ProSoft TECHNOLOGY

Where Automation Connects.



MVI56E-MNETR

Modbus® TCP/IP Interface Module
with Reduced Data Block

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 contact 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-MNETR User Manual
For Public Use.

October 13, 2025

ProSoft Technology ®, ProLinx ®, inRAx ®, ProTalk ®, and RadioLinx ® are Registered Trademarks 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

1 Start Here 7

1.1 What's New? 7
1.2 System Requirements 8
1.3 Package Contents 9
1.4 Setting Jumpers 9
1.5 Installing the Module in the Rack 10
1.6 Importing the Sample Add-On Instruction 10
1.6.1 Before You Begin 10
1.7 Creating a New RSLogix 5000 Project 11
1.7.1 Creating the Remote Network 12
1.7.2 Creating the Module - Remote Rack 14
1.7.3 Creating the Module - Local Rack 16
1.7.4 Importing the Add-On Instruction 18
1.8 Downloading the Sample Program to the Processor 29

2 Configuring the MVI56E-MNETR Module 30

2.1 Installing ProSoft Configuration Builder 30
2.2 Using ProSoft Configuration Builder Software 30
2.2.1 Upgrading from MVI56-MNETR in ProSoft Configuration Builder 30
2.2.2 Setting Up the Project 31
2.2.3 Setting Module Parameters 32
2.2.4 Module 33
2.2.5 MNET Client x 37
2.2.6 MNET Client x Commands 39
2.2.7 MNET Servers 46
2.2.8 Static ARP Table 48
2.2.9 Ethernet Configuration 49
2.3 Connecting Your PC to the Module 50
2.3.1 Setting Up a Temporary IP Address 50
2.4 Downloading the Project to the Module 54
2.4.1 Using CIPconnect to Connect to the Module 56
2.4.2 Using RSWho to Connect to the Module 64

3 Ladder Logic 65

3.1 Controller Tags 65
3.1.1 MVI56E-MNETR Controller Tags 65
3.2 User-Defined Data Types (UDTs) 67
3.2.1 MVI56(E)-MNETR User-Defined Data Types 67
3.3 Using Controller Tags 68
3.4 Controller Tag Overview 69
3.4.1 MNETR.DATA 69
3.4.2 MNETR.STATUS 72
3.4.3 MNETR.CONTROL 72

3.4.4 MNETR.UTIL 73

4 Diagnostics and Troubleshooting 74

4.1 LED Status Indicators 74
4.1.1 Scrolling LED Status Indicators 74
4.1.2 Ethernet LED Indicators 75
4.1.3 Non-Scrolling LED Status Indicators 76
4.1.4 Clearing a Fault Condition 76
4.1.5 Troubleshooting 76
4.2 Using the Diagnostics Menu in ProSoft Configuration Builder 78
4.2.1 Connecting to the Module's Webpage 81
4.2.2 The Diagnostics Menu 82
4.2.3 Monitoring Module Information 83
4.2.4 Monitoring Backplane Information 84
4.2.5 Monitoring MNET Client Information 84
4.2.6 Monitoring MNET Server Information 85
4.2.7 Monitoring Database Information 85
4.3 Reading Status Data from the Module 86
4.3.1 Status Data Definition 87
4.3.2 Client Configuration Error Word 88
4.3.3 Client Command Errors 89

5 Reference 91

5.1 Product Specifications 91
5.1.1 General Specifications 91
5.1.2 Functional Specifications 92
5.1.3 Hardware Specifications 92
5.2 Module Power Up 93
5.3 Backplane Data Transfer 94
5.3.1 Module's Internal Database Structure 95
5.3.2 Normal Data Transfer Blocks 96
5.3.3 Special Function Blocks 99
5.4 Data Flow between MVI56E-MNETR and ControlLogix Processor 109
5.4.1 Server Driver 110
5.4.2 Client Driver 112
5.5 Ethernet Cable Specifications 114
5.5.1 Ethernet Cable Configuration 114
5.5.2 Ethernet Performance 114
5.6 Modbus Protocol Specification 115
5.6.1 About the Modbus TCP/IP Protocol 115
5.6.2 Commands Supported by the Module 116
5.6.3 Read Coil Status (Function Code 01) 117
5.6.4 Read Input Status (Function Code 02) 118
5.6.5 Read Holding Registers (Function Code 03) 119
5.6.6 Read Input Registers (Function Code 04) 120
5.6.7 Force Single Coil (Function Code 05) 121
5.6.8 Preset Single Register (Function Code 06) 122
5.6.9 Diagnostics (Function Code 08) 123
5.6.10 Force Multiple Coils (Function Code 15) 125
5.6.11 Preset Multiple Registers (Function Code 16) 126
5.6.12 Modbus Exception Responses 127
5.7 Using the Optional Add-On Instruction Rung Import 129

5.7.1 Before You Begin 129
5.7.2 Overview 129
5.7.3 Installing the Rung Import with Optional Add-On Instruction 130
5.7.4 Reading the Ethernet Settings from the Module 134
5.7.5 Writing the Ethernet Settings to the Module 135
5.7.6 Reading the Clock Value from the Module 136
5.7.7 Writing the Clock Value to the Module 137
5.8 Adding the Module to an Existing Project 138
5.9 Using the Sample Program 141
5.9.1 Opening the Sample Program in RSLogix 141
5.9.2 Choosing the Controller Type 143
5.9.3 Selecting the Slot Number for the Module 144
5.9.4 Downloading the Sample Program to the Processor 145
5.9.5 Adding the Sample Ladder to an Existing Application 145

6 Support, Service & Warranty 146

6.1 Contacting Technical Support 146
6.2 Warranty Information 146

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-MNETR module's application port(s)

1.1 What's New?

MVI56E products are **backward compatible** with existing MVI56 products, ladder logic, and module configuration files already in use. Easily swap and upgrade products while benefiting from an array of new features designed to improve interoperability and enhance ease-of-use.

- **ProSoft Configuration Builder (PCB):** New Windows software for diagnostics, connecting via the module's Ethernet port or CIPconnect®, to upload/download module configuration information and access troubleshooting features and functions.
- **ProSoft Discovery Service (PDS):** Utility software to find and display a list of MVI56E modules on the network and to temporarily change an IP address to connect with a module's web page.
- **CIPconnect-enabled:** Allows PC-to-module configuration and diagnostics from the Ethernet network through a ControlLogix 1756-ENBT EtherNet/IP™ module.
- **Personality Module:** An industrial compact flash memory card storing the module's complete configuration and Ethernet settings, allowing quick and easy replacement.
- **LED Scrolling Diagnostic Display:** 4-character, alphanumeric display, providing standard English messages for status and alarm data, and for processor and network communication status.

1.2 System Requirements

The MVI56E-MNETR module requires the following minimum hardware and software components:

- Rockwell Automation ControlLogix® processor (firmware version 10 or higher), with compatible power supply, and one free slot in the rack for the MVI56E-MNETR module. The module requires 800 mA of available 5 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)
- ProSoft Discovery Service (PDS) (included in PCB)
- 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.

1.3 Package Contents

The following components are included with your MVI56E-MNETR module, and are all required for installation and configuration.

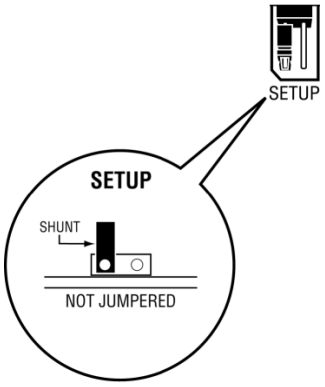
Important: Before beginning the installation, please verify that all of the following items are present.

Qty.	Part Name	Part Number	Part Description
1	MVI56E-MNETR Module	MVI56E-MNETR	Modbus TCP/IP Interface Module with Reduced Data Block

1.4 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 jumper configuration in the OFF position



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.

You should 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.

You should 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.

1.5 Installing the Module in the Rack

If you have not already installed and configured your ControlLogix processor and power supply, please do so before installing the MVI56E-MNETR 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-MNETR into the ControlLogix chassis. Use the same technique recommended by Rockwell Automation to remove and install ControlLogix modules.

You can install or remove ControlLogix system components while chassis power is applied and the system is operating. However, please note the following warning.

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 your 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.

1.6 Importing the Sample 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 141).

1.6.1 Before You Begin

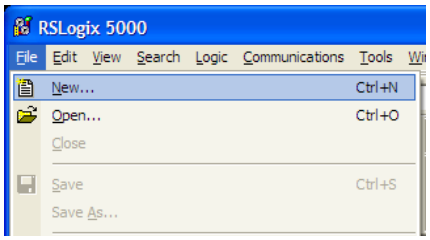
Two Add-On Instructions are provided for the MVI56E-MNETR module. The first is required for setting up the module; the second is optional.

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

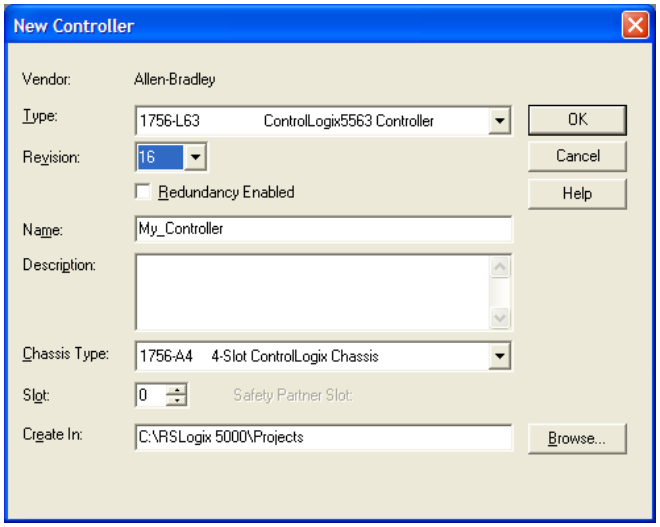
File Name	Description
MVI56EMNETR_AddOn_Rung_vXXX.L5X	L5X file containing Add-On Instruction, user defined data types, controller tags and ladder logic required to configure the MVI56E-MNETR module
MVI56EMNETR_Optional_Rung_vXXX.L5X	Optional L5X file containing additional Add-On Instruction with logic for changing Ethernet configuration and clock settings.

1.7 Creating a New RSLogix 5000 Project

1 Open the **FILE** menu, and then choose **NEW**.



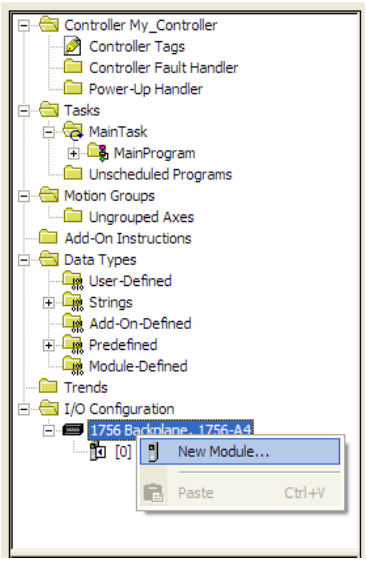
2 Select your ControlLogix controller model.
3 Select **REVISION 16**.
4 Enter a name for your controller, such as *My_Controller*.
5 Select your ControlLogix chassis type.
6 Select **SLOT 0** for the controller.



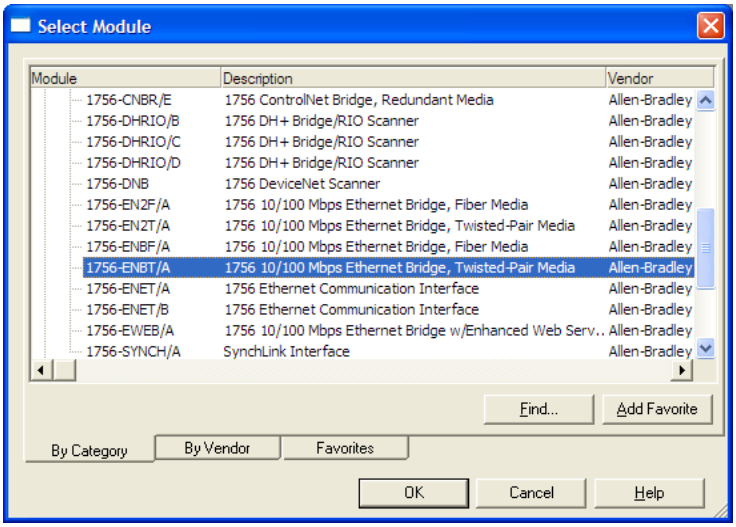
1.7.1 Creating the Remote Network

Note: If you are installing the MVI56E-MNETR module in a remote rack, follow these steps. If you are installing the module in a local rack, follow the steps in Creating the Module - Local Rack (page 16).

1 Right-click **I/O CONFIGURATION** and choose **NEW MODULE**.

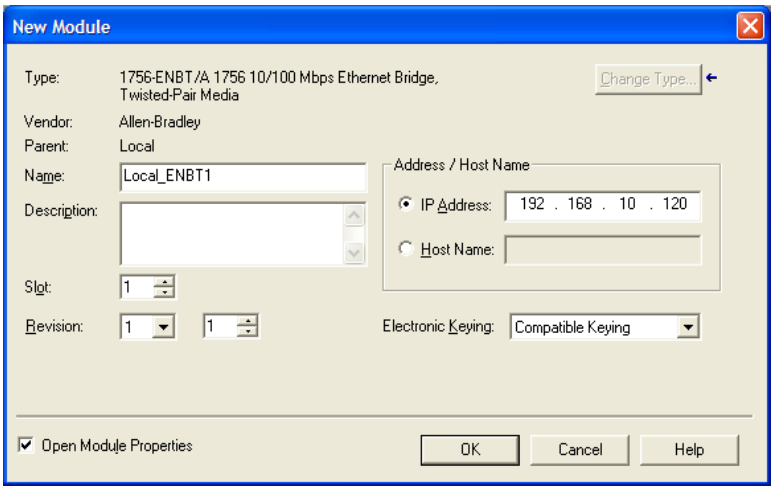


2 Expand the Communications module selections and then select the Ethernet Bridge module that matches your hardware. This example uses a 1756-ENBT/A module.



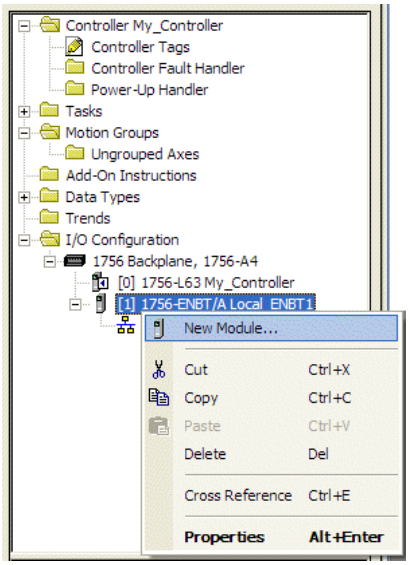
Note: If you are prompted to *Select Major Revision*, choose the lower of the available revision numbers.

3 Name the ENBT/A module, then set the IP Address and slot location in the local rack with the ControlLogix processor.



4 Click **OK**.

5 Next, select the **1756-ENBT** module that you just created in the *Controller Organization* pane and click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE**.

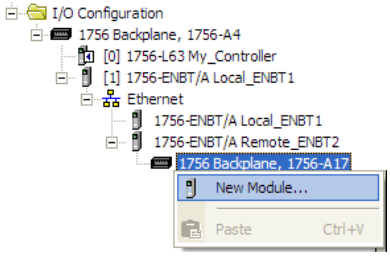


6 Repeat steps 2 and 3 to add the second EtherNet/IP module to the remote rack.

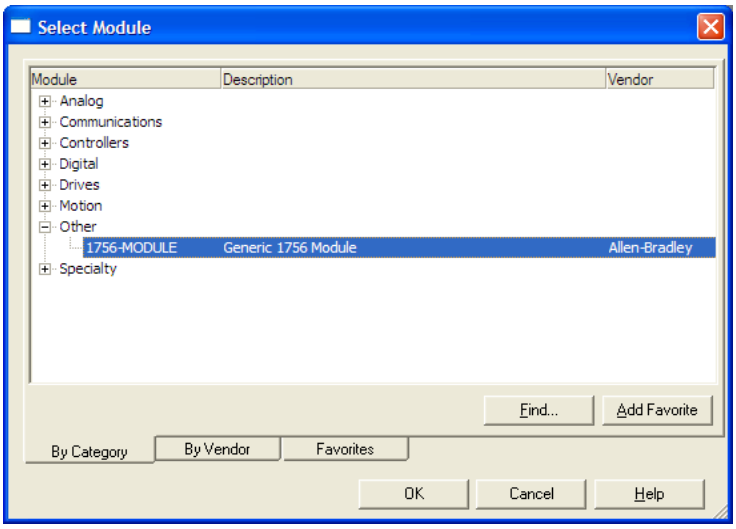
1.7.2 Creating the Module - Remote Rack

Note: To continue installing the MVI56E-MNETR module in a remote rack, follow the next steps. If you are installing the module in a local rack, follow the steps in Creating the Module - Local Rack (page 16).

1 Next, select the remote **1756 BACKPLANE** node in the Controller Organization pane underneath the remote rack EtherNet/IP module you just created 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.

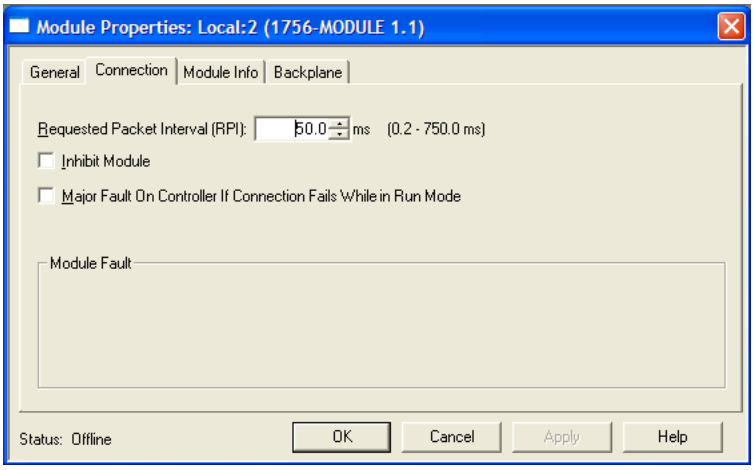


2 Select the **1756-MODULE (GENERIC 1756 MODULE)** from the list and click **OK**. This action opens the *New Module* dialog box.

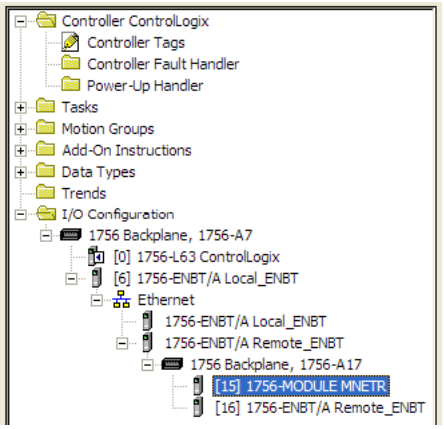
3 Set the Module Properties values as follows:

Parameter	Value
Name	Enter a module identification string. The recommended value is **MNETR**.
Description	Enter a description for the module. Example: **MODBUS TCP/IP INTERFACE MODULE WITH REDUCED DATA BLOCK**.
Comm Format	Select **DATA-INT (Very Important)**
Slot	Enter the slot number in the rack where the MVI56E-MNETR module will be installed.
Input Assembly Instance	**1**
Input Size	**42**
Output Assembly Instance	**2**
Output Size	**42**
Configuration Assembly Instance	**4**
Configuration Size	**0**

4 On the **CONNECTION** tab, set the *RPI* value for your project. Fifty (**50**) milliseconds is usually a good starting value.



The **MVI56E-MNETR** module is now visible in the **I/O CONFIGURATION** section.

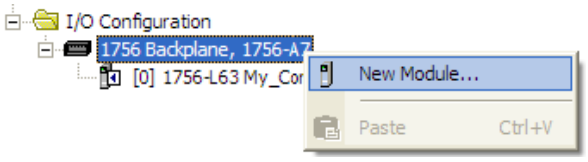


1.7.3 Creating the Module - Local Rack

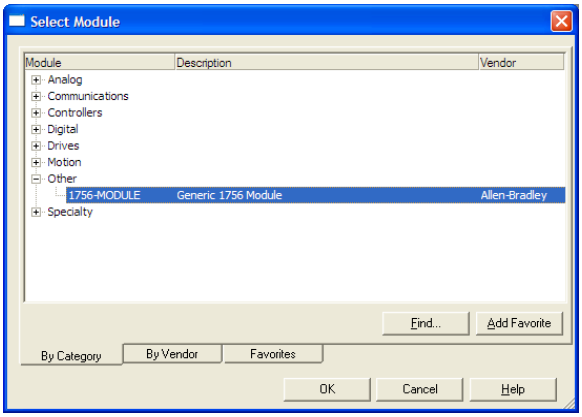
Note: If you are installing the MVI56E-MNETR module in a local rack, follow these steps. If you are installing the module in a remote rack, follow the steps in Creating the Module - Remote Rack (page 12).

1 Add the MVI56E-MNETR module to the project.

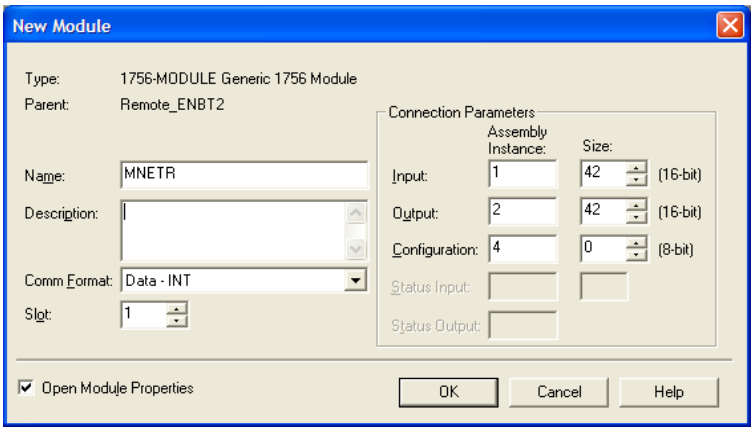
In the *Controller Organization* window, select **I/O CONFIGURATION** and click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE**.



This action opens the *Select Module* dialog box.



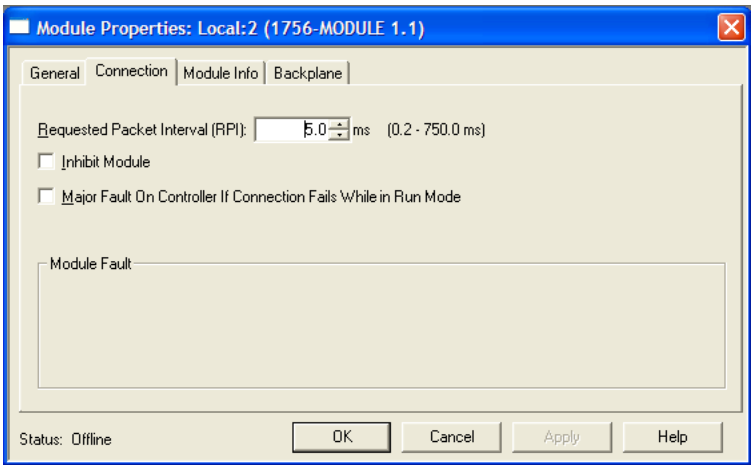
2 Select the **1756-MODULE (GENERIC 1756 MODULE)** from the list and click **OK**. This action opens the *New Module* dialog box.



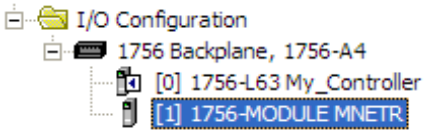
3 Set the Module Properties values as follows:

Parameter	Value
Name	Enter a module identification string. The recommended value is **MNETR**.
Description	Enter a description for the module. Example: **MODBUS TCP/IP INTERFACE MODULE WITH REDUCED DATA BLOCK**.
Comm Format	Select **DATA-INT (Very Important)**
Slot	Enter the slot number in the rack where the MVI56E-MNETR module is to be installed.
Input Assembly Instance	**1**
Input Size	**42**
Output Assembly Instance	**2**
Output Size	**42**
Configuration Assembly Instance	**4**
Configuration Size	**0**

4 On the **CONNECTION** tab, set the *RPI* value for your project. Five (**5**) milliseconds is usually a good starting value. Click **OK** to confirm.

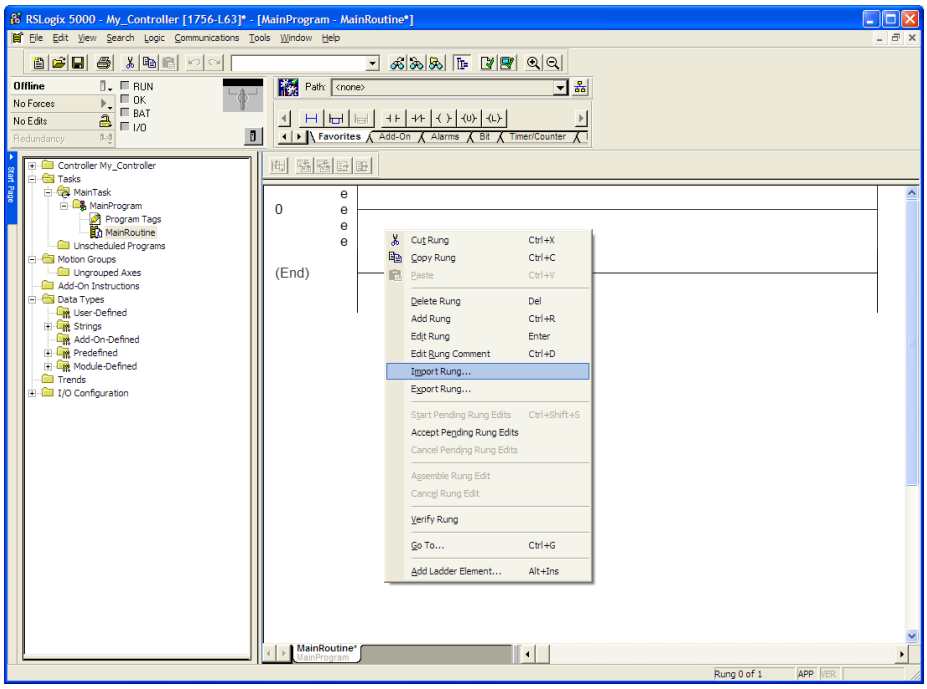


The **MVI56E-MNETR** module is now visible in the **I/O CONFIGURATION** section.

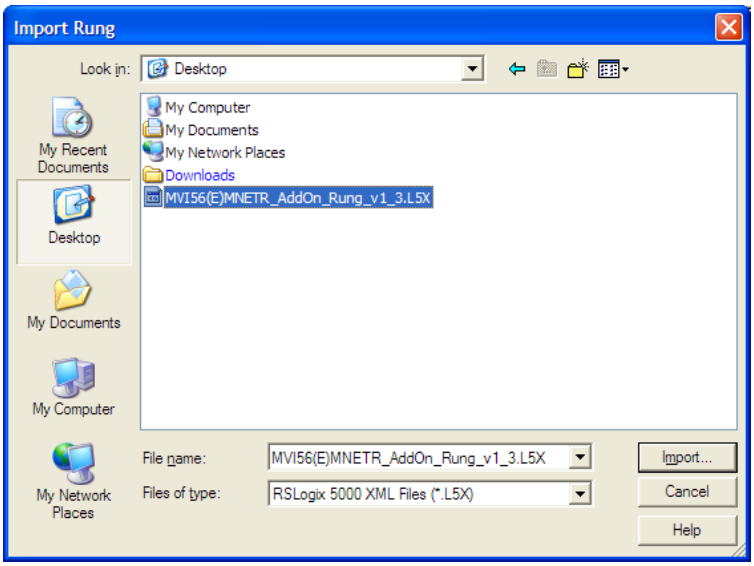


1.7.4 Importing the Add-On Instruction

1. In the *Controller Organization* window, expand the *Tasks* folder and subfolder 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 new routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNG**.

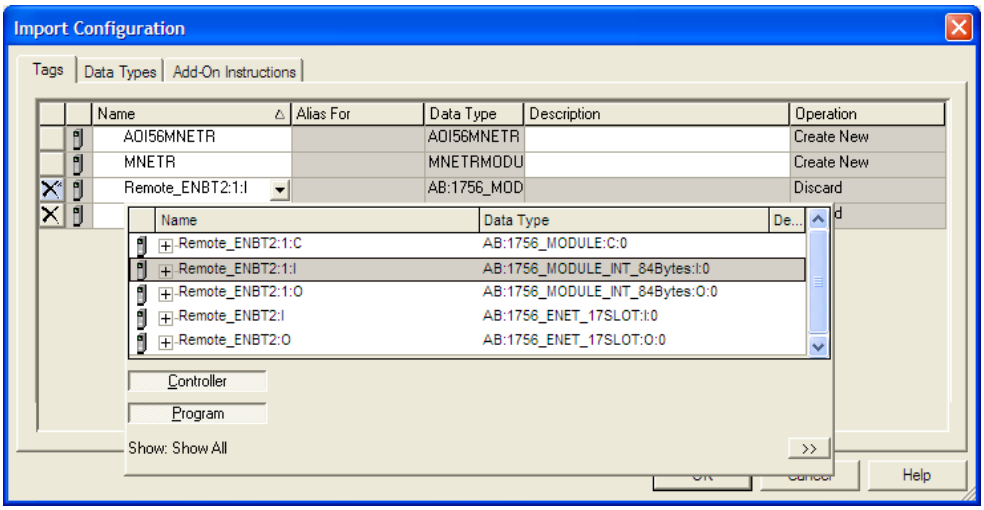


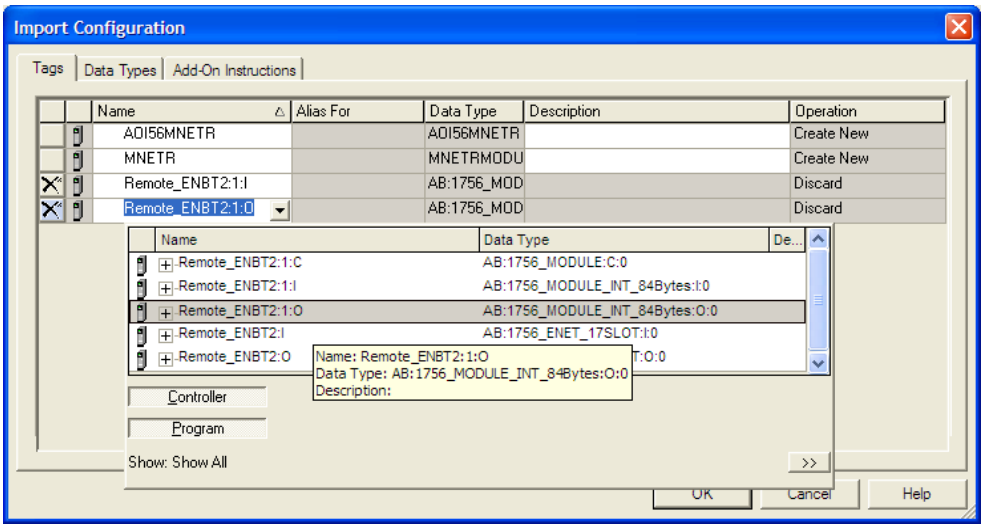
4. Navigate to the location on your PC where you saved (page 10) the Add-On Instruction (for example, *My Documents* or *Desktop*). Select the **MVI56EMNETR_ADDON_RUNG_V1_3.L5X** file.



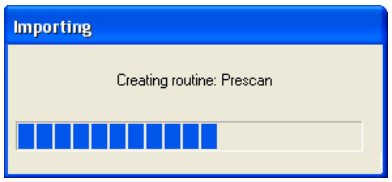
This action opens the *Import Configuration* dialog box, showing the controller tags that will be created.

- If you are installing the module in a Remote Rack, open the dropdown menus for the Input and Output tags, and select the MNETR module in the remote rack.

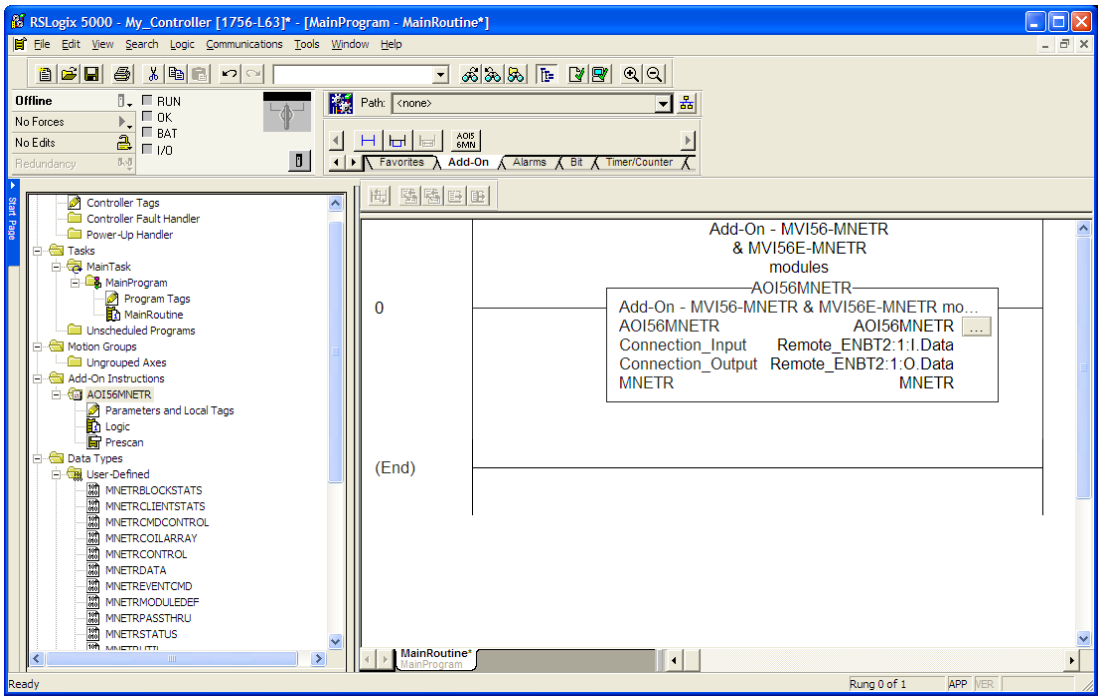




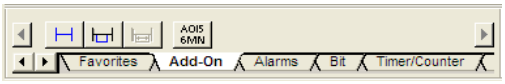
5 Click **OK** to confirm the import. RSLogix will indicate that the import is in progress:



When the import is complete, you will see the new Add-On Instruction rung in the ladder.



The procedure has also imported new user-defined data types, data objects and the Add-On instruction for your project.

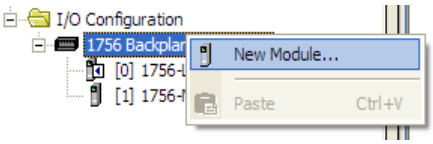


6 Save the application and then download the sample ladder logic into the processor.

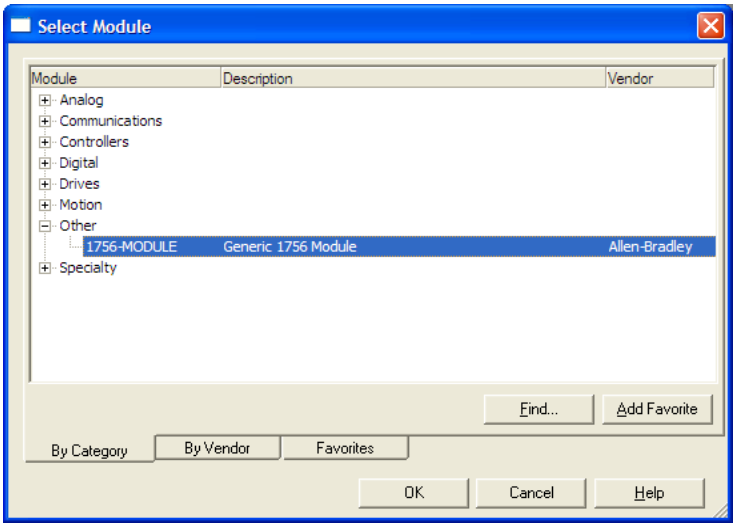
Adding Multiple Modules (Optional)

Important: If your application requires more than one MVI56-MNETR module into the same project, follow the steps below.

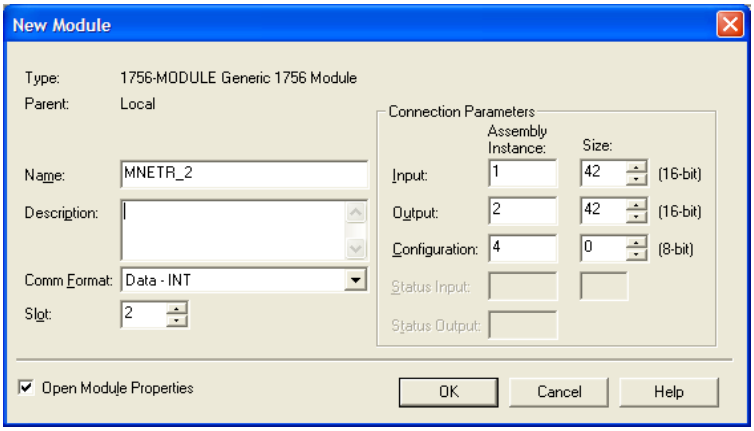
1 In the *I/O Configuration* folder, click the right mouse button to open a shortcut menu, and then choose **NEW MODULE**.



2 Select **1756-MODULE**.



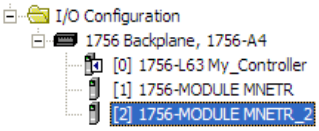
This action opens the *New Module* dialog box.



3 Fill in the module properties as follows:

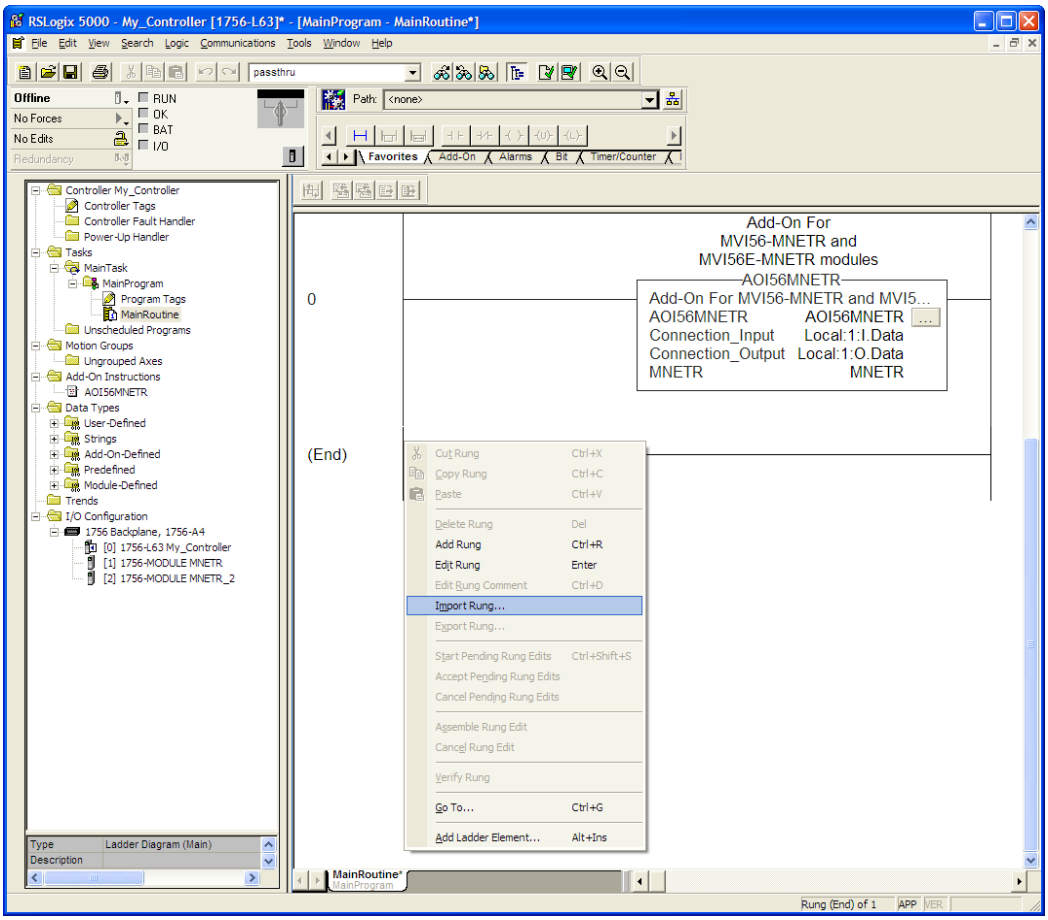
Parameter	Value
Name	Enter a module identification string. Example: **MNETR_2**
Description	Enter a description for the module. Example: **MODBUS TCP/IP INTERFACE MODULE WITH REDUCED DATA BLOCK**
Comm Format	Select **DATA-INT**
Slot	Enter the slot number in the rack where the MVI56E-MNETR module is located.
Input Assembly Instance	**1**
Input Size	**42**
Output Assembly Instance	**2**
Output Size	**42**
Configuration Assembly Instance	**4**
Configuration Size	**0**

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

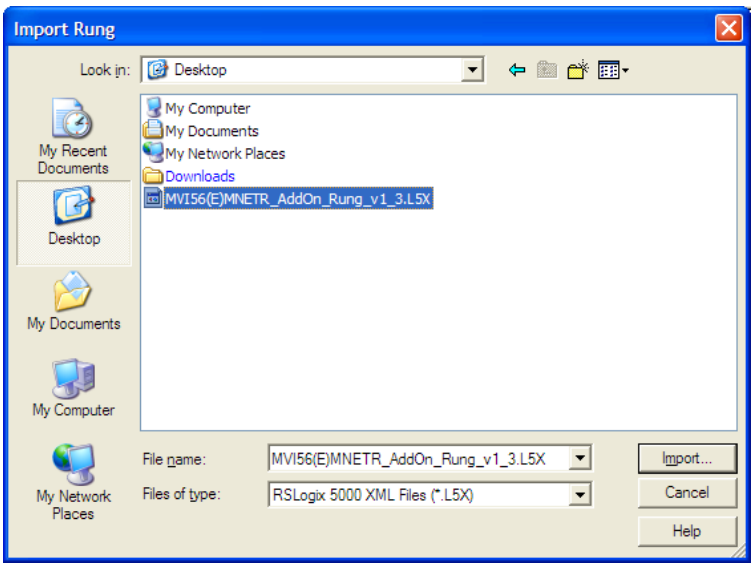


5 Expand the *Tasks* folder, and then expand the *MainTask* folder.
6 On the *MainProgram* folder, click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW ROUTINE**.
7 In the *New Routine* dialog box, enter the name and description of your routine, and then click **OK**.

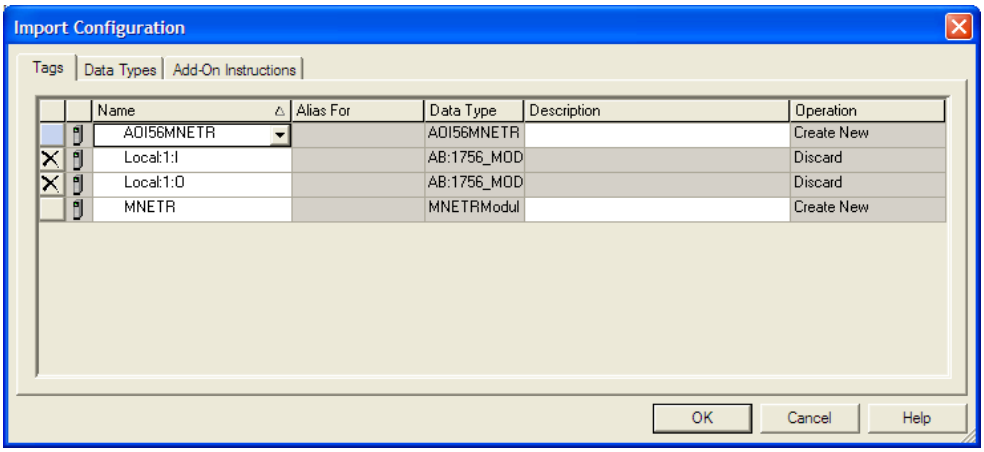
8 Select an empty rung in the new routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNG**.



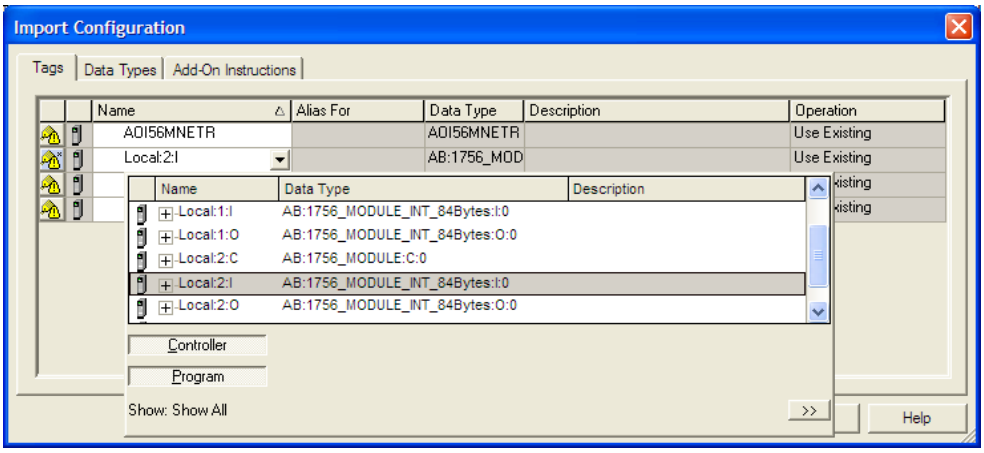
9 Select the file MVI56(E)MNETR_AddOn_Rung_<Version#>.L5X

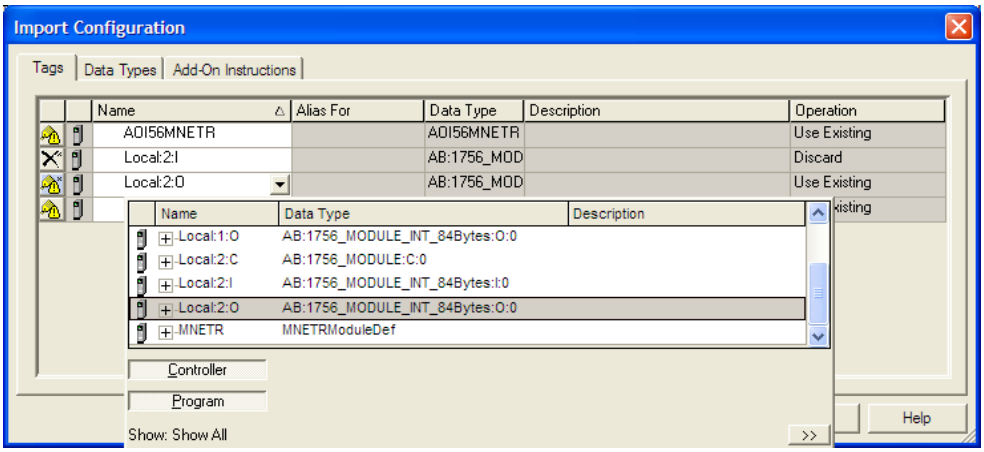


This action opens the **IMPORT CONFIGURATION** dialog box, showing the controller tags that will be created.

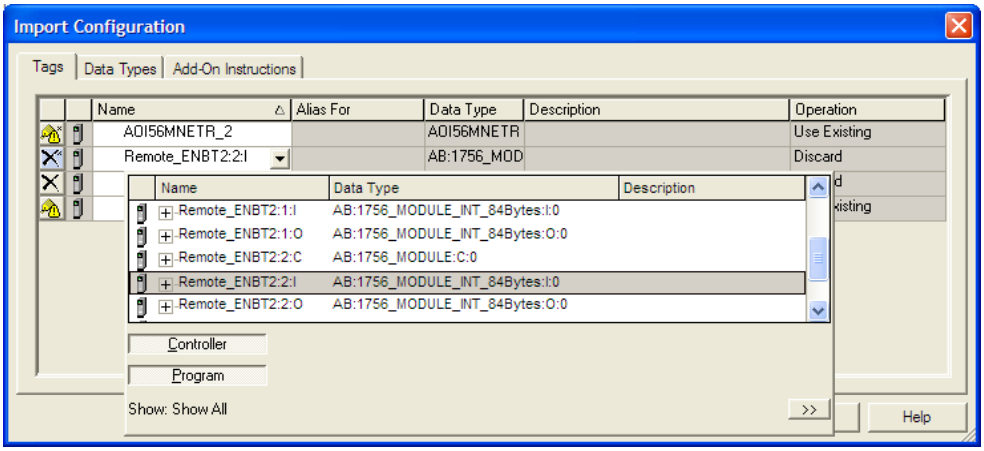


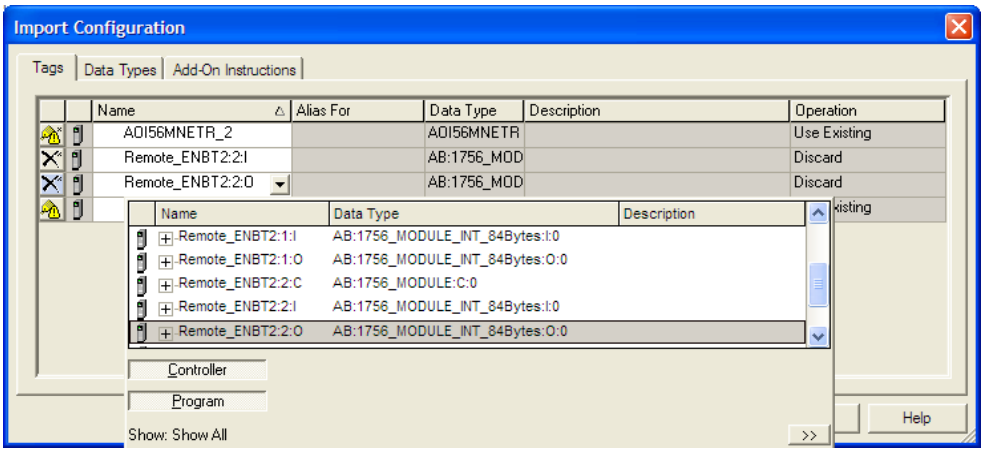
10 Associate the I/O connection variables to the correct module. The default values are Local:1:I and Local:1:O so these require change.



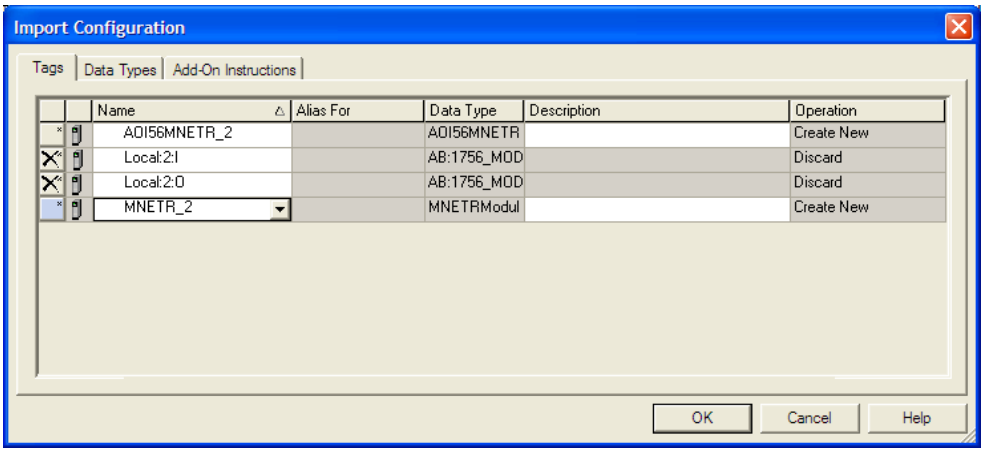


Or, in a Remote Rack application…

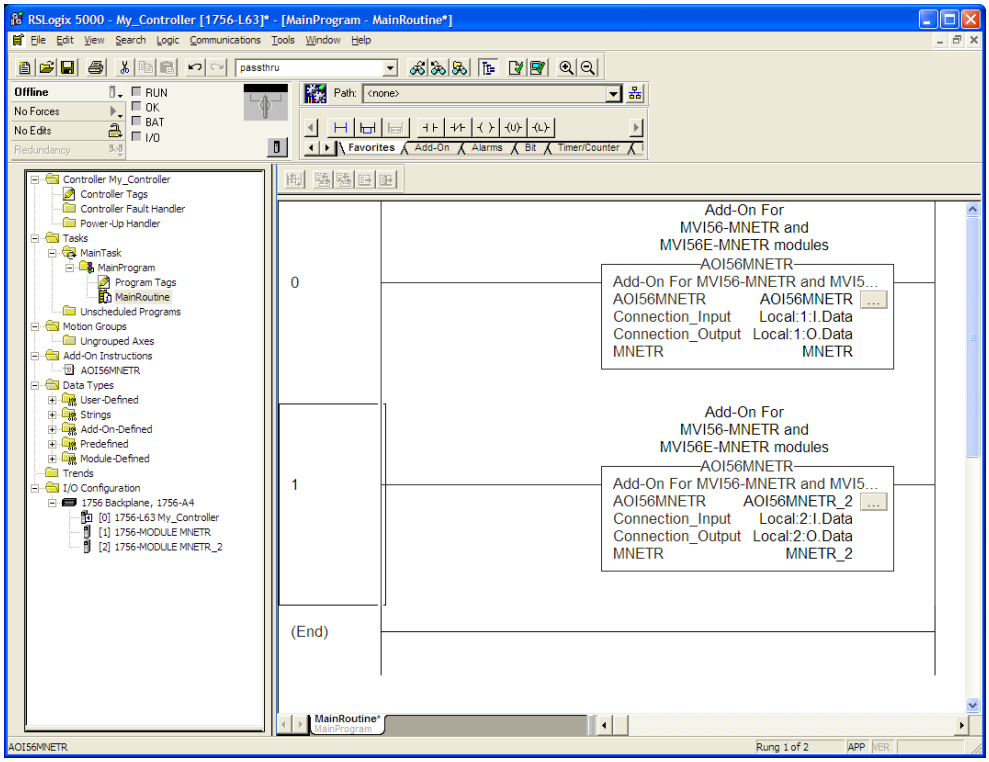




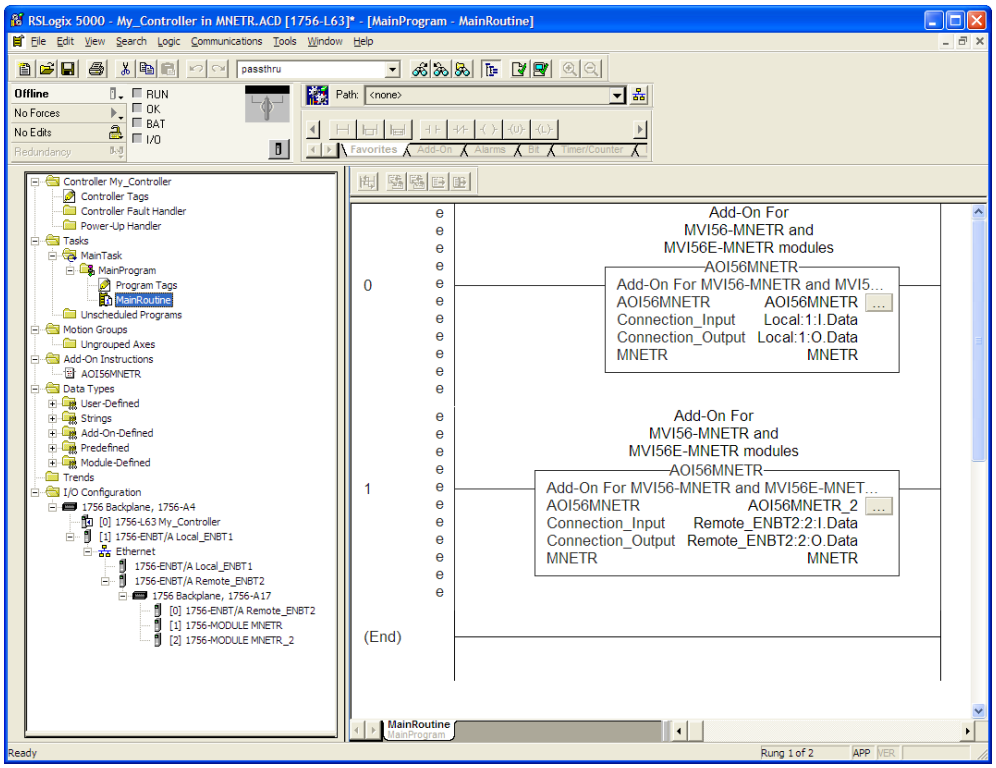
11 Change the default tags *MNETR* and *AOI56MNETR* to avoid conflict with existing tags. In this step, you should append a string to the default tag names, such as "_2", as shown in the following illustration.



12 Click **OK** to confirm.



Or, in a Remote Rack application…



Adjusting the Input and Output Array Sizes

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

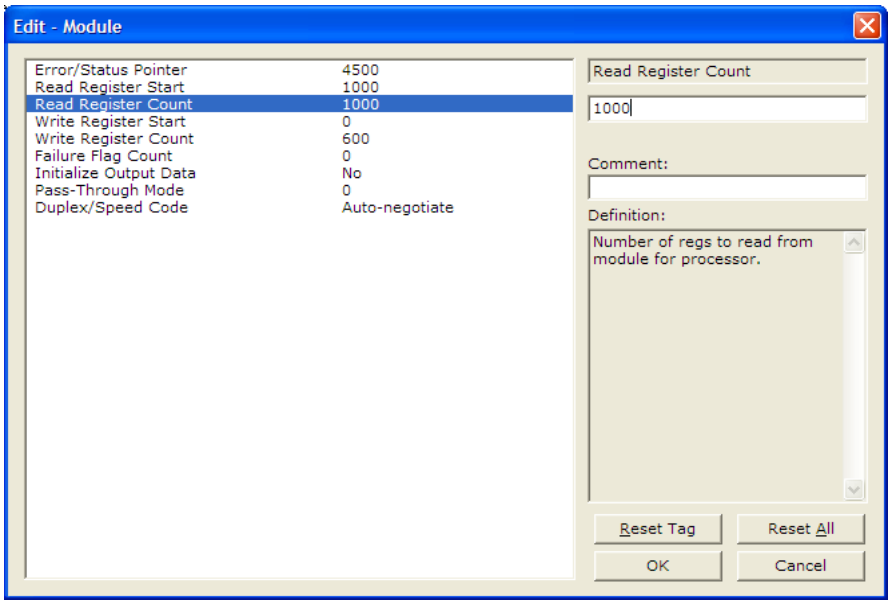
- Read Data
- Write Data

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.

The MVI56E-MNETR Add-On Instruction rung is configured for 600 registers of Read Data and 600 registers of Write Data, which is sufficient for most applications. However, you can configure the sizes of these data areas to meet the needs of your application.

1 In *ProSoft Configuration Builder*, expand the *Module* icon in the tree view and double-click **MODULE** to open an *Edit* window. Change the **READ REGISTER COUNT** to contain the number of words for your Read Data area.

Important: Because the module pages data in blocks of 40 registers at a time, you should configure your user data areas in multiples of 40 registers.



2 To modify the *WriteData* array, follow the above steps, substituting *WriteData* for *ReadData*.

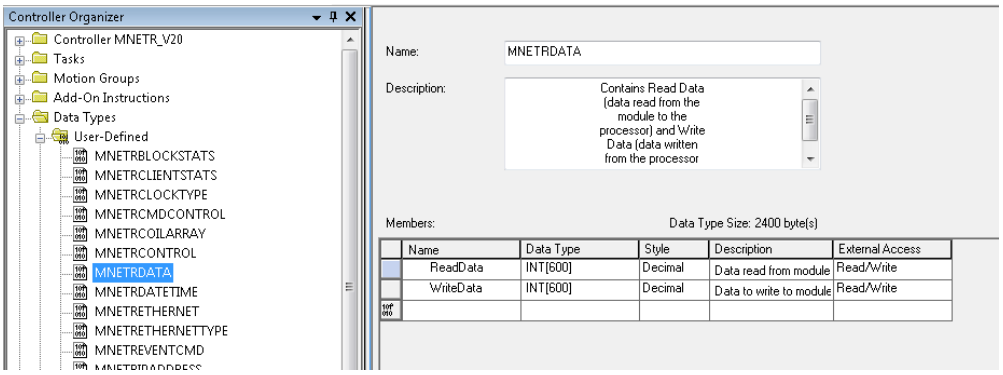
3 Save and download the configuration to the module (page 54) and reboot.

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 *Read Register Start* parameter must be set to a value of 2000 or greater.

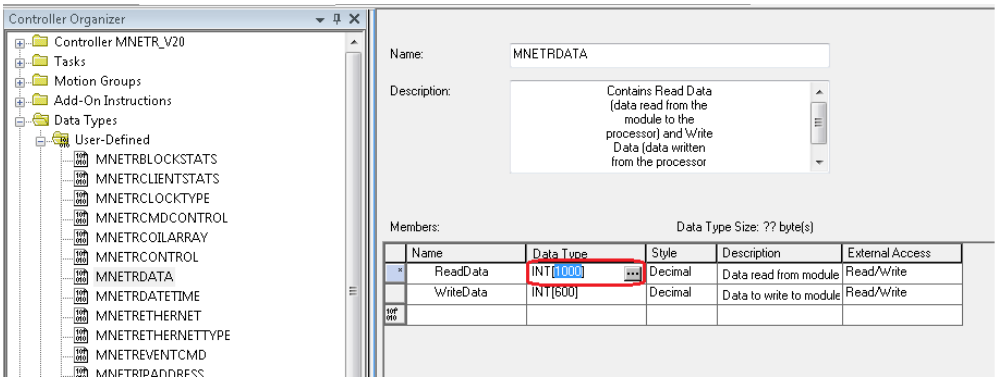
Adjusting the Input and Output Array Sizes in Logix 5000

You will also need to adjust the sizes of the *MNETR.DATA.ReadData* and *MNETR.DATA.WriteData* Controller Tag arrays to accommodate the *Read Register Count* and *Write Register Count* values configured in PCB.

1 In the *Controller Organizer* pane of RSLogix 5000, double-click on the *MNETRDATA* UDT.

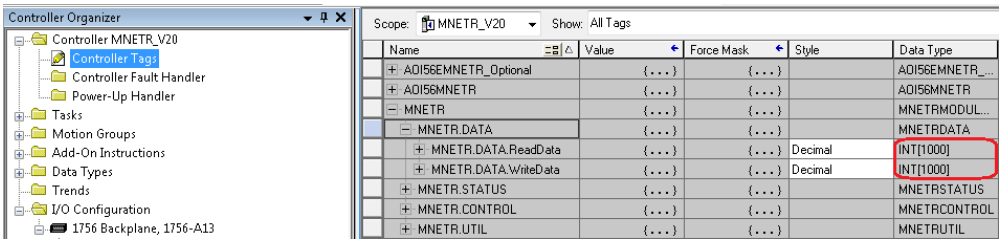


2 Edit the *ReadData* array size to match or exceed the value of the *Read Register Count* parameter in PCB.



3 Repeat Step **2** for the *WriteData* array size. Then click **OK**.

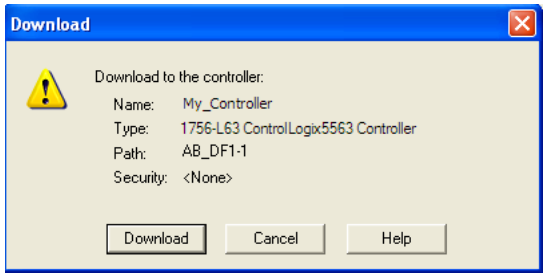
4 Once complete, the *MNETR.DATA.ReadData* and *MNETR.DATA.WriteData* Controller Tag array sizes are updated.



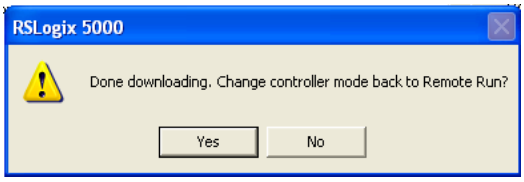
1.8 Downloading the Sample Program to the Processor

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, 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 Configuring the MVI56E-MNETR Module

2.1 Installing ProSoft Configuration Builder

1 Download the *ProSoft Configuration Builder* from www.prosoft-technology.com.
2 Run the installation file to start the installation wizard.

2.2 Using ProSoft Configuration Builder Software

ProSoft Configuration Builder (PCB) provides a convenient 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-MNETR 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.

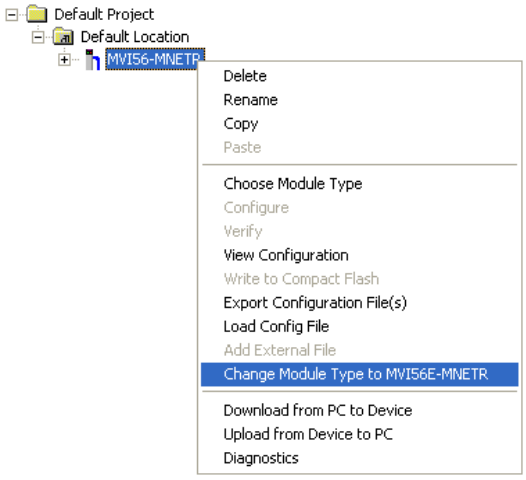
2.2.1 Upgrading from MVI56-MNETR in ProSoft Configuration Builder

MVI56E-MNETR modules are fully backward-compatible with MVI56-MNETR modules. However, you will need to convert your MVI56-MNETR configuration in *ProSoft Configuration Builder* to a form that your new MVI56E-MNETR 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, you need to have *ProSoft Configuration Builder* version 2.2.2 or later installed on your PC. You can download the latest version from www.prosoft-technology.com.

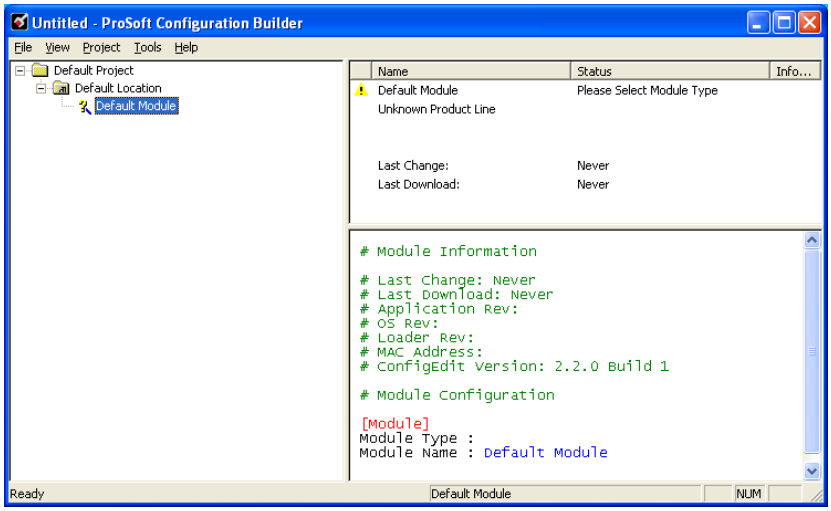
1 In *ProSoft Configuration Builder's* tree view, click the **MODULE** icon and right-click to open a shortcut menu.



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

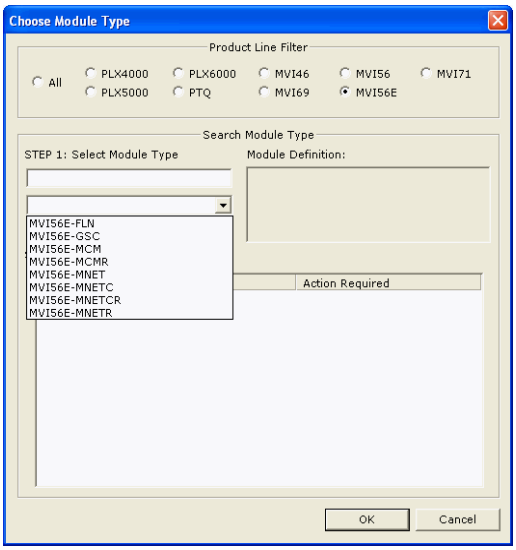
2.2.2 Setting Up the Project

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 select **DEFAULT MODULE** in the tree view, and then click the right mouse button to open a shortcut menu.

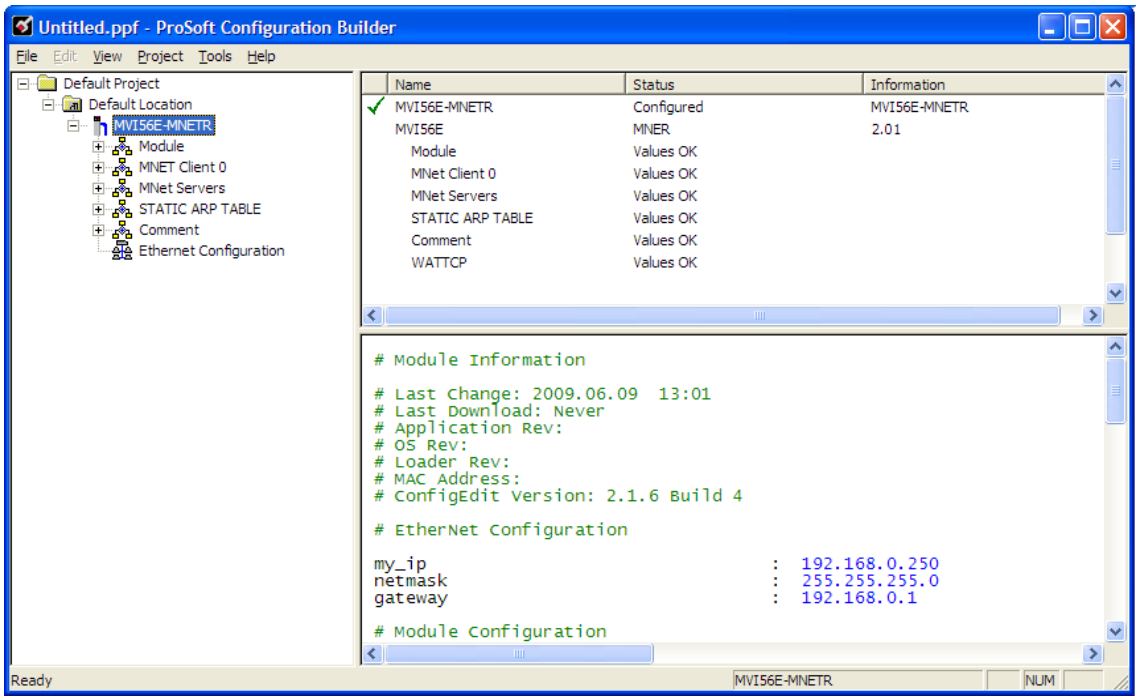
2 On the shortcut menu, select **CHOOSE MODULE TYPE**. This action opens the *Choose Module Type* dialog box.



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

2.2.3 Setting Module Parameters

Notice that the contents of the information pane and the configuration pane changed when you added the MVI56E-MNETR 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. Select the object, and then click the right mouse button to open a shortcut menu. From the shortcut menu, 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. Select the module icon, and then click the right mouse button to open a shortcut menu.
2. On the shortcut menu, choose **VIEW CONFIGURATION.** This action opens the *View Configuration* window.
3. In the *View Configuration* window, open the **FILE** menu, and choose **PRINT**. This action opens the *Print* dialog box.
4. In the *Print* dialog box, choose the printer to use from the drop-down list, select printing options, and then click **OK**.

2.2.4 Module

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

Backplane Error/Status Pointer

1 to **4955**

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 96).

Read Register Start

0 to **4999**

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.

Note: Total user database memory space is limited to the first 5000 registers of module memory, addresses 0 through 4999. Therefore, the practical limit for this parameter is 4999 minus the value entered for *Read Register Count*, so that the Read Data Area does not try to extend above address 4999. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

Read Register Count

0 to **5000**

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 5000 words.

Note: Total *Read Register Count* and *Write Register Count* cannot exceed 5000 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 **4999**

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 5000 registers of module memory, addresses 0 through 4999. Therefore, the practical limit for this parameter is 4999 minus the value entered for *Write Register Count*, so that the Write Data Area does not try to extend above address 4999. 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 **5000**

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 5000 words.

Note: Total *Read Register Count* and *Write Register Count* cannot exceed 5000 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

0 through **65535**

This parameter specifies the number of successive transfer errors that must occur before halting communication on the application port(s). If the parameter is set to **0**, the application port(s) will continue to operate under all conditions. If the value is set larger than **0** (**1** to **65535**), communications will cease if the specified number of failures occur.

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: If pass-through mode is used, the module's Write Data area must begin at 0 (*Write Register Start* = **0**).

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.

2.2.5 MNET Client x

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

Client Error/Status Pointer

1 to **4990**

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. 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 96).

Command Error Pointer

-1 to **4999**

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.

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.

Float Flag

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.

You will also need to set the *Float Start* and *Float Offset* parameters to appropriate values whenever the *Float Flag* parameter is set to **YES**.

Float Start

0 to **65535**

Whenever the *Float Flag* parameter is set to **YES**, this parameter determines the lowest Modbus Address, used in commands to a remote server, to consider as commands to read or write floating-point data. All commands with address values greater than or equal to this value will be considered floating-point data commands. All commands with address values less than this value will be considered normal 16-bit register data commands.

This parameter is used only if the *Float Flag* is set to **YES**. For example, if a value of 7000 is entered, all commands sent with addresses of 47001 (or 407001) and above will be considered as floating-point data commands and 32 bits of data will be sent or received for each count of one in the command.

You will also need to set the *Float Offset* parameter to an appropriate value whenever the *Float Flag* parameter is set to **YES**.

Float Offset

0 to **9999**

This parameter defines the start register for floating-point data in the internal database. This parameter is used only if the *Float Flag* is enabled. For example, if the *Float Offset* value is set to **3000** and the *Float Start* parameter is set to **7000**, data requests for register 7000 will use the internal Modbus register 3000.

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.

2.2.6 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-MNETR 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 MVI56E-MNETR 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 100 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.

Write commands have a special feature, as they can be set to execute only if the data in the write command changes. If the register data values in the command have not changed since the command was last issued, the command will not be executed.

If the data in the command has changed since the command was last issued, the command will be executed. Use of this feature can lighten the load on the network. To implement this feature, set the enable code for the command to **CONDITIONAL** (2).

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	Set Single Coil	X	X
6	Single Register Write	X	X
7	Read Exception Status		X
8	Diagnostics		X
15	Multiple Coil Write	X	X
16	Multiple Register Write	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.

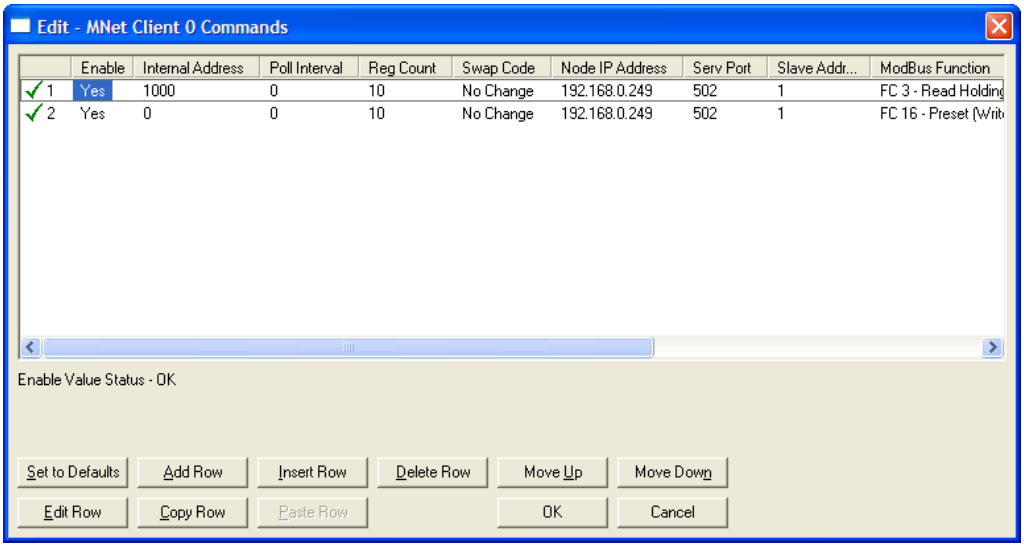
Command Entry Formats

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**
	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

The first part of the record is the module information, which relates to the MVI56E module, and the second part contains information required to interface to the server device.

Command list example:



Enable

NO (0), **YES** (1), or **CONDITIONAL** (2)

This field defines whether the command is to be executed and under what conditions.

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. If the *Poll Interval* time is set to a nonzero value, the command will be executed when the interval timer expires.
CONDITIONAL (2)	The command will execute only if the internal data associated with the command changes. This value is valid only for write commands.

Internal Address

0 to **65535** (for bit-level addressing)

or

0 to **4999** (for word-level addressing)

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

Regs: **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). If a server implementation supports another service port, enter the value here.

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 (page 43) to address commands to a remote Modbus TCP/IP device.

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**, **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
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)

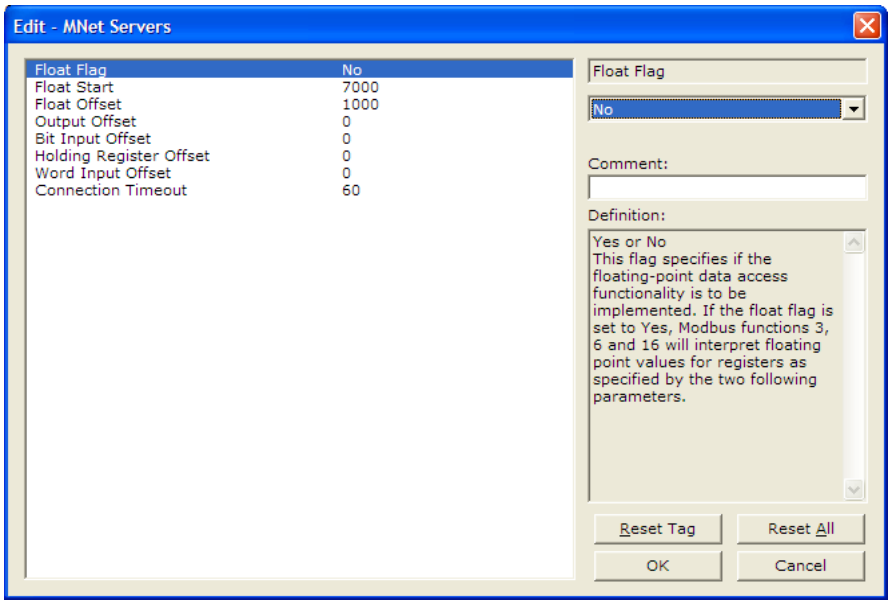
Note: If the documentation for a particular Modbus server device lists data addresses in hexadecimal (base16) notation, you will need to convert the hexadecimal value to a decimal value to enter in this parameter. In such cases, it is not usually necessary to subtract 1 from the converted decimal number,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

Comment

0 to 35 alphanumeric characters

2.2.7 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.



Float Flag

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 Client 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**, which is the default setting.

You will also need to set the *Float Start* and *Float Offset* parameters to appropriate values whenever the *Float Flag* parameter is set to **YES**.

Float Start

0 to **65535**

This parameter defines the first register of floating-point data. All requests with register values greater than or equal to this value will be considered floating-point data requests. This parameter is only used if the *Float Flag* is enabled. For example, if a value of **7000** is entered, all requests for registers 7000 and above will be considered as floating-point data.

Float Offset

0 to **9999**

This parameter defines the start register for floating-point data in the internal database. This parameter is used only if the *Float Flag* is enabled. For example, if the *Float Offset* value is set to **3000** and the *Float Start* parameter is set to **7000**, data requests for register 7000 will use the internal Modbus register 3000.

Output Offset

0 to **4999**

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 **4999**

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 **4999**

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 **4999**

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

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.

2.2.8 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-MNETR 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 accepts up to 40 static IP/MAC address 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 accepts up to 40 static IP/MAC address 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.

2.2.9 Ethernet Configuration

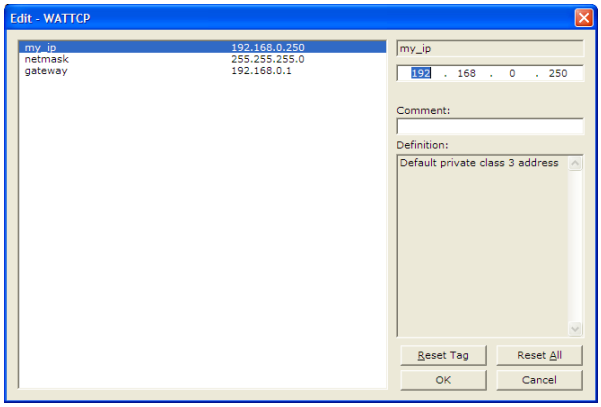
Use this procedure to configure the Ethernet settings for your module. You must assign an IP address, subnet mask and gateway address. After you complete this step, you can connect to the module with an Ethernet cable.

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.

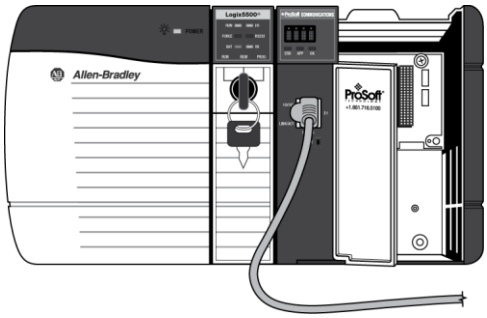


3 Edit the values for *my_ip*, *netmask* (subnet mask) and *gateway* (default gateway).

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

2.3 Connecting Your PC to the Module

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).

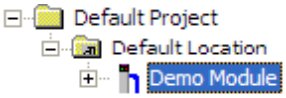


2.3.1 Setting Up a Temporary IP Address

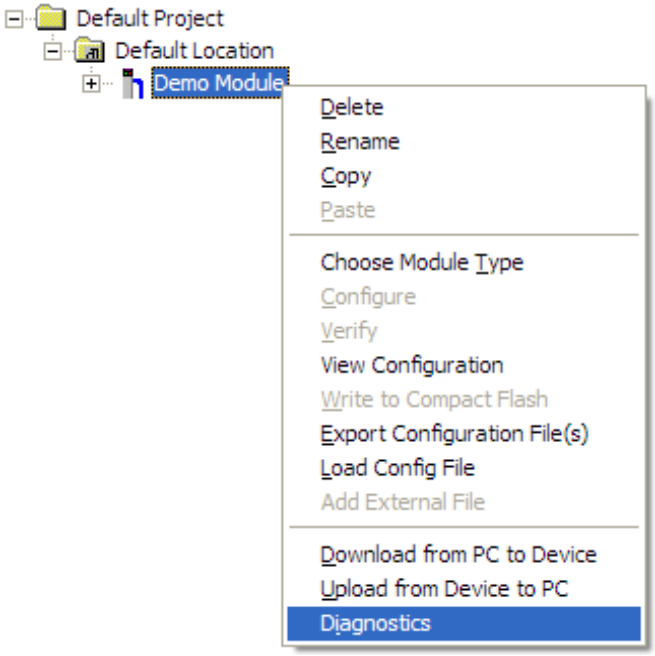
Important: *ProSoft Configuration Builder* locates MVI56E-MNETR modules through UDP broadcast messages. These messages may be blocked by routers or layer 3 switches. In that case, *ProSoft Discovery Service* will be unable to locate the modules.

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*, select the **MVI56E-MNETR** module.



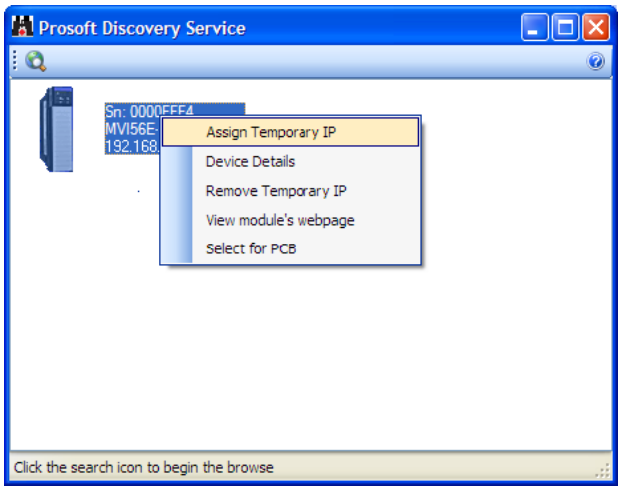
2 Click the right mouse button to open a shortcut menu. On the shortcut menu, choose **DIAGNOSTICS.**



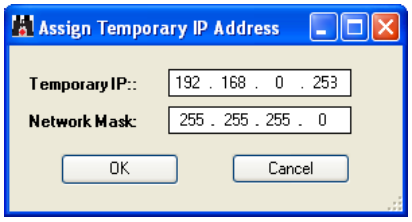
3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.



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**.

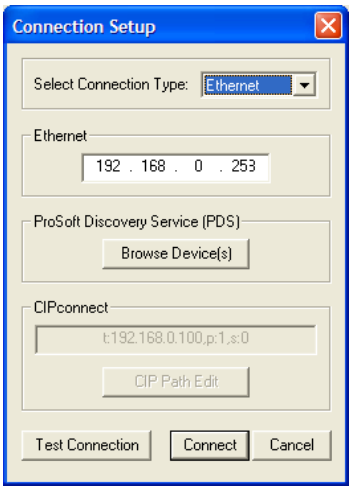


5 The module's default IP address is 192.168.0.250. Choose an unused IP within your subnet, and then click **OK**.

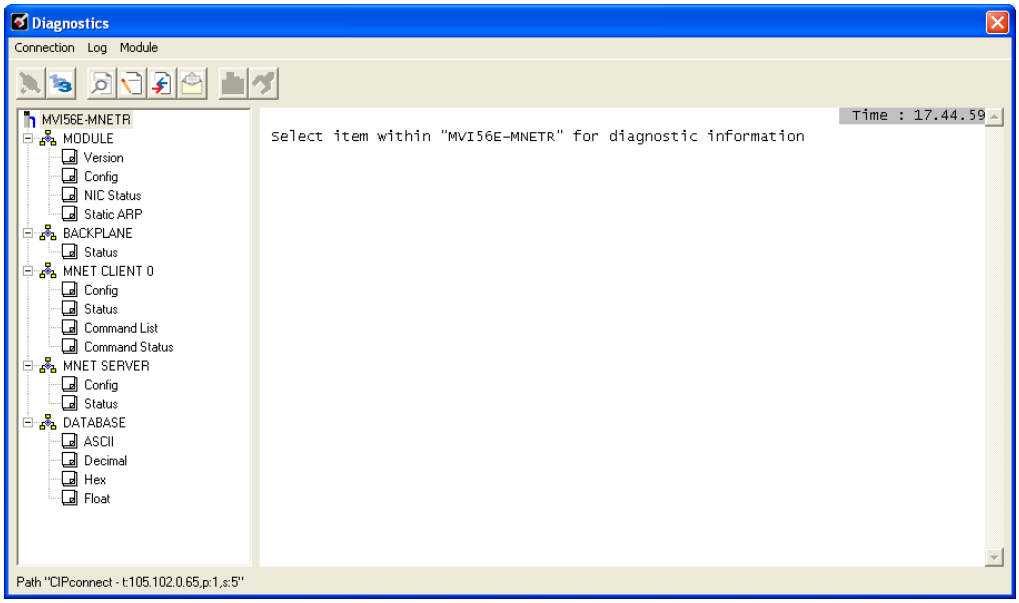


Important: The temporary IP address is only valid until the next time the module is initialized. For information on how to set the module's permanent IP address, see Ethernet Configuration (page 49).

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.



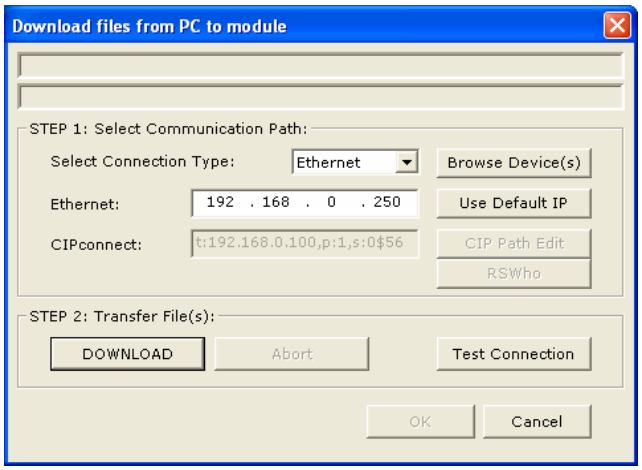
2.4 Downloading the Project to the Module

Note: For alternative methods of connecting to the module with your PC, refer to Using CIPconnect to Connect to the Module (page 56) or Using RSWho to Connect to the Module (page 64).

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

1. In the tree view in *ProSoft Configuration Builder*, click once to select the **MVI56E-MNETR** module.
2. Open the **PROJECT** menu, and then choose **MODULE / DOWNLOAD.**

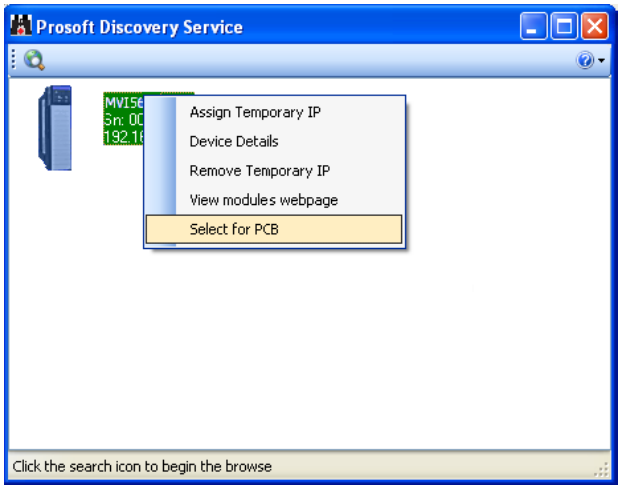
   This action opens the *Download* dialog box. Notice that the Ethernet address field contains the temporary IP address you assigned previously. *ProSoft Configuration Builder* will use this temporary IP address to connect to the module.



   Click **TEST CONNECTION** to verify that the IP address allows access to the module.
3. 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. Select the module, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **SELECT FOR PCB**.
4. Close *ProSoft Discovery Service*.
5. Click **DOWNLOAD** to transfer the configuration to the module.

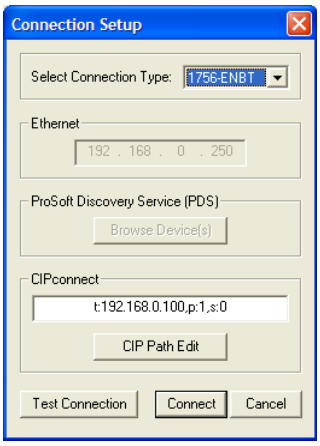
2.4.1 Using CIPconnect to Connect to the Module

You can use CIPconnect® to connect a PC to the MVI56E-MNETR module over Ethernet using Rockwell Automation's 1756-ENBT EtherNet/IP® module. This allows you to configure the MVI56E-MNETR 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:

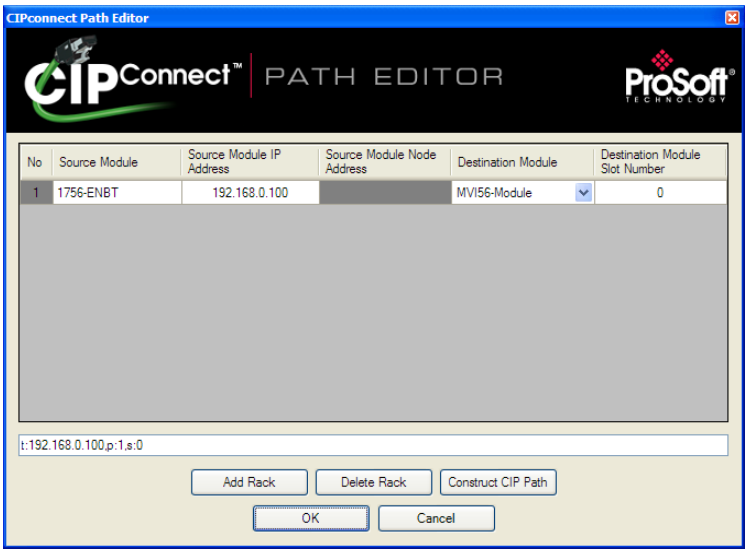
- The IP addresses and slot numbers of any 1756-ENBT modules in the path
- The ControlNet node numbers and slot numbers of any 1756-CNBx ControlNet Bridge modules in the path
- The slot number of the MVI56E-MNETR in the destination ControlLogix chassis (the last ENBT/CNBx and chassis in the path).

To use CIPconnect, follow these steps.

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



2 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-MNETR 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-MNETR 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-MNETR 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 module is located.

To use the CIPconnect Path Editor, follow these steps.

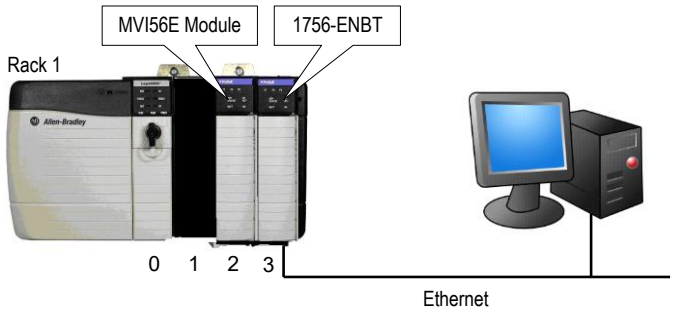
1 Configure the path between the 1756-ENBT connected to your PC and the MVI56E-MNETR module.
   - If the module is located in a remote rack, add more racks to configure the full path.
   - The path can only contain ControlNet or Ethernet/IP networks.
   - The maximum number of supported racks is six.

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

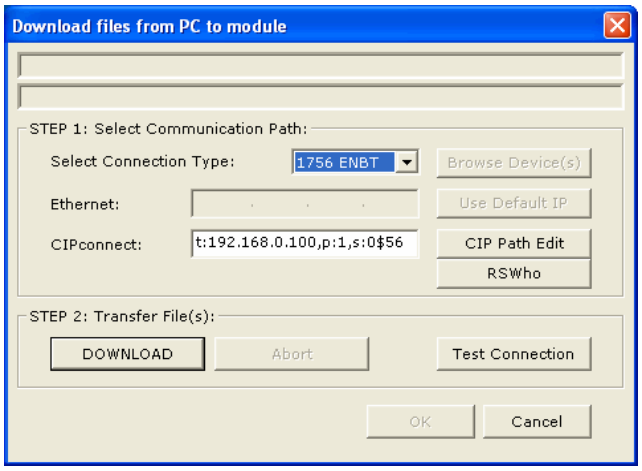
For this example, the MVI56E-MNETR module is located in the same rack as the 1756-ENBT that is connected to the PC.



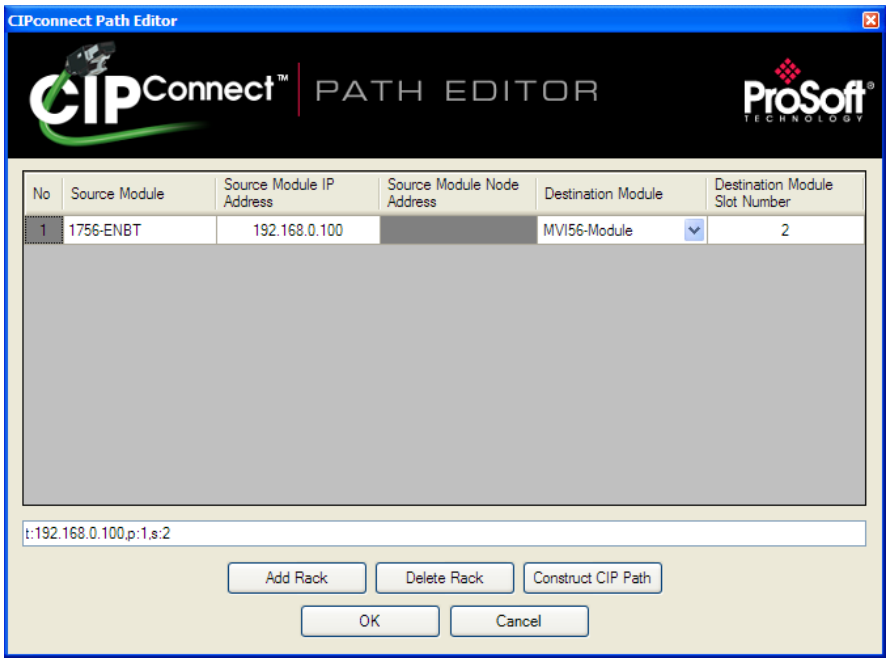
Rack 1

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

1 In the *Download* dialog box, click **CIP PATH EDIT.**

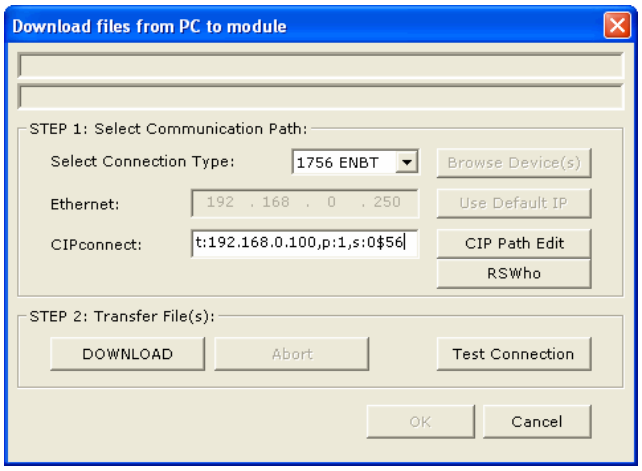


2 Configure the path as shown in the following illustration, and click **CONSTRUCT CIP PATH** to build the path in text format.

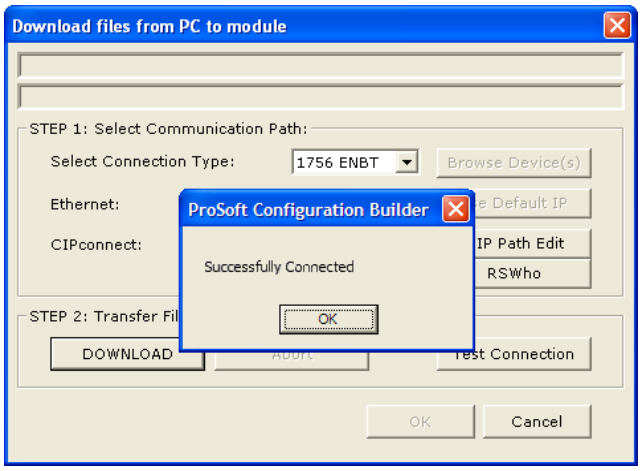


Click **OK** to close the *CIPconnect Path Editor* and return to the *Download* dialog box.

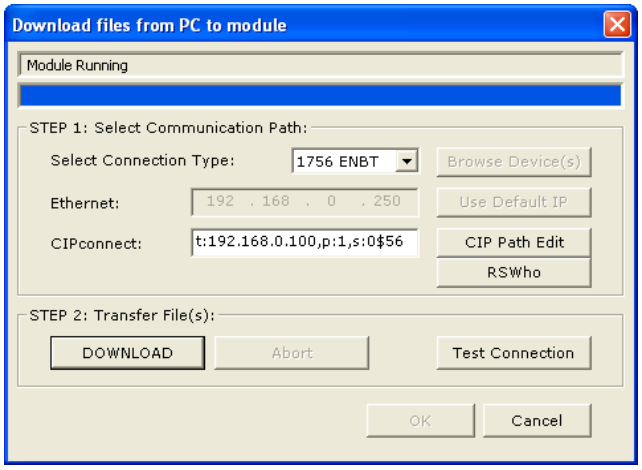
3 Check the new path in the *Download* dialog box.



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

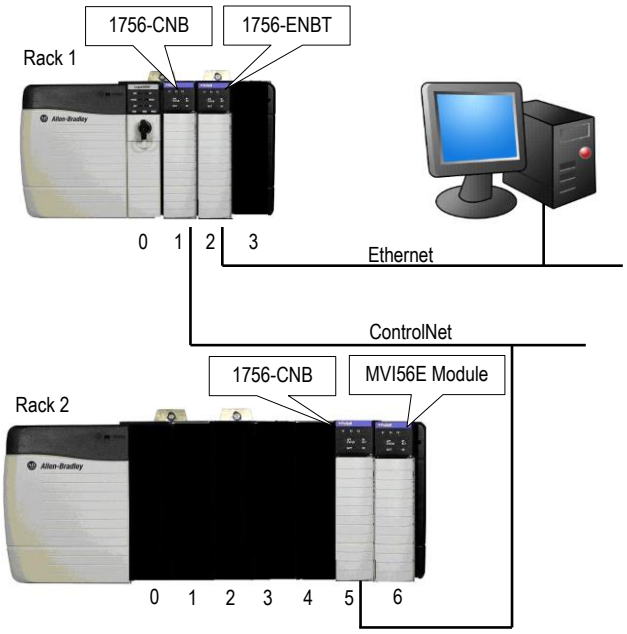


5 Click **OK** to close the Test Connection pop-up and then click **DOWNLOAD** to download the configuration files to the module through the path.



Example 2: Remote Rack Application

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



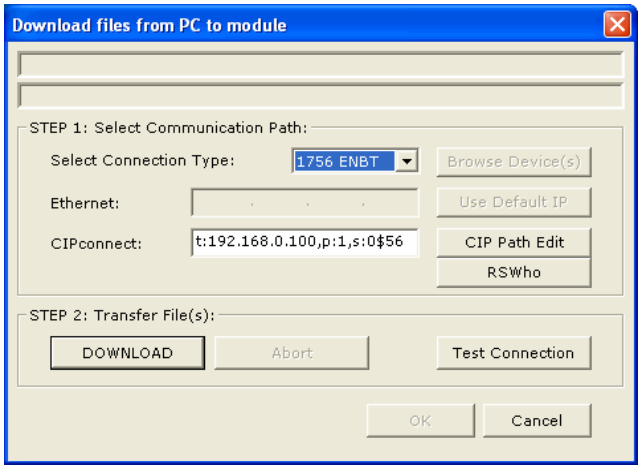
Rack 1

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

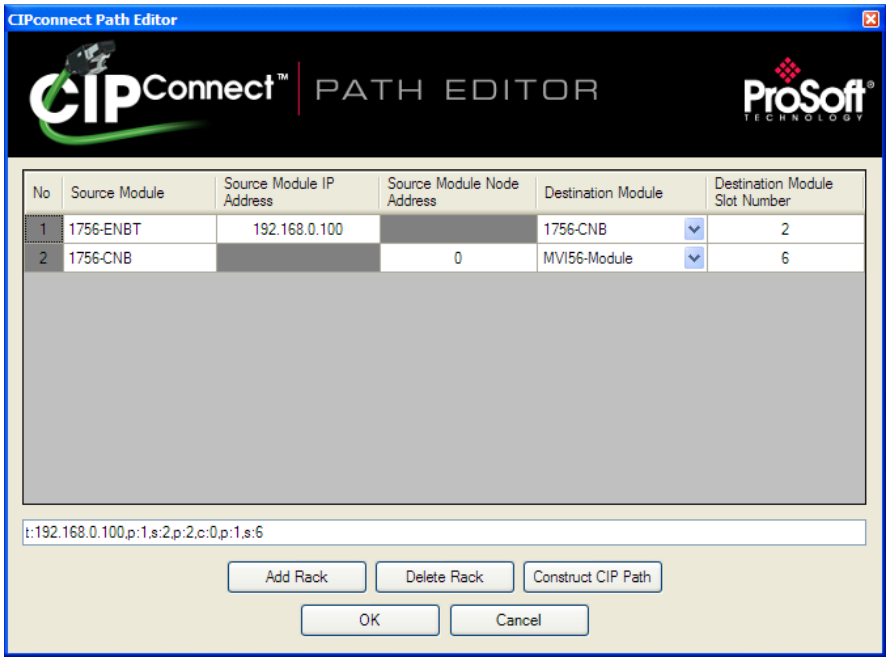
Rack 2

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

1 In the *Download* dialog box, click **CIP PATH EDIT**.

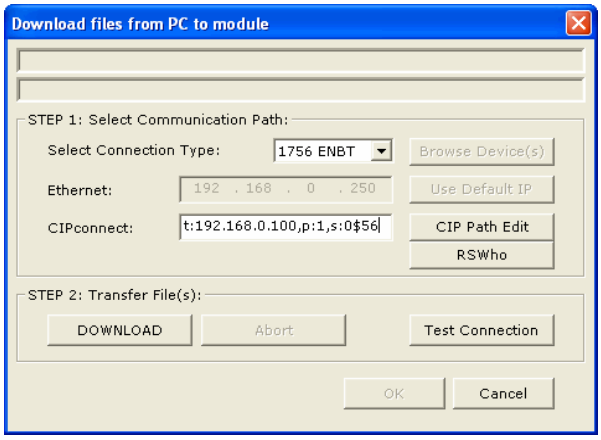


2 Configure the path as shown in the following illustration and click **CONSTRUCT CIP PATH** to build the path in text format.

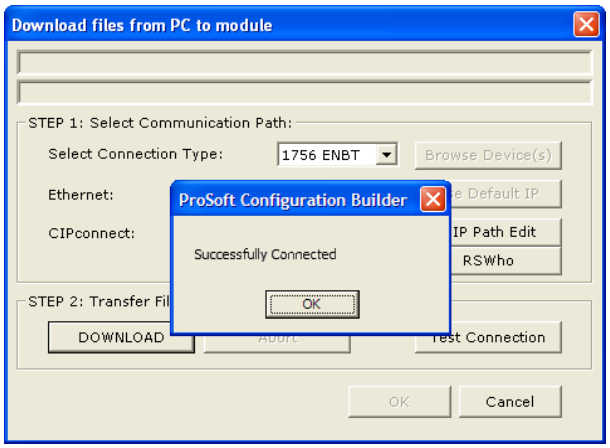


Click **OK** to close the *CIPconnect Path Editor* and return to the *Download* dialog box.

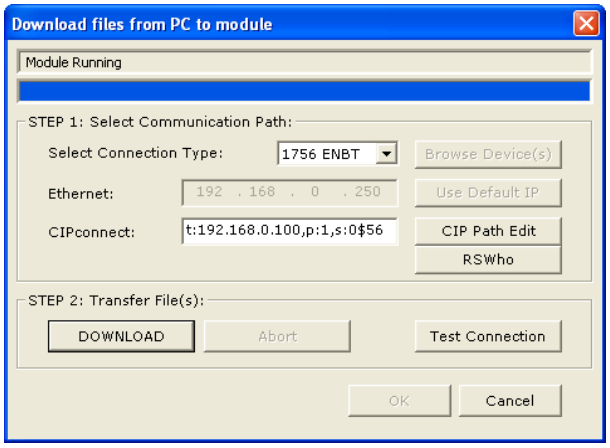
3 Check the new path in the *Download* dialog box.



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



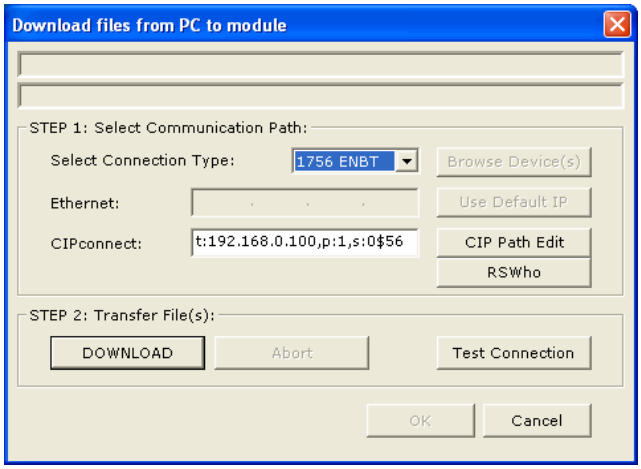
5 Click **DOWNLOAD** to download the configuration files to the module through the path.



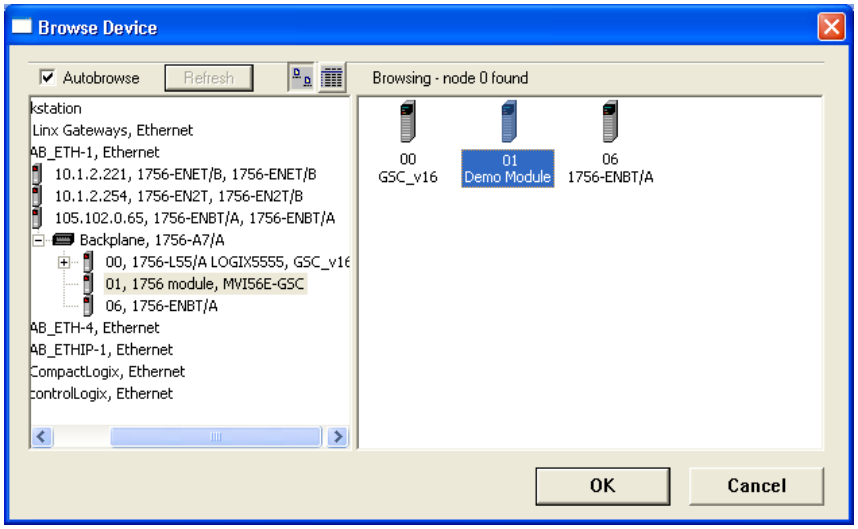
2.4.2 Using RSWho to Connect to the Module

Note: You need 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 56).

1. In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MNETR** 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-MNETR module will automatically be identified on the network.



5. Select the module, and then click **OK**.
6. In the *Download* dialog box, click **DOWNLOAD**.

3 Ladder Logic

Ladder logic is required for managing communication between the MVI56E-MNETR module and the processor. The ladder logic handles tasks such as:

- Module backplane data transfer
- Special block handling
- Status data receipt

Additionally, a power-up handler may be needed to initialize the module's database and may clear some processor fault conditions.

The sample Import Rung with 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 Import Rung with Add-On Instruction will work without modification.

3.1 Controller Tags

Data related to the MVI56E-MNETR is stored in the ladder logic in variables called controller tags. Individual controller tags can be grouped into collections of controller tags called controller tag structures. A controller tag structure can contain any combination of:

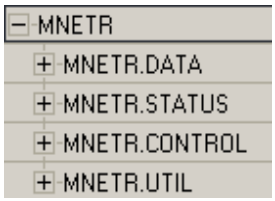
- Individual controller tags
- Controller tag arrays
- Lower-level controller tag structures

The controller tags for the module are pre-programmed into the Add-On Instruction Import Rung ladder logic. You can find them 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. Controller tag structures are declared with user-defined data types, which are collections of data types.

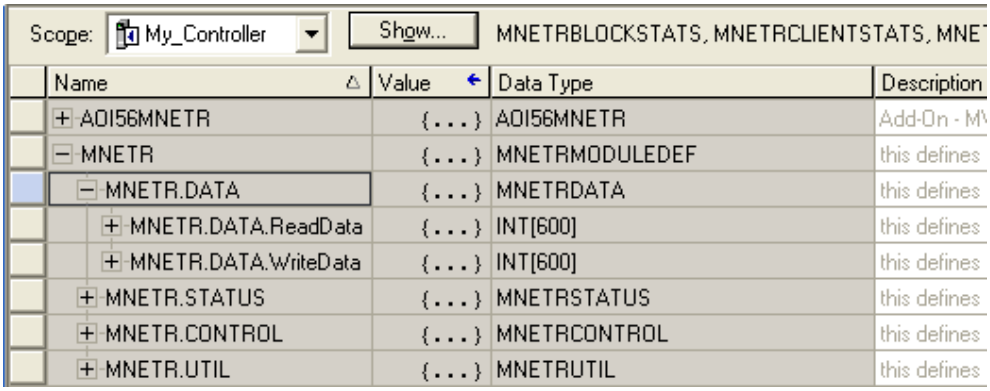
3.1.1 MVI56E-MNETR Controller Tags

The main controller tag structure, *MNETR*, 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 *MNETR.DATA* controller tag structure, you will see that it contains two controller tag arrays, *MNETR.DATA.ReadData* and *MNETR.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)

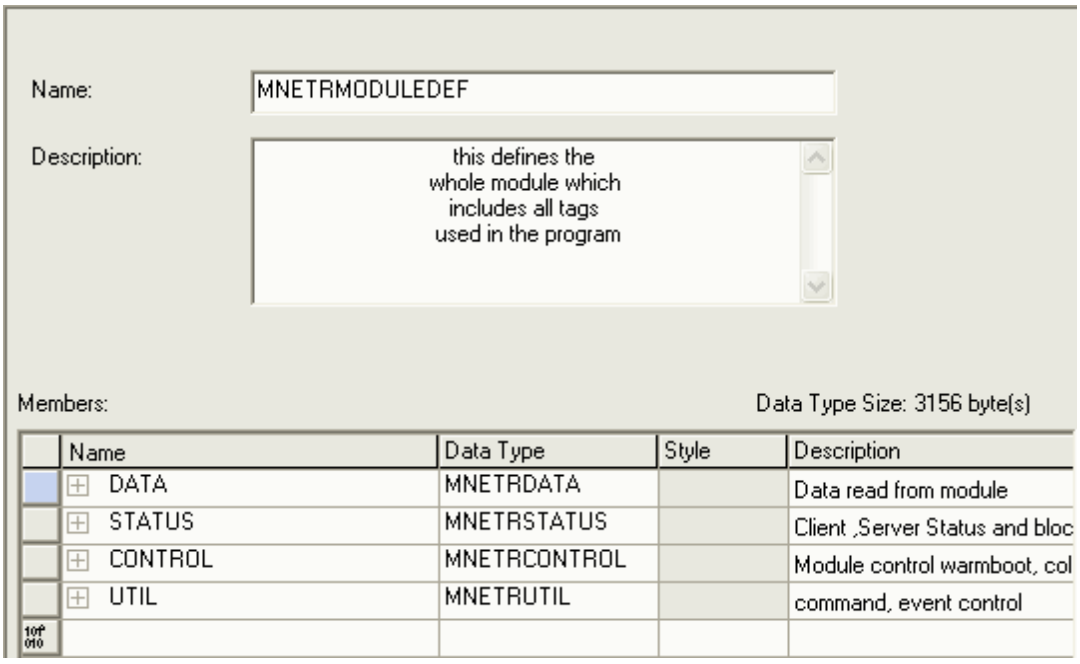
User-defined data types (UDTs) allow users to organize collections of data types into groupings. These groupings, or data type structures, can then be used to declare the data types for controller tag structures. Another advantage of defining a UDT is that it may be re-used in other controller tag structures that use the same data types.

The Add-On Instruction Import Rung ladder logic for the 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 MVI56(E)-MNETR User-Defined Data Types

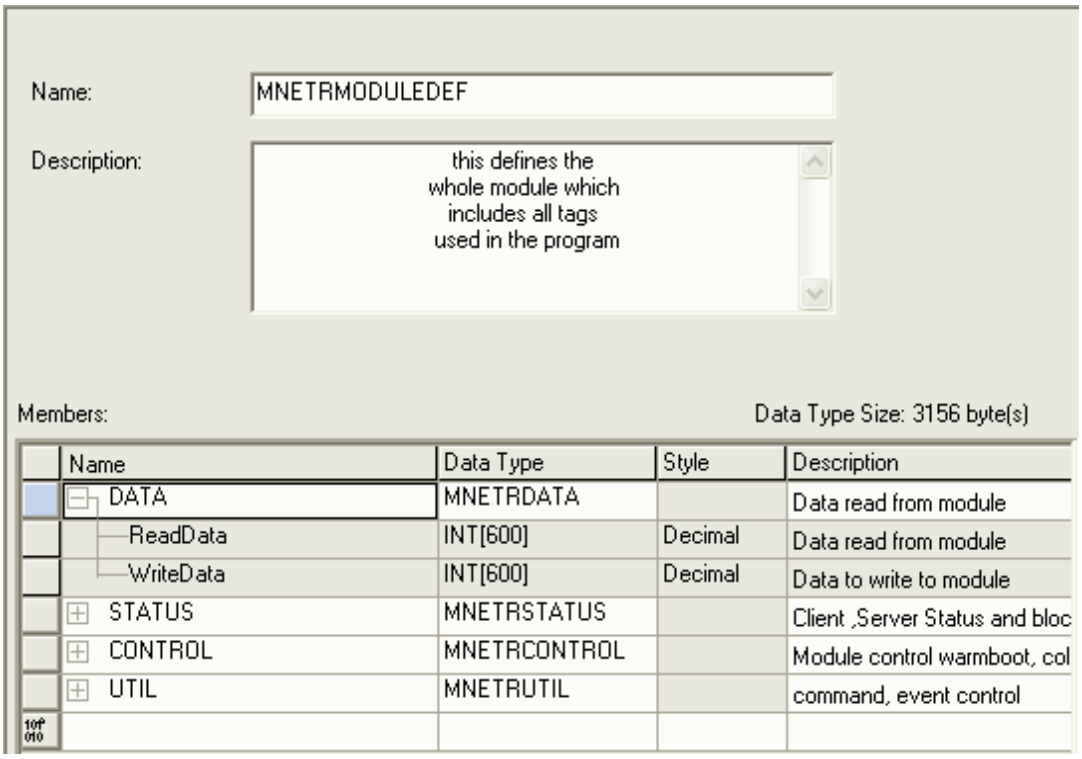
Twelve different UDTs are defined for the MVI56(E)-MNETR Add-On Instruction.

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



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

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



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

Each UDT is commented in the *Description* column.

3.3 Using Controller Tags

You can use controller tags to:

- View read and write data that is 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).

3.4 Controller Tag Overview

Controller Tag	Description
MNETR.DATA	Input and output data transferred between the processor and the module
MNETR.STATUS	Status information
MNETR.CONTROL	Governs the data movement between the PLC rack and the module
MNETR.UTIL	Generic tags used for internal ladder processing (DO NOT MODIFY)

The following sections describe each of these controller tag structures in more detail.

3.4.1 MNETR.DATA

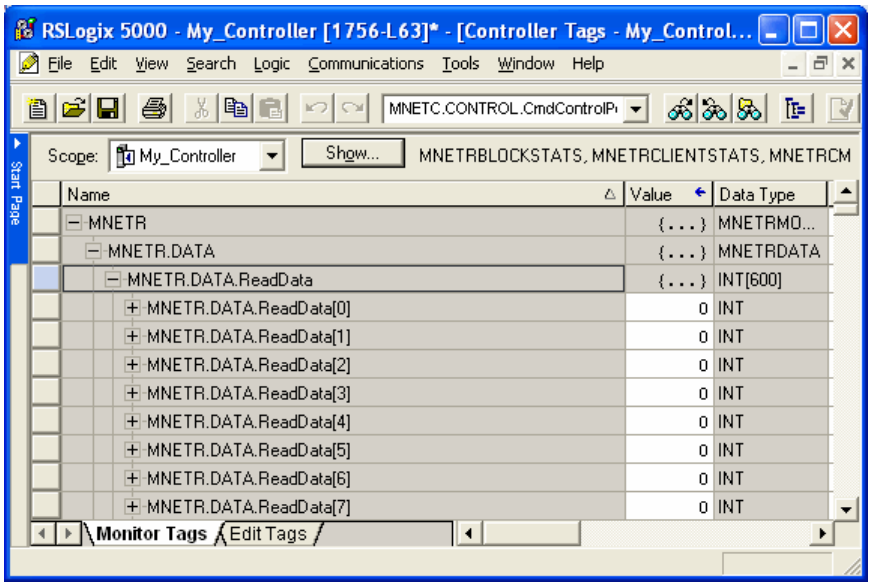
The controller tags in *MNETR.DATA* hold user data to be transferred between the processor and the MVI56E-MNETR module. This read and write data is transferred between the processor and the module as "pages," or blocks, of data up to 40 words long.

The data types for the *MNETR.DATA.ReadData* and *MNETR.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 Read Register Count set PCB.
WriteData	INT[x]	Data to write to module. Array size is equal to the *Write Register Count* set in PCB.

MNETR.DATA.ReadData

For ease of use, this array should be dimensioned as a multiple of 40 words. This data is paged up to 40 words at a time from the module to the processor. The ladder logic places the received data into the proper position in the *ReadData* array.



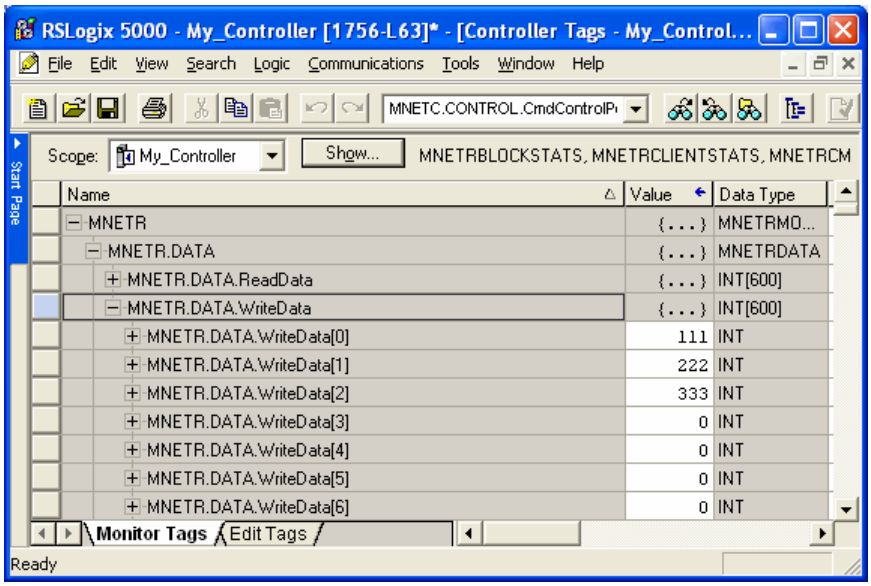
The *ReadData* array is related to the contents of the Read Data area of the module's internal database. To view the actual registers in the module's internal database, access the database display from *ProSoft Configuration Builder's Diagnostics* menu. For more information, see the section on *PCB* Diagnostics (page 82).

```
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
```

MNETR.DATA.WriteData

For ease of use, this array should be dimensioned as a multiple of 40 words. This data is paged up to 40 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.



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* Diagnostics (page 82).

```
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
```

3.4.2 MNETR.STATUS

Name	Data Type	Description
PassCnt	INT	Program cycle counter
ProductVersion	INT	Shows the module software version
ProductCode	INT[2]	Identifies the module product code
BlockStats	MNETRBLOCKSTATS	Status information for the data transfer operations between the processor and the module
Reserved1	INT	Reserved for future use
Reserved2	INT	Reserved for future use
MNetReq	INT	The number of MNET (Port 2000) requests received
MNetResp	INT	The number of MNET (Port 2000) responses sent
MBAPReq	INT	The number of MBAP (Port 502) requests received
MBAPResp	INT	The number of MBAP (Port 502) responses sent
ClientStatus	MNETRCLIENTSTATS	Client status data

For a more complete description of the *MNETR.STATUS* controller tag structure, refer to the Status Data Definition (page 87).

3.4.3 MNETR.CONTROL

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

Name	Data Type	Description
BootTimer	TIMER	Timer to clear Warm Boot and Cold Boot
WarmBoot	BOOL	Triggers a Warm Boot
ColdBoot	BOOL	Triggers a Cold Boot
EventCmdTrigger	BOOL	Triggers the Event Command.
EventCmd	MNETREVENTCMD	This object contains the values needed to build an Event Command. An array of these objects is used for each port.
CmdControl	MNETRCMDCONTROL	Controls the execution of the commands listed in the configuration's *MNET Client 0 Command List*.
PassThru	MNETRPASSTHRU	Transfers a remote Client's commands through the MNETR module straight into the processor's controller tags.
IPAddress	MNETRIPADDRESS	Getting and setting IP address to and from the module

3.4.4 MNETR.UTIL

Caution: These variables are for internal ladder usage only. Do not use these variables in your own application, otherwise unpredictable results could occur.

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
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. Array size is divided by 40.
WriteDataBlkCount	INT	Holds the value of the Block Counts of the *WriteData* array. Array size is divided by 40.
RBTSremainder	INT	Holds remainder calculation value from the *ReadData* array.
WBTSremainder	INT	Holds remainder calculation value from the WriteData array.
IPsetPending	BOOL	Allows setting module IP address
IPgetPending	BOOL	Allows getting module IP address
InitOutBlkIDLim	INT	Block Index Limit for *ReadData* size of the array

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 LED Status Indicators

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 Advanced Add-On Instruction.
C0 (Client): CmdCnt: X MinDly : X CmdOffs: X RespTmout : X Retries : X ErrOffs : X ARPTmout : X ErrDelay : X FltFlag : X FltSt : X FltOffs : X SVR (server) : BIOffs: X WIOffs : X OutOffs : X HoldOffs : X FltFlag : X FltSt : X FltSt : X CommTmout : X	After power up and every reconfiguration, the module will display the configuration of the application port(s). The information consists of: **Client** ▪ CmdCnt : number of commands configured for the Client ▪ MinDly : Minimum Command Delay parameter ▪ CmdOffs : Command Error Pointer parameter ▪ RespTmout : Response Timeout parameter ▪ Retries : Retry Count parameter ▪ ErrOffs : Error/Status Offset parameter ▪ ARPTmout : ARP Timeout parameter ▪ ErrDelay: Command Error Delay parameter ▪ FltFlag: Float Flag parameter ▪ Flt St : Float Start parameter ▪ FltOffs : Float Offset parameter **Server** ▪ BIOffs: Bit Input Offset parameter ▪ WIOffs : Word Input Offset parameter ▪ OutOffs : Output offset parameter ▪ HoldOffs : Holding Register offset parameter ▪ FltFlag: Float Flag parameter ▪ FltSt : Float Start parameter ▪ FltOffs : Float Offset parameter

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 74) for further information about the error.

4.1.2 Ethernet LED Indicators

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

LED	State	Description
10/100	Off	No activity on the Ethernet port.
	Green Flash	The Ethernet port is actively transmitting or receiving data.
LINK/ACT	Off	No physical network connection is detected. No Ethernet communication is possible. Check wiring and cables.
	Green Solid	Physical network connection detected. This LED must be On solid for Ethernet communication to be possible.

4.1.3 Non-Scrolling LED Status Indicators

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

LED Label	Color	Status	Indication
APP	Red or Green	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-MNETR 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	Red or Green	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.

4.1.4 Clearing a Fault Condition

Typically, if the OK LED on the front of the module turns RED for more than ten seconds, a hardware problem has been detected in the module 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 that the card 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.

4.1.5 Troubleshooting

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

Processor Errors

Problem Description	Steps to take
Processor Fault	Verify that the module is plugged into the slot that has been configured for the module in the I/O Configuration of RSLogix. Verify that 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-MNETR. Verify that all modules in the rack are correctly configured in the ladder logic.

Module Errors

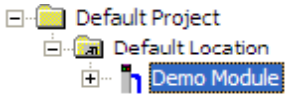
Problem Description	Steps to take
MVI56E modules with scrolling LED display: <*Backplane Status*> condition reads ERR	This indicates that backplane transfer operations are failing. Connect to the module's Configuration/Debug port to check this. To establish backplane communications, verify the following 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 Configuration/Debug 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.

4.2 Using the Diagnostics Menu in ProSoft Configuration Builder

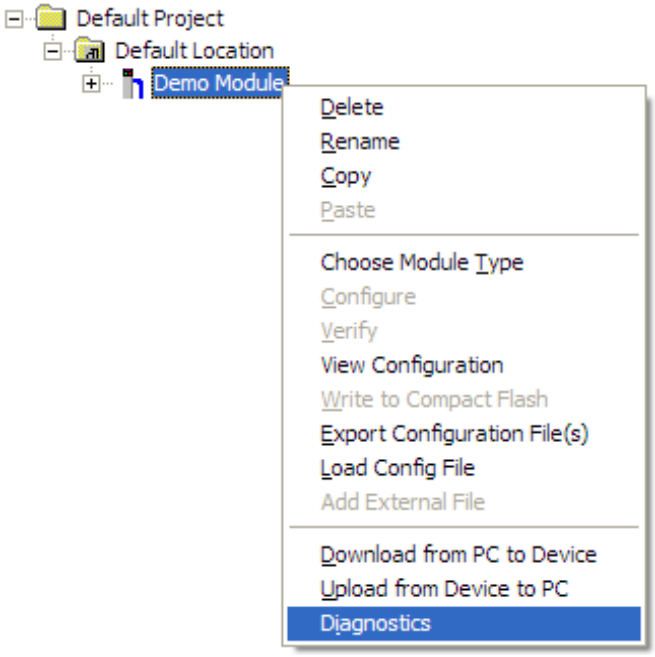
Tip: You can have a ProSoft Configuration Builder *Diagnostics* window open for more than one module at a time.

To connect to the module's Configuration/Debug Ethernet port:

1 In *ProSoft Configuration Builder*, select the module, and then click the right mouse button to open a shortcut menu.



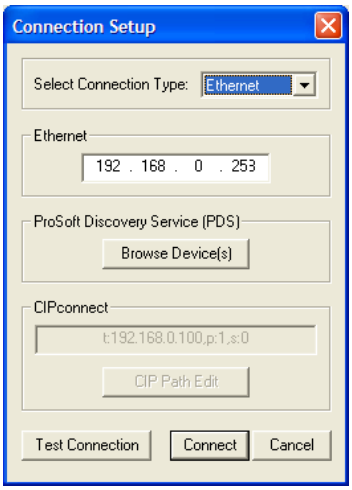
2 On the shortcut menu, choose **DIAGNOSTICS.**



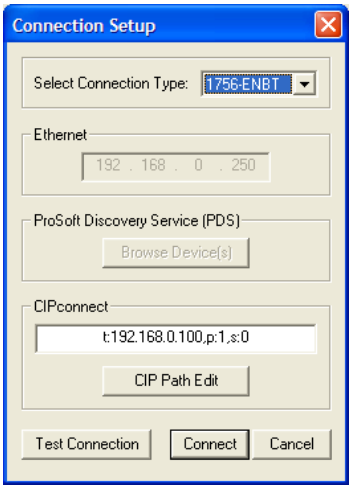
3 In the *Diagnostics* window, click the **SET UP CONNECTION** button to browse for the module's IP address.



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



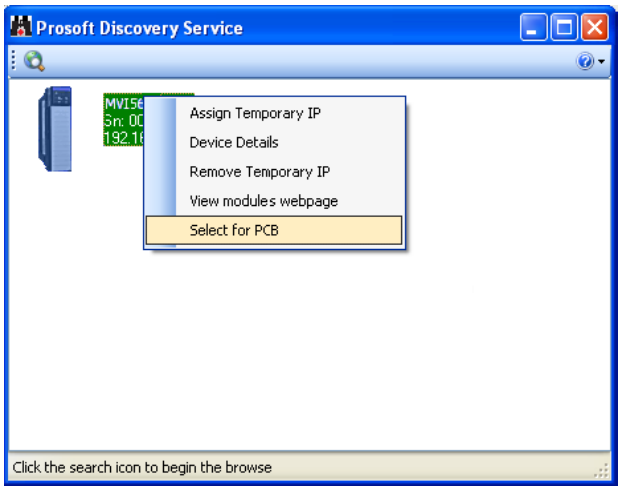
You can also use CIPconnect® to connect to the module through a 1756-ENBT card. Refer to Using CIPconnect to Connect to the Module (page 56) for information on how to construct a CIP path.



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.

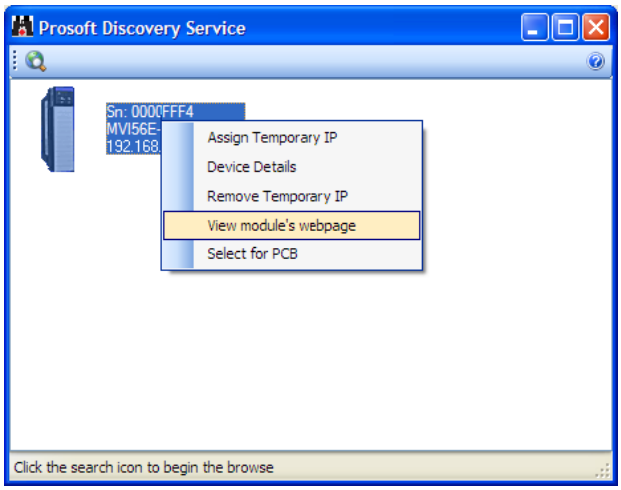
3 If these troubleshooting steps fail, verify that the Ethernet cable is connected properly between your computer and the module, either through a hub or switch (using the grey cable) or directly between your computer and the module (using the red cable).

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

4.2.1 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 *ProSoft Discovery Service*, select the module, and then click the right mouse button to open a shortcut menu.

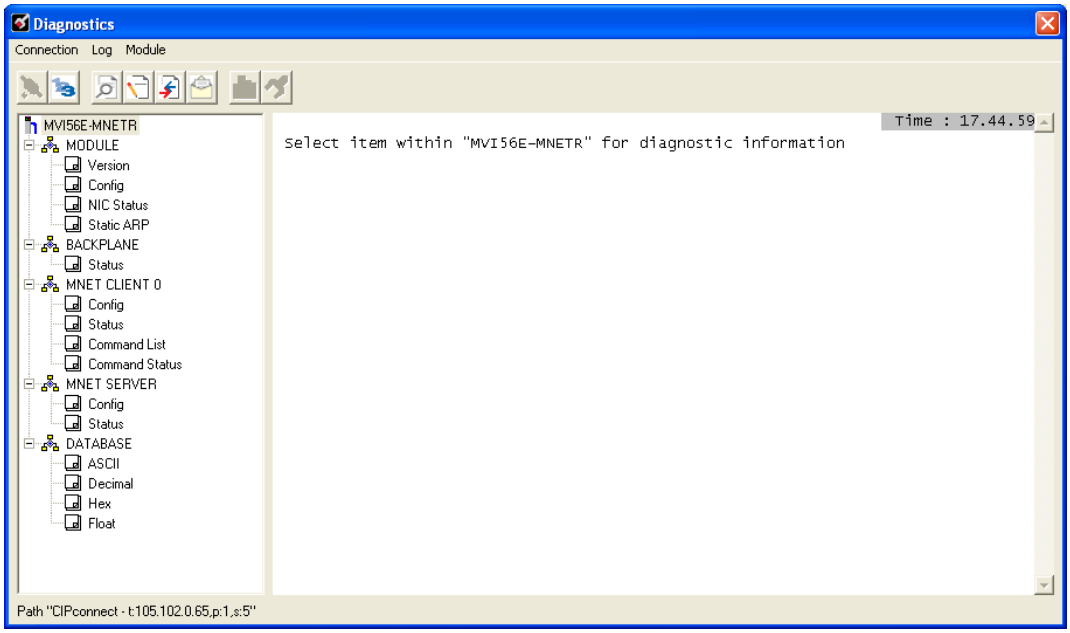


2 On the shortcut menu, choose **VIEW MODULE'S WEBPAGE**.



4.2.2 The Diagnostics Menu

The *Diagnostics* menu, available through the Ethernet configuration port for this module, is arranged as a tree structure, with the *Main* menu at the top of the tree, and one or more submenus for each menu command. The first menu you see when you connect to the module is the *Main* menu.

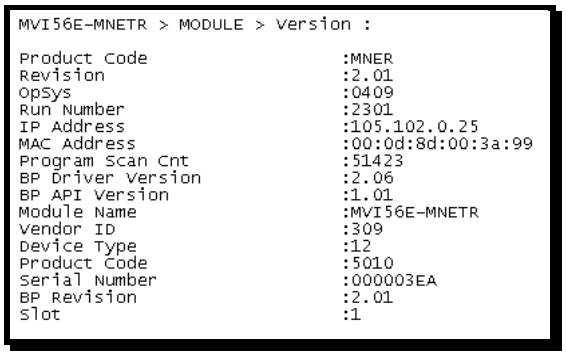


4.2.3 Monitoring Module Information

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

Version

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



Config

Use the *Configuration* menu to view backplane configuration settings for the MVI56E-MNETR 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-MNETR 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 48) defines a list of static IP addresses that the module will use when an ARP is required.

4.2.4 Monitoring Backplane Information

Use the *BACKPLANE* menu to view the backplane status information for the MVI56E-MNETR 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 (page 87).

4.2.5 Monitoring MNET Client Information

Use the *MNET CLIENT x* menu to view the configuration and status information for the MNET Client(s).

Config

Use the *Configuration* menu to view 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

Use the *Status* menu to view status for MNET Client x. During normal operation, the number of requests and responses should increment, while the number of errors should not change.

Command List

Use the *Command List* menu to view the command list settings for MNET Client x. 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

Use the *Command Status* menu to view MNET Client x Command status.

A zero indicates no error. A non-zero value indicates an error. For an explanation of each value, refer to Client Command Error (page 89).

4.2.6 Monitoring MNET Server Information

Use the *MNET SERVER* menu to view the configuration and status information for the MNET server.

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* (page 46).

Status

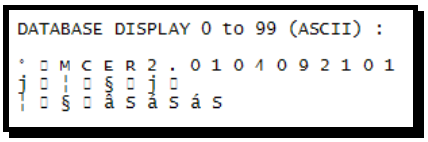
Use the *Status* menu to view the status of each MNET server connected to the MNET Client 0. During normal operation, the number of requests and responses should increment, while the number of errors should not change.

4.2.7 Monitoring Database Information

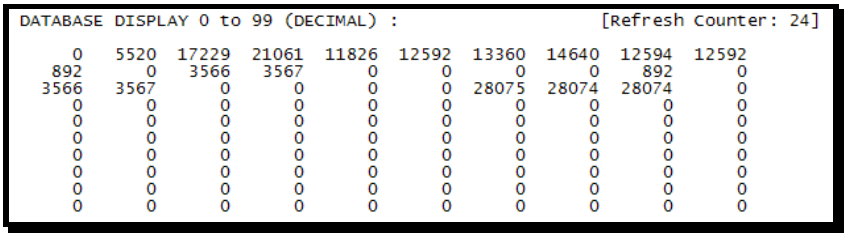
Use the *DATABASE* menu to view the contents of the MVI56E-MNETR module's internal database.

You can view data in the following formats:

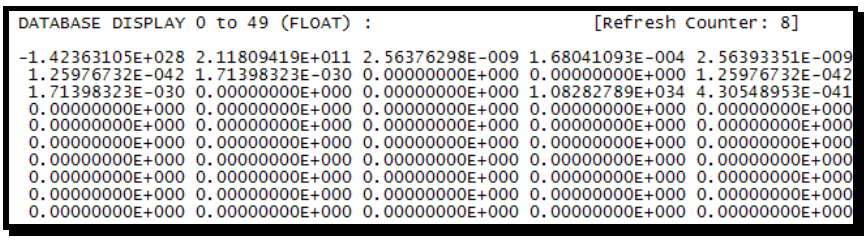
ASCII



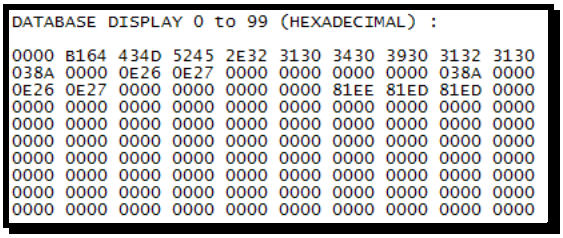
Decimal



Float



Hexadecimal



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

4.3 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 *MNETR.STATUS* controller tag structure.**

  The MVI56E-MNETR 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 block sequence (page 96). You can view this data in the *MNETR.STATUS* controller tag structure in the ladder logic. For more information, see the Status Data Definition (page 87).

- **In *ProSoft Configuration Builder's Diagnostics* screens.**

  For more information, see the section on PCB Diagnostics (page 82).

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

  If optional *Error/Status Pointers* are enabled, the 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 33), Client Error/Status Pointer (page 37) and Command Error Pointer (page 37).

4.3.1 Status Data Definition

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

Controller Tag	Description
Pass Count	This value is incremented each time a complete program cycle occurs in the module.
Product Version	Shows the module software version.
Product Code	Identifies the module product code.
Read Block Count	This field contains the total number of Read blocks transferred from the module to the processor.
Write Block Count	This field contains the total number of Write blocks transferred from the processor to the module.
Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
Command Event Block Count	This field contains the total number of Event Command blocks received from the processor.
Command Block Count	This field contains the total number of command blocks received from the processor.
Error Block Count	This field contains the total number of block errors recognized by the module.
Reserved1	Not used
Reserved2	Not used
MNet Request Count	This counter increments each time a MNet (port 2000) request is received.
MNet Response Count	This counter is incremented each time a MNet (port 2000) response message is sent.
MBAP Request Count	This counter increments each time a MBAP (port 502) request is received.
MBAP Response Count	This counter is incremented each time a MBAP (port 502) response message is sent.
Client Cmd Request	This value is incremented each time a command request is issued.
Client Cmd Response	This value is incremented each time a command response is received.
Client Cmd Error	This value is incremented each time an error message is received from a remote unit or a local error is generated for a command.
Client Request Count	Not used
Client Response Count	Not used
Client Error Sent Count	Not used
Client Error Received Count	Not used
Client Cfg Error Word	This word contains a bitmap that defines configuration errors in the configuration file for the Client.
Client Current Error Code	This value corresponds to the current error code for the Client.
Client Last Error Code	This value corresponds to the last error code recorded for the Client.

4.3.2 Client 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* is a controller tag in *MNETR.STATUS.ClientStatus* in the ladder logic.

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

Bit	Description	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.

4.3.3 Client Command Errors

There are several different ways to view Client Command Errors.

- In the *MNETR.STATUS.ClientStats.CurErr* and *MNETR.STATUS.ClientStats.LastErr* controller tags
- On the Client status data screens in the *ProSoft Configuration Builder Diagnostics*
- At a module database location specified by the configuration's *MNET Client 0 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
1	Command 1 Error
2	Command 2 Error
3	Command 3 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 (same as -11)
-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	ARP could not resolve MAC from IP (bad IP address, not part of a 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 user-adjustable ARP 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.

5 Reference

5.1 Product Specifications

The Modbus TCP/IP Client/Server Enhanced Communication Module with Reduced Data Block allows Rockwell Automation® ControlLogix® processors to interface easily with Modbus TCP/IP-compatible devices, such as Modicon Programmable Automation Controllers (PACs) and a wide variety of Modbus TCP/IP-compatible instruments and devices.

The MVI56E-MNETR uses a reduced Input/Output (I/O) data image block size for transferring data between itself and a ControlLogix processor. This makes it ideal for remote rack applications using ControlNet™ or EtherNet/IP™ process networks. The module also works well for applications that require redundant ControlLogix processors.

MVI56E-MNETR enhancements include local and remote configuration and diagnostics through the module's Ethernet port, and CIPconnect® technology for bridging through Rockwell Automation ControlNet and EtherNet/IP networks.

5.1.1 General Specifications

- Reduced I/O image size designed specifically to optimize remote rack implementations
- Backward compatible with previous MVI56-MNETR versions
- Single-slot 1756 ControlLogix backplane-compatible
- 10/100 Mbps auto crossover detection Ethernet port for data transfer and configuration
- User-definable module data memory mapping of up to 5000 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
- Four-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, allowing quick in-the-field product replacement by transferring the CF card

5.1.2 Functional Specifications

- The MVI56E-MNETR transfers data in smaller I/O blocks than the MVI56E-MNET, which makes it ideal for installations in remote racks or where bandwidth is limited.
- Works well with redundant ControlLogix Programmable Automation Controllers (PACs) using ControlNet
- Module appears to the ControlLogix processor as an input/output (I/O) module
- 40-word scheduled I/O image blocks used for data transfers allow module to use significantly less bandwidth than the MVI56E-MNET
- Retrieving module status and executing special functions (Event Commands, Command Control, etc.) are supported in ladder logic by special block transfer codes

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)
Storage temperature	-40°C to 85°C (-40°F to 185°F)
Shock	30 g operational 50 g non-operational
Vibration	5 g from 10 Hz 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 Module Power Up

On power up, the module begins performing the following logical functions:

1. Initialize hardware components
   - Initialize ControlLogix backplane driver
   - Test and clear all RAM
   - Read configuration for module from MNETR.CFG file on Compact Flash Disk
2. Initialize module register space
3. Enable server drivers
4. Enable Client driver

When the module has received the configuration, the module will begin communicating with other nodes on the network, depending on the configuration.

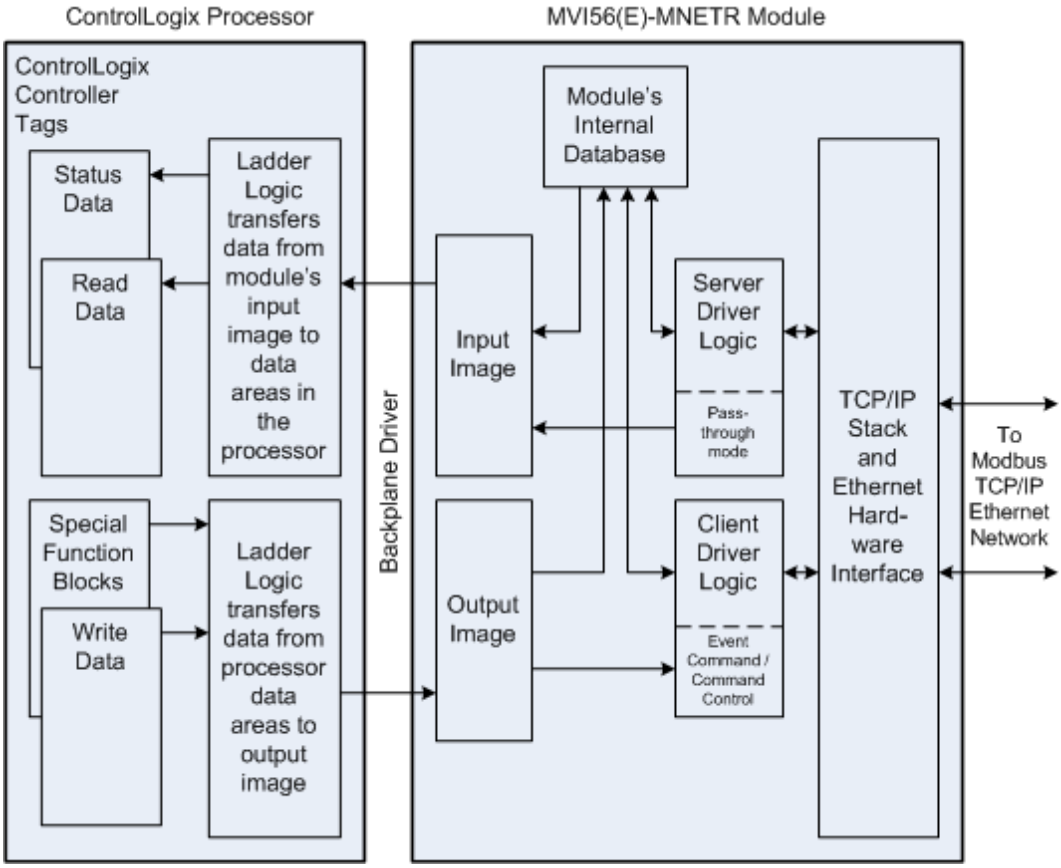
5.3 Backplane Data Transfer

The MVI56E-MNETR 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 transference 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 42 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 42 words.

The following illustration shows the data transfer method used to move data between the ControlLogix processor, the MVI56E-MNETR 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 4999 (45000 Modbus).

5.3.1 Module's Internal Database Structure



Data contained in this database is transferred in blocks, or pages, using the input and output images. ControlLogix ladder logic and the MVI56E-MNETR module's program work together to coordinate these block transfers. Up to 40 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 the data transfer function to be performed by the block. The module uses the following block identification numbers.

Block Range	Descriptions
-1	Status block
0	Status block
1 to 125	Read or Write blocks
1000 to 1124	Initialize Output Data blocks
2000	Event Command block
5001 to 5006	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
9996	Unformatted Pass-through block with raw Modbus message
9998	Warm-boot block
9999	Cold-boot block

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

Normal data transfer blocks

- Status, Read and Write blocks (-1 to 125)

Special function blocks

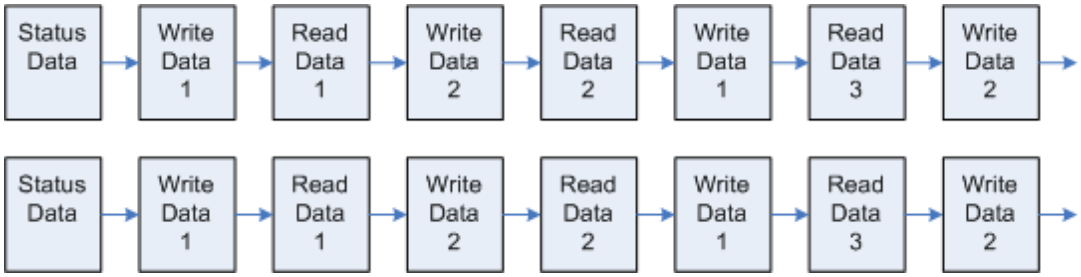
- Initialize Output Data blocks (1000 to 1124)
- Event Command block (2000)
- Command Control blocks (5001 to 5006)
- Pass-through blocks (9956 to 9961, 9970 and 9996)
- Module IP Address blocks (9990 and 9991)
- Warm-boot and Cold-boot blocks (9998 and 9999)

5.3.2 Normal Data Transfer Blocks

Normal data transfer includes the paging of user data between the processor's data areas and the module's internal database (registers 0 to 4999), as well as the paging of status data. These data are transferred through Read (input image), Write (output image) and Status blocks. The data is paged 40 words at a time.

During normal program operation, the module sequentially sends Read and Status Data blocks and receives Write blocks. The Status block is first in the sequence, followed by alternating Write and Read blocks.

As an example, assume that an application's Read Data area consists of 120 words and its Write Data area consists of 80 words. Since the Read and Write data is paged 40 words at a time, the module will use 3 Read blocks and 2 Write blocks to transfer the data. The Read, Write and Status blocks will be sequenced as follows.



This sequence will continue until interrupted by other special function blocks sent by the processor, by a command request from a node on the Modbus network, or by operator control through the module's Configuration/Debug port.

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	Write Block ID	1
1 to 40	Read Data	40
41	Read Block ID	1

The Read Block ID is an index value used to determine where the 40 words of data from module memory will be placed in the *ReadData[x]* controller tag array of the ControlLogix. Each transfer can move up to 40 words (block offsets 1 to 40) of data.

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
1 to 40	Write Data	40
41	Spare	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 40 words (block offsets 1 to 40) of data.

Status Block

The Status block is automatically copied from the module into the *MNETR.STATUS* controller tag structure when the block ID is 0 or -1, and contains status information about the module. For more information, see the Status Data Definition (page 87).

Status Block from Module to Processor

Word Offset	Description	Length
0	Write Block ID	1
1	Program Scan Counter	1
2 to 7	Block Transfer Status	6
8 to 9	Reserved Server Status	2
10 to 11	MNET Server Status	2
12 to 13	MBAP Server Status	2
14 to 23	MNET Client Status	10
24 to 25	Product Name	2
26	Product Version	1
27 to 40	Reserved	14
41	Read Block ID (-1 or 0)	1

Client Status Data

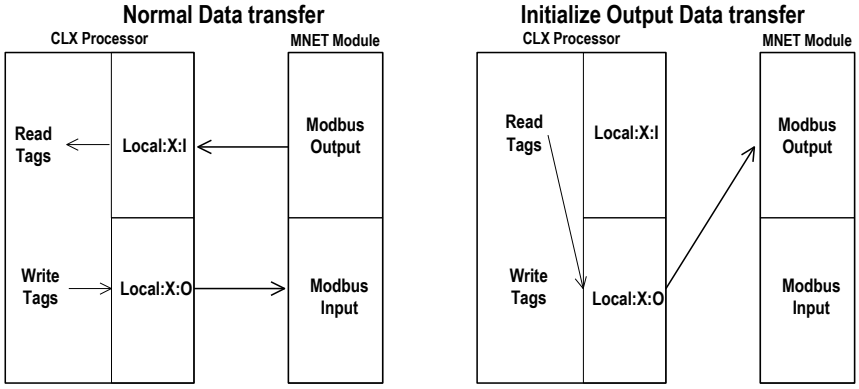
Word Offset	Client Status
14	Total number of command list requests
15	Total number of command list responses
16	Total number of command list errors
17	Not used
18	Not used
19	Not used
20	Not used
21	Configuration Error Word
22	Current Error
23	Last Error

5.3.3 Special Function Blocks

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

Initialize Output Data Blocks (1000 to 1124)

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	1000 to 1124	1
1 to 40	Spare	40
41	1000 to 1124	1

The block number in word 0 of the block determines the data set of up to 40 output words to transfer from the processor. Ladder logic in the processor must recognize these blocks and place the correct information in the output image to be returned to the module.

Block Response from Processor to Module

Word Offset	Description	Length
0	1000 to 1124	1
1 to 40	Output Data to preset in module.	40
41	Spare	1

Event Command Block (2000)

Note: Event Commands are not needed for normal Modbus command list polling operations and are needed only occasionally for special circumstances.

During routine operation, the module continuously cycles through the user-defined *MNET Client 0 Command List (page 39),* 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 0 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 0 Command List*.

Block Request from Processor to Module

Word Offset	Description
0	*Block ID* - This word contains the block 2000 identification code to indicate that the block contains a command to execute by the Client driver.
1 to 4	*IP Address* - These words contain the IP address for the server the message is intended for. Each digit (**0** to **255**) of the IP address is placed in one of the four registers. For example, to reach IP address 192.168.0.100, enter the following values in words 1 to 4: **192**, **168**, **0** and **100**. The module will construct the normal dotted IP address from the values entered. The values entered will be added with the mask 0x00ff to ensure the values are in the range of 0 to 255.
5	*Service Port* - This word contains the TCP service port the message will be interfaced with. For example, to interface with a MBAP device, the word should contain a value of **502**. To interface with a MNET device, a value of **2000** should be utilized. Any value from **0** to **65535** is permitted. A value of **502** will cause a MBAP formatted message to be generated. All other values will generate an encapsulated Modbus message.
6	*Slave Address* - This word contains the Modbus node address for the message. This field should have a value from **0** to **41**.
7	*Internal DB Address* - This word contains the internal Modbus address in the module to use with the command. This word can contain a value from **0** to **4999**.
8	*Point Count* - This word contains the count parameter that determines the number of digital points or registers to associate with the command.
9	*Swap Code* - The parameter specifies the swap type for the data. This function is only valid for function codes 3 and 4.
10	*Modbus Function Code* - This word contains the Modbus function code for the command.
11	*Device Database Address* - This word contains the Modbus address in the server device to be associated with the command.
12 to 41	Spare

When the module receives this request block, it builds the command, places the command in the command priority queue (if the queue is not already full; maximum capacity is 100 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	Write Block ID	1
1	0=Fail, 1=Success	1
2 to 40	Spare	39
41	2000	1

Word 2 of the block can be used by the ladder logic to determine whether or not the command was successfully added to the command priority queue. The command will fail if the queue for the port is already full at the time when the Event Command block is received by the module.

Controller Tags

The elements of the *MNETR.CONTROL.EventCmd* controller tag array contain all the values needed to build one Modbus TCP/IP command, have it sent to the module, and control the processing of the returned response block.

Name	Data Type	Description
IP0	INT	First digit of IP address
IP1	INT	Second digit of IP address
IP2	INT	Third digit of IP address
IP3	INT	Last digit of IP address
ServPort	INT	TCP Service Port number (0-65535), 502 for MBAP, 2000 for MNET
Node	INT	Modbus slave node address (0 to 247)
DBAddress	INT	Module internal database for message
Count	INT	Register or data point count
Swap	INT	Swap code for functions 3 and 4
Function	INT	Modbus function code for message
Address	INT	Address to interface with in device
Result	INT	Shows the result of the event that was sent

Command Control Blocks (5001 to 5006)

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

During routine operation, the module continuously cycles through the user-defined *MNET Client 0 Command List (page 39),* 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 0 Command List* and continues routine polling.

Like Event Command blocks, Command Control blocks place commands into the module's command priority queue. Unlike Event Commands blocks, which contain all the values needed for one command, Command Control is only used with commands already defined in the *MNET Client 0 Command List*.

Commands in the *MNET Client 0 Command List* may be either enabled for routine polling or disabled and excluded from routine polling. A disabled command has its *Enable* parameter set to **NO** (0) and is skipped during routine polling. An enabled command has its *Enable* parameter set to **YES** (1) and is sent during routine polling. However, Command Control allows any command in the predefined *MNET Client 0 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 *MNET Client 0 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 six commands from the *MNET Client 0 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 **5006**. The rightmost digit indicates the number of commands (**1** to **6**) to add to the command priority queue.	1
1	This word contains the Command Index for the first command to be entered into the queue.	1
2	This word contains the Command Index for the second command to be entered into the queue.	1
3	This word contains the Command Index for the third command to be entered into the queue.	1
4	This word contains the Command Index for the fourth command to be entered into the queue.	1
5	This word contains the Command Index for the fifth command to be entered into the queue.	
6	This word contains the Command Index for the sixth command to be entered into the queue.	1
7 to 41	Spare	35

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 six 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 six Command Indexes range from **0** to **99** and correspond to the *MNET Client 0 Command List* entries, which are numbered from 1 to 100. To determine the Command Index value, subtract one (**1**) from the row number of the command in the *MNET Client 0 Command List*, as seen in the *Command Editor* window of *ProSoft Configuration Builder (PCB)*.

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

Word Offset	Description	Length
0	Write Block ID	1
1	Number of commands added to command priority queue	1
2 to 40	Spare	39
41	This word contains the Block ID 5001 to 5006 requested by the processor.	1

Controller Tags

The *MNETR.CONTROL.CmdControl* 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.

Name	Data Type	Description
CmdIndex	INT[6]	Enter the **ROW NUMBER** of the command in the *MNET Client 0 Command List* in *Prosoft Configuration Builder* minus **1**. This is a six-element array. Each element holds one Command Index.and to execute from the Client command list.
NumberOfCommands	INT	Enter a decimal value representing the quantity of commands to be requested in the Command Control block (**1** to **6**).
CommandsAddedtoQueue	INT	Returned decimal value representing the quantity of commands added from the *MNET Client 0 Command List* to the command priority queue by the most recent Command Control block.
CmdControlTrigger	BOOL	Set this tag to **1** to trigger the execution of a command 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

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 and 9961 from Module to Processor

Word Offset	Description	Length
0	9956, 9957, 9958, 9960 or 9961	1
1	Number of word registers in Modbus data set	1
2	Starting address for Modbus data set	1
3 to 40	Modbus data set	38
41	9956, 9957, 9958, 9960 or 9961	1

Pass-Through Block 9959 from Module to Processor

Word Offset	Description	Length
0	9959	1
1	Number of word registers in Modbus data set	1
2	Starting word address for Modbus data set	1
3 to 21	Modbus data set	19
22 to 40	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.	19
41	9959	1

Pass-Through Block 9970 from Module to Processor

Word Offset	Description	Length
0	9970	1
1	1	1
2	0	1
3 to 40	Spare data area	38
41	9996	1

The ladder logic should copy and parse the received message and control 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 41	Spare data area	41

Unformatted Pass-Through Blocks

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	9996	1
1	Number of bytes in Modbus msg	1
2	Reserved (always 0)	1
3 to 40	Modbus message received	38
41	9996	1

The ladder logic should copy and parse the received message and control 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 41	Spare	41

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 41	Reserved	36

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 41	Spare data area	35

Get Module IP Address Block (9991)

Block Request from Processor to Module

Word Offset	Description	Length
0	9991	1
1 to 41	Spare data area	40

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 41	Spare data area	35

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

Word Offset	Description	Length
0	9998	1
1 to 41	Spare	41

Cold Boot Block (9999)

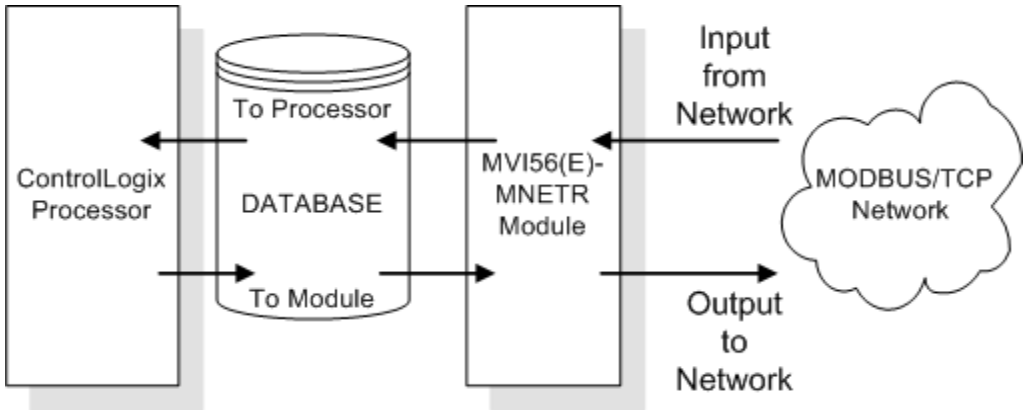
This block is sent from the ControlLogix processor to the module (output image) when the module is required to perform the cold boot (hardware reset) operation. This block is sent to the module when a hardware problem is detected by the ladder logic that requires a hardware reset.

Block Request from Processor to Module

Word Offset	Description	Length
0	9999	1
1 to 41	Spare	41

5.4 Data Flow between MVI56E-MNETR and ControlLogix Processor

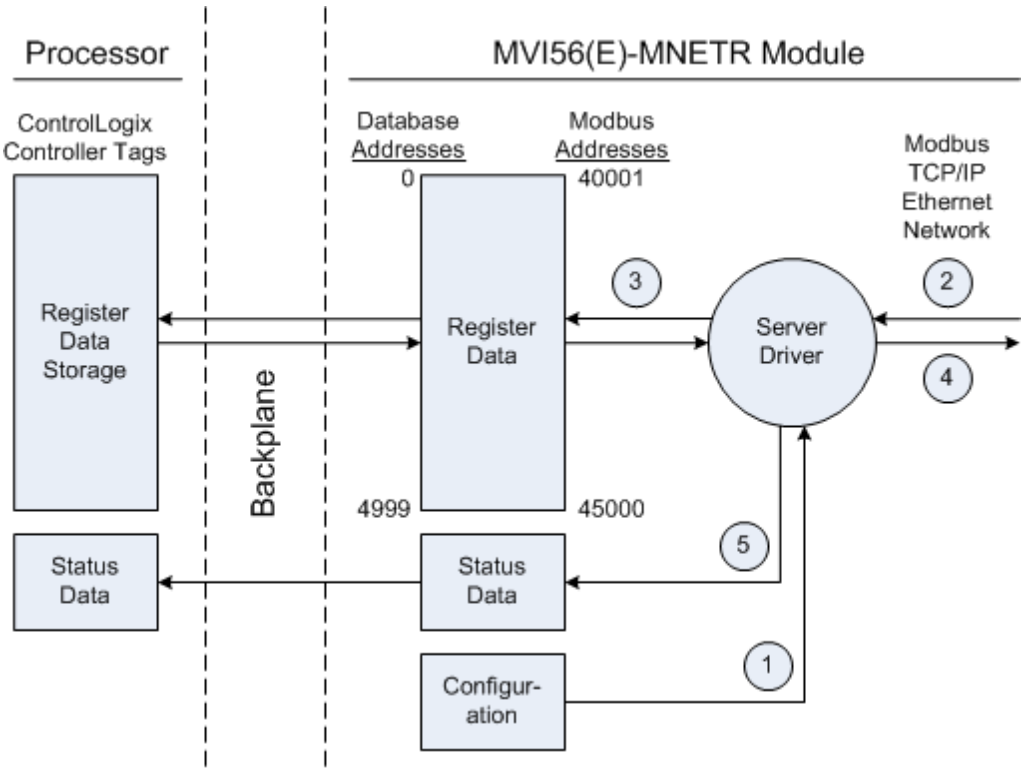
The following topics describe the flow of data between the two pieces of hardware (ControlLogix processor and MVI56E-MNETR module) and other nodes on the Modbus TCP/IP network under the module's different operating modes. The module has both server and Client capability. The server accepts TCP/IP connections on service ports 502 (MBAP) (10 server connections) and 2000 (MNET) (10 server connections). The Client can generate either MBAP or MNET requests dependent on the service port selected in the command.



The following topics discuss the operation of the server and Client drivers.

5.4.1 Server Driver

The server driver allows the MVI56E-MNETR 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 Client 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.

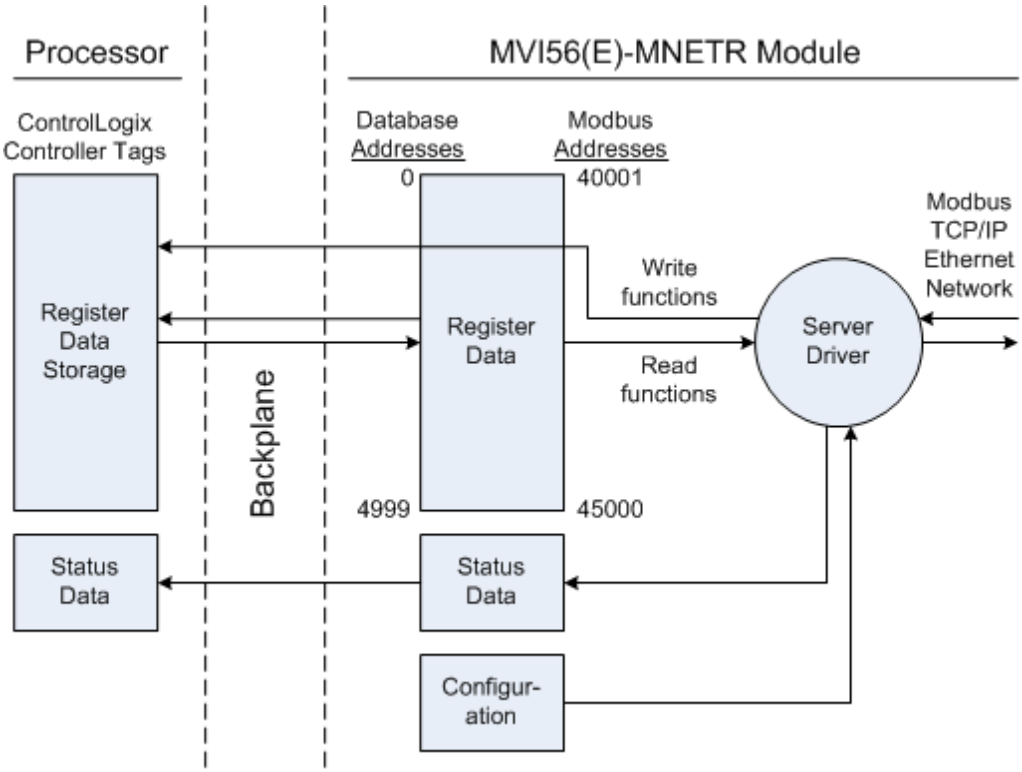
If the command is a read command, the data is read out of the database and a response message is built.
If the command is a write command, the data is written directly into the database and a response message is built.
If the command is a write command and the pass-through feature is utilized, the write 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 When the data processing has been completed in Step 3, the response is issued to the originating Client node.

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

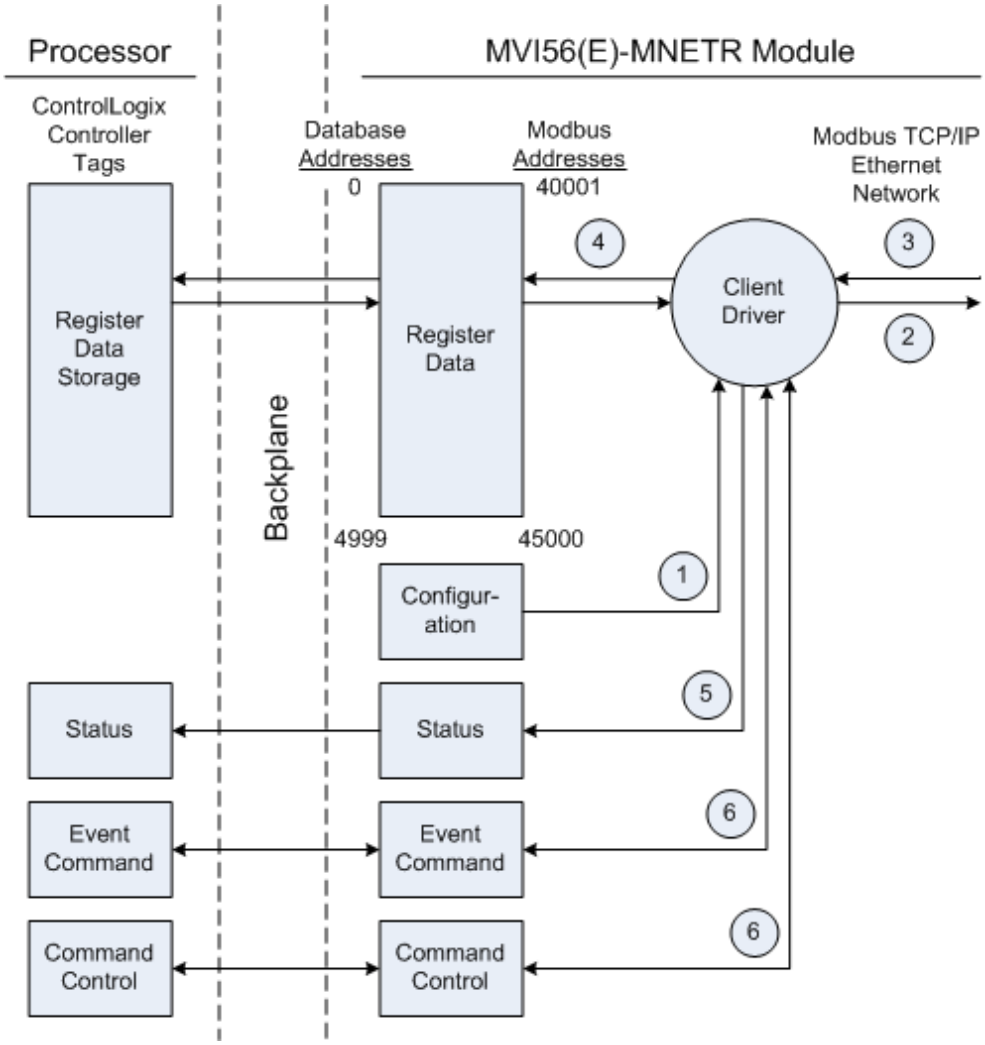
After the server socket is open, it must receive messages within a one minute period, or else it will close the socket. After closing, the socket will be reused.

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.



5.4.2 Client Driver

In the Client driver, the MVI56E-MNETR module issues read or write commands to servers on the Modbus TCP/IP network. The commands originate either from the module's user-configured *Client 0 Command List*, or directly from the processor as Event Commands. The commands from the *Client 0 Command List* are executed either via routine polling or as a result of special Command Control block requests from the processor. 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 type of commands to be issued to the other nodes on the Modbus TCP/IP network.

2 When configured, the Client driver begins transmitting read and/or write commands to the other nodes on the network. The data for write commands is obtained from the module's internal database.

3 Assuming successful processing by the node 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.
5 Status information is routinely sent to the processor in a special Status block among with the Read data blocks in the normal data transfer sequence.
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 *MNET Client x Command List* must be defined in the configuration. This list contains up to 100 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
  - (2) conditional
- 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 - Selects 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 89).

5.5 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 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 will detect the cable type and use the appropriate pins to send and receive Ethernet signals.

Ethernet cabling is like U.S. telephone cables, except that it has eight conductors. 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 114) for a diagram of how to configure Ethernet cable.

5.5.1 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-

5.5.2 Ethernet Performance

Ethernet performance on the MVI56E-MNETR module can affect the operation of the MNETR application ports in the following ways.

- Accessing the web interface (refreshing the page, downloading files, and so on) may affect MNETR performance
- High Ethernet traffic may impact MNETR performance (consider CIPconnect (page 56) for these applications and disconnect the module Ethernet port from the network).

5.6 Modbus Protocol Specification

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

5.6.1 About the Modbus TCP/IP 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 Client 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 Client establishes a connection to the remote server. When the connection is established, the Client sends the Modbus TCP/IP commands to the server. The MVI56E-MNETR module works both as a Client and as 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 a TCP/IP network. The efficiency, scalability, and low cost of a Modbus TCP/IP network make this an ideal solution for industrial applications.

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

5.6.2 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	Set Single Coil	X	X
6	Single Register Write	X	X
7	Read Exception Status		X
8	Diagnostics		X
15	Multiple Coil Write	X	X
16	Multiple Register Write	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.

5.6.3 Read Coil Status (Function Code 01)

Query

This function allows the user to obtain the ON/OFF status of logic coils 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 that will be 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 read output status request to read coils 0020 to 0056 from server device number 11.

Adr	Func	Data Start Pt Hi	Data Start Pt Lo	Data # Of Pts Ho	Data # Of Pts Lo	Error Check Field
11	01	00	13	00	25	CRC

Response

An example response to Read Coil Status is as shown in Figure C2. 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 will be 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 follow. For coil quantities that are not even multiples of eight, the last characters will be 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 will reflect coil status at the end of the scan. Some servers will 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.

Addr	Func	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
11	01	05	CD	6B	B2	OE	1B	CRC

The status of coils 20 to 27 is shown as CD(HEX) = 1100 1101 (Binary). Reading left to right, this shows that coils 27, 26, 23, 22, and 20 are all on. The other coil data 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.

5.6.4 Read Input Status (Function Code 02)

Query

This function allows the user to obtain the ON/OFF status of discrete inputs 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 will be 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 from server number 11.

Addr	Func	Data Start Pt Hi	Data Start Pt Lo	Data #of Pts Hi	Data #of Pts Lo	Error Check Field
11	02	00	C4	00	16	CRC

Response

An example response to Read Input Status is as shown in Figure C4. 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 will be 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 follow. For input quantities that are not even multiples of eight, the last characters will be 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, data will reflect input status at the end of the scan. Some servers will 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.

Addr	Func	Byte Count	Data Discrete Input 10197 to 10204	Data Discrete Input 10205 to 10212	Data Discrete Input 10213 to 10218	Error Check Field
11	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.

5.6.5 Read Holding Registers (Function Code 03)

Query

Read Holding Registers (03) allows the user to obtain the binary contents of holding registers 4xxxx 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 up to 125 registers to obtained at each request; however, the specific server device may have restriction 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 from server 584 number 11.

Addr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
11	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 will reflect the register content at the end of the scan. Some servers will limit the quantity of register content provided each scan; thus for large register quantities, multiple transmissions will be 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.

Addr	Func	ByteCnt	Hi Data	Lo Data	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
11	03	06	02	2B	00	00	00	64	CRC

5.6.6 Read Input Registers (Function Code 04)

Query

Function code 04 obtains the contents of the controller's input registers at addresses 3xxxx. 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 up to 125 registers to be obtained 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 3009 in server number 11.

Addr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
11	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 will reflect the register content at the end of the scan. Each PC will limit the quantity of register contents provided each scan; thus for large register quantities, multiple PC scans will be required, and the data provided will be form sequential scans.

In the example below the register 3009 contains the decimal value 0.

Addr	Func	Byte Count	Data Input Reg Hi	Data Input Reg Lo	Error Check Field
11	04	02	00	00	E9

5.6.7 Force Single Coil (Function Code 05)

Query

This message forces a single coil 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) will set the coil ON and the value zero will turn it OFF; all other values are illegal and will not affect that coil.

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

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

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

Addr	Func	Data Coil # Hi	Data Coil # Lo	Data On/Off Ind	Data	Error Check Field
11	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.

Addr	Func	Data Coil # Hi	Data Coil # Lo	Data On/ Off	Data	Error Check Field
11	05	00	AC	FF	00	CRC

The forcing of a coil via Modbus function 5 will be accomplished regardless of whether the addressed coil is disabled or not (In ProSoft products, the coil is only affected if the necessary ladder logic is implemented).

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 will be "hot".

5.6.8 Preset Single Register (Function Code 06)

Query

Function (06) allows the user to modify the contents of a holding register. 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 will load the specified register with the contents specified.

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

Addr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
11	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.

Addr	Func	Data Reg Hi	Data Reg Lo	Data Input Reg Hi	Data Input Reg Lo	Error Check Field
11	06	00	01	00	03	CRC

5.6.9 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-MNETR 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.

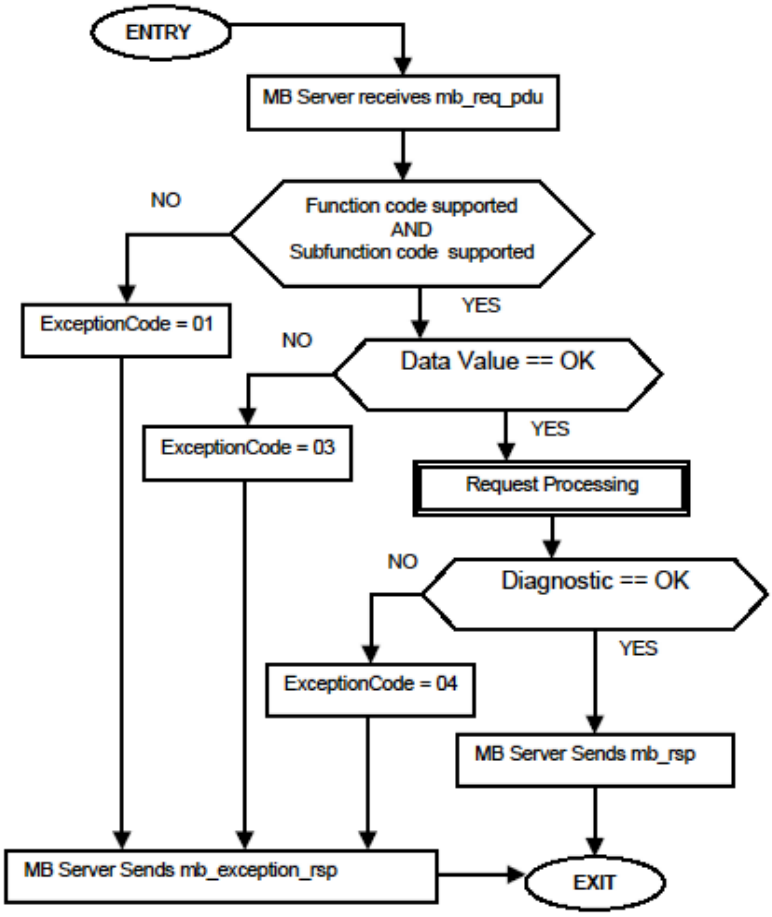
Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

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.



5.6.10 Force Multiple Coils (Function Code 15)

Query

This message forces each coil 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) will force 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 will be 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.

Addr	Func	Hi Add	Lo Add	Quantity	Byte Cnt	Data Coil Status 20 to 27	Data Coil Status 28 to 29	Error Check Field	
11	0F	00	13	00	0A	02	CD	00	CRC

Response

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

Addr	Func	Hi Addr	Lo Addr	Quantity	Error Check Field
11	0F	00	13	00	0A	CRC

The writing of coils via Modbus function 15 will be accomplished regardless of whether the addressed coils are disabled or not.

Coils that are unprogrammed 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 will be hot.

5.6.11 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.

Addr	Func	Hi Add	Lo Add	Quantity		Byte Cnt	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
11	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.

Addr	Func	Hi Addr	Lo Addr	Quantity		Error Check Field
11	10	00	87	00	02	56

5.6.12 Modbus Exception Responses

When a Modbus Client sends a request to a server device, it expects a normal response. One of four possible events can occur from the Client'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 Client 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 Client 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 Client 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 Client'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 Client 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 Client 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 unconfigured 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 Client. The Client 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 Client 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 Client 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.

5.7 Using the Optional Add-On Instruction Rung Import

5.7.1 Before You Begin

- Make sure that you have installed RSLogix 5000 version 16 (or later).
- Download the Optional Add-On file *MVI56EMNETR_Optional_Rung_vXXX.L5X* from www.prosoft-technology.com.
- Save a copy in a folder in your PC.

5.7.2 Overview

The Optional Add-On Instruction Rung Import contains optional logic for MVI56E-MNETR 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 PCB (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.

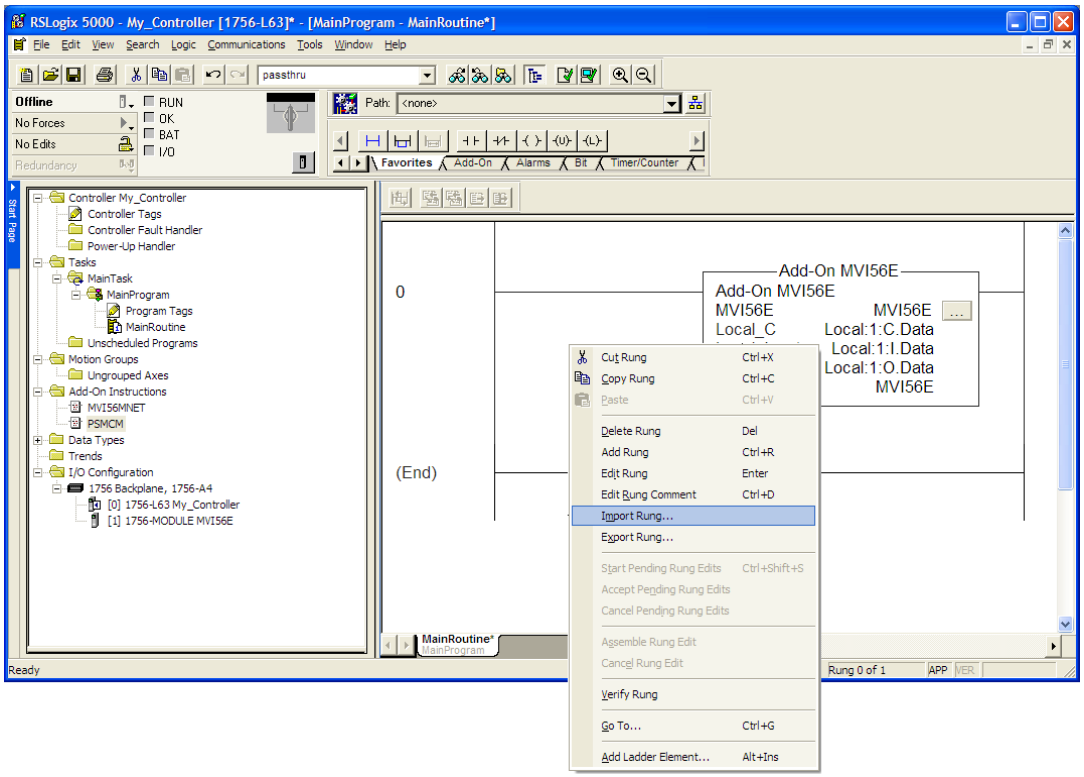
- Read/Write Module Clock Value

  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.

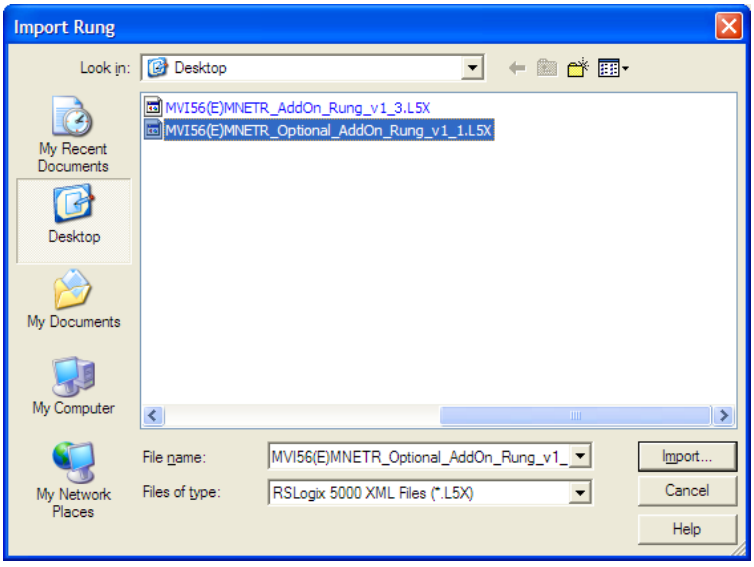
Important: The Optional Add-On Instruction only supports the two features listed above. You must use the sample ladder logic for all other features including backplane transfer of Modbus TCP/IP data.

5.7.3 Installing the Rung Import with Optional Add-On Instruction

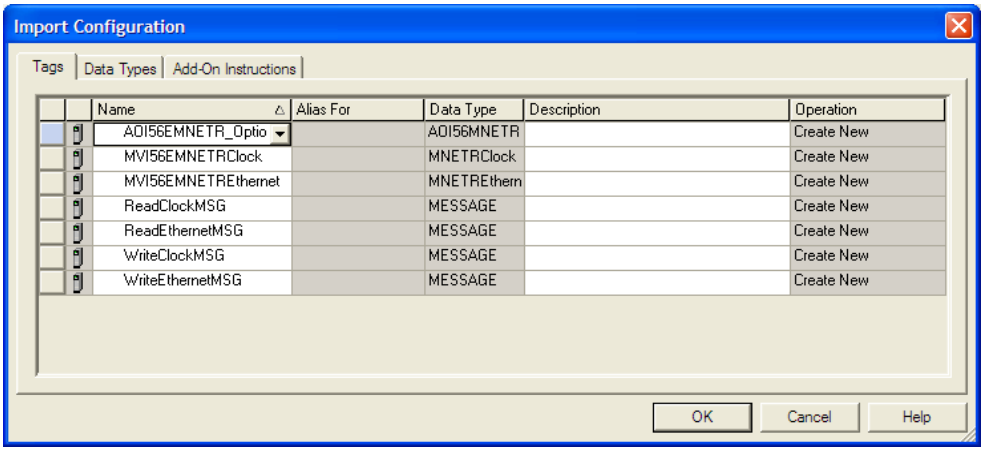
1 Right-click an empty rung in the main routine of your existing ladder logic and choose **IMPORT RUNG**.



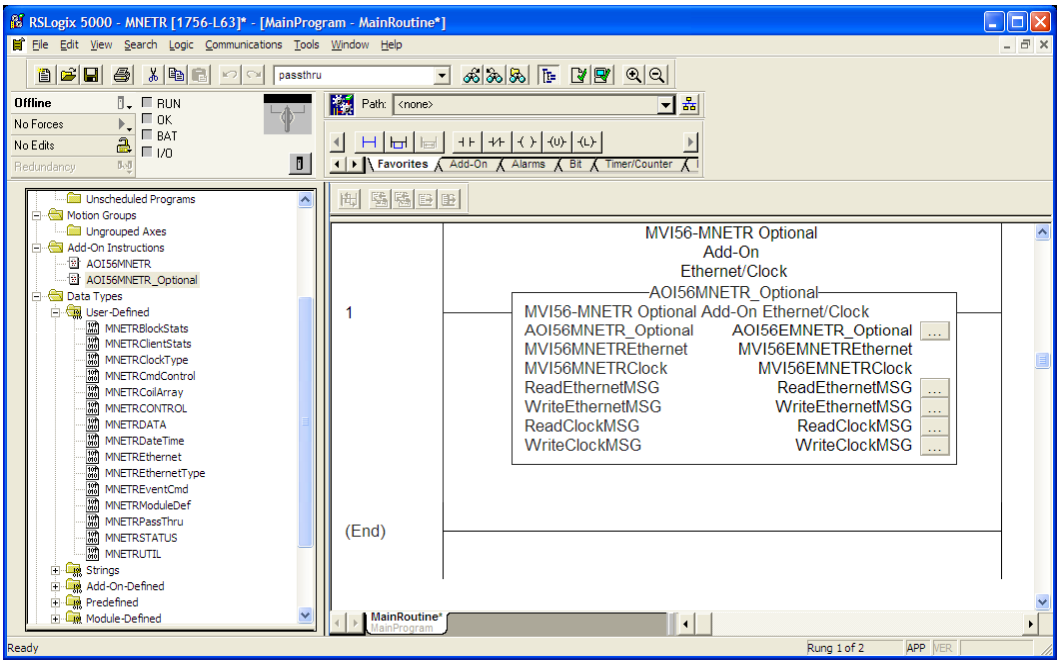
2 Navigate to the folder where you saved MVI56EMNETR_Optional_Rung_v1_1.L5X and select the file.



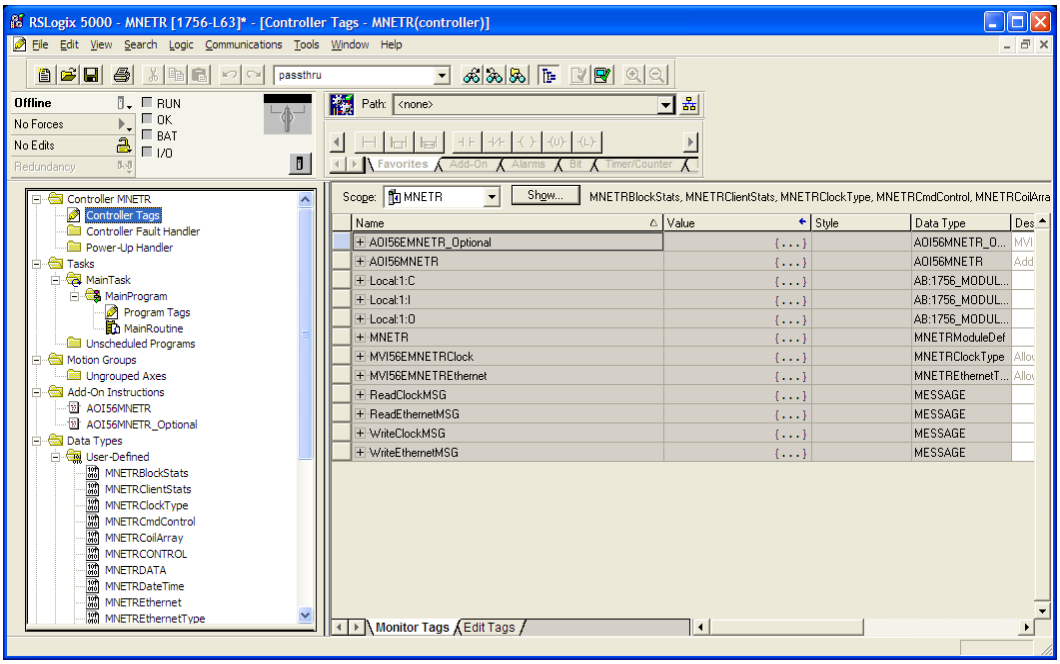
3 In the **IMPORT CONFIGURATION** window, click **OK.**



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

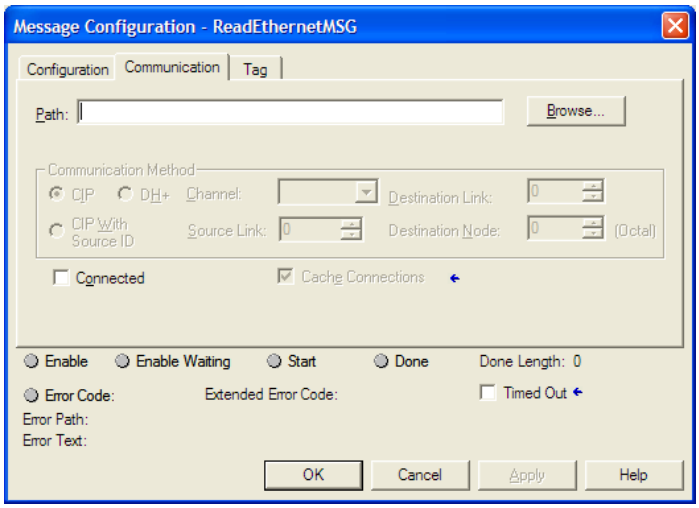


You will notice that new tags have been imported: four **MESSAGE** tags, **MVI56MNETRCLOCK** and **MVI56MNETRETHERNET** tags.

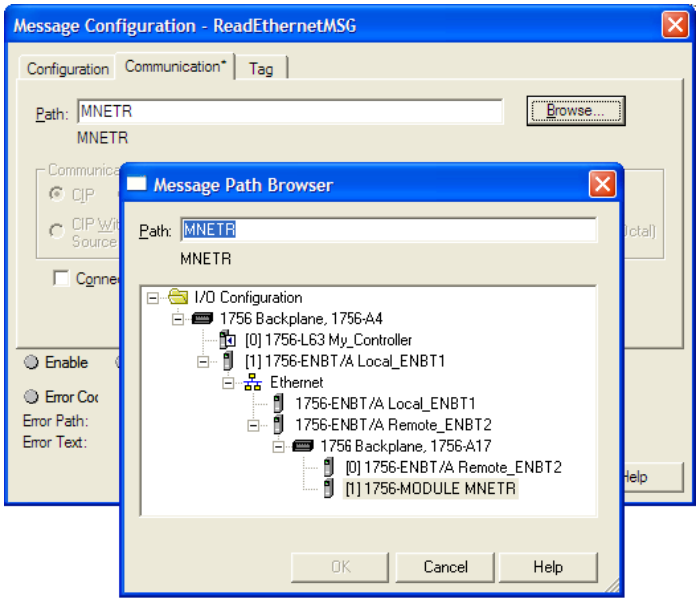


4 In the Add-On Instruction, click the **[...]** button next to each **MSG** tag to open the **MESSAGE CONFIGURATION TAG**.

5 Click the **COMMUNICATION** tab and click the **BROWSE** button as follows.



6 Select the module to configure the message path.



5.7.4 Reading the Ethernet Settings from the Module

1 Expand the **MVI56MNETRETHERNET** controller tag and move a value of 1 to **MVI56MNETRETHERNET.READ.**

MVI56EMNETREthernet	{...}
MVI56EMNETREthernet.Read	1
MVI56EMNETREthernet.Write	0
MVI56EMNETREthernet.Config	{...}
MVI56EMNETREthernet.Config.IP	{...}
MVI56EMNETREthernet.Config.IP[0]	0
MVI56EMNETREthernet.Config.IP[1]	0
MVI56EMNETREthernet.Config.IP[2]	0
MVI56EMNETREthernet.Config.IP[3]	0
MVI56EMNETREthernet.Config.Netmask	{...}
MVI56EMNETREthernet.Config.Netmask[0]	0
MVI56EMNETREthernet.Config.Netmask[1]	0
MVI56EMNETREthernet.Config.Netmask[2]	0
MVI56EMNETREthernet.Config.Netmask[3]	0
MVI56EMNETREthernet.Config.Gateway	{...}
MVI56EMNETREthernet.Config.Gateway[0]	0
MVI56EMNETREthernet.Config.Gateway[1]	0
MVI56EMNETREthernet.Config.Gateway[2]	0
MVI56EMNETREthernet.Config.Gateway[3]	0

2 The bit will be automatically reset and the current Ethernet settings will be copied to **MVI56MNETRETHERNET** controller tag as follows.

MVI56EMNETREthernet	{...}
MVI56EMNETREthernet.Read	0
MVI56EMNETREthernet.Write	0
MVI56EMNETREthernet.Config	{...}
MVI56EMNETREthernet.Config.IP	{...}
MVI56EMNETREthernet.Config.IP[0]	105
MVI56EMNETREthernet.Config.IP[1]	102
MVI56EMNETREthernet.Config.IP[2]	0
MVI56EMNETREthernet.Config.IP[3]	32
MVI56EMNETREthernet.Config.Netmask	{...}
MVI56EMNETREthernet.Config.Netmask[0]	255
MVI56EMNETREthernet.Config.Netmask[1]	255
MVI56EMNETREthernet.Config.Netmask[2]	255
MVI56EMNETREthernet.Config.Netmask[3]	0
MVI56EMNETREthernet.Config.Gateway	{...}
MVI56EMNETREthernet.Config.Gateway[0]	192
MVI56EMNETREthernet.Config.Gateway[1]	168
MVI56EMNETREthernet.Config.Gateway[2]	0
MVI56EMNETREthernet.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

5.7.5 Writing the Ethernet Settings to the Module

1. Expand the **MVI56EMNETRETHERNET** controller tag.
2. Set the new Ethernet configuration in **MVI56EMNETRETHERNET.CONFIG**
3. Move a value of 1 to **MVI56MNETRETHERNET.WRITE**

MVI56EMNETREthernet	{...}
MVI56EMNETREthernet.Read	0
MVI56EMNETREthernet.Write	1
MVI56EMNETREthernet.Config	{...}
MVI56EMNETREthernet.Config.IP	{...}
MVI56EMNETREthernet.Config.IP[0]	105
MVI56EMNETREthernet.Config.IP[1]	102
MVI56EMNETREthernet.Config.IP[2]	0
MVI56EMNETREthernet.Config.IP[3]	32
MVI56EMNETREthernet.Config.Netmask	{...}
MVI56EMNETREthernet.Config.Netmask[0]	255
MVI56EMNETREthernet.Config.Netmask[1]	255
MVI56EMNETREthernet.Config.Netmask[2]	255
MVI56EMNETREthernet.Config.Netmask[3]	0
MVI56EMNETREthernet.Config.Gateway	{...}
MVI56EMNETREthernet.Config.Gateway[0]	192
MVI56EMNETREthernet.Config.Gateway[1]	168
MVI56EMNETREthernet.Config.Gateway[2]	0
MVI56EMNETREthernet.Config.Gateway[3]	1

4. After the message is executed, the **MVI56MNETRETHERNET.WRITE** bit resets to 0.

MVI56EMNETREthernet	{...}
MVI56EMNETREthernet.Read	0
MVI56EMNETREthernet.Write	0
MVI56EMNETREthernet.Config	{...}
MVI56EMNETREthernet.Config.IP	{...}
MVI56EMNETREthernet.Config.IP[0]	105
MVI56EMNETREthernet.Config.IP[1]	102
MVI56EMNETREthernet.Config.IP[2]	0
MVI56EMNETREthernet.Config.IP[3]	32
MVI56EMNETREthernet.Config.Netmask	{...}
MVI56EMNETREthernet.Config.Netmask[0]	255
MVI56EMNETREthernet.Config.Netmask[1]	255
MVI56EMNETREthernet.Config.Netmask[2]	255
MVI56EMNETREthernet.Config.Netmask[3]	0
MVI56EMNETREthernet.Config.Gateway	{...}
MVI56EMNETREthernet.Config.Gateway[0]	192
MVI56EMNETREthernet.Config.Gateway[1]	168
MVI56EMNETREthernet.Config.Gateway[2]	0
MVI56EMNETREthernet.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

5.7.6 Reading the Clock Value from the Module

1 Expand the **MVI56MNETRCLOCK** controller tag and move a value of 1 to **MVI56MNETRCLOCK.READ**

MVI56EMNETRClock	{...}
MVI56EMNETRClock.Read	1
MVI56EMNETRClock.Write	0
MVI56EMNETRClock.Config	{...}
MVI56EMNETRClock.Config.Year	0
MVI56EMNETRClock.Config.Month	0
MVI56EMNETRClock.Config.Day	0
MVI56EMNETRClock.Config.Hour	0
MVI56EMNETRClock.Config.Minute	0
MVI56EMNETRClock.Config.Seconds	0

2 The bit will be automatically reset and the current clock value will be copied to **MVI56MNETRCLOCK.CONFIG** controller tag as follows.

MVI56EMNETRClock	{...}
MVI56EMNETRClock.Read	0
MVI56EMNETRClock.Write	0
MVI56EMNETRClock.Config	{...}
MVI56EMNETRClock.Config.Year	2009
MVI56EMNETRClock.Config.Month	5
MVI56EMNETRClock.Config.Day	4
MVI56EMNETRClock.Config.Hour	15
MVI56EMNETRClock.Config.Minute	38
MVI56EMNETRClock.Config.Seconds	9

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

5.7.7 Writing the Clock Value to the Module

1. Expand the **MVI56MNETRCLOCK** controller tag.
2. Set the new Clock value in **MVI56MNETRCLOCK.CONFIG**
3. Move a value of 1 to **MVI56MNETRCLOCK.WRITE**

MVI56EMNETRClock	{...}
MVI56EMNETRClock.Read	0
MVI56EMNETRClock.Write	1
MVI56EMNETRClock.Config	{...}
MVI56EMNETRClock.Config.Year	2009
MVI56EMNETRClock.Config.Month	5
MVI56EMNETRClock.Config.Day	4
MVI56EMNETRClock.Config.Hour	15
MVI56EMNETRClock.Config.Minute	38
MVI56EMNETRClock.Config.Seconds	9

4. The bit will be automatically reset to 0.

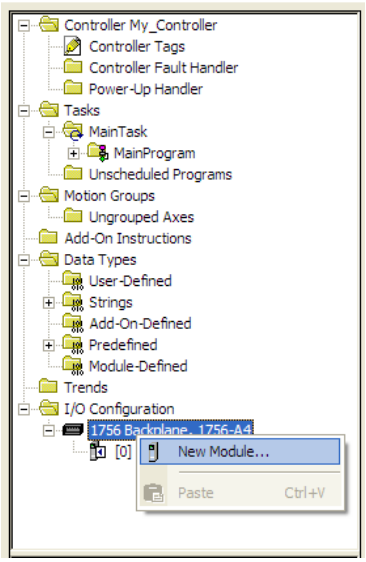
MVI56EMNETRClock	{...}
MVI56EMNETRClock.Read	0
MVI56EMNETRClock.Write	0
MVI56EMNETRClock.Config	{...}
MVI56EMNETRClock.Config.Year	2009
MVI56EMNETRClock.Config.Month	5
MVI56EMNETRClock.Config.Day	4
MVI56EMNETRClock.Config.Hour	15
MVI56EMNETRClock.Config.Minute	38
MVI56EMNETRClock.Config.Seconds	9

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

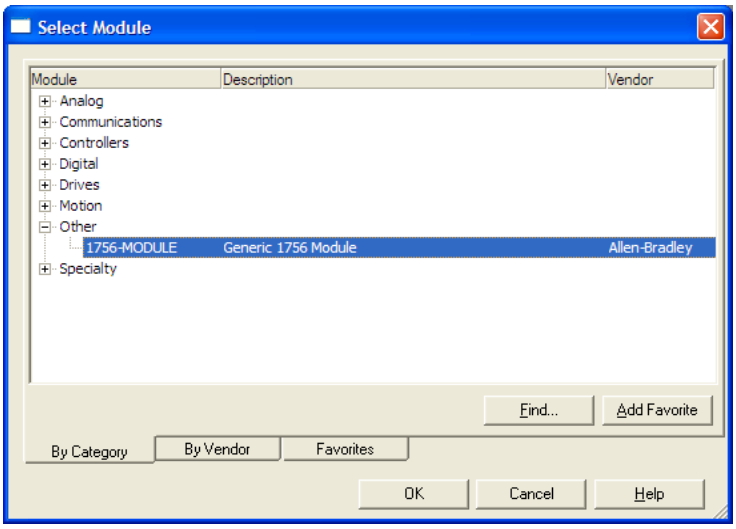
WriteClockMSG	{...}
WriteClockMSG.Flags	16#0200
WriteClockMSG.EW	0
WriteClockMSG.ER	0
WriteClockMSG.DN	0
WriteClockMSG.ST	0
WriteClockMSG.EN	0
WriteClockMSG.TO	0
WriteClockMSG.EN_CC	1
WriteClockMSG.ERR	16#0000
WriteClockMSG.EXERR	16#0000_0000
WriteClockMSG.ERR_SRC	0
WriteClockMSG.DN_LEN	0
WriteClockMSG.REQ_LEN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

5.8 Adding the Module to an Existing Project

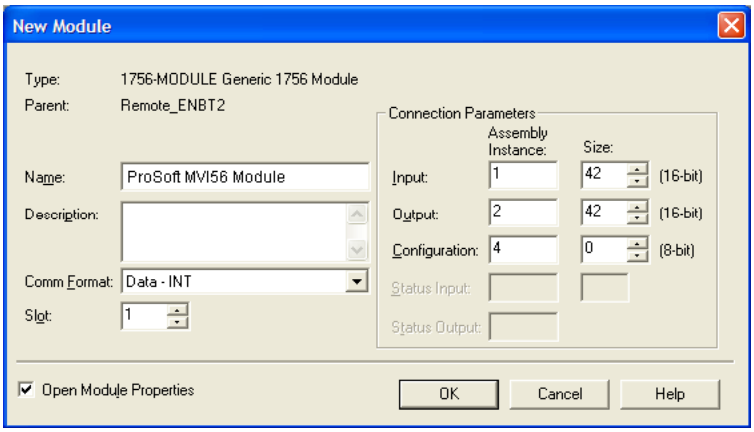
1 **Add the MVI56E-MNETR module to the project.** Select the **I/O CONFIGURATION** folder in the *Controller Organization* window, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE**.



This action opens the *Select Module* dialog box:



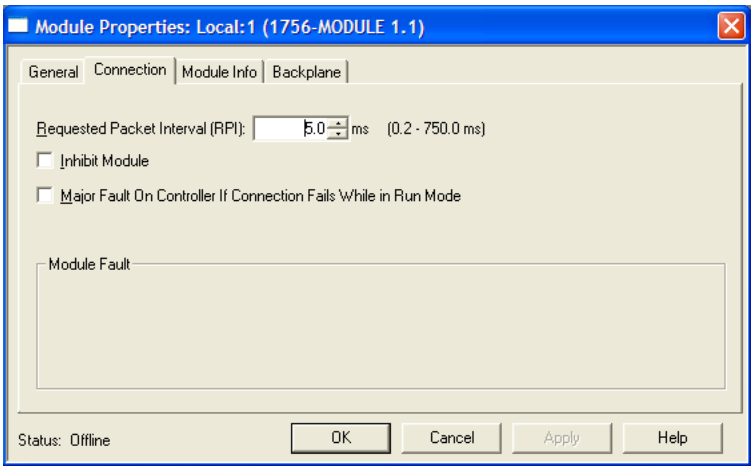
2 Select the **1756-MODULE** (Generic 1756 Module) from the list and click **OK.** This action opens the *New Module* dialog box.



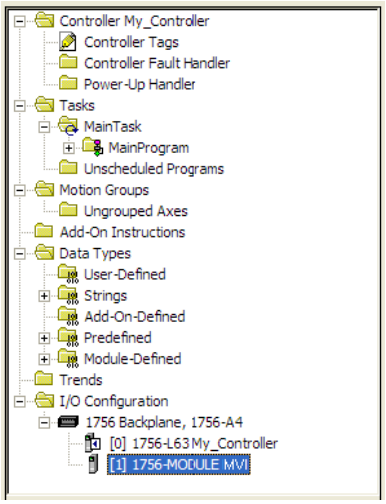
Parameter	Value
Name	Enter a module identification string. The recommended value is **MNETR**.
Description	Enter a description for the module. Example: **MODBUS TCP/IP INTERFACE MODULE WITH REDUCED DATA BLOCK**.
Comm Format	Select **DATA-INT (Very Important)**
Slot	Enter the slot number in the rack where the MVI56E-MNETR module will be installed.
Input Assembly Instance	**1**
Input Size	**42**
Output Assembly Instance	**2**
Output Size	**42**
Configuration Assembly Instance	**4**
Configuration Size	**0**

Enter the Name, Description and Slot options for your application. 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. Click **OK** to continue.

3 In the *Connection* tab of the *Module Properties* dialog, 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.



4 Save the module and click **OK** to dismiss the dialog box. The *Controller Organization* window now displays the module's presence.



5 Copy the Controller Tags (page 65) from the sample program.
6 Copy the User Defined Data Types (page 67) from the sample program.
7 Copy the Ladder Rungs from the sample program.
8 Save and Download (page 29, page 145) the new application to the controller and place the processor in run mode.

5.9 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.

5.9.1 Opening the Sample Program in RSLogix

The sample program for your MVI56E-MNETR 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.

Download the manuals and sample program from the ProSoft Technology web site

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

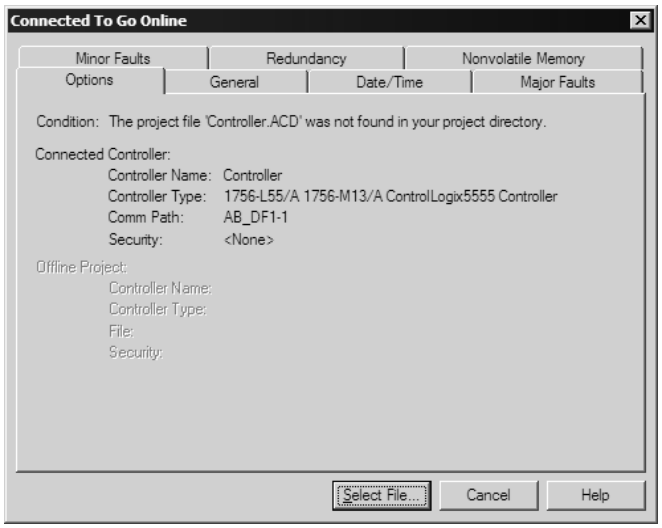
From that link, navigate to the download page for your module and choose the sample program to download for your version of RSLogix 5000 and your processor.

To determine the firmware version of your processor

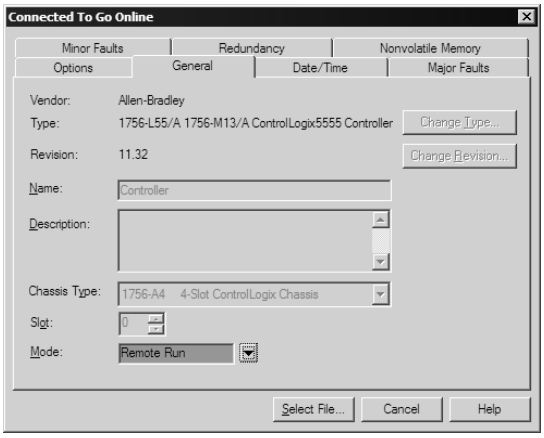
Important: The RSLinx service must be installed and running on your computer in order for RSLogix to communicate with the processor. Refer to your RSLinx and RSLogix documentation for help configuring and troubleshooting these applications.

1. Connect an RS-232 serial cable from the COM (serial) port on your PC to the communication port on the front of the processor.
2. Start RSLogix 5000 and close any existing project that may be loaded.
3. Open the **COMMUNICATIONS** menu and choose **GO ONLINE**. RSLogix will establish communication with the processor. This may take a few moments.

4 When RSLogix has established communication with the processor, the *Connected To Go Online* dialog box will open.



5 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



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

To open the sample program

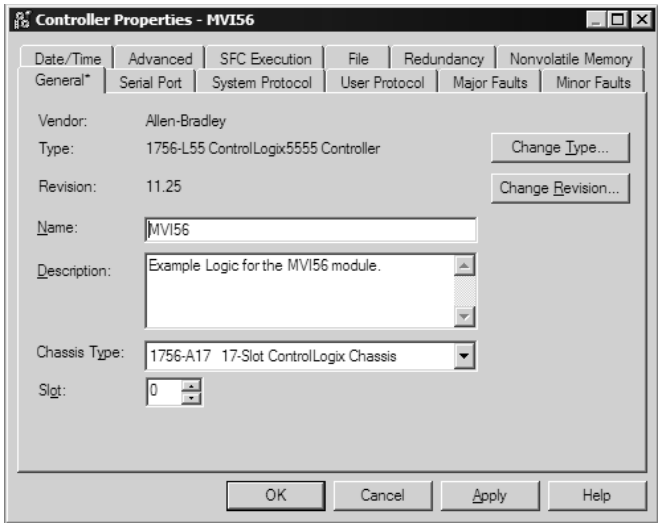
1 On the *Connected to Go Online* dialog box, click the **SELECT FILE** button.
2 Choose the sample program file that matches your firmware version, and then click the **SELECT** button.
3 RSLogix will load the sample program.

The next step is to configure the correct controller type and slot number for your application.

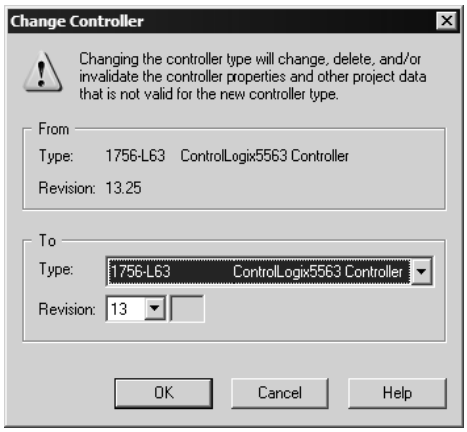
5.9.2 Choosing the Controller Type

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 Organization* list, select the folder for the controller and then click the right mouse button to open a shortcut menu.
2. On the shortcut menu, choose **PROPERTIES.** This action opens the *Controller Properties* dialog box.



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



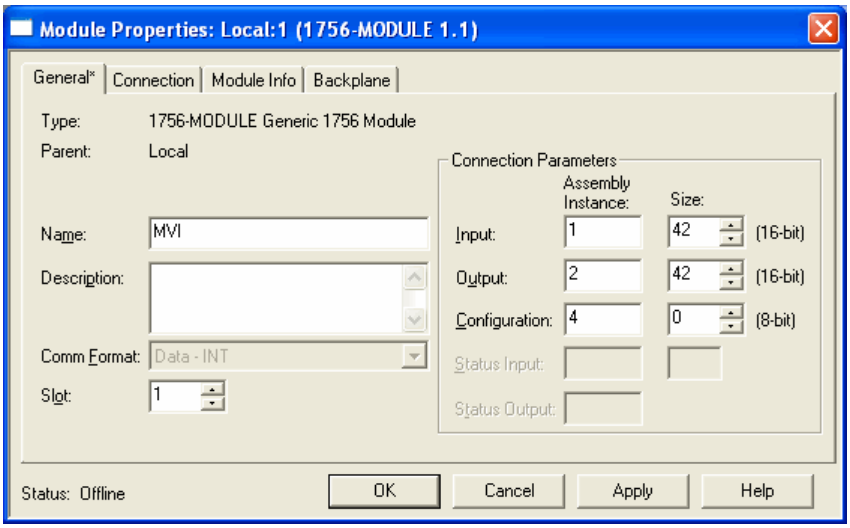
4. Open the **TYPE** dropdown list, and then select your ControlLogix controller.
5. Select the correct firmware revision for your controller, if necessary.
6. Click **OK** to save your changes and return to the previous window.

5.9.3 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 Organization* list, select the module, and then click the right mouse button to open a shortcut menu.
2. On the shortcut menu, choose **PROPERTIES.** This action opens the *Module Properties* dialog box.



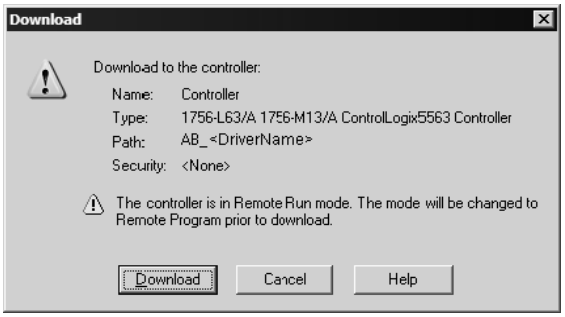
3. In the **SLOT** field, use the up and down arrows on the right side of the field to select the slot number where the module will reside in the rack, and then click **OK.**

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

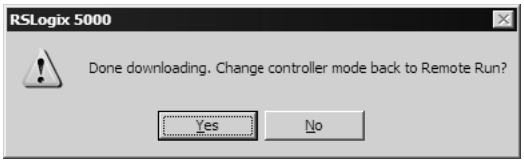
5.9.4 Downloading the Sample Program to the Processor

Note: The key switch on the front of the ControlLogix processor 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.

5.9.5 Adding the Sample Ladder to an Existing Application

1. Copy the Controller Tags (page 65) from the sample program.
2. Copy the User-Defined Data Types (page 67) from the sample program.
3. Copy the Ladder Rungs from the sample program.
4. Save and Download (page 29, page 145) 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