands), and returns a response block to tell the ladder logic whether or not the command has been successfully added to the queue.

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	The next read request block identification code.	1
2	The result of the event request. 1 indicates the command was issued. 0 indicates no room was found in the command priority queue.	1
3 to 248	Reserved	246
249	The block identification code 4000 to 4029 requested by the processor.	1

Controller Tags

The elements of the *MNETC.CONTROL.EventSeqCmd[x]* controller tag array contain all the values needed to build one Modbus TCP/IP command, have it sent to a specific Client on the module, and control the processing of the returned response block.

Controller Tag	Description
IP0	Enter the first octet of the IP address of the target Modbus server.
IP1	Enter the second octet of the IP address of the target Modbus server.
IP2	Enter the third octet of the IP address of the target Modbus server.
IP3	Enter the fourth octet of the IP address of the target Modbus server.
ServPort	Enter **502** for a MBAP message or **2000** for a MNET message.
Node	Enter the Modbus slave node address. Enter **1** to **247**. Enter **0** if not needed.
DBAddress	Enter the module internal database address to associate with the command.
Count	Enter the number of words or bits to be transferred by the Client.
Swap	Enter the swap code for the data. This function is only valid for function codes 3 and 4.
Function	Enter the Modbus function code for the command.
Address	Enter the database address for the server.
Sequence	Enter the sequence number for the command.

When these values have been entered, set the bit in *MNETC.CONTROL.EventSeqCmd.Trigger* to one (**1**) to trigger the execution of the Event Command.

Event Sequence Command Error Report (4100 to 4129)

This block displays the result of each command sent to the Client. The request includes the Client identification and the command sequence number. The response is the event count and error code for each event. A value of 0 in the error code means there was no error detected.

Block Request from Processor to Module

Word Offset	Description	Length
0	4100 to 4129 (last digits indicate which Client to utilize)	1
1 to 247	Reserved	247

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2	Number of Event Sequence Commands executed by the specified Client for which status has not yet been retrieved	1
3 to 32	Sequence number and error code for each Event Sequence Command (up to the 15 most recent commands)	30
33 to 248	Reserved	216
249	Read Block ID: 4100 to 4129	1

Notes:

- The module stores command status data in a queue for the last 15 event sequence commands it has sent out on the Modbus TCP/IP network. The Event Command Sequence Number identifies the status data for an Event Sequence Command.
- When you retrieve event status data for a Client using Block 4100 to 4129, the Client's event status data queue is deleted. If more than 15 commands have been issued since the last retrieval, the early commands are deleted and only the last 15 are saved in the queue.

Controller Tags

To retrieve status data for one client at a time, enter the appropriate values in the following members of the MNETC.STATUS.EventSeqCmd controller tag in the ladder logic.

Controller Tag	Data Type	Description
ClientID	INT	Enter the Client **(0-29**) to request status data for.
Trigger	BOOL	Set the value of this tag to **1** to trigger the Event Sequence Command block request.

Event Sequence Command Count Status (4200)

This block displays the command queue status data for all clients. The response is the priority command queue count and the Event Sequence Command status queue count.

Block Request from Processor to Module

Word Offset	Description	Length
0	4200	1
1 to 247	Reserved	247

Block Response from Module to Processor

Word Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2 to 31	Command queue status data for each Client	30
32 to 248	Reserved	217
249	Read Block ID: 4200	1

In words 2 to 31 of this block, each word has two bytes:

- Byte 1: Number of Event sequence commands for which status has not yet been retrieved (up to 15). This corresponds to the MNETC.STATUS.EventSeqCmdPending.Client[x]_QueueCount controller tag.
- Byte 2: Total number of commands waiting in the command queue. This includes Event Commands, Event Commands with Sequence Numbers, and Command Control messages. This corresponds to the MNETC.STATUS.EventSeqCmdPending.Client[x]_WaitingMsgs tag.

Controller Tags

To retrieve status data for all 30 clients at a time, enter the appropriate values in the MNETC.STATUS.EventSeqCmd controller tag in the ladder logic.

Controller Tag	Data Type	Description
Trigger	BOOL	Triggers the Command queue status data request.

Command Control Blocks

Note: Command Control is not needed for normal Modbus command list polling operations and is needed only occasionally for special circumstances.

During routine operation, the module continuously cycles through the user-defined *MNETC Client x Commands* (page 37) for each Client, examining commands in the order they are listed, and sending enabled commands on the network. However, the module also has a special command priority queue, which is an internal buffer that holds commands from special function blocks until they can be sent on the network.

When one or more commands appear in the command priority queue:

1. The routine polling process is temporarily interrupted.
2. The commands in the command priority queue are executed until the queue is empty.
3. Then the module goes back to where it left off on the *MNETC Client x Command List* and continues routine polling.

Like Event Command blocks, Command Control blocks place commands into the module's command priority queue. Unlike Event Command blocks, which contain all the values needed for one command, Command Control is used with commands already defined in the *MNETC Client x Command List*.

Commands in the *MNETC Client x Command List* may be either enabled for routine polling or disabled and excluded from routine polling. A disabled command has its bit in the *MNETC.CONTROL.CmdControl.WriteCmdBits* controller tag set to zero (**0**) and is skipped during routine polling. An enabled command has its bit in the *WriteCmdBits* controller tag set to one (**1**) and is sent during routine polling. However, Command Control allows any command in the predefined *MNETC Client x Command List* to be added to the command priority queue, whether it is enabled for routine polling or not.

Command Control also gives you the option to use ladder logic to have commands from the *MNETC Client x Command List* executed at a higher priority and out of routine order, if such an option might be required in special circumstances.

A single Command Control block request can place up to 16 commands from the *MNETC Client x Command List* into the command priority queue.

Block Request from Processor to Module

Word Offset	Description	Length
0	Command Control block identification code of **5001** to **5016**. The rightmost digit indicates the number of commands (**1** to **16**) to add to the command priority queue.	1
1	Client index	1
2	This word contains the Command Index for the first command to be entered into the queue.	1
3	Command Index 2	1
4	Command Index 3	1
5	Command Index 4	1
6	Command Index 5	1
7	Command Index 6	1
8	Command Index 7	1
9	Command Index 8	1
10	Command Index 9	1
11	Command Index 10	1
12	Command Index 11	1
13	Command Index 12	1
14	Command Index 13	1
15	Command Index 14	1
16	Command Index 15	1
17	Command Index 16	1
18 to 247	Spare	230

The last digit in the block identification code indicates the number of commands to process. For example, a block identification code of **5003** indicates that three commands are to be placed in the queue. In this case, the first three of the 16 available Command Indexes will be used to determine exactly which three commands will be added to the queue, and to set their order of execution.

Values to enter for the 16 Command Indexes range from **0** to **15** and correspond to the *MNETC Client x Command List* entries, which are numbered from 1 to 16. To determine the Command Index value, subtract one (**1**) from the row number of the command in the *MNETC Client x Command List*, as seen in the *Command Entry* Formats (page 38).

The module responds to a Command Control block request with a response block, indicating the number of commands added to the command priority queue.

Block Response from Module to Processor

Offset	Description	Length
0	Reserved	1
1	Write Block ID	1
2	Number of commands added to command queue	1
3 to 248	Spare	246
249	5001 to 5016	1

Controller Tags

The *MNETC.CONTROL* controller tag array holds all the values needed to create one Command Control block, have it sent to the module, and control the processing of the returned response block.

Controller Tag	Description
CmdID	Enter a decimal value representing the quantity of commands to be requested in the Command Control block (**1** to **16**). This value is used by the ladder logic to generate the Command Control Block ID. The rightmost digits of the Command Control Block ID are the number of commands requested by the block.
CmdControl.ClientID	Enter the Client to issue the commands to (**0** to **29**)
CmdControl.CMDqty	Not used
CmdControl.CmdIndex	Enter the **ROW NUMBER** of the command in the *MNETC Client x Command List* in *Prosoft Configuration Builder* minus **1**. This is a 16-element array. Each element holds one Command Index.
CmdControl.WriteCmdBits	Enter a **1** (enable) or a **0** (disable) to select which commands on the configuration's *Client x Command List* will be executed during routine polling. There is one 16-bit word for each of the 30 Clients. Each of the 16 bits corresponds to one of the 16 commands available to each Client. The state of these *WriteCmdBits* overrides whatever value may be assigned to the *Enable* parameter in the configuration. **Note:** This parameter only affects routine polling. It has no effect on Command Control blocks.
CmdControlPending	Not used
CmdControlTrigger	Set this tag to **1** to trigger the execution of a Command Control block after all the other parameters have been entered.

Pass-Through Blocks (9956 to 9961, 9970 and 9996)

In Pass-Through mode, write messages sent to a server port are passed directly through to the processor. In this mode, the module sends special blocks to the processor when a write request is received from a Client. Ladder logic must handle the receipt of these blocks and place the enclosed data into the proper controller tags in the processor.

There are two basic modes of operation when the pass-through feature is utilized: Unformatted (code 1) and Formatted (code 2 or 3). In the unformatted mode, messages received on the server are passed directly to the processor without any processing. These unformatted blocks require more decoding than the formatted blocks.

The Modbus protocol supports control of binary output (coils - functions 5 and 15) and registers (functions 6 and 16).

Any Modbus function 5, 6, 15 or 16 commands will be passed from the server to the processor using the block identification numbers 9956 to 9961, 9970 and 9996.

Formatted Pass-Through Blocks (9956 to 9961, 9970)

In formatted pass-through mode, the module processes the received write request and generates a special block dependent on the function received. There are two modes of operation when the formatted pass-through mode is selected. If code 2 is utilized (bytes swapped), the data received in the message is presented in the order expected by the processor. If code 3 is utilized (no swap), the bytes in the data area of the message will be swapped. This selection is applied to all received write requests. The block identification code used with the request depends on the Modbus function requested.

Block ID	Modbus Function
9956	6, 16 (word type data)
9957	6, 16 (floating-point)
9958	5
9959	15
9960	22
9961	23
9970	99

Pass-Through Blocks 9956, 9957, 9958, 9960 or 9961 from Module to Processor

Word Offset	Description	Length
0	0	1
1	9956, 9957, 9958, 9960 or 9961	1
2	Number of word registers in Modbus data set	1
3	Starting address for Modbus data set	1
4 to 248	Modbus data set	245
249	9956, 9957, 9958, 9960 or 9961	1

Pass-Through Block 9959 from Module to Processor

Word Offset	Description	Length
0	0	1
1	9959	1
2	Number of word registers in Modbus data set	1
3	Starting word address for Modbus data set	1
4 to 53	Modbus data set	50
54 to 103	Bit mask for the data set. Each bit to be considered with the data set will have a value of 1 in the mask. Bits to ignore in the data set will have a value of 0 in the mask.	50
104 to 248	Spare data area	145
249	9959	1

Pass-Through Block 9970 from Module to Processor (Function 99 request)

Word Offset	Description	Length
0	0	1
1	Read block ID: 9970	1
2	1	1
3	0	1
4 to 248	Spare data area	245
249	Write block ID: 9970	1

The ladder logic copies and parses the received message, and controls the processor as expected by the Client device. The processor responds to the formatted pass-through blocks with a write block.

Response Blocks 9956, 9957, 9958, 9959, 9960, 9961, or 9970 from Processor to Module

Word Offset	Description	Length
0	9956, 9957, 9958, 9959, 9960, 9961, or 9970	1
1 to 249	Spare data area	247

Unformatted Pass-Through Block (9996)

When the unformatted pass-through mode (code 1) is selected, information is passed from the module to the processor with a block identification code of 9996. Word 2 of this block contains the length of the message, and the message starts at word 3. Other controller tags are required to store the controlled values contained in these messages.

Pass-Through Block 9996 from Module to Processor

Word Offset	Description	Length
0	0	1
1	9996	1
2	Number of bytes in Modbus msg	1
3	Reserved (always 0)	1
4 to 248	Modbus message received	245
249	9996	1

The ladder logic copies and parses the received message, and controls the processor as expected by the Client device. The processor responds to the pass-through block with a write block.

Response Block 9996 from Processor to Module

Word Offset	Description	Length
0	9996	1
1 to 247	Spare	247

This informs the module that the command has been processed and can be cleared from the pass-through queue.

Set Module IP Address Block (9990)

Block Request from Processor to Module

Word Offset	Description	Length
0	9990	1
1	First digit of dotted IP address	1
2	Second digit of dotted IP address	1
3	Third digit of dotted IP address	1
4	Last digit of dotted IP address	1
5 to 247	Reserved	243

Block Response from Module to Processor

Word Offset	Description	Length
0	0	1
1	Write Block ID	1
2	First digit of dotted IP address	1
3	Second digit of dotted IP address	1
4	Third digit of dotted IP address	1
5	Last digit of dotted IP address	1
6 to 248	Spare data area	243
249	9990	1

Get Module IP Address Block (9991)

Block Request from Processor to Module

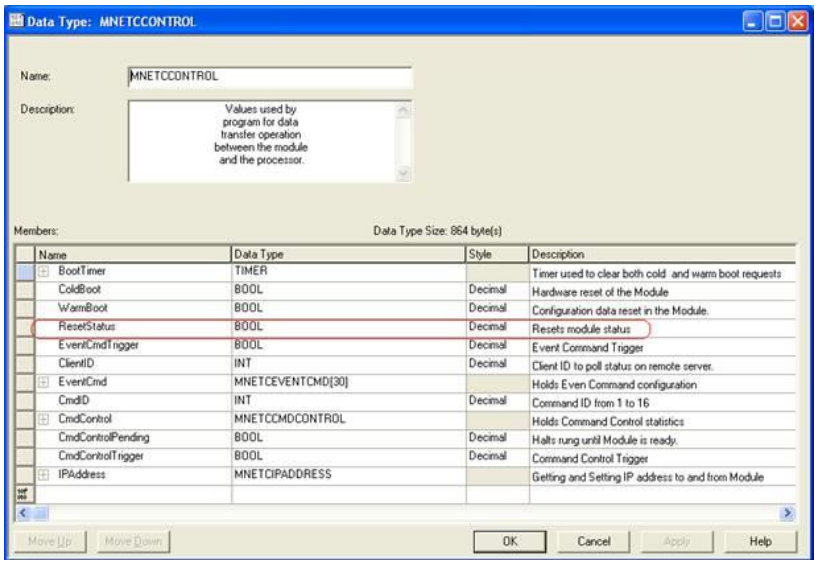
Word Offset	Description	Length
0	9991	1
1 to 247	Spare data area	247

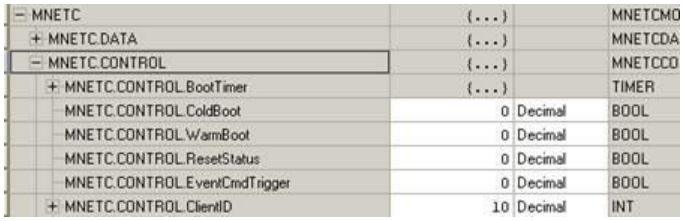
Block Response from Module to Processor

Word Offset	Description	Length
0	0	1
1	Write Block ID	1
2	First digit of dotted IP address	1
3	Second digit of dotted IP address	1
4	Third digit of dotted IP address	1
5	Last digit of dotted IP address	1
6 to 248	Spare data area	243
249	9991	1

Reset Module Status Block (9997)

This block allows the processor to reset all status values available from the module to the processor or through the PCB Diagnostics menu. This block is triggered through the following data type and controller tag elements:





Warm Boot Block (9998)

This block is sent from the ControlLogix processor to the module (output image) when the module is required to perform a warm-boot (software reset) operation. This block is commonly sent to the module any time configuration data modifications are made in the controller tags data area. This will cause the module to read the new configuration information and to restart.

Block Request from Processor to Module

Offset	Description	Length
0	9998	1
1 to 247	Spare	247

Cold Boot Block (9999)

This block is sent from the ControlLogix processor to the module (output image) when the module is required to perform the cold boot (hardware reset) operation. This block is sent to the module when a hardware problem is detected by the ladder logic that requires a hardware reset.

Block Request from Processor to Module

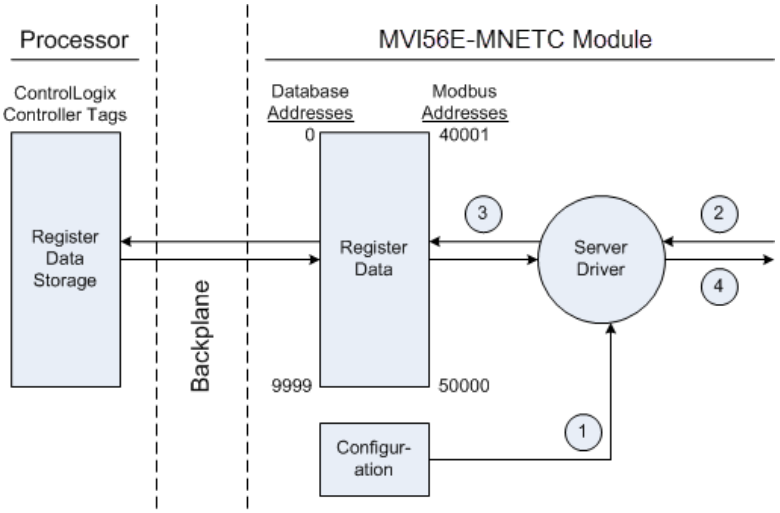
Word Offset	Description	Length
0	9999	1
1 to 247	Spare	247

5.2.4 Data Flow between MVI56E-MNETC/MNETCXT Module and Processor

The following topics describe the flow of data between the two pieces of hardware (processor and MVI56E-MNETC/MNETCXT module) and other nodes on the Modbus TCP/IP network. The module contains up to 30 Clients, which can generate either MBAP (Modbus API for network communications) or MNET requests dependent on the service port selected in the command.

Server Driver

The server driver allows the MVI56E-MNETC/MNETCXT module to respond to data read and write commands issued by Clients on the Modbus TCP/IP network. The following illustration describes the flow of data into and out of the module.



1 The server driver receives the configuration information from the configuration file on the Personality Module (compact flash card), and the module initializes the server.

2 A host device, such as a Modicon PLC or an HMI application, issues a read or write command to the module's node address. The server driver validates the message before accepting it into the module. If the message is considered invalid, an error response is returned to the originating Client node.

3 After the module accepts the command, the module processes the data contained in the command.

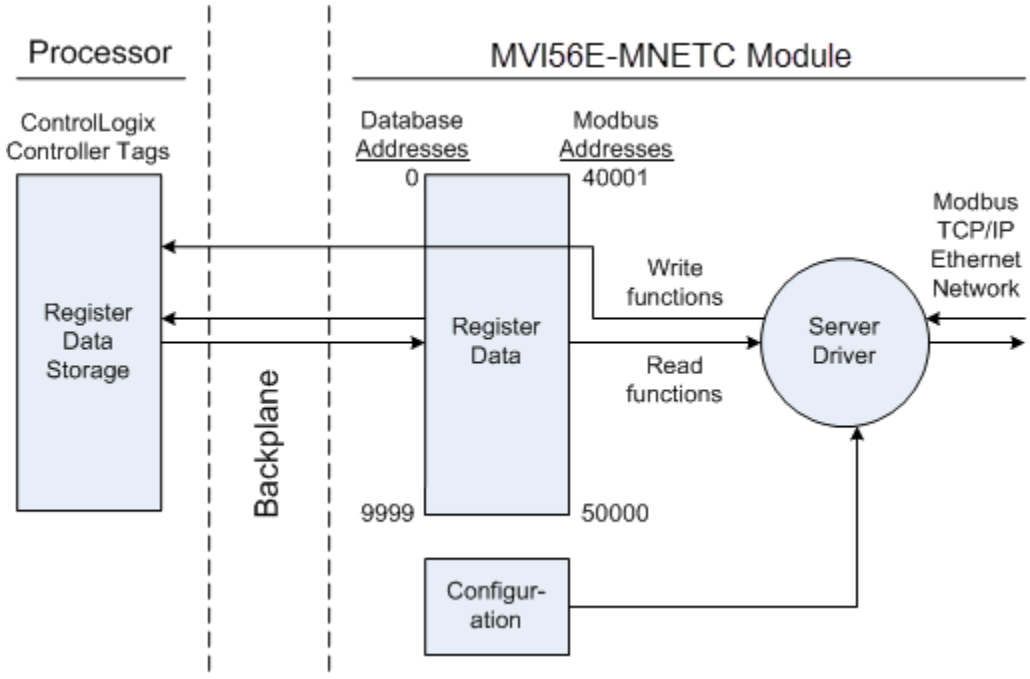
- If the command is a read command, the data is read out of the database and a response message is built.

- If the command is a write command, the data is written directly into the database and a response message is built.
- If the command is a write command and the pass-through feature is utilized, the write message is transferred to the processor ladder logic and is not written directly into the module's database, unless it is returned as a change in the output image that overwrites data in the *WriteData* area as a result of such ladder logic processing.

4 After the data processing has been completed in Step 3, a response is issued to the originating Client node.

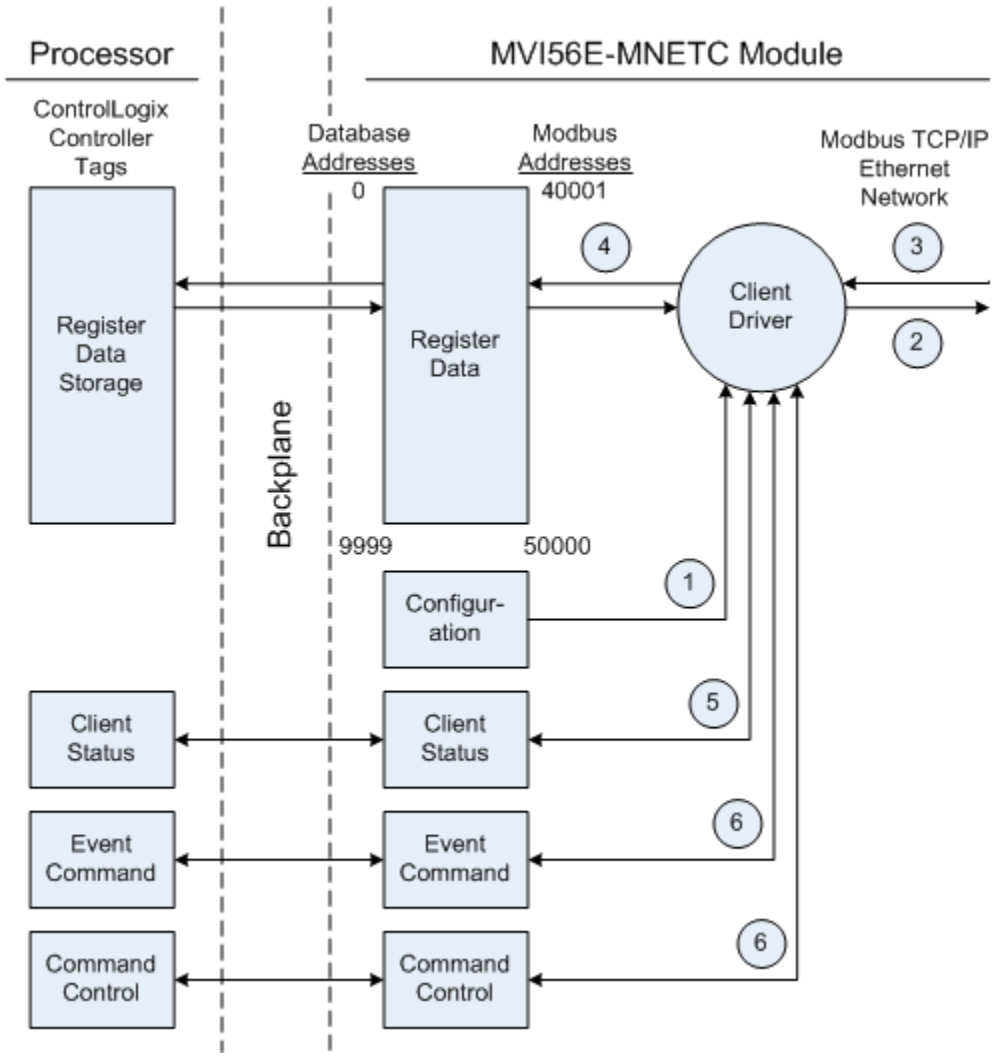
Counters are available in the Status Block that permit the ladder logic program to determine the level of activity of the server driver.

An exception to normal processing is when the pass-through mode is implemented. In this mode, all write requests are passed directly to the processor and are not placed in the database. This permits direct, remote control of the processor without changes in the intermediate database. This mode is especially useful for Client devices that do not send both states of control. For example, a SCADA system may only send a SET command to a digital control point and never send a CLEAR command to that same digital point address because it expects the processor logic to reset the control bit. Pass-through must be used to simulate this mode. The following illustration shows the data flow for a server port with pass-through enabled.



Client Driver

In the Client driver, the MVI56E-MNETC/MNETCXT module issues read or write commands to servers on the Modbus TCP/IP network using up to 30 simulated Clients. The commands originate either from the module's user-configured *Client x Command List* for each Client, or directly from the processor as Event Commands. The commands from the *Client x Command List* are executed either via routine polling or as a result of special Command Control block requests from the processor. Client status data is returned to the processor in special Client Status blocks. The following flowchart describes the flow of data into and out of the module.



1. The Client driver obtains configuration data when the module restarts. This includes the timeout parameters and the Command List. These values are used by the driver to determine the types of commands to be issued to servers on the Modbus TCP/IP network.
2. When configured, the Client driver begins transmitting read and/or write commands to servers on the network. The data for write commands is obtained from the module's internal database.
3. Assuming successful processing by the server specified in the command, a response message is received into the Client driver for processing.
4. Data received from the server is passed into the module's internal database, if the command was a read command. General module status information is routinely returned to the processor in the input images.
5. Status data for a specific Client can be requested by the processor and returned in a special Client Status block.
6. Special functions, such as Event Commands and Command Control options, can be generated by the processor and sent to the Client driver for action.

Client Command List

In order for the Client to function, the module's Client Command List must be defined in the *MNET Client x Commands* section of the configuration. This list contains up to 16 individual entries, with each entry containing the information required to construct a valid command. This includes the following:

- Command enable mode: (**0**) disabled, (**1**) continuous, or (**2**) conditional. Conditional enabling applies only to write commands.
- IP address and service port to connect to on the remote server
- Slave Node Address
- Command Type - Read or Write up to 100 words per command
- Database Source and Destination Register Address - Determines where data will be placed and/or obtained
- Count - Select the number of words to be transferred - 1 to 100
- Poll Delay - $1/10^{th}$ seconds

For information on troubleshooting commands, see Client Command Errors (page 86).

5.3 Ethernet Cable Specifications

The recommended cable is Category 5 or better. A Category 5 cable has four twisted pairs of wires, which are color-coded and cannot be swapped. The module uses only two of the four pairs.

The Ethernet port or ports on the module are Auto-Sensing. You can use either a standard Ethernet straight-through cable or a crossover cable when connecting the module to an Ethernet hub, a 10/100 Base-T Ethernet switch, or directly to a PC. The module detects the cable type and uses the appropriate pins to send and receive Ethernet signals.

Some hubs have one input that can accept either a straight-through or crossover cable, depending on a switch position. In this case, you must ensure that the switch position and cable type agree.

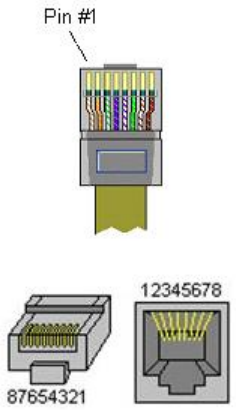
Refer to Ethernet Cable Configuration (page 121) for a diagram of how to configure Ethernet cable.

5.3.1 Ethernet Cable Configuration

Note: The standard connector view shown is color-coded for a straight-through cable.

Crossover cable

RJ-45 PIN	RJ-45 PIN
1 Rx+	3 Tx+
2 Rx-	6 Tx-
3 Tx+	1 Rx+
6 Tx-	2 Rx-



Straight- through cable

RJ-45 PIN	RJ-45 PIN
1 Rx+	1 Tx+
2 Rx-	2 Tx-
3 Tx+	3 Rx+
6 Tx-	6 Rx-

5.3.2 Ethernet Performance

Ethernet performance in the MVI56E-MNETC/MNETCXT module can be affected in the following way:

- Accessing the web interface (refreshing the page, downloading files, and so on) may affect performance
- Also, high Ethernet traffic may impact performance, so consider one of these options:
  - Use managed switches to reduce traffic coming to module port
  - Use CIPconnect for these applications and disconnect the module Ethernet port from the network

5.4 Modbus Protocol Specification

The following pages give additional reference information regarding the Modbus protocol commands supported by the MVI56E-MNETC/MNETCXT.

5.4.1 About the Modbus Protocol

Modbus is a widely-used protocol originally developed by Modicon in 1978. Since that time, the protocol has been adopted as a standard throughout the automation industry.

The original Modbus specification uses a serial connection to communicate commands and data between master and server devices on a network. Later enhancements to the protocol allow communication over Ethernet networks using TCP/IP as a "wrapper" for the Modbus protocol. This protocol is known as Modbus TCP/IP.

Modbus TCP/IP is a client/server protocol. The master establishes a connection to the remote server. When the connection is established, the master sends the Modbus TCP/IP commands to the server. The MVI56E-MNETC/MNETCXT Module module simulates up to 30 masters, and works both as a master and a server.

Aside from the benefits of Ethernet versus serial communications (including performance, distance, and flexibility) for industrial networks, the Modbus TCP/IP protocol allows for remote administration and control of devices over an Internet connection. It is important to note that not all Internet protocols are implemented in the module; for example, HTTP and SMTP protocols are not available. Nevertheless, the efficiency, scalability, and low cost of a Modbus TCP/IP network make this an ideal solution for industrial applications.

The MVI56E-MNETC/MNETCXT Module module acts as an input/output module between devices on a Modbus TCP/IP network and the Rockwell Automation backplane and processor. The module uses an internal database to pass data and commands between the processor and the master and server devices on the Modbus TCP/IP network.

5.4.2 Read Coil Status (Function Code 01)

Query

This function allows you to obtain the ON/OFF status of logic coils (Modbus 0x range) used to control discrete outputs from the addressed server only. Broadcast mode is not supported with this function code. In addition to the server address and function fields, the message requires that the information field contain the initial coil address to be read (Starting Address) and the number of locations interrogated to obtain status data.

The addressing allows up to 2000 coils to be obtained at each request; however, the specific server device may have restrictions that lower the maximum quantity. The coils are numbered from zero; (coil number 1 = zero, coil number 2 = one, coil number 3 = two, and so on).

The following table is a sample Modbus message structure to read output status request to read coils 0020 to 0056 (37 coils) from server device number 11.

Note: The byte values below are in hexadecimal display

Node Address	Function Code	Data Start Point High	Data Start Point Low	Number of Points High	Number of Points Low	Error Check Field (2 bytes)
0B	01	00	13	00	25	CRC

Response

An example response to Read Coil Status is as shown in the table below. The data is packed one bit for each coil. The response includes the server address, function code, quantity of data characters, the data characters, and error checking. Data is packed with one bit for each coil (1 = ON, 0 = OFF). The low order bit of the first character contains the addressed coil, and the remainder follows. For coil quantities that are not even multiples of eight, the last characters are filled in with zeros at high order end. The quantity of data characters is always specified as quantity of RTU characters, that is, the number is the same whether RTU or ASCII is used.

Because the server interface device is serviced at the end of a controller's scan, data reflects coil status at the end of the scan. Some servers limit the quantity of coils provided each scan; thus, for large coil quantities, multiple PC transactions must be made using coil status from sequential scans.

Node Address	Func Code	Byte Count	Data Coil Status 20 to 27	Data Coil Status 28 to 35	Data Coil Status 36 to 43	Data Coil Status 44 to 51	Data Coil Status 52 to 56	Error Check Field (2 bytes)
0B	01	05	CD	6B	B2	OE	1B	CRC

The status of coils 20 to 27 is shown as CD (HEX) = 1100 1101 (Binary). Reading from left to right, this shows that coils 27, 26, 23, 22, and 20 are all on. The other Data Coil Status bytes are decoded similarly. Due to the quantity of coil statuses requested, the last data field, which is shown 1B (HEX) = 0001 1011 (Binary), contains the status of only 5 coils (52 to 56) instead of 8 coils. The 3 left most bits are provided as zeros to fill the 8-bit format.

5.4.3 Read Input Status (Function Code 02)

Query

This function allows you to obtain the ON/OFF status of discrete inputs (Modbus 1x range) in the addressed server. PC Broadcast mode is not supported with this function code. In addition to the server address and function fields, the message requires that the information field contain the initial input address to be read (Starting Address) and the number of locations that are interrogated to obtain status data.

The addressing allows up to 2000 inputs to be obtained at each request; however, the specific server device may have restrictions that lower the maximum quantity. The inputs are numbered form zero; (input 10001 = zero, input 10002 = one, input 10003 = two, and so on, for a 584).

The following table is a sample read input status request to read inputs 10197 to 10218 (22 coils) from server number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Node Address	Function Code	Data Start Point High	Data Start Point Low	Number of Points High	Number of Points Low	Error Check Field (2 bytes)
0B	02	00	C4	00	16	CRC

Response

An example response to Read Input Status is as shown in the table below. The data is packed one bit for each input. The response includes the server address, function code, quantity of data characters, the data characters, and error checking. Data is packed with one bit for each input (1=ON, 0=OFF). The lower order bit of the first character contains the addressed input, and the remainder follows. For input quantities that are not even multiples of eight, the last characters are filled in with zeros at high order end. The quantity of data characters is always specified as a quantity of RTU characters, that is, the number is the same whether RTU or ASCII is used.

Because the server interface device is serviced at the end of a controller's scan, the data reflect input status at the end of the scan. Some servers limit the quantity of inputs provided each scan; thus, for large coil quantities, multiple PC transactions must be made using coil status for sequential scans.

Node Address	Func Code	Byte Count	Data Discrete Input 10197 to 10204	Data Discrete Input 10205 to 10212	Data Discrete Input 10213 to 10218	Error Check Field (2 bytes)
0B	02	03	AC	DB	35	CRC

The status of inputs 10197 to 10204 is shown as AC (HEX) = 10101 1100 (binary). Reading left to right, this show that inputs 10204, 10202, and 10199 are all on. The other input data bytes are decoded similar.

Due to the quantity of input statuses requested, the last data field which is shown as 35 HEX = 0011 0101 (binary) contains the status of only 6 inputs (10213 to 102180) instead of 8 inputs. The two left-most bits are provided as zeros to fill the 8-bit format.

5.4.4 Read Holding Registers (Function Code 03)

Query

This function allows you to retrieve the contents of holding registers 4xxxx (Modbus 4x range) in the addressed server. The registers can store the numerical values of associated timers and counters which can be driven to external devices. The addressing allows retrieving up to 125 registers at each request; however, the specific server device may have restrictions that lower this maximum quantity. The registers are numbered form zero (40001 = zero, 40002 = one, and so on). The broadcast mode is not allowed.

The example below reads registers 40108 through 40110 (three registers) from server number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Node Address	Function Code	Data Start Registers High	Data Start Registers Low	Data Number of Registers High	Data Number of Registers Low	Error Check Field (2 bytes)
0B	03	00	6B	00	03	CRC

Response

The addressed server responds with its address and the function code, followed by the information field. The information field contains 1 byte describing the quantity of data bytes to be returned. The contents of the registers requested (DATA) are two bytes each, with the binary content right justified within each pair of characters. The first byte includes the high order bits and the second, the low order bits.

Because the server interface device is normally serviced at the end of the controller's scan, the data reflect the register content at the end of the scan. Some servers limit the quantity of register content provided each scan; thus for large register quantities, multiple transmissions are made using register content from sequential scans.

In the example below, the registers 40108 to 40110 have the decimal contents 555, 0, and 100 respectively.

Node Address	Function Code	Byte Count	High Data	Low Data	High Data	Low Data	High Data	Low Data	Error Check Field (2 bytes)
0B	03	06	02	2B	00	00	00	64	CRC

5.4.5 Read Input Registers (Function Code 04)

Query

This function retrieves the contents of the controller's input registers from the Modbus 3x range. These locations receive their values from devices connected to the I/O structure and can only be referenced, not altered from within the controller, The addressing allows retrieving up to 125 registers at each request; however, the specific server device may have restrictions that lower this maximum quantity. The registers are numbered for zero (30001 = zero, 30002 = one, and so on). Broadcast mode is not allowed.

The example below requests the contents of register 30009 in server number 11.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Node Address	Function Code	Data Start Point High	Data Start Point Low	Data Number of Points High	Data Number of Points Low	Error Check Field (2 bytes)
0B	04	00	08	00	01	CRC

Response

The addressed server responds with its address and the function code followed by the information field. The information field contains 1 byte describing the quantity of data bytes to be returned. The contents of the registers requested (DATA) are 2 bytes each, with the binary content right justified within each pair of characters. The first byte includes the high order bits and the second, the low order bits.

Because the server interface is normally serviced at the end of the controller's scan, the data reflect the register content at the end of the scan. Each PC limits the quantity of register contents provided each scan; thus for large register quantities, multiple PC scans are required, and the data provided is from sequential scans.

In the example below the register 30009 contains the decimal value 0.

Node Address	Function Code	Byte Count	Data Input Register High	Data Input Register Low	Error Check Field (2 bytes)
0B	04	02	00	00	CRC

5.4.6 Force Single Coil (Function Code 05)

Query

This Function Code forces a single coil (Modbus 0x range) either ON or OFF. Any coil that exists within the controller can be forced to either state (ON or OFF). However, because the controller is actively scanning, unless the coil is disabled, the controller can also alter the state of the coil. Coils are numbered from zero (coil 0001 = zero, coil 0002 = one, and so on). The data value 65,280 (FF00 HEX) sets the coil ON and the value zero turns it OFF; all other values are illegal and do not affect that coil.

The use of server address 00 (Broadcast Mode) forces all attached servers to modify the desired coil.

Note: Functions 5, 6, 15, and 16 are the only messages that are recognized as valid for broadcast.

The example below is a request to server number 11 to turn ON coil 0173.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Node Address	Function Code	Data Start Bit High	Data Start Bit Low	Number of Bits High	Number of Bits Low	Error Check Field (2 bytes)
0B	05	00	AC	FF	00	CRC

Response

The normal response to the Command Request is to re-transmit the message as received after the coil state has been altered.

Node Address	Function Code	Data Coil Bit High	Data Coil Bit Low	Data On/Off	Data	Error Check Field (2 bytes)
0B	05	00	AC	FF	00	CRC

The forcing of a coil via Modbus function 5 happens regardless of whether the addressed coil is disabled or not (*In ProSoft products,* the coil *is only affected if you implement the necessary ladder logic).*

Note: The Modbus protocol does not include standard functions for testing or changing the DISABLE state of discrete inputs or outputs. Where applicable, this may be accomplished via device specific Program commands (*In ProSoft products, this is only accomplished through ladder logic programming).*

Coils that are reprogrammed in the controller logic program are not automatically cleared upon power up. Thus, if such a coil is set ON by function Code 5 and (even months later), an output is connected to that coil, the output is "hot".

5.4.7 Preset Single Register (Function Code 06)

Query

This Function Code allows you to modify the contents of a Modbus 4x range in the server. This writes to a single register only. Any holding register that exists within the controller can have its contents changed by this message. However, because the controller is actively scanning, it also can alter the content of any holding register at any time. The values are provided in binary up to the maximum capacity of the controller. Unused high order bits must be set to zero. When used with server address zero (Broadcast mode), all server controllers load the specified register with the contents specified.

Note: Functions 5, 6, 15, and 16 are the only messages that are recognized as valid for broadcast.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

The example below is a request to write the value '3' to register 40002 in server 11.

Node Address	Function Code	Data Start Bit High	Data Start Bit Low	Preset Data Register High	Preset Data Register Low	Error Check Field (2 bytes)
0B	06	00	01	00	03	CRC

Response

The response to a preset single register request is to re-transmit the query message after the register has been altered.

Node Address	Function Code	Data Register High	Data Register Low	Preset Data Register High	Preset Data Register Low	Error Check Field (2 bytes)
0B	06	00	01	00	03	CRC

5.4.8 Read Exception Status (Function Code 07)

This function code is used to read the contents of eight Exception Status outputs in a remote device. Function code 7 provides a method for accessing this information because the Exception Output references are known (no output reference is needed in the function). The normal response contains the status of the eight Exception Status outputs. The outputs are packed into one data byte, with one bit per output. The status of the lowest output reference is contained in the least significant bit of the byte.

5.4.9 Diagnostics (Function Code 08)

Modbus function code 08 provides a series of tests for checking the communication system between a Client device and a server, or for checking various internal error conditions within a server.

The function uses a two-byte sub-function code field in the query to define the type of test to be performed. The server echoes both the function code and sub-function code in a normal response. Some of the diagnostics cause data to be returned from the remote device in the data field of a normal response.

In general, issuing a diagnostic function to a remote device does not affect the running of the user program in the remote device. Device memory bit and register data addresses are not accessed by the diagnostics. However, certain functions can optionally reset error counters in some remote devices.

A server device can, however, be forced into 'Listen Only Mode' in which it will monitor the messages on the communications system but not respond to them. This can affect the outcome of your application program if it depends upon any further exchange of data with the remote device. Generally, the mode is forced to remove a malfunctioning remote device from the communications system.

Sub-function Codes Supported

Only Sub-function 00 is supported by the MVI56E-MNETC/MNETCXT module.

00 Return Query Data

The data passed in the request data field is to be returned (looped back) in the response. The entire response message should be identical to the request.

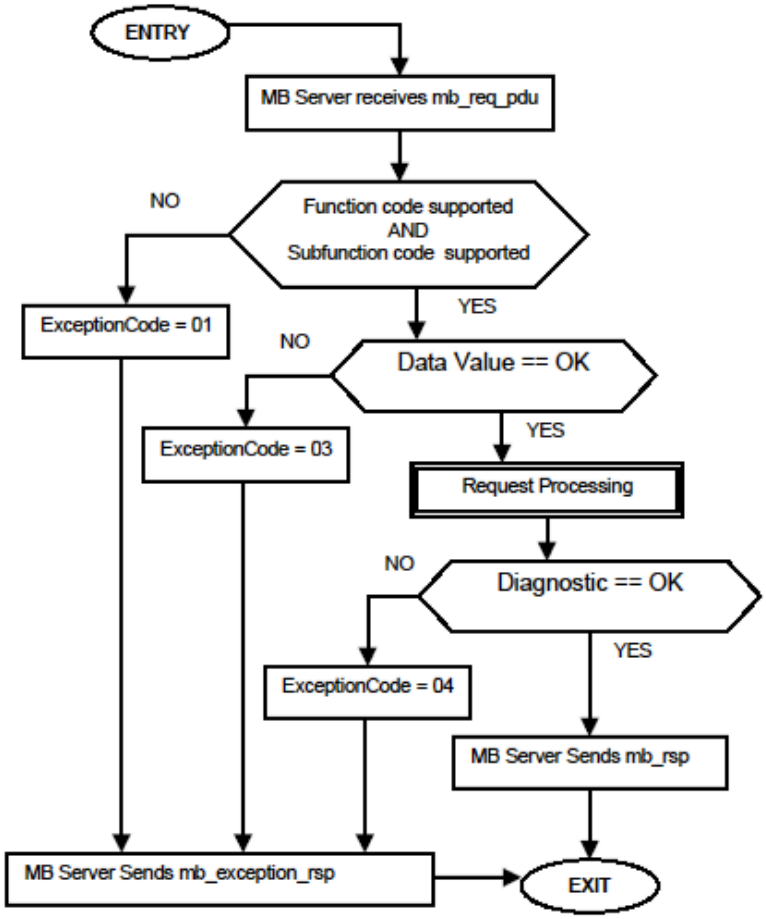
Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

Example and State Diagram

Here is an example of a request to remote device to Return Query Data. This uses a sub-function code of zero (00 00 hex in the two-byte field). The data to be returned is sent in the two-byte data field (A5 37 hex).

Request		Response	
Field Name	**(Hex)**	**Field Name**	**(Hex)**
Function	08	Function	08
Sub-function Hi	00	Sub-function Hi	00
Sub-function Lo	00	Sub-function Lo	00
Data Hi	A5	Data Hi	A5
Data Lo	37	Data Lo	27

The data fields in responses to other kinds of queries could contain error counts or other data requested by the sub-function code.



5.4.10 Force Multiple Coils (Function Code 15)

Query

This function forces each coil (Modbus 0x range) in a consecutive block of coils to a desired ON or OFF state. Any coil that exists within the controller can be forced to either state (ON or OFF). However, because the controller is actively scanning, unless the coils are disabled, the controller can also alter the state of the coil. Coils are numbered from zero (coil 00001 = zero, coil 00002 = one, and so on). The desired status of each coil is packed in the data field, one bit for each coil (1= ON, 0= OFF). The use of server address 0 (Broadcast Mode) forces all attached servers to modify the desired coils.

Note: Functions 5, 6, 15, and 16 are the only messages (other than Loopback Diagnostic Test) that are recognized as valid for broadcast.

The following example forces 10 coils starting at address 20 (13 HEX). The two data fields, CD =1100 and 00 = 0000 000, indicate that coils 27, 26, 23, 22, and 20 are to be forced on.

Note: This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Node Address	Func Code	Coil Address High	Coil Address Low	Number of Coils High	Number of Coils Low	Byte Count	Force Data High 20 to 27	Force Data Low 28 to 29	Error Check Field (2 bytes)
0B	0F	00	13	00	0A	02	CD	01	CRC

Response

The normal response is an echo of the server address, function code, starting address, and quantity of coils forced.

Node Address	Func Code	Coil Address High	Coil Address Low	Number of Coils High	Number of Coils Low	Error Check Field (2 bytes)
0B	0F	00	13	00	0A	CRC

Writing to coils with Modbus function 15 is accomplished regardless of whether the addressed coils are disabled or not.

Coils that are not programmed in the controller logic program are not automatically cleared upon power up. Thus, if such a coil is set ON by function code 15 and (even months later) an output is connected to that coil, the output is hot.

5.4.11 Preset Multiple Registers (Function Code 16)

Query

Holding registers existing within the controller can have their contents changed by this message (a maximum of 60 registers). However, because the controller is actively scanning, it also can alter the content of any holding register at any time. The values are provided in binary up to the maximum capacity of the controller (16-bit for the 184/384 and 584); unused high order bits must be set to zero.

Note: Function codes 5, 6, 15, and 16 are the only messages that will be recognized as valid for broadcast.

This is the structure of the message being sent out to the Modbus network. The byte values below are in hexadecimal display.

Adr	Func	Hi Add	Lo Add	Quantity		Byte Cnt	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
0B	10	00	87	00	02	04	00	0A	01	02	CRC

Response

The normal response to a function 16 query is to echo the address, function code, starting address and number of registers to be loaded.

Adr	Func	Hi Addr	Lo Addr	Quantity		Error Check Field
0B	10	00	87	00	02	56

5.4.12 Modbus Exception Responses

When a Modbus master sends a request to a server device, it expects a normal response. One of four possible events can occur from the master's query:

- If the server device receives the request without a communication error, and can handle the query normally, it returns a normal response.
- If the server does not receive the request due to a communication error, no response is returned. The master program will eventually process a timeout condition for the request.
- If the server receives the request, but detects a communication error (parity, LRC, CRC, ...), no response is returned. The master program will eventually process a timeout condition for the request.
- If the server receives the request without a communication error, but cannot handle it (for example, if the request is to read a non-existent output or register), the server will return an exception response informing the master of the nature of the error.

The exception response message has two fields that differentiate it from a normal response:

Function Code Field: In a normal response, the server echoes the function code of the original request in the function code field of the response. All function codes have a most-significant bit (MSB) of 0 (their values are all below 80 hexadecimal). In an exception response, the server sets the MSB of the function code to 1. This makes the function code value in an exception response exactly 80 hexadecimal higher than the value would be for a normal response.

With the function code's MSB set, the master's application program can recognize the exception response and can examine the data field for the exception code.

Data Field: In a normal response, the server may return data or statistics in the data field (any information that was requested in the request). In an exception response, the server returns an exception code in the data field. This defines the server condition that caused the exception.

The following table shows an example of a master request and server exception response.

Request		**Response**	
Field Name	**(Hex)**	**Field Name**	**(Hex)**
Function	01	Function	81
Starting Address Hi	04	Exception Code	02
Starting Address Lo	A1		
Quantity of Outputs Hi	00		
Quantity of Outputs Lo	01		

In this example, the master addresses a request to server device. The function code (01) is for a Read Output Status operation. It requests the status of the output at address 1245 (04A1 hex). Note that only that one output is to be read, as specified by the number of outputs field (0001).

If the output address is non-existent in the server device, the server will return the exception response with the exception code shown (02). This specifies an illegal data address for the server.

Modbus Exception Codes

Code	Name	Meaning
01	Illegal Function	The function code received in the query is not an allowable action for the server. This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the server is in the wrong state to process a request of this type, for example because it is not configured and is being asked to return register values.
02	Illegal Data Address	The data address received in the query is not an allowable address for the server. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed; a request with offset 96 and length 5 will generate exception 02.
03	Illegal Data Value	A value contained in the query data field is not an allowable value for server. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expectation of the application program, because the Modbus protocol is unaware of the significance of any particular value of any particular register.
04	Slave Device Failure	An unrecoverable error occurred while the server was attempting to perform the requested action.
05	Acknowledge	Specialized use in conjunction with programming commands. The server has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the master. The master can next issue a poll program complete message to determine if processing is completed.
06	Slave Device Busy	Specialized use in conjunction with programming commands. The server is engaged in processing a long-duration program command. The master should retransmit the message later when the server is free.
08	Memory Parity Error	Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server attempted to read record file, but detected a parity error in the memory. The master can retry the request, but service may be required on the server device.
0A	Gateway Path Unavailable	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
0B	Gateway Target Device Failed To Respond	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

5.5 Using the Optional Add-On Instruction

5.5.1 Before You Begin

Make sure that you have installed RSLogix 5000 version 16 (or later).

Download the files from www.prosoft-technology.com. Save them to a convenient location in your PC, such as *Desktop* or *My Documents*.

File Name	Description
MVI56EMNETC_AddOn_Rung_v1_x.L5X A newer version may be available at: www.prosoft-technology.com	L5X file containing Add-On Instruction, user defined data types, controller tags and ladder logic required to configure the MVI56E-MNETC/MNETCXT module
MVI56EMNETC_Optional_AddOn_Rung_v1_x.L5X A newer version may be available at: www.prosoft-technology.com	Optional L5X file containing additional Add-On Instruction with logic for changing Ethernet configuration and clock settings.

5.5.2 Overview

The Optional Add-On Instruction Rung Import contains optional logic for MVI56E-MNETC/MNETCXT applications to perform the following tasks.

- Read/Write Ethernet Configuration

  Allows the processor to read or write the module IP address, netmask and gateway values.

Note: This is an optional feature. You can perform the same task through ProSoft Configuration Builder. Even if your PC is in a different network group you can still access the module through PCB by setting a temporary IP address.

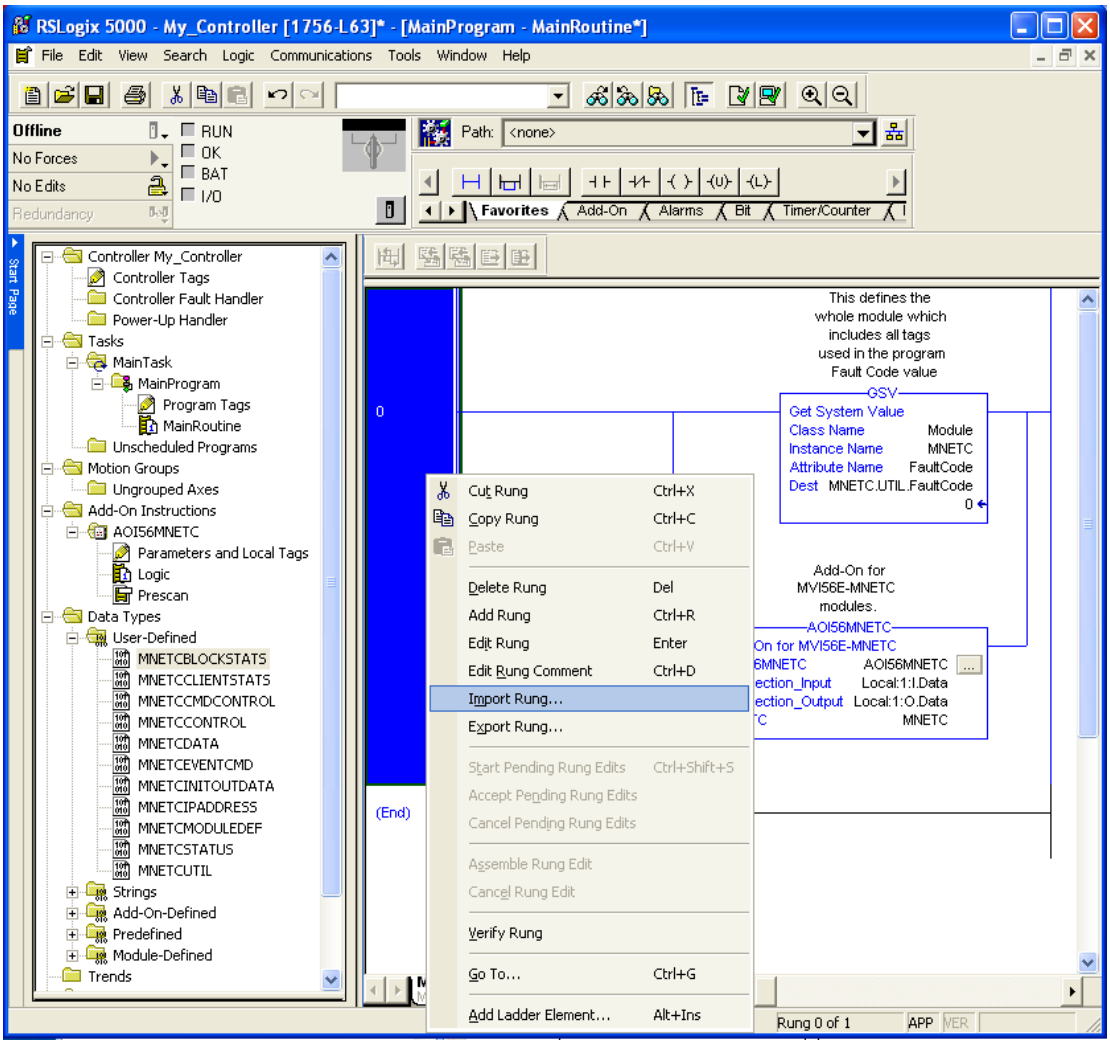
- Read/Write Module Clock Value

  Allows the processor to read and write the module clock settings. The module clock stores the last time that the Ethernet configuration was changed. The date and time of the last Ethernet configuration change is displayed in the scrolling LED during module power up.

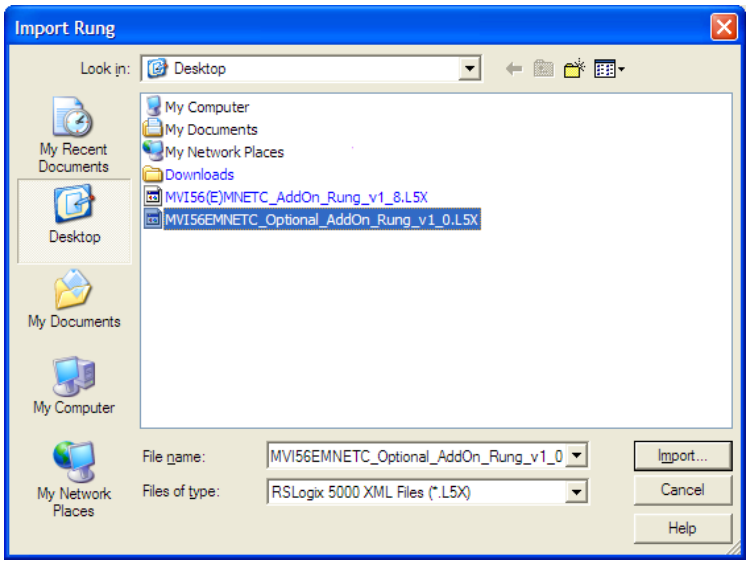
Important: The Optional Add-On Instruction only supports the two features listed above. You must use the sample ladder logic for all other features including backplane transfer of Modbus TCP/IP data.

5.5.3 Importing the Optional Add-On Instruction

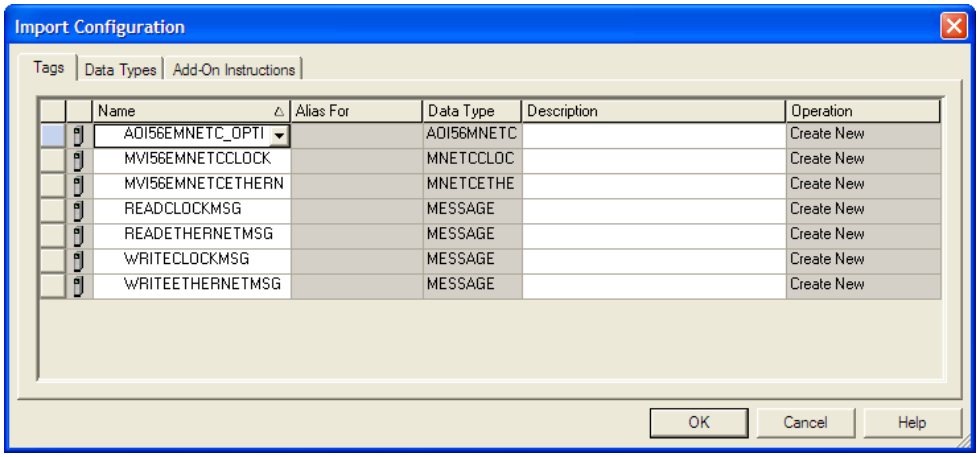
1 Right-click an empty rung in the main routine of your existing ladder logic and choose **IMPORT RUNG**.



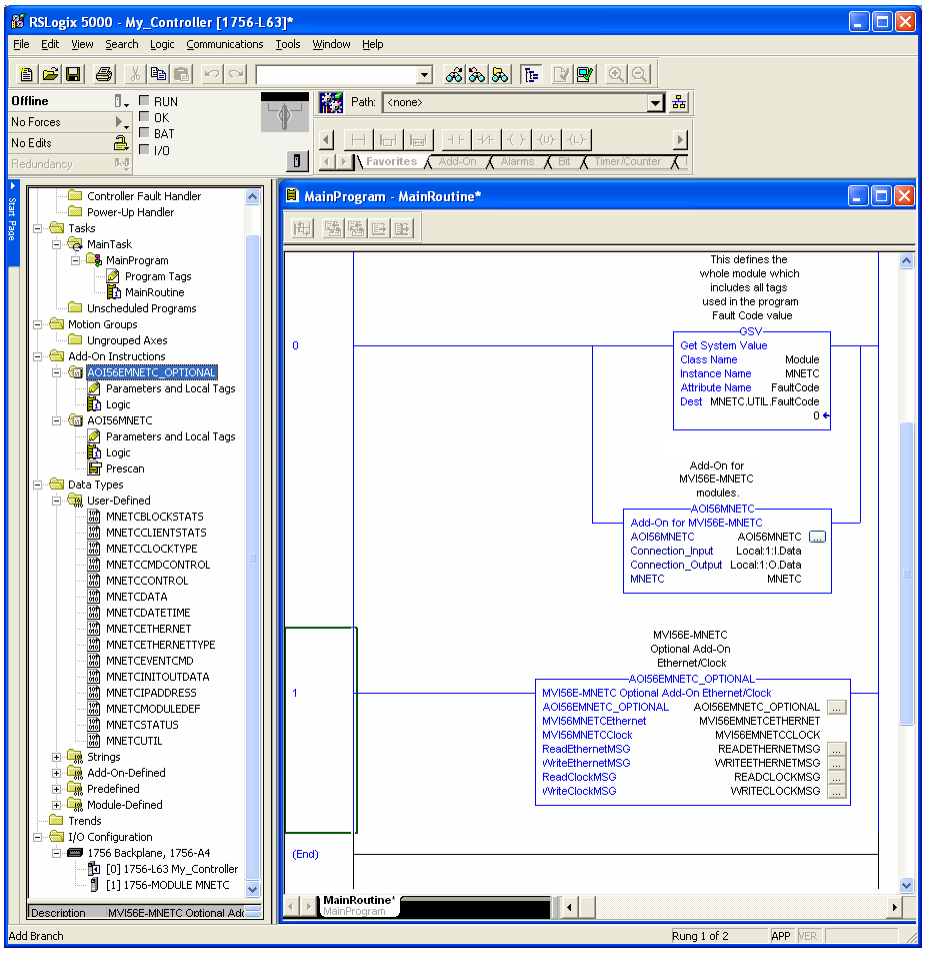
2 Navigate to the folder where you saved MVI56EMNETC_Optional_AddOn_Rung_v1_0.L5X and select the file.



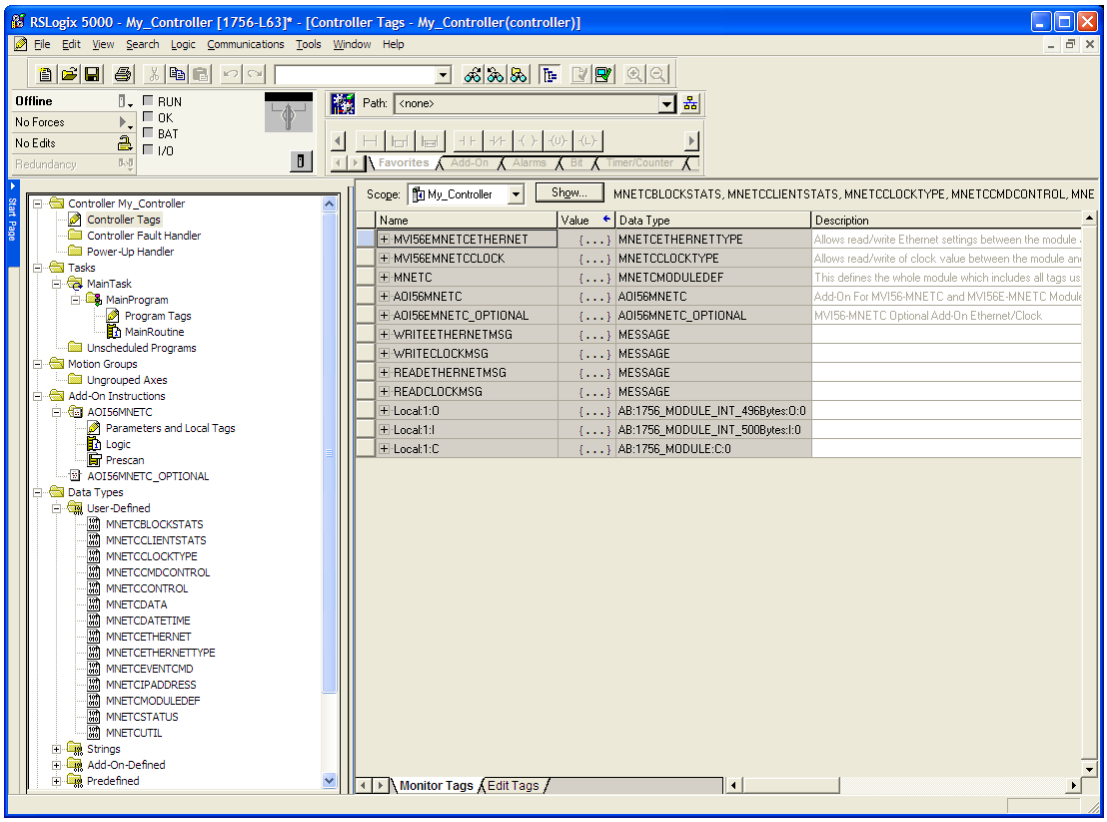
In the *Import Configuration* window, click **OK.**



The Add-On Instruction is now visible in the ladder logic. Observe that the procedure has also imported data types and controller tags associated with the Add-On Instruction.

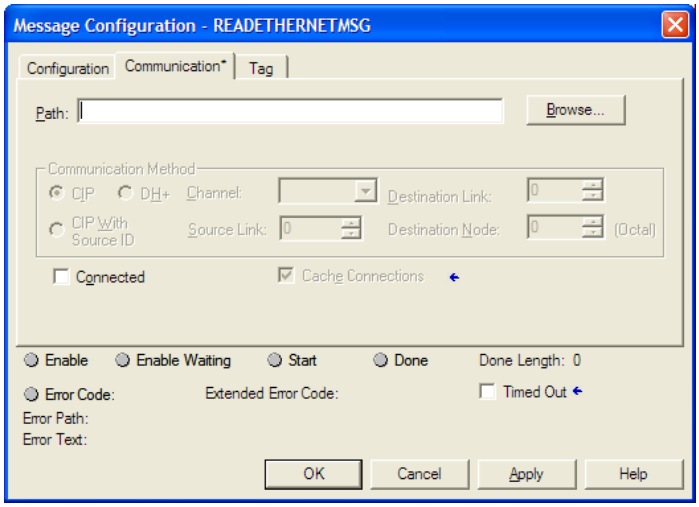


You will notice that new tags have been imported: *MVI56MNETCEthernet*, *MVI56MNETCClock*, and four *MESSAGE* tags.

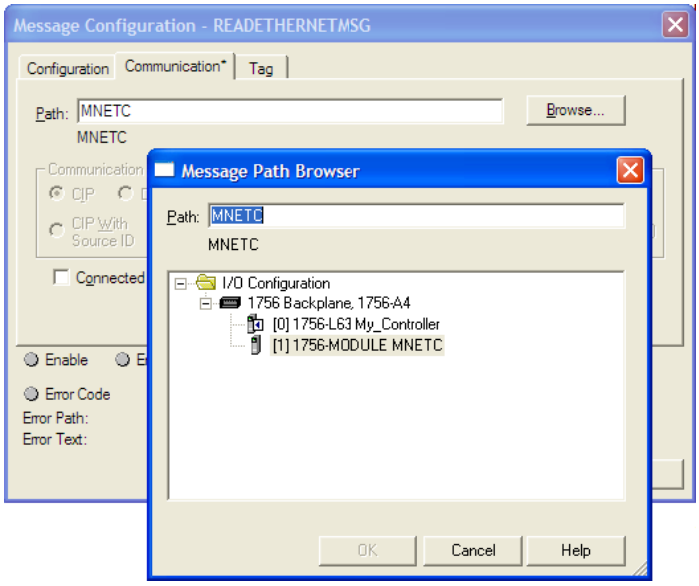


3 In the Add-On Instruction, click the **[...]** button next to any *MSG* tag to open the *Message Configuration* dialog box.

4 Click the **COMMUNICATION** tab and then click the **BROWSE** button.



5 Select the module to configure the message path.



5.5.4 Reading the Ethernet Settings from the Module

1 Expand the *MVI56EMNETCETHERNET* controller tag and move a value of **1** to *MVI56EMNETCETHERNET.Read*.

MVI56EMNETCETHERNET	{...}
MVI56EMNETCETHERNET.Read	1
MVI56EMNETCETHERNET.Write	0
MVI56EMNETCETHERNET.Config	{...}
MVI56EMNETCETHERNET.Config.IP	{...}
MVI56EMNETCETHERNET.Config.IP[0]	0
MVI56EMNETCETHERNET.Config.IP[1]	0
MVI56EMNETCETHERNET.Config.IP[2]	0
MVI56EMNETCETHERNET.Config.IP[3]	0
MVI56EMNETCETHERNET.Config.Netmask	{...}
MVI56EMNETCETHERNET.Config.Netmask[0]	0
MVI56EMNETCETHERNET.Config.Netmask[1]	0
MVI56EMNETCETHERNET.Config.Netmask[2]	0
MVI56EMNETCETHERNET.Config.Netmask[3]	0
MVI56EMNETCETHERNET.Config.Gateway	{...}
MVI56EMNETCETHERNET.Config.Gateway[0]	0
MVI56EMNETCETHERNET.Config.Gateway[1]	0
MVI56EMNETCETHERNET.Config.Gateway[2]	0
MVI56EMNETCETHERNET.Config.Gateway[3]	0

2 The bit will be automatically reset and the current Ethernet settings will be copied to *MVI56EMNETCETHERNET* controller tag as follows.

MVI56EMNETCETHERNET	{...}
MVI56EMNETCETHERNET.Read	0
MVI56EMNETCETHERNET.Write	0
MVI56EMNETCETHERNET.Config	{...}
MVI56EMNETCETHERNET.Config.IP	{...}
MVI56EMNETCETHERNET.Config.IP[0]	105
MVI56EMNETCETHERNET.Config.IP[1]	102
MVI56EMNETCETHERNET.Config.IP[2]	0
MVI56EMNETCETHERNET.Config.IP[3]	12
MVI56EMNETCETHERNET.Config.Netmask	{...}
MVI56EMNETCETHERNET.Config.Netmask[0]	255
MVI56EMNETCETHERNET.Config.Netmask[1]	255
MVI56EMNETCETHERNET.Config.Netmask[2]	255
MVI56EMNETCETHERNET.Config.Netmask[3]	0
MVI56EMNETCETHERNET.Config.Gateway	{...}
MVI56EMNETCETHERNET.Config.Gateway[0]	192
MVI56EMNETCETHERNET.Config.Gateway[1]	168
MVI56EMNETCETHERNET.Config.Gateway[2]	0
MVI56EMNETCETHERNET.Config.Gateway[3]	1

3 To check the status of the message, refer to the *ReadEthernetMSG* tag.

ReadEthernetMSG	{...}
ReadEthernetMSG.Flags	16#0200
ReadEthernetMSG.EW	0
ReadEthernetMSG.ER	0
ReadEthernetMSG.DN	0
ReadEthernetMSG.ST	0
ReadEthernetMSG.EN	0
ReadEthernetMSG.TO	0
ReadEthernetMSG.EN_CC	1
ReadEthernetMSG.ERR	16#0000
ReadEthernetMSG.EXERR	16#0000_0000
ReadEthernetMSG.ERR_SRC	0
ReadEthernetMSG.DN_LEN	0
ReadEthernetMSG.REQ_LEN	0

5.5.5 Writing the Ethernet Settings to the Module

1. Expand the *MVI56EMNETCETHERNET* controller tag.
2. Set the new Ethernet configuration in *MVI56EMNETCETHERNET.Config*:
3. Move a value of **1** to *MVI56EMNETCETHERNET.Write*.

MVI56EMNETCETHERNET	{...}
MVI56EMNETCETHERNET.Read	0
MVI56EMNETCETHERNET.Write	1
MVI56EMNETCETHERNET.Config	{...}
MVI56EMNETCETHERNET.Config.IP	{...}
MVI56EMNETCETHERNET.Config.IP[0]	105
MVI56EMNETCETHERNET.Config.IP[1]	102
MVI56EMNETCETHERNET.Config.IP[2]	0
MVI56EMNETCETHERNET.Config.IP[3]	12
MVI56EMNETCETHERNET.Config.Netmask	{...}
MVI56EMNETCETHERNET.Config.Netmask[0]	255
MVI56EMNETCETHERNET.Config.Netmask[1]	255
MVI56EMNETCETHERNET.Config.Netmask[2]	255
MVI56EMNETCETHERNET.Config.Netmask[3]	0
MVI56EMNETCETHERNET.Config.Gateway	{...}
MVI56EMNETCETHERNET.Config.Gateway[0]	192
MVI56EMNETCETHERNET.Config.Gateway[1]	168
MVI56EMNETCETHERNET.Config.Gateway[2]	0
MVI56EMNETCETHERNET.Config.Gateway[3]	1

4. After the message is executed, the *MVI56EMNETCETHERNET.Write* bit resets to **0**.

MVI56EMNETCETHERNET	{...}
MVI56EMNETCETHERNET.Read	0
MVI56EMNETCETHERNET.Write	0
MVI56EMNETCETHERNET.Config	{...}
MVI56EMNETCETHERNET.Config.IP	{...}
MVI56EMNETCETHERNET.Config.IP[0]	105
MVI56EMNETCETHERNET.Config.IP[1]	102
MVI56EMNETCETHERNET.Config.IP[2]	0
MVI56EMNETCETHERNET.Config.IP[3]	12
MVI56EMNETCETHERNET.Config.Netmask	{...}
MVI56EMNETCETHERNET.Config.Netmask[0]	255
MVI56EMNETCETHERNET.Config.Netmask[1]	255
MVI56EMNETCETHERNET.Config.Netmask[2]	255
MVI56EMNETCETHERNET.Config.Netmask[3]	0
MVI56EMNETCETHERNET.Config.Gateway	{...}
MVI56EMNETCETHERNET.Config.Gateway[0]	192
MVI56EMNETCETHERNET.Config.Gateway[1]	168
MVI56EMNETCETHERNET.Config.Gateway[2]	0
MVI56EMNETCETHERNET.Config.Gateway[3]	1

5. To check the status of the message, refer to the *WriteEthernetMSG* tag.

WriteEthernetMSG	{...}
WriteEthernetMSG.Flags	16#0200
WriteEthernetMSG.EW	0
WriteEthernetMSG.ER	0
WriteEthernetMSG.DN	0
WriteEthernetMSG.ST	0
WriteEthernetMSG.EN	0
WriteEthernetMSG.TO	0
WriteEthernetMSG.EN_CC	1
WriteEthernetMSG.ERR	16#0000
WriteEthernetMSG.EXERR	16#0000_0000
WriteEthernetMSG.ERR_SRC	0
WriteEthernetMSG.DN_LEN	0
WriteEthernetMSG.REQ_LEN	24

5.5.6 Reading the Clock Value from the Module

1 Expand the *MVI56EMNETCCLOCK* controller tag and move a value of **1** to *MVI56EMNETCCLOCK.Read*

MVI56EMNETCCLOCK	{...}
MVI56EMNETCCLOCK.Read	1
MVI56EMNETCCLOCK.Write	0
MVI56EMNETCCLOCK.Config	{...}
MVI56EMNETCCLOCK.Config.Year	0
MVI56EMNETCCLOCK.Config.Month	0
MVI56EMNETCCLOCK.Config.Day	0
MVI56EMNETCCLOCK.Config.Hour	0
MVI56EMNETCCLOCK.Config.Minute	0
MVI56EMNETCCLOCK.Config.Seconds	0

2 The bit will be automatically reset and the current clock value will be copied to *MVI56EMNETCCLOCK.Config* controller tag as follows.

MVI56EMNETCCLOCK	{...}
MVI56EMNETCCLOCK.Read	0
MVI56EMNETCCLOCK.Write	0
MVI56EMNETCCLOCK.Config	{...}
MVI56EMNETCCLOCK.Config.Year	2009
MVI56EMNETCCLOCK.Config.Month	11
MVI56EMNETCCLOCK.Config.Day	18
MVI56EMNETCCLOCK.Config.Hour	10
MVI56EMNETCCLOCK.Config.Minute	21
MVI56EMNETCCLOCK.Config.Seconds	45

3 To check the status of the message, refer to the *ReadClockMSG* tag.

ReadClockMSG	{...}
ReadClockMSG.Flags	16#0200
ReadClockMSG.EW	0
ReadClockMSG.ER	0
ReadClockMSG.DN	0
ReadClockMSG.ST	0
ReadClockMSG.EN	0
ReadClockMSG.TO	0
ReadClockMSG.EN_CC	1
ReadClockMSG.ERR	16#0000
ReadClockMSG.EXERR	16#0000_0000
ReadClockMSG.ERR_SRC	0
ReadClockMSG.DN_LEN	0
ReadClockMSG.REQ_LEN	0

5.5.7 Writing the Clock Value to the Module

1. Expand the *MVI56EMNETCCLOCK* controller tag.
2. Set the new Clock value in *MVI56EMNETCCLOCK.Config*:
3. Move a value of **1** to *MVI56EMNETCCLOCK.Write*.

MVI56EMNETCCLOCK	{...}
MVI56EMNETCCLOCK.Read	0
MVI56EMNETCCLOCK.Write	1
MVI56EMNETCCLOCK.Config	{...}
MVI56EMNETCCLOCK.Config.Year	2009
MVI56EMNETCCLOCK.Config.Month	11
MVI56EMNETCCLOCK.Config.Day	18
MVI56EMNETCCLOCK.Config.Hour	10
MVI56EMNETCCLOCK.Config.Minute	21
MVI56EMNETCCLOCK.Config.Seconds	45

4. The bit will be automatically reset to **0**.

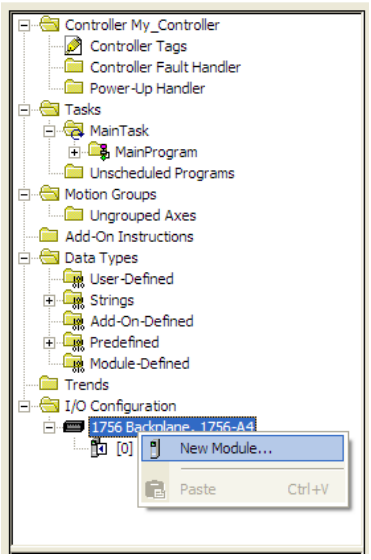
MVI56EMNETCCLOCK	{...}
MVI56EMNETCCLOCK.Read	0
MVI56EMNETCCLOCK.Write	0
MVI56EMNETCCLOCK.Config	{...}
MVI56EMNETCCLOCK.Config.Year	2009
MVI56EMNETCCLOCK.Config.Month	11
MVI56EMNETCCLOCK.Config.Day	18
MVI56EMNETCCLOCK.Config.Hour	10
MVI56EMNETCCLOCK.Config.Minute	21
MVI56EMNETCCLOCK.Config.Seconds	45

5. To check the status of the message, refer to the *WriteClockMSG* tag.

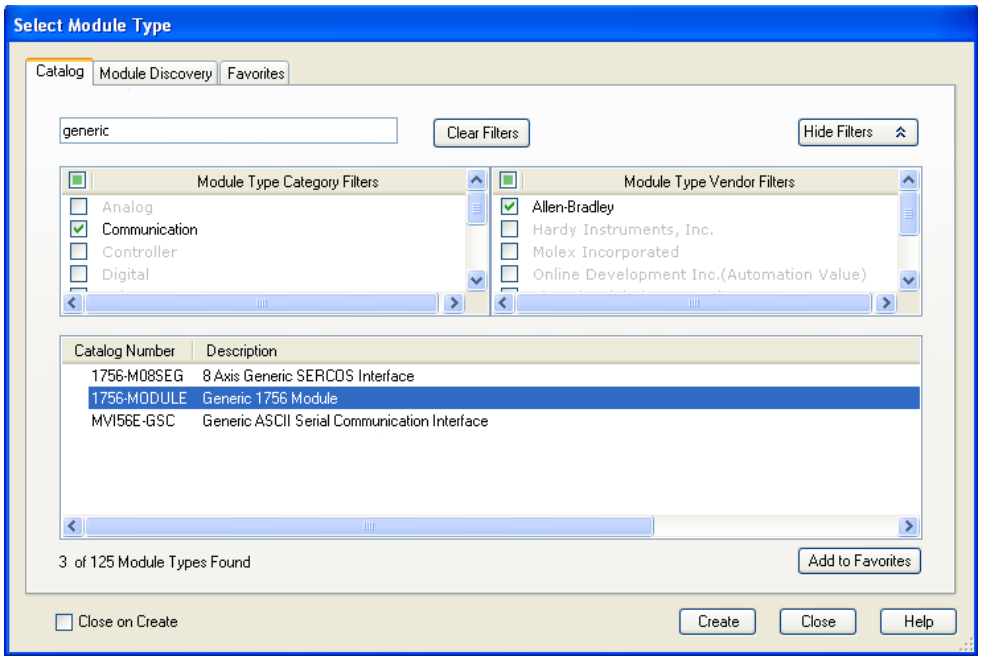
WriteClockMSG	{...}
WriteClockMSG.Flags	16#0200
WriteClockMSG.EW	0
WriteClockMSG.ER	0
WriteClockMSG.DN	0
WriteClockMSG.ST	0
WriteClockMSG.EN	0
WriteClockMSG.TO	0
WriteClockMSG.EN_CC	1
WriteClockMSG.ERR	16#0000
WriteClockMSG.EXERR	16#0000_0000
WriteClockMSG.ERR_SRC	0
WriteClockMSG.DN_LEN	0
WriteClockMSG.REQ_LEN	24

5.6 Adding the Module to an Existing Project

1 Select the *I/O Configuration* folder in the *Controller Organization* window of RSLogix 5000, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE**.



This action opens the *Select Module* dialog box .Enter **GENERIC** in the text box and select the **GENERIC 1756 MODULE**. If you're using an earlier version of RSLogix, expand **OTHER** in the Select Module dialog box, and then select the **GENERIC 1756 MODULE**.



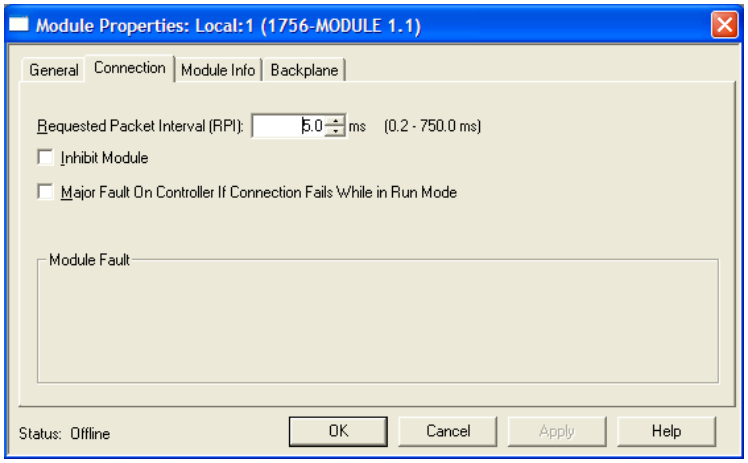
2 Select the **1756-MODULE (GENERIC 1756 MODULE)** from the list and click **OK**. This action opens the *New Module* dialog box.

3 Fill in the module properties as follows. You must select the *Comm Format* as **DATA - INT** in the dialog box, otherwise the module will not communicate.

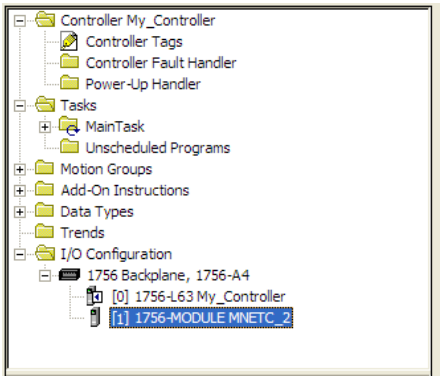
Parameter	Value
Name	Enter a module identification string. Example: **MNETC_2**
Description	Enter a description for the module. Example: **MODBUS TCP/IP CLIENT ENHANCED COMMUNICATION MODULE - CLIENT/SERVER**
Comm Format	Select **DATA-INT**.
Slot	Enter the slot number in the rack where the MVI56E-MNETC/MNETCXT module is located.
Input Assembly Instance	**1**
Input Size	**250**
Output Assembly Instance	**2**
Output Size	**248**
Configuration Assembly Instance	**4**
Configuration Size	**0**

4 Click **OK** to continue.

5 Select the *Requested Packet Interval* value for scanning the I/O on the module. This value represents the minimum frequency that the module will handle scheduled events. This value should not be set to less than **1** millisecond. The default value is **5** milliseconds. Values between **1** and **10** milliseconds should work with most applications.



6 Save the module. Click **OK** to dismiss the dialog box. The *Controller Organization* window now shows the module.



7 Copy the *User-Defined Data Types* from the sample program into your existing RSLogix 5000 project.
8 Copy the *Controller Tags* from the sample program into your project.
9 Copy the *Ladder Rungs* from the sample program into your project.

5.7 Using the Sample Program

If your processor uses RSLogix 5000 version 15 or earlier, you will not be able to use the Add-On Instruction for your module. Follow the steps below to obtain and use a sample program for your application.

5.7.1 Opening the Sample Program in RSLogix

The sample program for your MVI56E-MNETC/MNETCXT module includes custom tags, data types and ladder logic for data I/O, status and command control. For most applications, you can run the sample program without modification, or, for advanced applications, you can incorporate the sample program into your existing application.

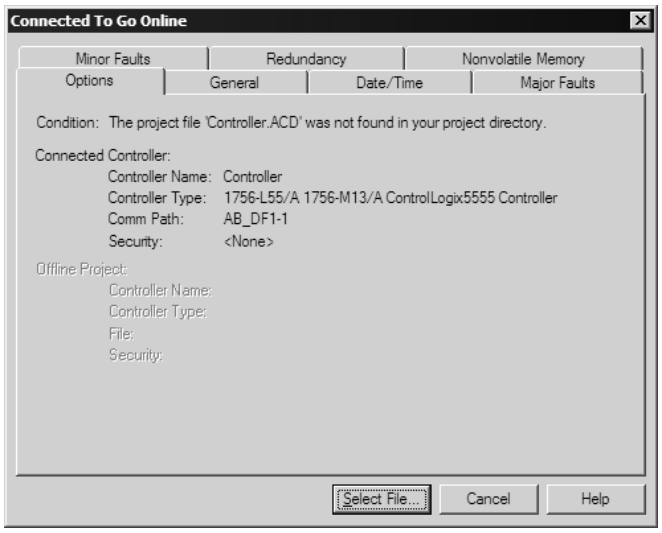
You can always download the latest version of the sample ladder logic and user manuals for the MVI56E-MNETC/MNETCXT module from the ProSoft Technology website, at www.prosoft-technology.com.

From that link, navigate to the download page for your module and choose the sample program to download for your version of RSLogix 5000 and your processor.

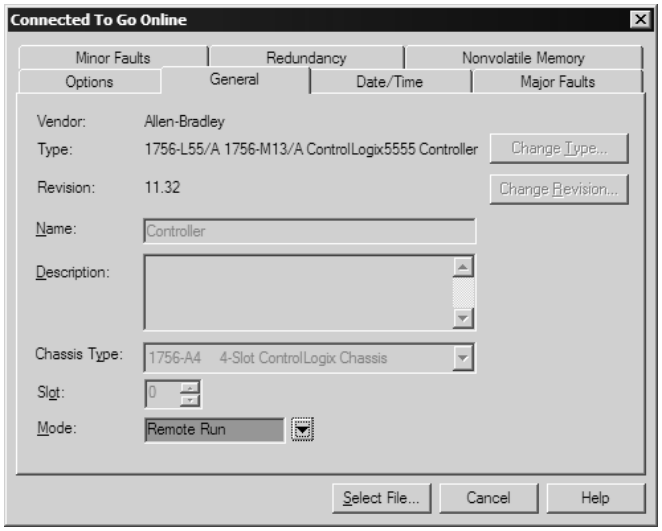
To determine the firmware version of your processor

Important: The RSLinx service must be installed and running on your computer for RSLogix to communicate with the processor. Refer to your RSLinx and RSLogix documentation for help configuring and troubleshooting these applications.

1. Start RSLogix 5000 and close any existing project that may be loaded.
2. Open the **COMMUNICATIONS** menu and choose **GO ONLINE**. RSLogix will establish communication with the processor. This may take a few moments.
3. When RSLogix has established communication with the processor, the *Connected To Go Online* dialog box will open.



4 In the *Connected To Go Online* dialog box, click the **GENERAL** tab. This tab shows information about the processor, including the Revision (firmware) version. In the following illustration, the firmware version is 11.32

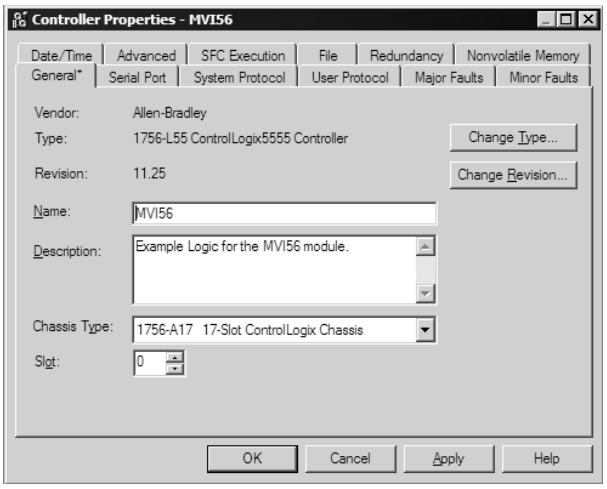


5 Select the sample ladder logic file for your firmware version.

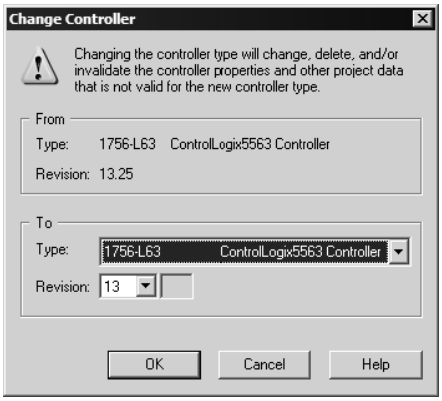
5.7.2 Choosing the Controller Type

The sample application is for a 1756-L63 ControlLogix 5563 Controller. If you are using a different model of the ControlLogix processor, you must configure the sample program to use the correct processor model.

1 In the *Controller Organizer* list, right-click the folder for the controller and then choose **PROPERTIES.** This action opens the *Controller Properties* dialog box.



2 Click the **CHANGE TYPE** or **CHANGE CONTROLLER** button. This action opens the *Change Controller* dialog box.



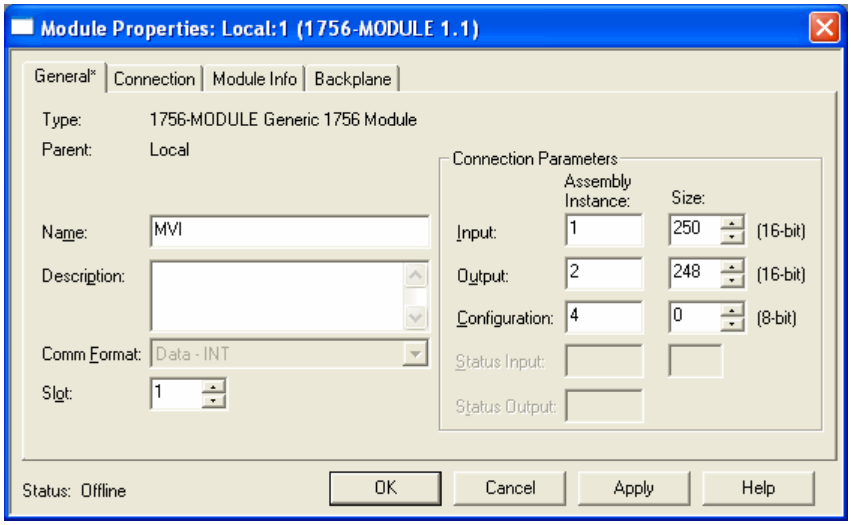
3 Open the **TYPE** dropdown list, and then select your ControlLogix controller.
4 Select the correct firmware revision for your controller, if necessary.
5 Click **OK** to save your changes and return to the previous window.

5.7.3 Selecting the Slot Number for the Module

The sample application is for a module installed in Slot 1 in a ControlLogix rack. The ladder logic uses the slot number to identify the module. If you are installing the module in a different slot, you must update the ladder logic so that program tags and variables are correct, and do not conflict with other modules in the rack.

To change the slot number

1 In the *Controller Organizer* list, right-click the module and then choose **PROPERTIES.** This action opens the *Module Properties* dialog box.



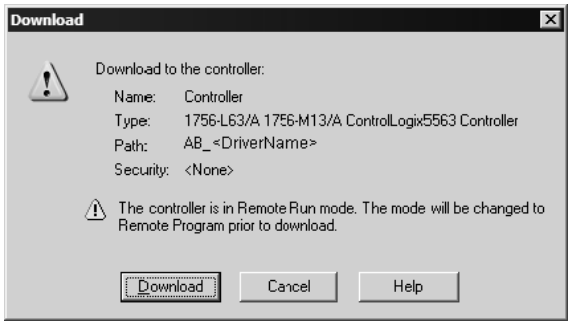
2 In the **SLOT** field, use the up and down arrows on the right side of the field to select the slot number where the module will reside in the rack, and then click **OK.**

RSLogix will automatically apply the slot number change to all tags, variables and ladder logic rungs that use the MVI56E-MNETC/MNETCXT slot number for computation.

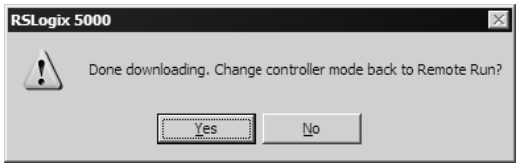
5.7.4 Downloading the Sample Program to the Processor

Note: The key switch on the front of the ControlLogix module must be in the REM position.

1. If you are not already online to the processor, open the **COMMUNICATIONS** menu, and then choose **DOWNLOAD**. RSLogix will establish communication with the processor.
2. When communication is established, RSLogix will open a confirmation dialog box. Click the **DOWNLOAD** button to transfer the sample program to the processor.



3. RSLogix will compile the program and transfer it to the processor. This process may take a few minutes.
4. When the download is complete, RSLogix will open another confirmation dialog box. Click **OK** to switch the processor from PROGRAM mode to RUN mode.



Note: If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

5.7.5 Adding the Sample Ladder to an Existing Application

1. Copy the Controller Tags from the sample program.
2. Copy the User-Defined Data Types from the sample program.
3. Copy the Ladder Rungs from the sample program.
4. Save and Download (page 23, page 152) the new application to the controller and place the processor in RUN mode.

6 Support, Service & Warranty

6.1 Contacting Technical Support

ProSoft Technology, Inc. is committed to providing the most efficient and effective support possible. Before calling, please gather the following information to assist in expediting this process:

1. Product Version Number
2. System architecture
3. Network details

If the issue is hardware related, we will also need information regarding:

1. Module configuration and associated ladder files, if any
2. Module operation and any unusual behavior
3. Configuration/Debug status information
4. LED patterns
5. Details about the interfaced serial, Ethernet or Fieldbus devices

North America (Corporate Location)	Europe / Middle East / Africa Regional Office
Phone: +1 661-716-5100 ps.prosofttechnology@belden.com Languages spoken: English, Spanish REGIONAL TECH SUPPORT ps.support@belden.com	Phone: +33.(0)5.34.36.87.20 ps.europe@belden.com Languages spoken: English, French, Hindi, Italian REGIONAL TECH SUPPORT ps.support.emea@belden.com
Latin America Regional Office	**Asia Pacific Regional Office**
Phone: +52.222.264.1814 ps.latinam@belden.com Languages spoken: English, Spanish, Portuguese REGIONAL TECH SUPPORT ps.support.la@belden.com	Phone: +60.3.2247.1898 ps.asiapc@belden.com Languages spoken: Bahasa, Chinese, English, Hindi, Japanese, Korean, Malay REGIONAL TECH SUPPORT ps.support.ap@belden.com

For additional ProSoft Technology contacts in your area, please see:
www.prosoft-technology.com/About-Us/Contact-Us

6.2 Warranty Information

For details regarding ProSoft Technology's legal terms and conditions, please see:
www.prosoft-technology.com/ProSoft-Technology-Legal-Terms-and-Conditions

For Return Material Authorization information, please see:
www.prosoft-technology.com/Services-Support/Return-Material-Instructions