ProSoft
TECHNOLOGY

Where Automation Connects.

MVI56E-MCMR

ControlLogix Platform

Modbus Communication Module with Reduced Data Block

USER MANUAL

Your Feedback Please

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

How to Contact Us

ProSoft Technology
5201 Truxtun Ave., 3rd Floor
Bakersfield, CA 93309
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
support@prosoft-technology.com

ProSoft Technology® Product Documentation

In an effort to conserve paper, ProSoft Technology no longer includes printed manuals with our product shipments. User Manuals, Datasheets, Sample Ladder Files, and Configuration Files are provided on the enclosed DVD, and are available at no charge from our web site: www.prosoft-technology.com

MVI (Multi Vendor Interface) Modules

WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.

AVERTISSEMENT - RISQUE D'EXPLOSION - AVANT DE DÉCONNECTER L'ÉQUIPEMENT, COUPER LE COURANT OU S'ASSURER QUE L'EMPLACEMENT EST DÉSIGNÉ NON DANGEREUX.

Important Safety Information

North America Warnings

A Warning - Explosion Hazard - Substitution of components may impair suitability for Class I, Division 2.
B Warning - Explosion Hazard - When in hazardous locations, turn off power before replacing or rewiring modules.
  Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be nonhazardous.
C Suitable for use in Class I, Division 2 Groups A, B, C, and D, T5 Hazardous Locations or Non-Hazardous Locations.

ATEX Warnings and Conditions of Safe Usage

Power, Input, and Output (I/O) wiring must be in accordance with the authority having jurisdiction

A Warning - Explosion Hazard - When in hazardous locations, turn off power before replacing or wiring modules.
B Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be non-hazardous.
C These products are intended to be mounted in an IP54 enclosure. The devices shall provide external means to prevent the rated voltage being exceeded by transient disturbances of more than 40%. This device must be used only with ATEX certified backplanes.
D DO NOT OPEN WHEN ENERGIZED.

Electrical Ratings

- 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 to 150 Hz
- Relative Humidity 5% to 95% (without condensation)
- All phase conductor sizes must be at least 1.3 mm (squared) and all earth ground conductors must be at least 4mm (squared).

Label Markings

<cULus>
E183151
Class I, DIV 2, groups A,B,C,D
T5 for all models
0°C to +60°C

<Ex>

II 3 G

Ex nA T5

0°C <= Ta <= 60°C

II – Equipment intended for above ground use (not for use in mines).

3 – Category 3 equipment, investigated for normal operation only.

G – Equipment protected against explosive gasses.

Agency Approvals and Certifications

Agency
RoHS
ATEX
CSA
CE
CSA CB Safety
cULus
GOST-R

RoHS ⟨Ex⟩ CSA CE cUL us P⌐ ETL us

Battery Life Advisory

Note: Modules manufactured after April 1st, 2011 do not contain a battery. For modules manufactured before that date the following applies:

The module uses a rechargeable Lithium Vanadium Pentoxide battery to back up the real-time clock and CMOS settings. The battery itself should last for the life of the module. However, if left in an unpowered state for 14 to 21 days, the battery may become fully discharged and require recharging by being placed in a powered-up ControlLogix chassis. The time required to fully recharge the battery may be as long as 24 hours.

Once it is fully charged, the battery provides backup power for the CMOS setup and the real-time clock for approximately 21 days. Before you remove a module from its power source, ensure that the battery within the module is fully charged (the BATT LED on the front of the module goes OFF when the battery is fully charged). If the battery is allowed to become fully discharged, the module will revert to the default BIOS and clock settings.

Note: The battery is not user-replaceable or serviceable.

Contents

Guide to the MVI56E-MCMR User Manual

Function		Section to Read	Details
Introduction (Must Do)	→	Start Here (page 11)	This section introduces the customer to the module. Included are: package contents, system requirements, hardware installation, and basic configuration.
Diagnostic and Troubleshooting	→	Diagnostics and Troubleshooting (page 122, page 130)	This section describes Diagnostic and Troubleshooting procedures.
Reference Product Specifications Functional Overview	→	Reference (page 143) Product Specifications (page 145) Functional Overview (page 148)	These sections contain general references associated with this product, Specifications, and the Functional Overview.
Support, Service, and Warranty Index	→	Support, Service and Warranty (page 219) Index	This section contains Support, Service and Warranty information. Index of chapters.

1   Start Here

In This Chapter

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 MCMR and ControlLogix devices to a power source and to the MVI56E-MCMR 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-MCMR 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-MCMR module. The module requires 800 mA of available 5 Vdc power and 3 mA of available 24 VDC power.

- Rockwell Automation RSLogix 5000 programming software
  - o Version 16 or higher required for Add-On Instruction
  - o 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:
  - o Microsoft Windows® Vista
  - o Microsoft Windows XP Professional with Service Pack 1 or 2
  - o Microsoft Windows 2000 Professional with Service Pack 1, 2, or 3
  - o 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)
- 256-color VGA graphics adapter, 800 x 600 minimum resolution (True Color 1024 × 768 recommended)
- DVD drive

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 Deployment Checklist

Before you begin configuring the module, consider the following questions. Your answers will help you determine the scope of your project, and the configuration requirements for a successful deployment.

1 _____ Are you creating a new application or integrating the module into an existing application?

Most applications can use the Sample Add-On Instruction or Sample Ladder Logic without any edits to the Sample Program.

2 _____ Which slot number in the chassis will the MVI56E-MCMR module occupy?

For communication to occur, you must enter the correct slot number in the sample program.

3 _____ Are RSLogix 5000 and RSLinx installed?

RSLogix and RSLinx are required to communicate to the ControlLogix processor (1756-L1, L55, L61 & L63). Sample Ladder programs are available for different versions of RSLogix 5000.

4 _____ How many words of data do you need to transfer in your application (from ControlLogix to Module / to ControlLogix from Module)?

The MVI56E-MCMR module can transfer a maximum of 5000 (16-bit) registers to and from the ControlLogix processor. The Sample Ladder transfers 600 words to the ControlLogix processor (into the Read Data array), and obtains 600 words from the ControlLogix processor (from the Write Data array)

5 _____ Will you be using the module as a Modbus Master or Modbus Slave? Will you be transferring data using Modbus RTU or Modbus ASCII?

Modbus is a Master/Slave network. Only one Master is allowed on the serial communications line (max 32 devices/RS485). The Master is responsible for polling data from the Slaves on the network.

6 _____ For a Modbus Master, what Slave Device Addresses and Modbus Data Addresses do you need to exchange data with on the Modbus network?

For a Modbus Master, you must know the Slave Device Address number of each Slave device to poll. You also need the Modbus address (for example, coil 00001, register 40001) of the data to read from or write to each Slave device.

7 _____ For a Modbus Slave, how many words or bits of data do you need to send to the Master device?

The MVI56E-MCMR module can send data to a Modbus Master as 0x coil data, 1x input coil data, 3x input registers, and 4x holding registers. The sample program transfers 600 (16-bit) words or 9600 bits to the ControlLogix processor, and 600 (16-bit) words or 9600 bits from the ControlLogix processor.

8 Serial Communication Parameters for the Modbus network:

_____ Baud rate?

_____ Data bits?

_____ Parity?

_____ Stop bits?

Required for both Master and Slave configurations.

9 _____ Wiring type to use (RS232, 422 or 485). Configured by jumper settings.

Required for proper implementation of the module in Master and Slave configurations.

Note: If you are installing your module into a new system, and plan to use our Sample Ladder Logic, refer to the printed Quick Start Guide in the module package for simple installation procedures.

- For version 16 or newer of RSLogix 5000, refer to Upload the Add-On Instruction from the Module.
- For EXISTING system installations, refer to Using the Sample Program - RSLogix Version 15 and earlier (page 203).

Note: Most applications can use the Sample Ladder Logic *without modifying the sample program.*

1.4   Package Contents

The following components are included with your MVI56E-MCMR 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-MCMR Module	MVI56E-MCMR	Modbus Communication Module with Reduced Data Block
1	Cable	RL-CBL025	5 foot Ethernet Straight-Through Cable (Gray)
2	Cable	Cable #14, RJ45 to DB9 Male Adapter cable	For DB9 Connection to Module's Application Serial Port
2	Adapter	1454-9F	Two Adapters, DB9 Female to Screw Terminal. For RS422 or RS485 Connections to Port 1 and 2 of the Module
1	ProSoft Solutions DVD	DVD-001	Contains ProSoft Configuration Builder (PCB) and ProSoft Discovery Service (PDS) for the MVI56E-MCMR modules.

If any of these components are missing, please contact ProSoft Technology Support for replacement parts.

1.5   Setting Jumpers

There are three jumpers located at the bottom of the module. The first two jumpers (P1 and P2) set the serial communication mode: RS-232, RS-422 or RS-485.

The following illustration shows the MVI56E-MCMR jumper configuration, with the Setup Jumper OFF.
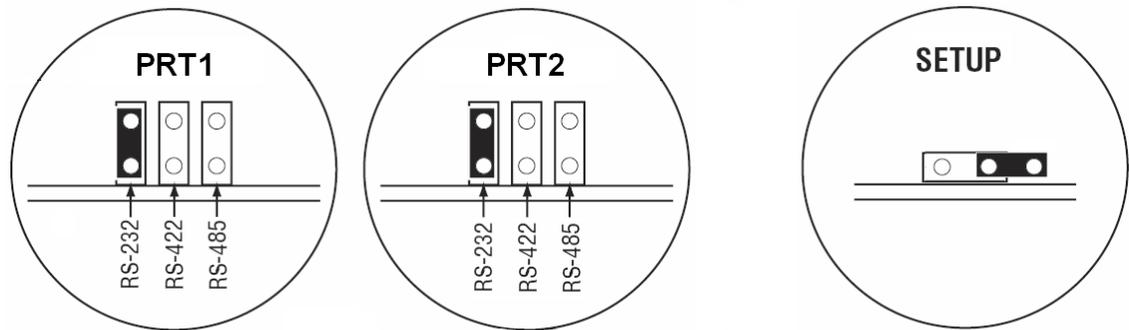
The Setup Jumper acts as "write protection" for the module's firmware. In "write protected" mode, the Setup pins are not connected, and the module's firmware cannot be overwritten. The module is shipped with the Setup jumper OFF. If you need to update the firmware, apply the Setup jumper to both pins.

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

1.6   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-MCMR 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-MCMR 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 Align the module with the top and bottom guides, and then slide it into the rack until the module is firmly against the backplane connector.

MVI56E Module

Slot 1
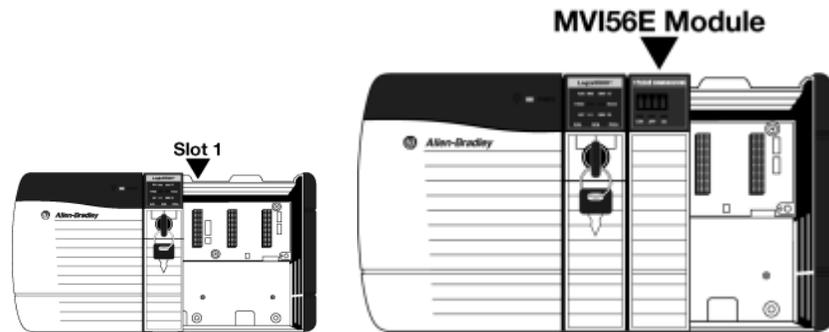
2 With a firm, steady push, snap the module into place.
3 Check that the holding clips on the top and bottom of the module are securely in the locking holes of the rack.
4 Make a note of the slot location. You must identify the slot in which the module is installed in order for the sample program to work correctly. Slot numbers are identified on the green circuit board (backplane) of the ControlLogix rack.
5 Turn power ON.

Note: If you insert the module improperly, the system may stop working or may behave unpredictably.

1.7   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 203).

Before You Begin

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

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

File Name	Description
MVI56(E)MCMR_AddOn_Rung_<VersionNumPri>.L5X	L5X file containing Add-On Instruction, user defined data types, controller tags and ladder logic required to configure the MVI56E-MCMR module
MVI56(E)MCMR_Optional_AddOn_Rung_v1_1.L5X	Optional L5X file containing additional Add-On Instruction with logic for changing Ethernet configuration and clock settings.

1.7.1   About the Optional Add-On Instruction

The Optional Add-On Instruction performs the following tasks:

▪   Read/Write Ethernet Configuration

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

▪   Read/Write Module Clock Value

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

Note: You can also set the date and time from the module's home page (page 129).
Important: The Optional Add-On Instruction supports only the two features listed above. You must use the regular MVI56E-MCMR Add-On Instruction for all other features including backplane transfer and Modbus data communication.

1.8 Creating a New RSLogix 5000 Project

1 Open the **FILE** menu, and then choose **NEW**.
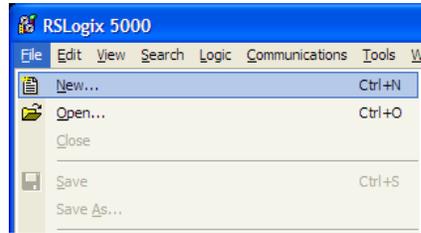
2 Select your ControlLogix controller model.
3 Select the **REVISION** of the controller.
4 Enter a name for your controller, such as *My_Controller*.
5 Select your ControlLogix chassis type.
6 Select **SLOT 0** for the controller.
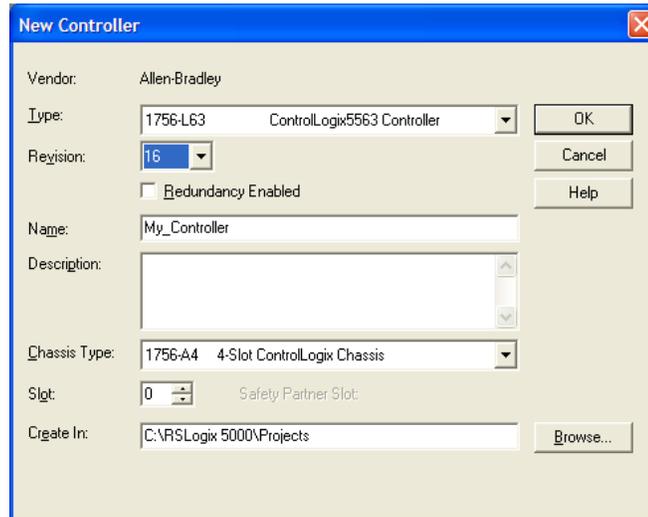
1.8.1 Creating the Remote Network

Note: If you are installing the MVI56E-MCMR 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 27).

1   Right-click **I/O CONFIGURATION** and choose **NEW MODULE**.
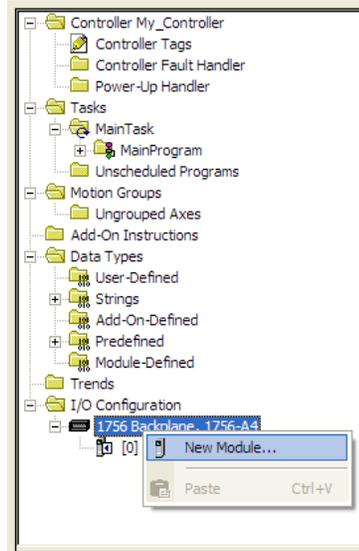
2   Expand the Communications module selections and then select the Ethernet Bridge module that matches your hardware. This example uses a 1756-ENBT/A module.
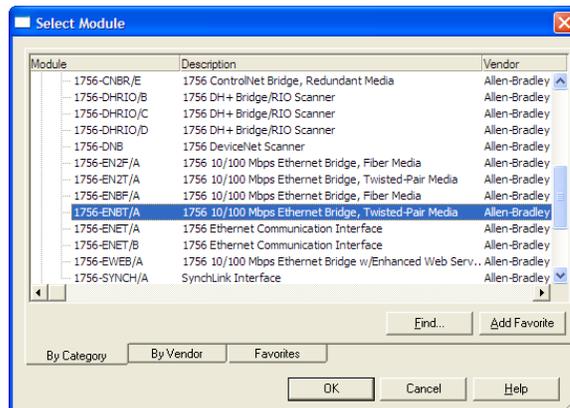
Note: If you are prompted to *Select Major Revision*, choose the lower of the available revision numbers.

3 Name the ENBT/A module, then set the *IP Address* and *Slot* location in the local rack with the ControlLogix processor.
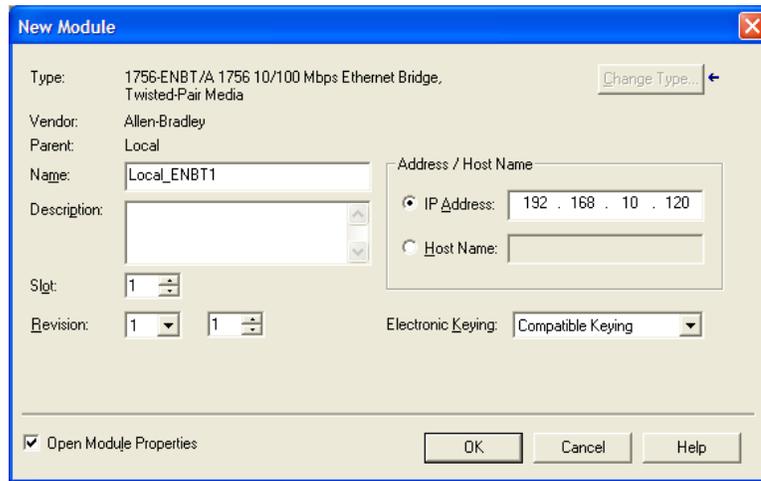
4 Click **OK**.
5 Next, select the **1756-ENBT** module that you just created in the *Controller Organization* pane and click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE**.
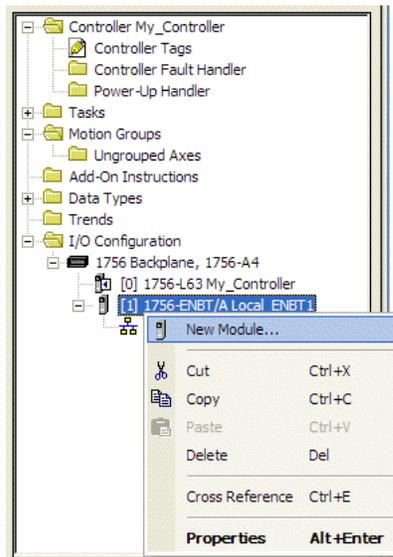
6 Repeat steps 2 and 3 to add the second EtherNet/IP module to the remote rack.

1.8.2 Creating the Module in a Remote Rack

Note: To continue installing the MVI56E-MCMR 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 27).

1   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 MCMR, as this name will be linked automatically with the MSG paths, irrespective of the slot location.
Description	Enter a description for the module. Example: ProSoft communication module for Modbus Serial protocol communications.
Comm Format	Select **DATA-INT** *(*Very Important*)*
Slot	Enter the slot number in the rack where the MVI56E-MCMR 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. Fifty (50) milliseconds is usually a good starting value.
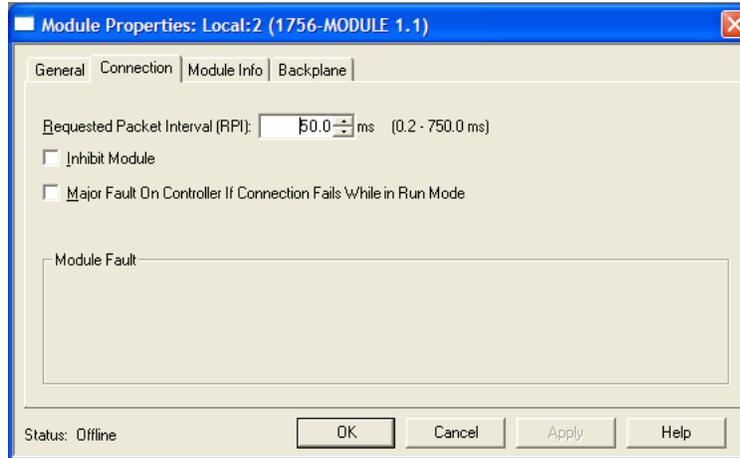
The **MVI56E-MCMR** module is now visible in the **I/O CONFIGURATION** section
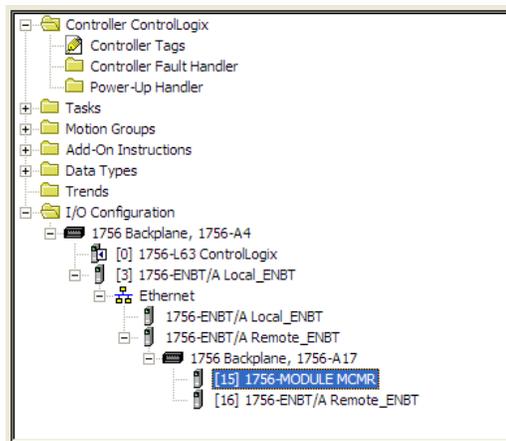
1.8.3 Creating the Module in a Local Rack

Note: If you are installing the MVI56E-MCMR 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 22).

1 Add the MVI56E-MCMR 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 MCMR, as this name will be linked automatically with the MSG paths, irrespective of the slot location.
Description	Enter a description for the module. Example: ProSoft communication module for Modbus Serial protocol communications.
Comm Format	Select **DATA-INT** *(*Very Important*)*
Slot	Enter the slot number in the rack where the MVI56E-MCMR 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.
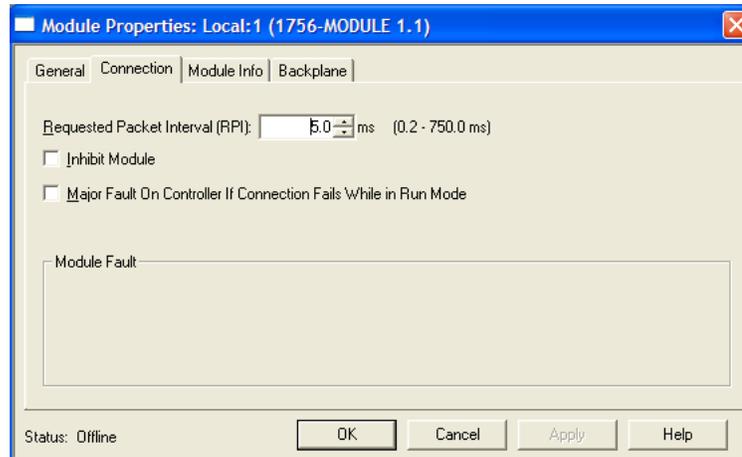
The **MVI56E-MCMR** module is now visible in the **I/O CONFIGURATION** section
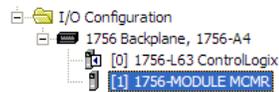
1.8.4 Importing the Ladder Rung

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…**
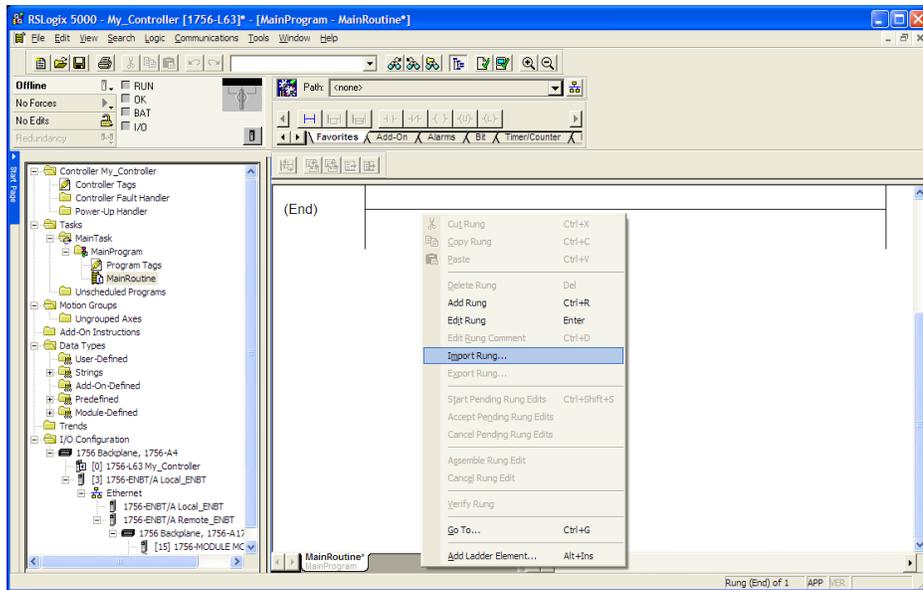
4 Navigate to the location on your PC where you saved (page 20) the Add-On Instruction (for example, "My Documents" or "Desktop"). Select the **MVI56(E)MCMR_ADDON_RUNG_V1_X.L5X** file
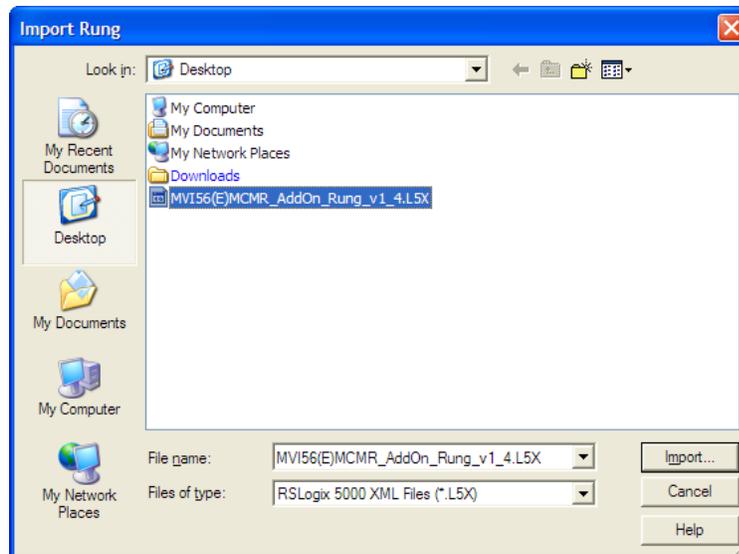
This action opens the **IMPORT CONFIGURATION** dialog box, showing the controller tags that will be created.
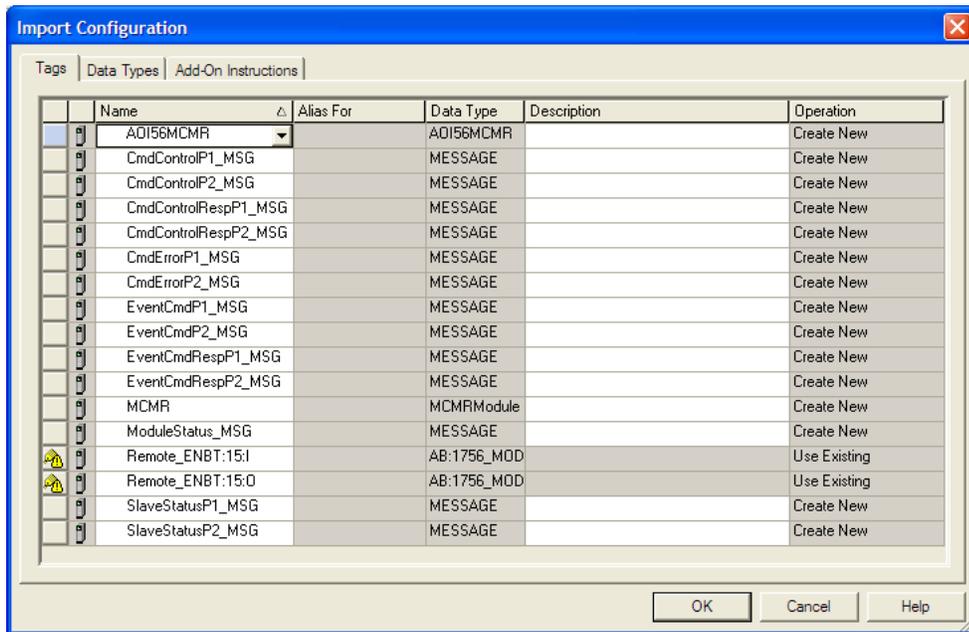
5 Locate the *Remote_ENBT:x:I Tag*, where x is the slot number of the module within the local rack. Rename this tag to: *Local:x:I*. Do the same for *Local:x:O*. This defines the backplane path to the module in a local rack.
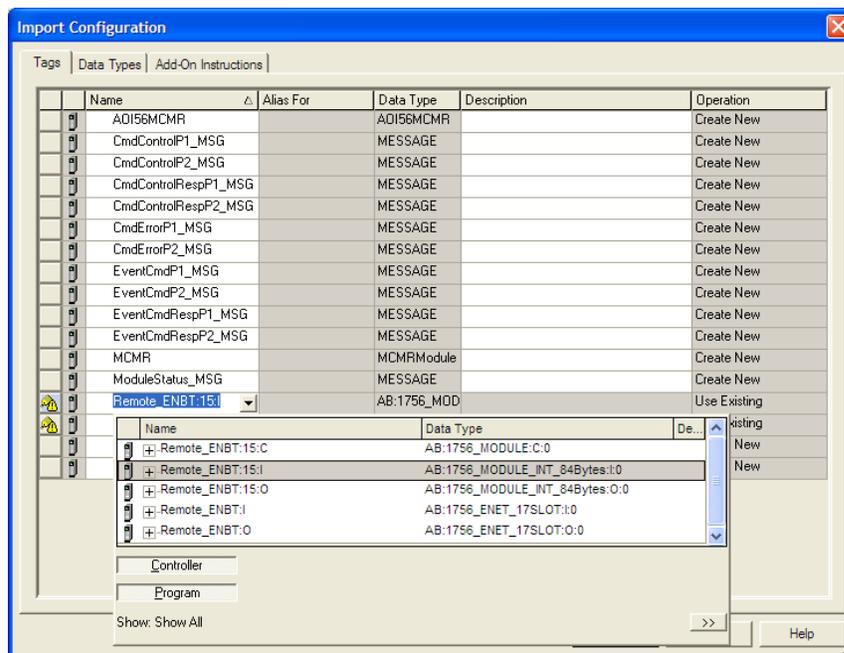
6 Click **OK** to confirm the import. RSLogix will indicate that the import is in progress:
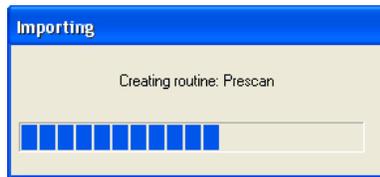
When the import is completed, the new rung with the Add-On Instruction will be visible as shown in the following illustration.
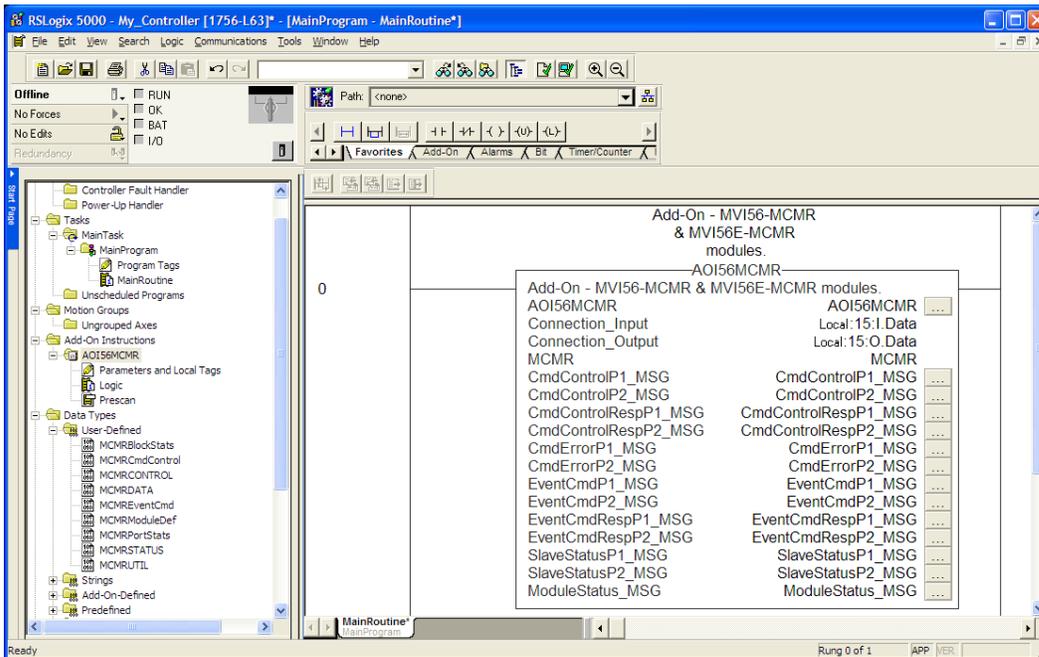
The procedure has also imported new User Defined Data Types, Controller Tags, and the Add-On instruction for your project.

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

Adding Multiple Modules (Optional)

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

1   In the **I/O CONFIGURATION** folder, click the right mouse button to open a shortcut menu, and then choose **NEW MODULE**.
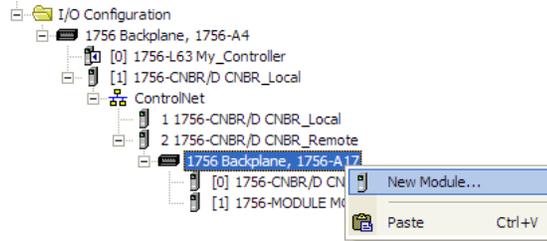
2   Select **1756-MODULE**
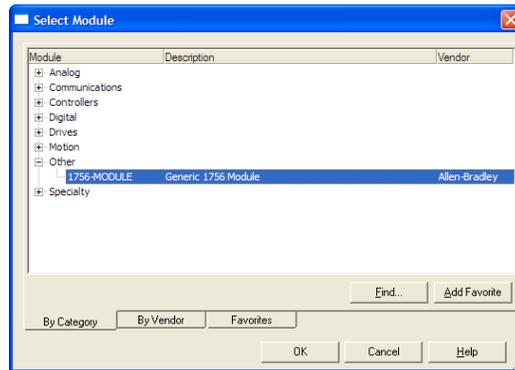
3   Fill the module properties as follows:

Parameter	Value
Name	Enter a module identification string. The recommended value is MCMR_2. You will need to link this name with the MSG paths for the AOI.
Description	Enter a description for the module. Example: Modbus Communication Module with Reduced Data Block
Comm Format	Select **DATA-INT** *(Very Important)*
Slot	Enter the slot number in the rack where the MVI56E-MCMR 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:
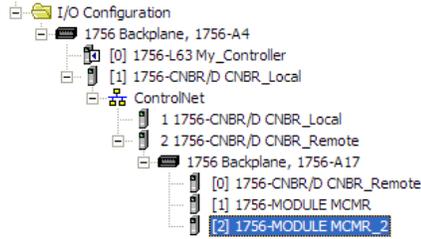
5 Expand the **TASKS** folder, and then expand the **MAINTASK** folder.
6 On the **MAINPROGRAM** folder, click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW ROUTINE**. As an alternative to creating a separate New Routine, you could skip to Step 8 and import the AOI for the second module into the same routine you created for the first module.
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 or an existing routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNG…**
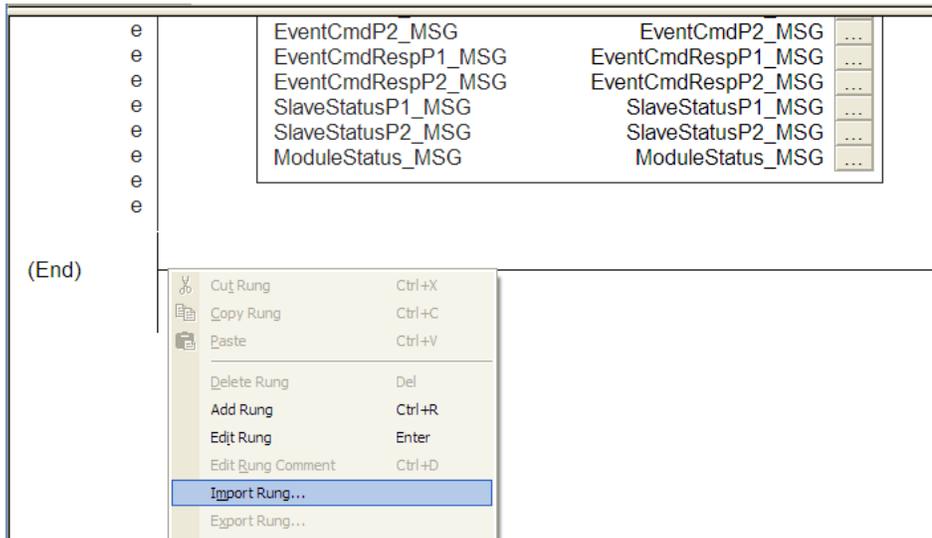
9 Select the **MVI56(E)MCMR_ADDON_RUNG_V1_4.L5X** file, and then click **IMPORT.**
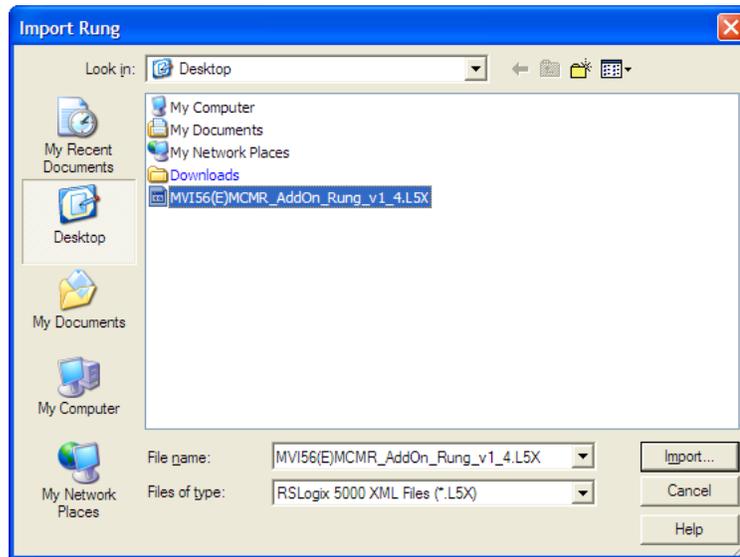
10 This action opens the **IMPORT CONFIGURATION** window, which shows the tags that will be imported.
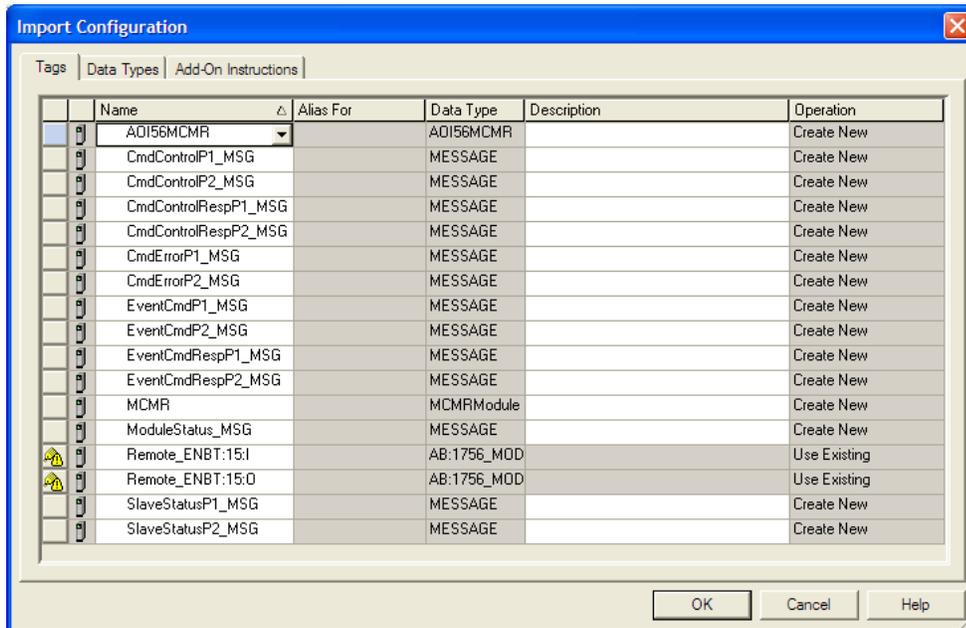
11 Associate the I/O connection variables to the correct module. The default values are Remote_ENBT:15:I and Remote_ENBT:15:I so these require change.
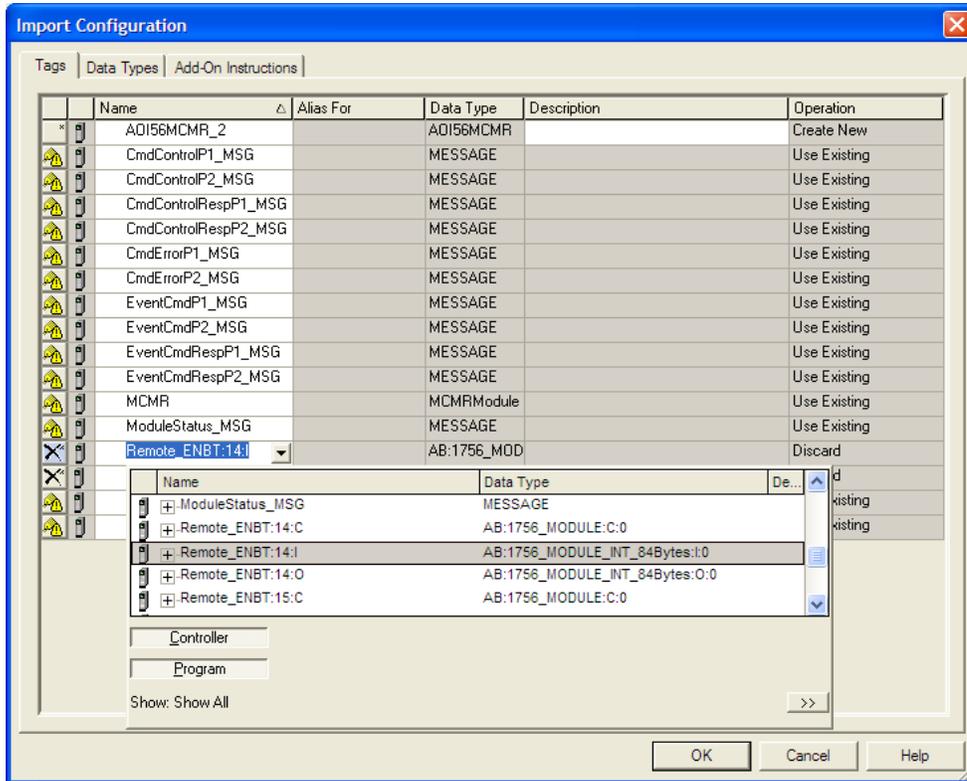
12 Change the default tags **MCMR** and **AOI56MCMR** to avoid conflict with existing tags. In this procedure, you will append the string "_2" to all tags to be imported as shown in the following illustration.
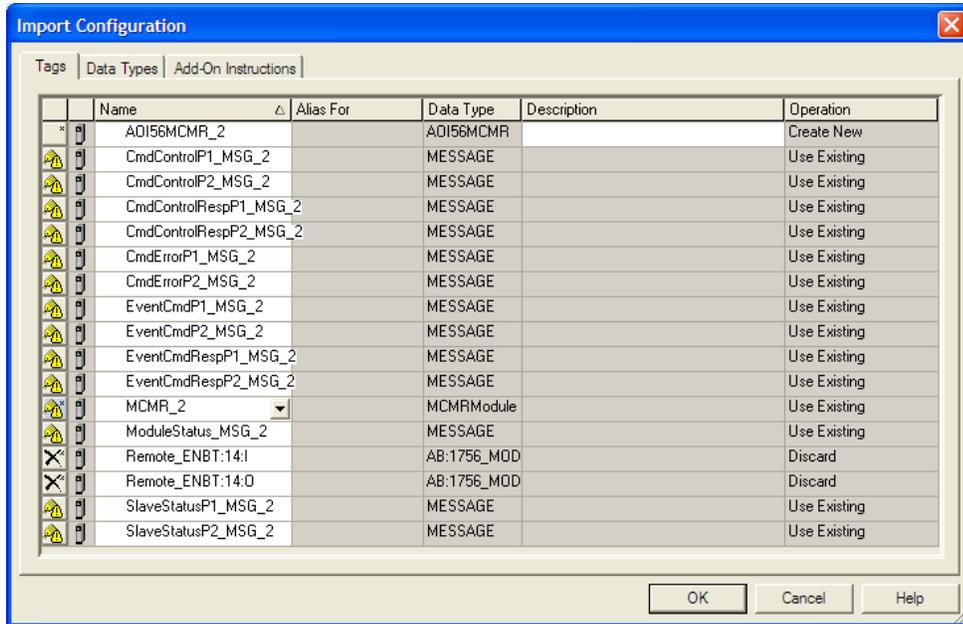
13 Click **OK** to confirm.
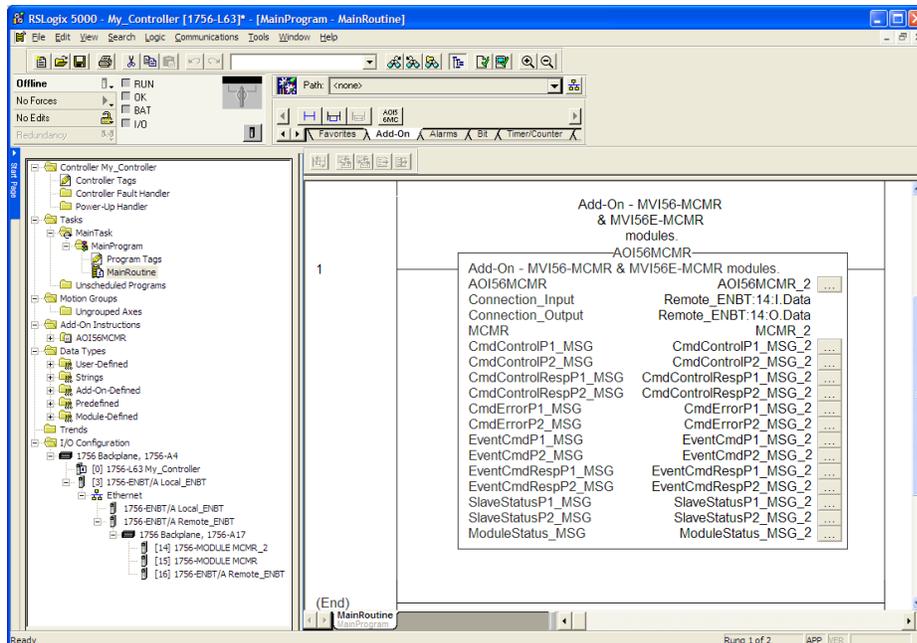
Configuring the Path for Message Blocks

If you used the recommended name for the module (MCMR) to import the first Add-On Instruction, the MSG paths will be associated correctly with the module. For additional modules, you must configure the message path for each MSG instruction to address the correct module.

1　In the Add-On Instruction, click the **[...]** button next to each **MSG** tag to open the **MESSAGE CONFIGURATION TAG**.
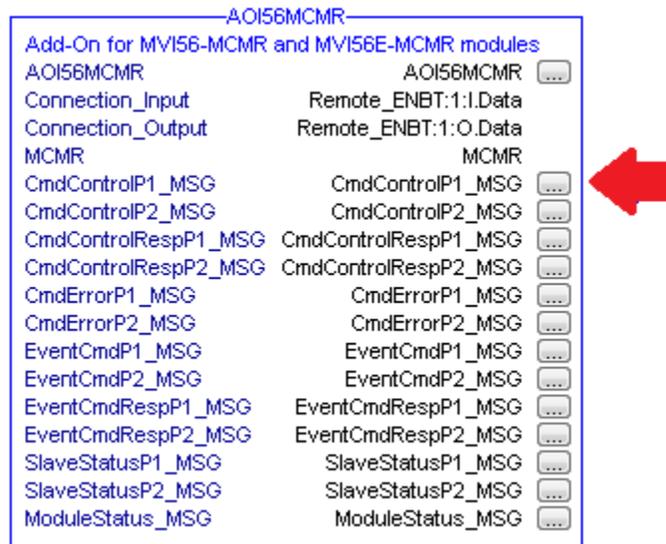
2　Click the **COMMUNICATION** tab and click the **BROWSE** button as follows.
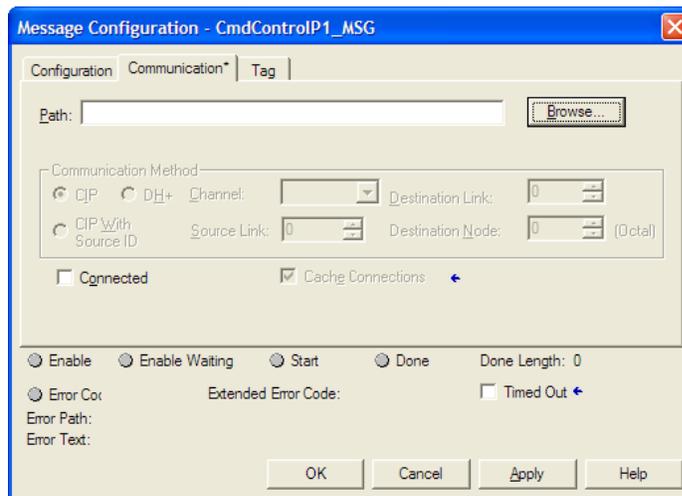
3 Select the module to configure the message path.
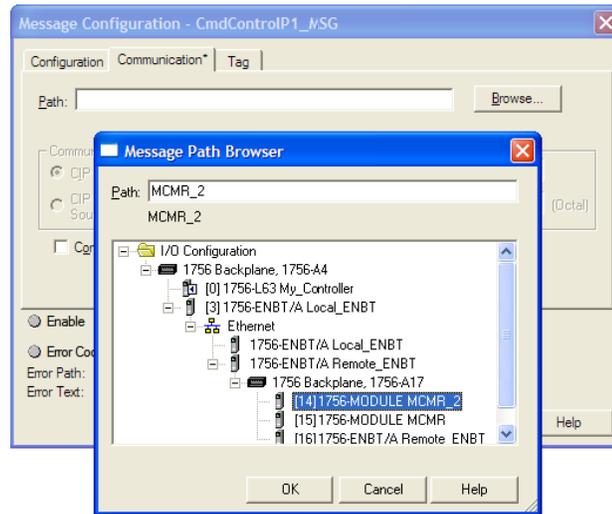
4 Repeat these steps for each **MSG** tag, and for each additional MVI56E-MCMR module.

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

1.8.5 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. You can configure the start register and size of each area. The size of each area you configure must match the Add-On instruction controller tag array sizes for the **READDATA** and **WRITEDATA** arrays.

The MVI56E-MCMR sample program is configured for 600 registers of **READDATA** and 600 registers of **WRITEDATA,** which is sufficient for most applications. This topic describes how to configure user data for applications requiring more than 600 registers of ReadData and WriteData.

Important: Because the module pages data in blocks of 40 registers at a time, you must configure your user data in multiples of 40 registers.
Caution: When you change the array size, RSLogix may reset the MCMR tag values to zero. To avoid data loss, be sure to save your settings before continuing.

1   In the **CONTROLLER ORGANIZATION** window, expand the **DATA TYPES** and
USER-DEFINED folders, and then double-click **MCMRDATA.** This action opens
an edit window for the MCMRDATA data type.
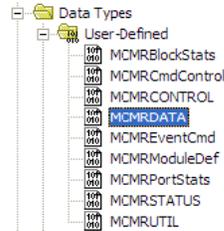
2   In the edit window, change the value of the **READDATA** array from **INT[600]** to
INT[1000] as shown, and then click **APPLY.**

> **Note**: If RSLogix resets your data values, refer to the backup copy of your program to re-enter your
> configuration parameters.
> **Important:** When you change the ReadData and WriteData array sizes in RSLogix, you must also
> change the Read Register Count and Write Register Count values in ProSoft Configuration Builder
> (page 56).

3   In ProSoft Configuration Builder, navigate to the **BACKPLANE CONFIGURATION**
tag (page 56), and double click to open an edit window. Change the **READ
REGISTER COUNT** value to match the value you entered in RSLogix for the
ReadData data type.
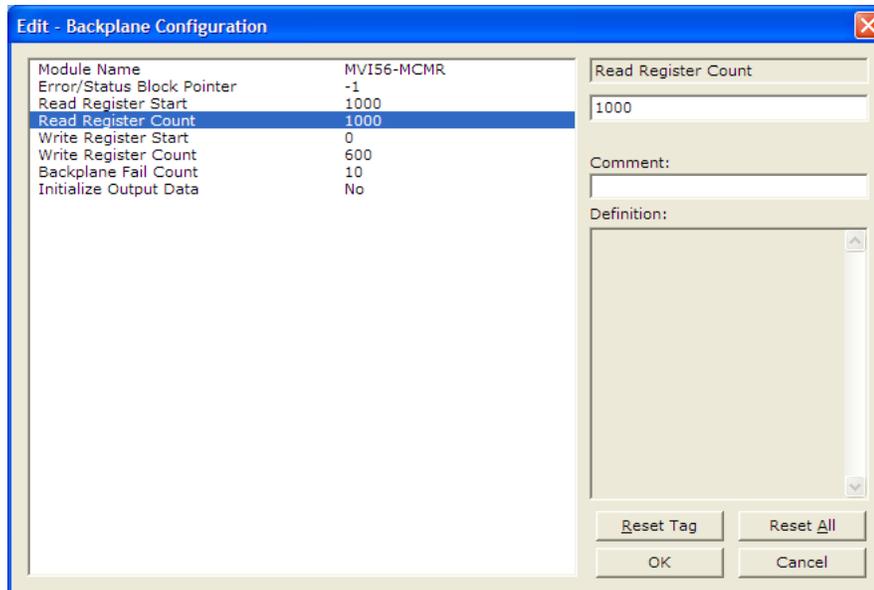
4 Save and download the sample program to the processor.

To modify the **WRITEDATA** array, follow the steps in this topic, but substitute **WRITEDATA** for ReadData throughout. Also, 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 in ProSoft Configuration Builder.

1.9   Connecting Your PC to the ControlLogix Processor

There are several ways to establish communication between your PC and the ControlLogix processor. The following steps show how to establish communication through the serial interface. It is not mandatory that you use the processor's serial interface. You may access the processor through whatever network interface is available on your system. Refer to your Rockwell Automation documentation for information on other connection methods.

1   Connect the right-angle connector end of the cable to your controller at the communications port.
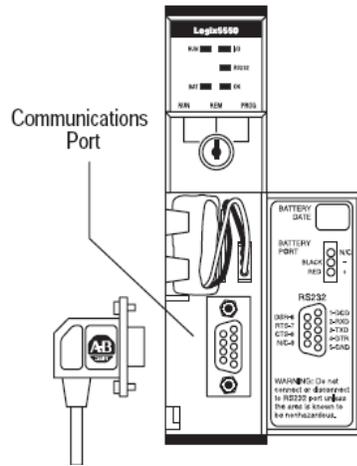
2   Connect the straight connector end of the cable to the serial port on your computer.
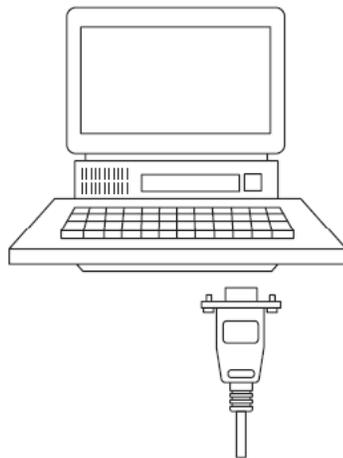
1.10 Downloading the Sample Program to the Processor

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.

1.10.1 Configuring the RSLinx Driver for the PC COM Port

If RSLogix is unable to establish communication with the processor, follow these steps.

1 Open *RSLinx*.

2 Open the **COMMUNICATIONS** menu, and choose **CONFIGURE DRIVERS**.
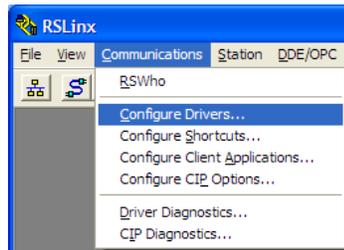
This action opens the *Configure Drivers* dialog box.
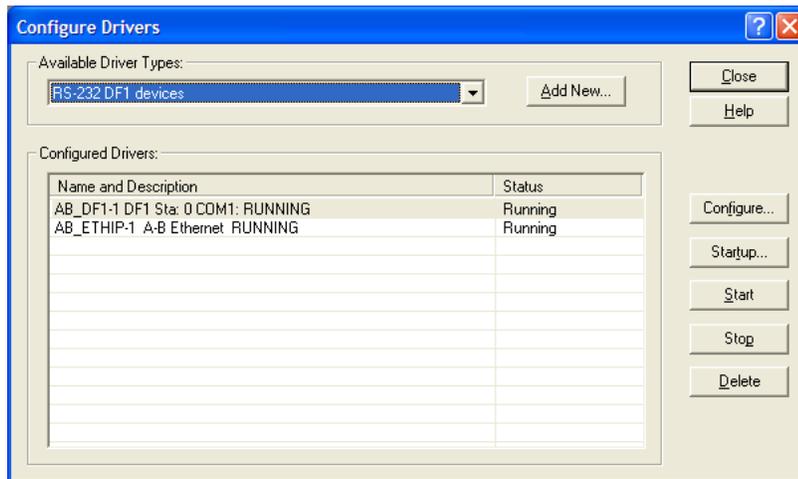
Note: If the list of configured drivers is blank, you must first choose and configure a driver from the Available Driver Types list. The recommended driver type to choose for serial communication with the processor is *RS-232 DF1 Devices*.

3 Click to select the driver, and then click **CONFIGURE**. This action opens the
Configure RS-232 DF1 Devices dialog box.
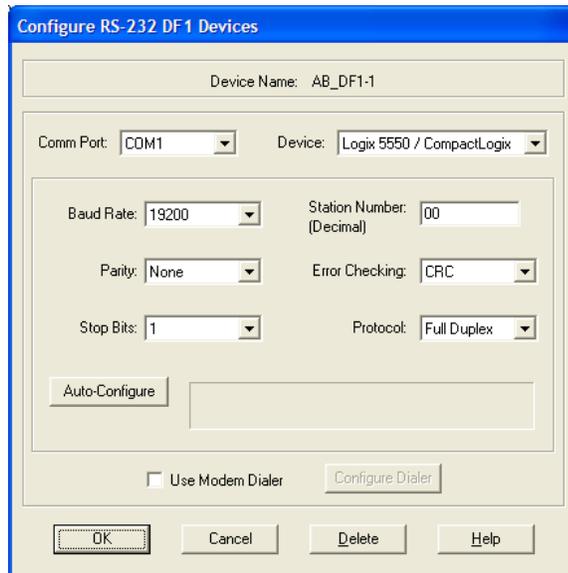
4 Click the **AUTO-CONFIGURE** button. RSLinx will attempt to configure your
serial port to work with the selected driver.
5 When you see the message *Auto Configuration Successful*, click the **OK**
button to dismiss the dialog box.

Note: If the auto-configuration procedure fails, verify that the cables are connected correctly
between the processor and the serial port on your computer, and then try again. If you are still
unable to auto-configure the port, refer to your RSLinx documentation for further troubleshooting
steps.

2   Configuring the MVI56E-MCMR Module

In This Chapter

2.1 Installing ProSoft Configuration Builder

The ProSoft Configuration Builder (PCB) software is used to configure the module. You can find the latest version of the ProSoft Configuration Builder (PCB) on our web site: http://www.prosoft-technology.com, or you can install it from the ProSoft Solutions DVD. The installation filename contains the PCB version number. For example, **PCB_4.1.0.4.0206.EXE**.

If you are installing PCB from the ProSoft website:

1   Open a browser window and navigate to http://www.prosoft-technology.com/pcb.
2   Click the download link for ProSoft Configuration Builder, and save the file to your Windows desktop.
3   After the download completes, double-click on the PCB installation file, and follow the instructions that appear on the screen.
4   If you want to find additional software specific to your MVI56E-MCMR, enter the model number into the website search box and press the Enter key.

If you are installing PCB from the ProSoft Solutions DVD:

1   Insert the ProSoft Solutions DVD into your computer's DVD drive and wait for the ProSoft Installation program to start.
2   If the ProSoft Installation program does not start, open the Windows file Explorer, navigate to the DVD, and double-click on the ProSoft_DVD.exe file.
3   In the ProSoft Installation program window, click **PROSOFT CONFIGURATION BUILDER**.
4   Follow the instructions that appear on the screen.
5   If you want to find additional software specific to your MVI56E-MCMR, select the correct **PLATFORM** and **PRODUCT** before proceeding.

2.2 Using ProSoft Configuration Builder Software

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

Note: During startup and initialization, the MVI56E-MCMR 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-MCMR in ProSoft Configuration Builder

MVI56E-MCMR modules are fully backward-compatible with MVI56-MCMR modules. However, you will need to convert your MVI56-MCMR configuration in *ProSoft Configuration Builder* to a form that your new MVI56E-MCMR 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.
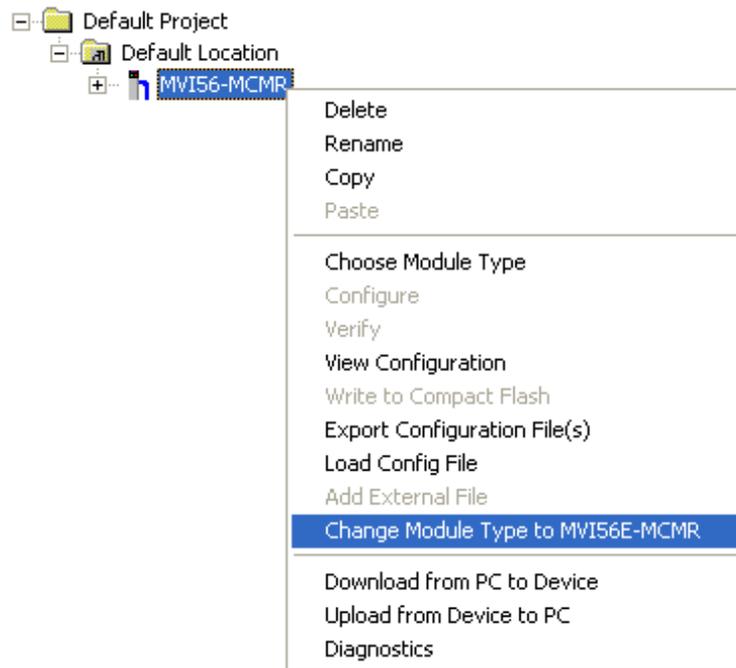
2 On the shortcut menu, select **CHANGE MODULE TYPE TO MVI56E-MCMR**.

2.2.2 Setting Up the Project

To begin, start **PROSOFT CONFIGURATION BUILDER** (PCB).
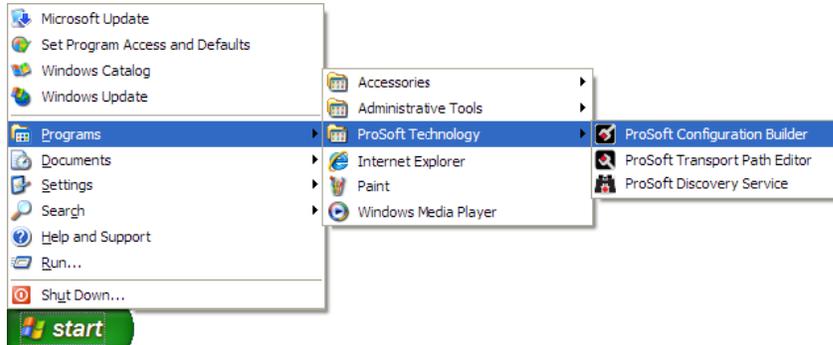
If you have used other Windows configuration tools before, you will find the screen layout familiar. *PCB*'s window consists of a tree view on the left, and an information pane and a configuration pane on the right side of the window. When you first start *PCB*, the tree view consists of folders for *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.
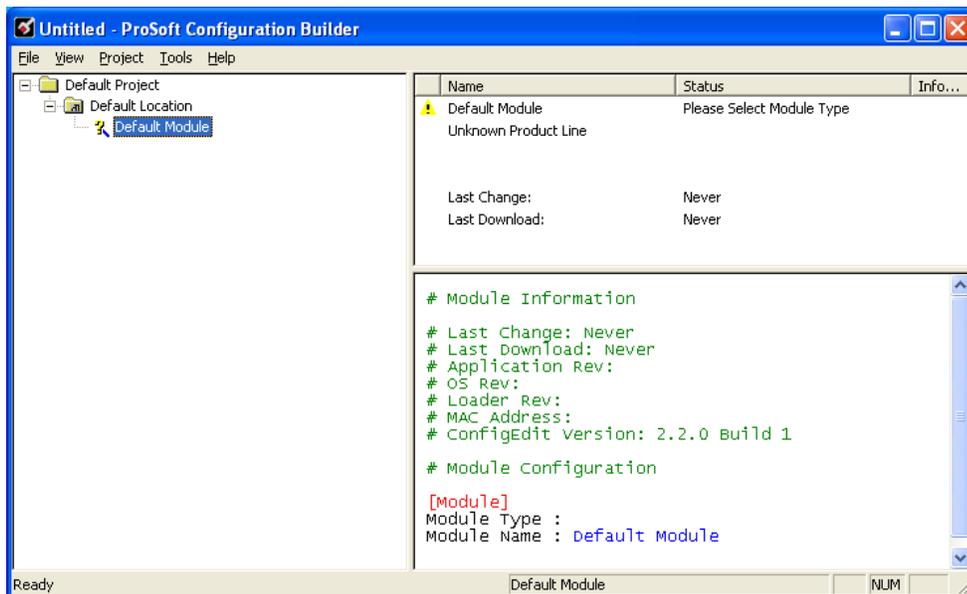
Your first task is to add the MVI56E-MCMR module to the project.

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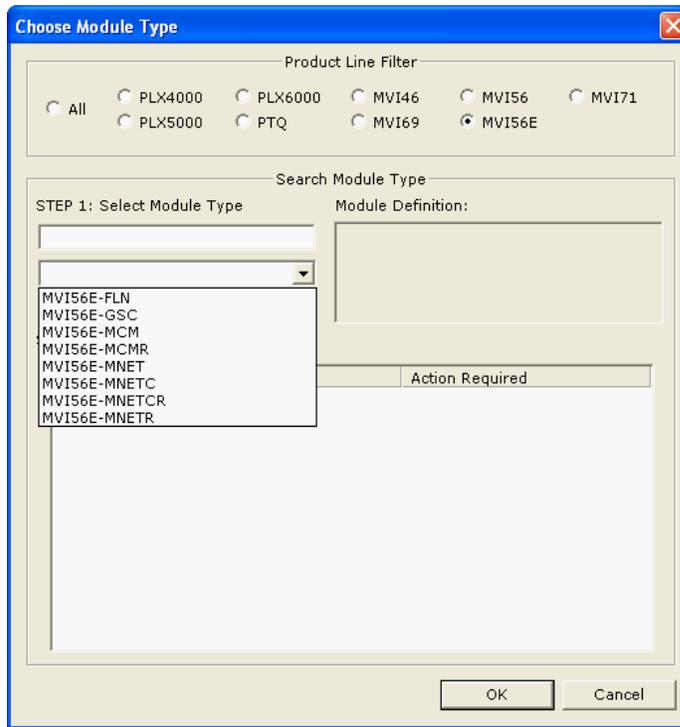
3 In the *Product Line Filter* area of the dialog box, select **MVI56E**. In the *Select Module Type* dropdown list, select **MVI56E-MCMR**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window.

2.2.3 Setting Module Parameters

Notice that the contents of the information pane and the configuration pane changed when you added the MVI56E-MCMR module to the project.
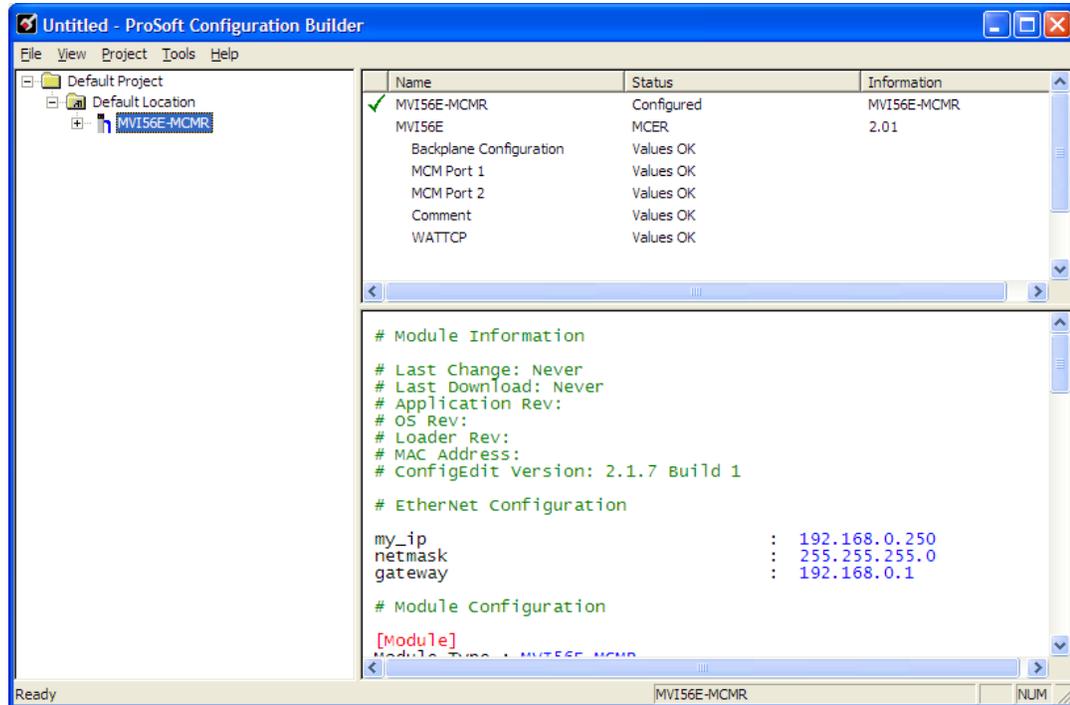
At this time, you may wish to rename the *Default Project* and *Default Location* folders in the tree view.

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 on the **[+]** sign next to the module icon to expand module information.
2 Click on 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.

Creating Optional Comment Entries

1 Click the **[+]** to the left of the ⊞ 🕂 Comment icon to expand the module comments.

2 Double-click the ⏚🅱 Module Comment icon. The *Edit - Module Comment* dialog box appears.
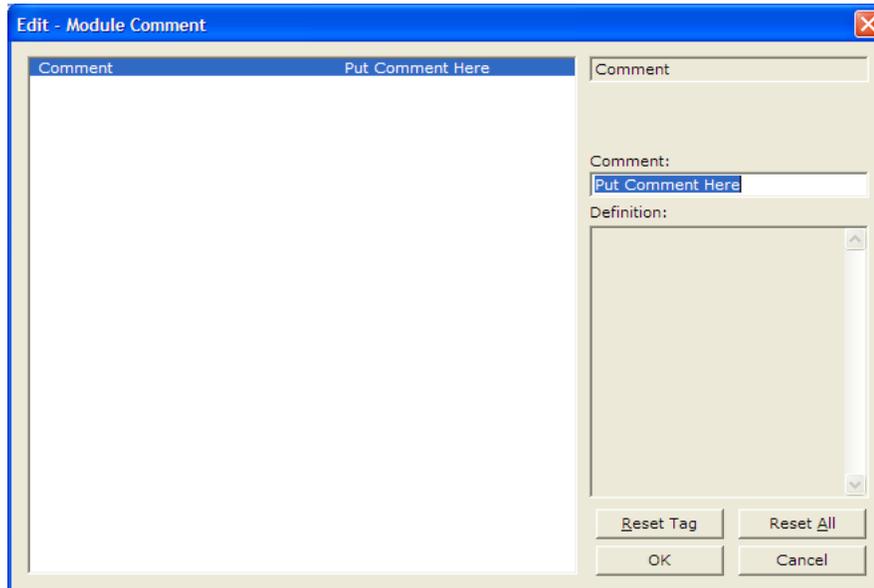
3 Enter your comment and click **OK** to save your changes.

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.**

2.3 Configuration as a Modbus Master

2.3.1 Overview

This section describes how to configure the module as a **MODBUS MASTER** device. The Master is the only device on a Modbus network that can initiate communications. A Master device issues a request message, and then waits for the slave to respond. When the slave responds, or when a timeout has occurred, the Modbus Master will then execute the next command in the list.

The following ProSoft Configuration Builder sections contain the Modbus Master configuration. You must configure all three sections.

1   The **BACKPLANE CONFIGURATION** section sets up the backplane communication between the MVI56E-MCMR module and the ControlLogix processor (page 56). These settings include register addresses for ReadData and WriteData. You can configure up to 5000 data registers in the module to exchange data with the ControlLogix processor.

2   The **MODBUS PORT1** and **MODBUS PORT 2** sections configure the Modbus application serial ports (page 57). These sections configure parameters such as baud rate, parity, data bits, stop bits, and command response timeout.

3   The **MODBUS PORT 1 COMMANDS** and **MODBUS PORT 2 COMMANDS** sections define a polling table (command list) for the Modbus Master (page 60). These sections contain the addresses for devices on the network, the types of data (Modbus Function Codes) to read from and write to those devices, and the location to store the data within the module's 5000 data registers.
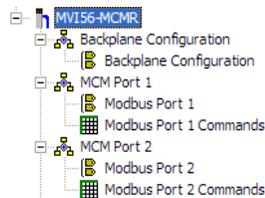
2.3.2 Backplane Configuration

The **BACKPLANE CONFIGURATION** section defines the 5000 data registers to use for read and write data within the MVI56E-MCMR module. You will use these data read and write locations in the Internal Address tag within each Master Command (page 60). The following illustration shows the values from the sample program.
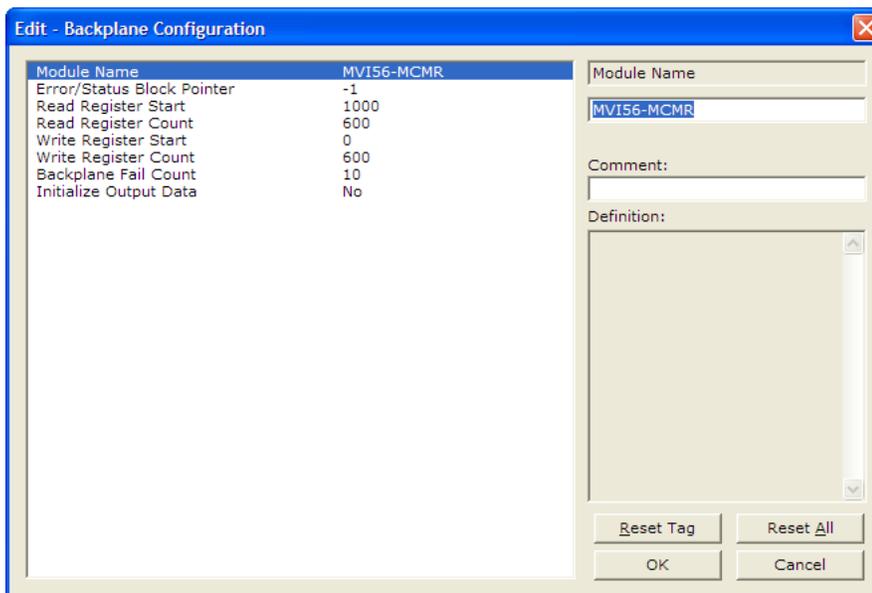
The **WRITE REGISTER START** parameter determines the starting register location for **WRITEDATA[0** to **599]**. The **WRITE REGISTER COUNT** determines how many of the 5000 registers to use send data to the module. The sample ladder file uses 600 registers for write data, labeled **MCMR.DATA.WRITEDATA[0** to **599].**

Parameter	Description
Error/Status Block Pointer	Used mainly when the module is configured as a Slave. This parameter places the STATUS data into the database of the module.
Read Register Start	Specifies the starting register in the module's database for sending data to the ReadData controller tag array in the ControlLogix processor.
Read Register Count	Sets how many registers of data the MVI56E-MCMR module will send to the ControlLogix processor's ReadData array. This value is best if set to a multiple of 200 (40 for MCMR).
Write Register Start	Specifies where in the 5000 register module memory to start placing data sent from the WriteData tag array in the ControlLogix processor.
Write Register Count	Specifies how many registers of data the MVI56E-MCMR module will request from the ControlLogix processor. Because the module pages data in blocks of 40 words, this number is best if it is evenly divisible by 40.
Backplane Fail Count	Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network. Typically used when the module is configured as a Slave.

The sample configuration values configure the module database to store
WRITEDATA[0 to **599]** in registers 0 to 599, and **READDATA[0 TO 599]** in registers
1000 to 1599, as shown in the following illustration.
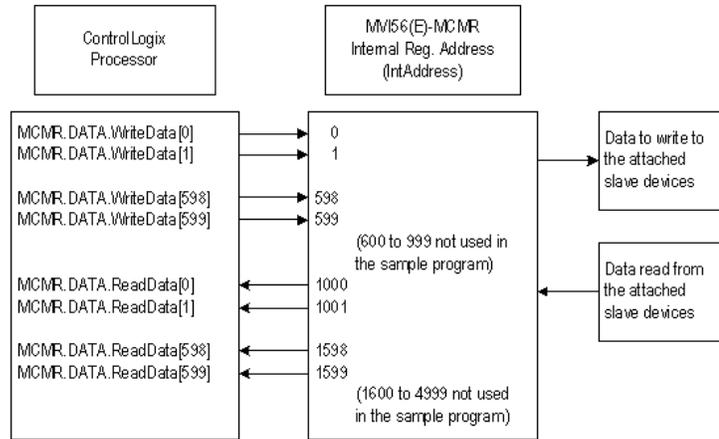
Important: If you need to configure different values for the Read Register Count and Write
Register Count parameters, you must also configure the same values in the user-defined data type
MCMRData in the sample program (page 39).

2.3.3 Port Configuration

The **MODBUS PORT X** configuration parameters are used when the module is
configured as a Modbus Master device. Port 1 and Port 2 each have their own
set of configuration parameters.

Note: Any changes made within the configuration file must be downloaded to the MVI56E-MCMR
module from ProSoft Configuration Builder.

In ProSoft Configuration Builder, expand the **MVI56E-MCMR** node, and then
expand the **MCM PORT 1** node. Double-click the **MODBUS PORT 1** icon. In the
EDIT - MODBUS PORT 1 dialog box, click to highlight the *Type* parameter, and then
select **MASTER** from the dropdown list.

The following parameters are displayed when the *Type* parameter is set to **MASTER**.
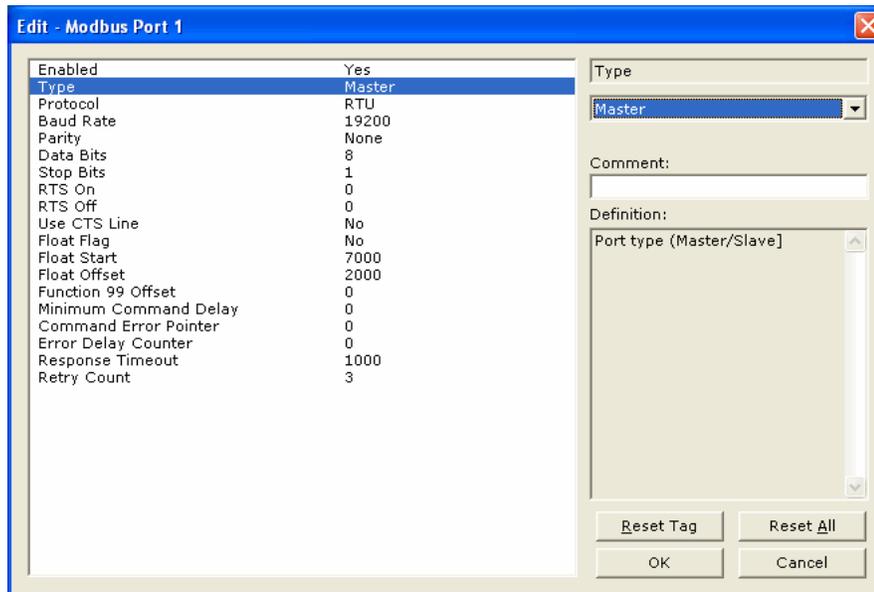
The following table describes the parameters in the **EDIT – MODBUS PORT 1** dialog box when the *Type* parameter is set to **MASTER**.

Parameter	Description
Enabled	**1 = ENABLE PORT**, 0 = disable port
Type	**0 = MASTER**, 1 = Slave
Protocol	**0 = MODBUS RTU MODE**, 1 = Modbus ASCII mode
Baud Rate	Sets the baud rate for the port. Valid values for this field are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 384 or 3840 (for 38,400 baud), 576 or 5760 (for 57,600 baud) and 115,1152, or 11520 (for 115,200 baud)
Parity	**0** = None, **1** = Odd, **2** = Even
Data Bits	Modbus RTU mode = 8 Modbus ASCII mode = **8** or **7**
Stop Bits	Valid values are **1** or **2**.
RTS On	**0** to **65535** milliseconds to delay after RTS line is asserted on the port before data message transmission begins. This delay can be used to allow for radio keying or modem dialing before data transmission begins.
RTS Off	**0** to **65535** milliseconds to delay after data message is complete before RTS line is dropped on the port.
Use CTS Line	**NO** or **YES** This parameter is used to enable or disable hardware handshaking. The default setting is **No** hardware handshaking, CTS Line not used. Set to **No** if the connected devices do not need hardware handshaking. Set to **YES** if the device(s) connected to the port require hardware handshaking (most modern devices do not). If you set this parameter to **YES**, be sure to pay attention to the pinout and wiring requirements to ensure that the hardware handshaking signal lines are properly connected; otherwise communication will fail.

Parameter	Description
Float Flag	**YES** or **No** Enables or disables use of floating data type
Float Start	**0** to **32767** Register offset in message for floats
Float Offset	**0** to **3998** Internal address for floats
Function 99 Offset	**1** to **247** Modbus node address for this port on the network
Minimum Command Delay	**0-65535** milliseconds The amount of delay in milliseconds to be inserted after receiving a Slave response or encountering a response timeout before retrying the command or sending the next command on the list. Use this parameter to slow down overall polling speed and spread out commands on networks with Slaves that require additional gaps between messages.
Command Error Pointer	Internal DB location to place command error list Each command will reserve one word for the command error code for that command. See Verify Communication (page 107). **CMDERRPTR** value should be within the range of the **READDATA** array. See Backplane Configuration (page 56).
Error Delay Counter	This parameter specifies the number of poll attempts to be skipped before trying to re-establish communications with a slave that has failed to respond to a command within the time limit set by the *Response Timeout* parameter. After the slave fails to respond, the master will skip sending commands that should have been sent to the slave until the number of skipped commands matches the value entered in this parameter. This creates a sort of *slow poll* mode for slaves that are experiencing communication problems.
Response Timeout	**0** to **65535** milliseconds response timeout for command before it will either reissue the command, if **RETRYCOUNT > 0.** If the RetryCount =0 or if the designated number of retries have been accomplished, then the Master will move on to the next command in the list.
Retry Count	Number of times to retry a failed command request before moving to the next command on the list.

2.3.4 Master Command Configuration

This topic describes the communications with the Master port and slave devices that are connected to that port.

In ProSoft Configuration Builder, expand the **MVI56E-MCMR** node, and then double-click the **MODBUS PORT 1 COMMANDS** icon.

Parameter	Description
Enable	**0** = Disabled
	Command will not be executed, but can be enabled using the Command Control option in ladder logic.
	1 = Enabled
	Command is enabled and will be sent out to the target device.
	2 = Conditional Write
	Only for Func 5, 15, 6, or 16. Data will be sent to the target device only when the data to be written has changed in the source registers of the module's internal database.

Parameter	Description
Internal Address	**0** to **4999** for Register-level commands
	0 to **65535** for Bit-level commands
	Determines the starting address in the module's 5000-register database that will be affected by the command. For a Read command, this will determine where the data will begin to be placed in the module database after it has been read from a slave. For read commands, you should configure this value so that the data will be placed in the range of module memory designated for ReadData, as defined in the Backplane Configuration section of this configuration file. For write commands, the **INTERNAL ADDRESS** determines where to begin obtaining the data to write to the slave device. This must be a location that is in the WriteData area of module memory, as defined in the Backplane Configuration section of this configuration file.
	Note: When using a bit-level command, you must define this field at the bit level. For example, when using a Function Code 1, 2 for a Read command, you must have a value of 16000 to place the data in MCM.ReadData[0] (ReadStartRegister = 1000 * 16 bits per register = 16000).
Poll Interval	**0** to **65535**
	The Poll Interval is the number of seconds that the Master will wait between successive executions of this command. Set to zero (0) for the fastest possible polling.
	This parameter can be used to prioritize and optimize network traffic by assigning low values to high-priority poll requests and assigning higher values to less important data poll commands.
Reg Count	**1** to **125** words for Function Codes 3, 4, and 16 (Register-level)
	1 to **2000** for Function Codes 1, 2, and 15 (Bit-level)
	Sets how many continuous words (Function Codes 3, 4, and 16) or bits (Function Codes 1, 2, and 15) to request from the slave device.
	Note: These values are the maximum allowed in the Modbus protocol. Some devices may support fewer words or bits per command than these maximum values.
Swap Code	**NO CHANGE**, **SWAP WORDS**, **SWAP WORDS & BYTES**, **SWAP BYTES**
	Typically used when reading floating-point data. Swaps the data read from the slave device before it is placed into the module memory. For example, you receive 4 bytes of data from the slave (ABCD).
	NO CHANGE = No swapping (ABCD)
	SWAP WORDS = Word pairs switched (CDAB)
	SWAP WORDS AND BYTES = Bytes and words switched (DCBA)
	SWAP BYTES = Bytes swapped (BADC)
Node Address	**1** to **247**
	Modbus Slave Device Address of the device on the network to read data from, or write data to. Valid addresses are **1** to **247**. Address **0** is reserved for broadcast write commands (will broadcast a Write command to all devices on the network).

Parameter	Description
ModBus Function	**1**, **2**, **3**, **4**, **5**, **6**, **15**, and **16** (when viewed in the .CFG text file)
	The Modbus Function Code determines what kind of command to send to the slave device. Valid code numbers and descriptions for this field are as follows:
	Note: The Modbus protocol specifies that the valid address range for each Modbus data type can be x00001 to x65535. Most newer Modbus devices support this addressing range. However, some older Modbus devices may only support addresses that range from x0001 to x9999.
	FC 1 = Read Coil (0X)
	Use this Function Code to read Modbus Coil addresses 000001 to 065535 (or 0x0001 to 0x9999). These are read/write single-bit binary values. Use Function Code 5 or 15 to write to these Coil addresses.
	FC 2 = Read Input (1X)
	Use this Function Code to read Modbus Input Status addresses 100001 to 165535 (or 1x0001 to 1x9999). These are read-only single-bit binary values.
	FC 3 = Read Holding Registers (4X)
	Use this Function Code to read Modbus Holding Register addresses 400001 to 465535 (or 4x0001 to 4x9999). These are read/write 16-bit word values. Use Function Code 6 or 16 to write to these Holding Registers.
	FC 4 = Read Input Registers (3X)
	Use this Function Code to read Modbus Input Register addresses 300001 to 365535 (or 3x0001 to 3x9999). These are read-only 16-bit word values.
	FC 5 = Force Single Coil (0X)
	Use this Function Code to write to Modbus Coil addresses. This command will write to only one coil per command. Use Function Code 15 to write to multiple coils in the same command.
	FC 6 = Preset Single Register (4X)
	Use this Function Code to write to Modbus Holding Registers. This command will write to only one register per command. Use Function Code 16 to write to multiple registers in the same command.
	FC 15 = Force Multiple Coils (0X)
	Use this Function Code to write multiple Coil values with one command.
	FC 16 = Preset Multiple Registers (4X)
	Use this Function Code to write multiple Holding Register values with one command.

Parameter	Description
MB Address in Device	Specifies the starting Modbus bit or register address where data will begin being read from or written to the slave device. With Modbus, to read an address of 40001, what will actually be transmitted out port is Function Code 03 (one byte) with an address of 00 00 (two bytes). This means that to read an address of 40501, use FC 3 with a MB Address in Device of 500.
	This applies to all Modbus addresses. Below are some examples that will help with your **MB ADDRESS IN DEVICE** configuration:
	Function Codes 1, 5, or 15 for reading or writing Modbus Coils
	MB Address in Device setting = Modbus Coil address in the Slave device – 0001
	For Modbus Coil address 0001: MB Address in Device = 0
	For Modbus Coil address 1378: MB Address in Device = 1377
	Function Code 2
	MB Address in Device setting = Modbus Input Status address in the Slave device - 10001
	For Modbus address 10001: MB Address in Device = 0
	For Modbus Input Status address 10345: MB Address in Device = 344
	Function Codes 3, 6, or 16
	MB Address in Device setting = Modbus Holding Register address in the Slave device – 40001
	For Modbus Holding Register address 40001; MB Address in Device = 0
	For Modbus Holding Register address 40591; MB Address in Device = 590
	Function Code 4
	MB Address in Device setting = Modbus Input Register address in the Slave device – 30001
	For Modbus Input Register address 30001: MB Address in Device = 0
	For Modbus Input Register address 34290; MB Address in Device = 4289

2.3.5 Other Modbus Addressing Schemes

While the above information will handle most devices, some device manufacturers show their Modbus addressing differently.

The two most common schemes are six-digit addressing (400101, 301000, and so on) and some devices show their addressing already as an offset address (the address that actually goes out on the Modbus communication line). When addresses are given as actual offset addresses, they are usually given as a hexadecimal (base 16) number. This is an example.

Actual Values (Input Registers) Addresses: 0200 to 0E1F

STATUS	0200	Switch Input Status
	0201	LED Status Flags
	0202	LED Attribute Flags
	0203	Output Relay Status Flags

If your device manufacturer gives you addressing like this "Input Registers" example above, then you will use Function Code 4 to convert the hexadecimal value to a decimal equivalent value, and place the decimal value in the **MB ADDRESS IN DEVICE** field. So for this example device, use Modbus Function = 4 (Input Registers) with a **MB ADDRESS IN DEVICE** of 512 decimal (200h) to read the "Switch Input Status" value.

What if my slave shows addresses such as 400,001 or 301,345?

For 6-digit addressing, use the same function codes and configuration as shown above, but subtract higher values; 100001 instead of 10001; 300001 instead of 30001; and 400001 instead of 40001.

Function Codes 1, 5, or 15 **MB Address in Device** = Modbus Coil address in slave device - 000001

- For Modbus Coil address 000001; MB Address in Device = 0
- For Modbus Coil address 001378; MB Address in Device = 1377

Function Code 2 **MB Address in Device =** Modbus Input Status address in slave device - 100001

- For Modbus Input Status address 100001; MB Address in Device = 0
- For Modbus Input Status address 100345; MB Address in Device = 344

Function Codes 3, 6, or 16 **MB Address in Device** = Modbus Holding Register address in slave device - 400001

- For Modbus Holding Register address 400001; MB Address in Device = 0
- For Modbus Holding Register address 400591; MB Address in Device = 590

Function Code 4 **MB Address in Device** = Modbus Input Register address in device - 300001

- For Modbus Input Register address 300001; MB Address in Device = 0
- For Modbus Input Register address 304290; MB Address in Device = 4289
    For example:

If our device listed above shows its addressing as follows:

Variable Name	Data Type		Address
Switch_Input_Status	INT	▼	300513
LED_Status_Flags	INT	▼	300514
LED_Attribute_Flags	INT	▼	300515
Output_Relay_Status_Flags	INT	▼	300516

Then: To read "Switch_Input_Status", you would use Function Code 4 and use a MB Address in Device of 512.

2.3.6 Master Command Examples

Read Holding Registers 4x (Modbus Function Code 3)

The 4x Holding Registers are used for storing analog values such as pressure, temperature, current, program counters, timer accumulators and presets, and so on. Holding Registers store values in 16-bit memory registers. These 16-bit values can be interpreted in different ways that allow Holding Registers to hold many different data types, such as 8-bit, 16-bit, 32-bit, or 64-bit signed or unsigned integers, as well as 32-bit or 64-bit floating-point data (page 74) and other data types.

The following illustration shows the correct parameter values to create a command to read Modbus addresses 40001 to 40010 from Modbus Slave Device Address 1.
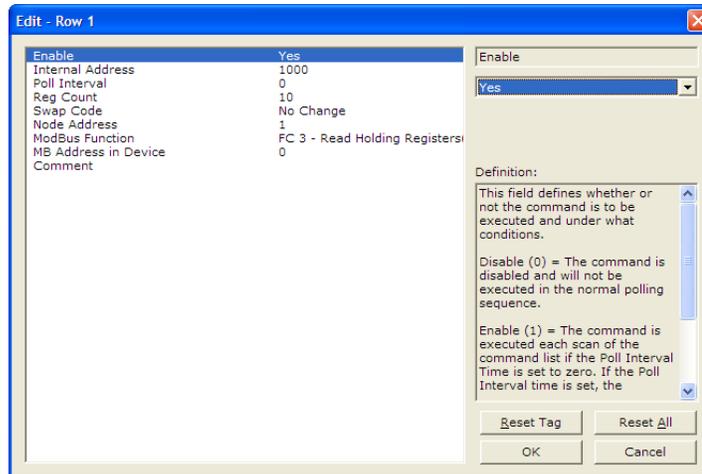
Parameter	Description
Enable = YES	The module will send the command every time it goes through the command list.
Internal Address = 1000	Begins placing the data read from the slave device into the module at address 1000. Internal Address 1000 of the module memory will be copied into the tag **MCMR.DATA.READDATA[0]**, assuming MCMR.CONFIG.ReadStartReg = 1000.
Reg Count = 10	Read 10 consecutive registers from the Slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =3	Issues Modbus Function Code 3 to Read Holding Registers.
MB Address in Device = 0	Using Function Code 3, MB Address in Device of 0 will read Holding Register address 40001 (or 400001, if using 6-digit addressing)
	With a count of 10, this command reads 40001 to 40010 (400001 to 400010).

Read Input Registers 3xxxxx (Modbus Function Code 4)

Like the 4x holding registers, 3x input registers are used for reading analog values that are 16-bit register values. You can also use these registers to store floating-point data (page 74). Unlike the 4x registers, 3x registers are Read Only.

The following illustration shows a sample command to read Modbus addresses 30021 to 30030 of Modbus Slave Device Address 1.
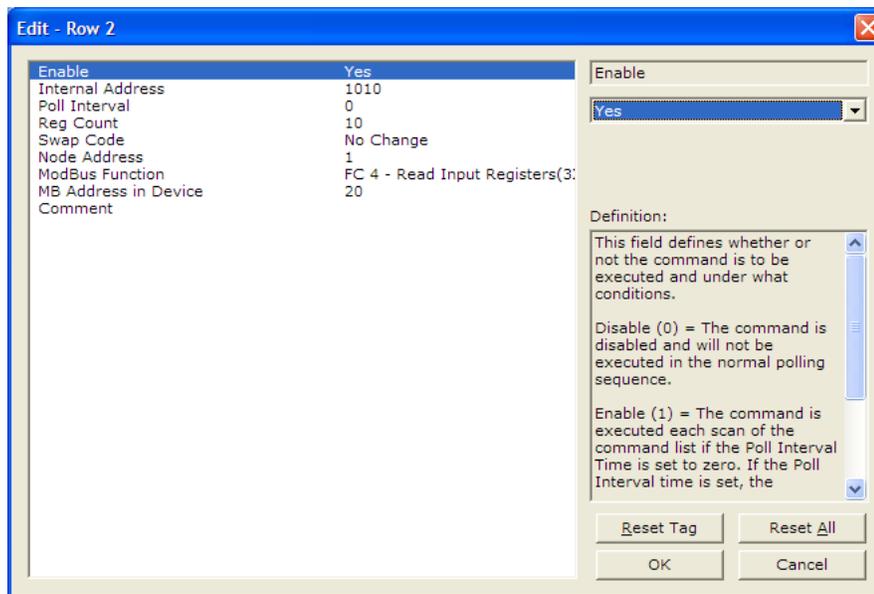
Parameter	Description
Enable = 1	The module will send the command every time it goes through the command list.
Internal Address = 1010	Places the data read from the slave device into the module at address 1010. Internal Address 1010 of the module memory will be copied into the tag **MCMR.DATA.READDATA[10].**
Reg Count = 10	Reads 10 consecutive registers from the slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =4	Issues Modbus Function Code 4 to Read Input Registers.
MB Address in Device =20	Function Code 4 MB Address in Device of 20 will read address 30021 Along with a count of 10, this command reads 30021 to 30030.

Read Coil Status 0x (Modbus Function Code 1)

Modbus Function Code 1 reads the Coils addressed at 0001 to 9999 from a slave device. These are bit values that are read using Modbus Function Code 1, and can be written to using Function Code 5 or 15. Within a Slave device, this is an individual bit value. Thus, the Internal Address field must be defined down to the bit level within your MasterCmd.

The following illustration shows a sample command to read Modbus addresses 0321 to 0480 from Modbus Slave Device Address 1.

Parameter	Description
Enable = 1	The module will send the command every time it goes through the command list.
Internal Address = 16320	Places the data read from the slave device into the module at address 16320. Internal Address 16320 of the module memory will be copied into the tag **MCMR.DATA.READDATA[20]** because 16320 represents a bit address within the memory of the MVI56E-MCMR module (16320 / 16 = register 1020).
Reg Count = 160	Reads 160 consecutive bits from the Slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =1	Issues Modbus Function Code 1 to Read Coils.
MB Address in Device = 320	Function Code 1, MB Address in Device of 320 will read address 0321 Along with a count of 160, this command reads 0321 to 0480.

Read Input Status 1x (Modbus Function Code 2)

Use this command to read Input Coils from a slave device. These are single bit addresses within a Modbus slave device. Unlike Coils 0x, the Input Coils are Read Only values and cannot be written to by a Modbus Master device. Also like the Coils 0x, the Internal Address field of this command is defined down to the bit level within the module memory.

The following illustration shows a sample command to read Modbus addresses 10081 to 10090 of Modbus Slave Device Address 1.

Parameter	Description
Enable = 1	The module will send the command every time it goes through the command list.
Internal Address = 16480	Places the data read from the slave device into the module at address 16480. Internal Address 16480 of the module memory will be copied into the tag **MCMR.DATA.READDATA[30]** (bit16480 / 16 = register 1030).
Reg Count = 16	Reads 16 consecutive registers from the slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =2	Issues Modbus Function Code 2 to Read Input Coils.
MB Address in Device = 80	Function Code 2, MB Address in Device of 80 will read address 10081 Along with a count of 16, this command reads 10081 to 10096.

Preset (Write) Single Coil 0x (Modbus Function Code 5)

Used to write a Coil of a slave device, these are single-bit addresses within a Modbus slave device. The Internal Address field of this command is defined down to the bit level within the module memory, and should come from an area of memory that has been defined within the **MCMR.DATA.WRITEDATA** area (this is configured within **BACKPLANE CONFIGURATION**).

The following illustration shows a sample command to write Modbus addresses 0513 of Modbus Slave Device Address 1, only when the data associated with the Internal Address has changed.
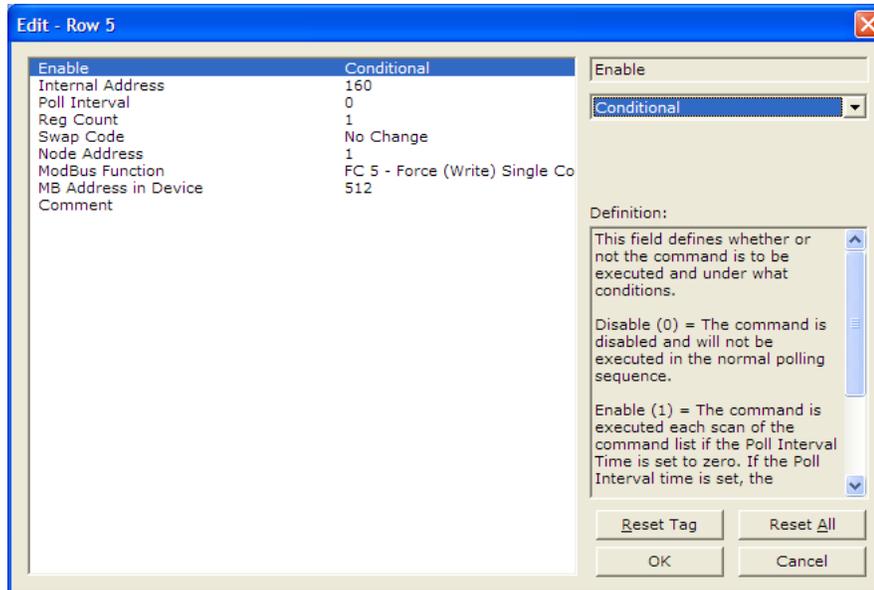
Parameter	Description
Enable = 2	The module will send the command only when the data within the Internal Address field of the module has changed.
Internal Address = 160	Will write the data to the slave device when the value at WriteData[10].0 has changed. Because this is a bit-level command, the Internal Address field must be defined down to the bit level.
Reg Count = 1	Will write a single bit to the device (Function Code 5 will 1 support a count of 1).
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function = 5	Issues Modbus Function Code 5 to write a single coil.
MB Address in Device = 512	Function Code 5, MB Address in Device of 512 will read address 0513

Write Multiple Coils 0xxx (Modbus Function Code 15)

Use this function code to write multiple Coils in the 0x address range. This function code sets multiple Coils within a slave device using the same Modbus command. Not all devices support this function code. Refer to your slave device documentation before implementing this function code.

This function code will also support the Enable code of 2, to write the data to the slave device only when the data associated within the Internal Address field of the module has changed. The Internal Address is once again defined down to the bit level as a Function Code 15 is a bit level Modbus function.

The following illustration shows a sample command to write Modbus addresses 0001 to 0016 of Modbus Slave Device Address 1.
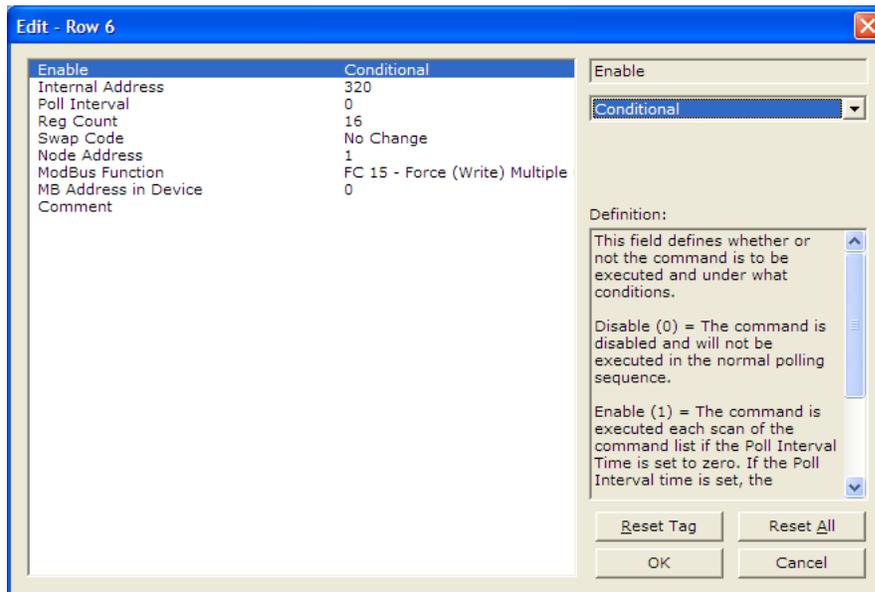
Parameter	Description
Enable = 2	The module will send the command to the slave device only when the data associated within the Internal Address of the MVI56E-MCMR module memory has changed.
Internal Address = 320	Writes the data in bit 320 of the module memory to the slave device. Based on the **BACKPLANE CONFIGURATION** setting, this would be the data in **MCMR.DATA.WRITEDATA[20].0** to **[20].15** in the ladder logic.
Reg Count = 16	Writes 16 consecutive bits to the slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =15	Issues Modbus Function Code 15 to write multiple coils.
MB Address in Device = 0	Function Code 15, MB Address in Device of 0 will read address 0001 Along with a count of 16, this command writes to 0001 to 0016.

Preset (Write) Single Register 4x (Modbus Function Code 6)

Used to write to Modbus Holding Registers 4x, this function code will write a single register to the slave device. The Enable code can be set to a value of 1 for a continuous write, or a value of 2 to write the data to the slave device only when the data associated with the Internal Address field has changed.

The following illustration shows a sample command to write Modbus addresses 41041 of Modbus Slave Device Address 1.
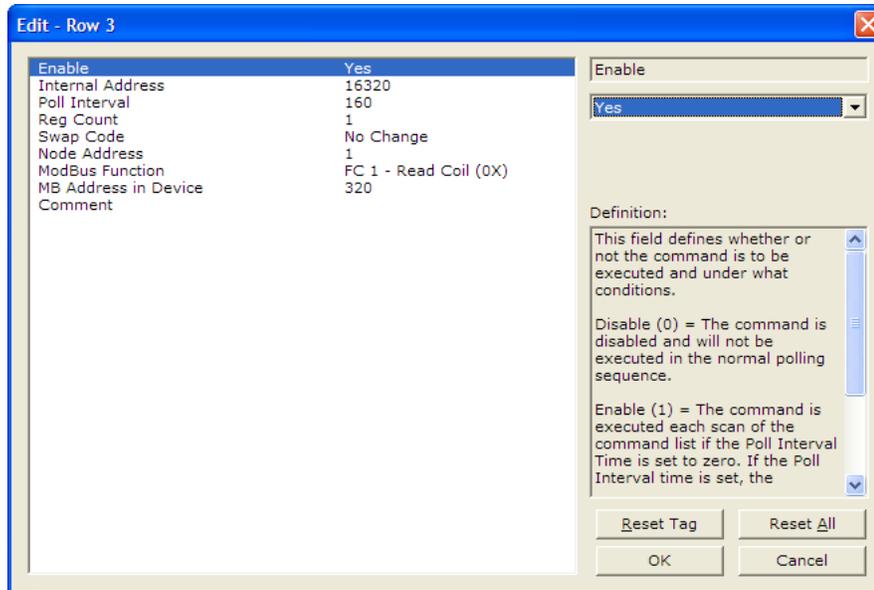
Parameter	Description
Enable = 1	The module will send the command every time it goes through the command list.
Internal Address = 5	Writes the data from address 5 of the module memory to the slave device. Based on the **BACKPLANE CONFIGURATION**, this will take the data from **MCMR.DATA.WRITEDATA[5]** and write that information out to the slave device.
Reg Count = 1	Writes 1 register (16-bit) to the slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =2	Issues Modbus Function Code 6 to write a single register.
MB Address in Device = 1040	Function Code 6, MB Address in Device of 1040 will write to address 41041 of the Modbus slave device.

Preset (Write) Multiple Registers 4x (Modbus Function Code 16)

Used to write to Modbus Holding Registers 4x, this function code will write multiple registers to the slave device. The Enable code can be set to a value of 1 for a continuous write, or a value of 2 to write the data to the slave device only when the data associated with the Internal Address field has changed.

The following illustration shows a sample command to write Modbus addresses 41051 to 41060 of Modbus Slave Device Address 1.

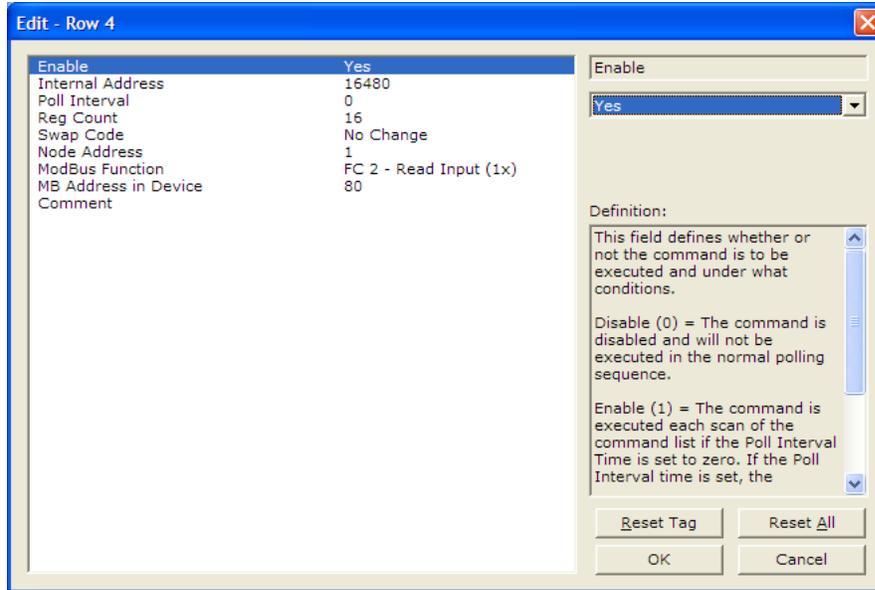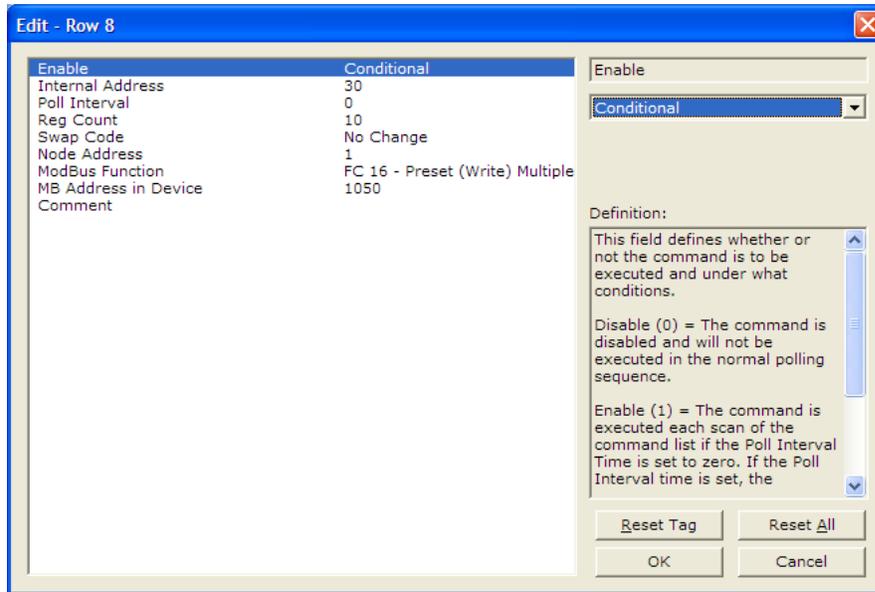| Parameter	Description
Enable = 2	The module will send the command only when the data associated with the Internal Address of the module has changed.
Internal Address =30	Writes the data from Internal Address 30 of the module memory to the Slave device. Based on the **BACKPLANE CONFIGURATION,** this will write the data from **MCMR.DATA.WRITEDATA[30] TO [39]** to the Slave device.
Reg Count = 10	Writes 10 consecutive registers to the slave device.
Node Address = 1	Issues the Modbus command to Modbus Slave Device Address 1.
Modbus Function =16	Issues Modbus Function Code 16 to write Holding Registers.
MB Address in Device = 1050	Function Code 16, MB Address in Device of 1050 will write address 41051. Along with a count of 10, this command writes 41051 to 41060 of the slave device.

2.3.7 Floating-Point Data Handling (Modbus Master)

In many applications, it is necessary to read or write floating-point data to the slave device. The sample program only provides an INT array for the ReadData and Write Data array (16-bit signed integer value). In order to read/write floating-point data to and from the slave device, you must add additional ladder logic to handle the conversion of the data to a REAL data type within the ControlLogix processor. This is very easy to accomplish.

The following topics show how to read or write data to a slave device. These topics also show when to use the Float Flag and Float Start parameters within the module configuration. For all applications, floating-point data can be read from a device without any changes to the Float Flag and Float Start parameters. You only need to configure these parameters to issue a Write command to a device that uses a single Modbus address, such as 47001, to represent a single floating-point value.

Read Floating-Point Data

Here is the addressing of a slave device, with a parameter "Energy Consumption" that is shown as two registers 40257 and 40258.

Value			Description	Type
40257	--------	KWH	Energy Consumption	Float, lower 16 bits
40258		KWH	Energy Consumption	Float, upper 16 bits

To issue a Read command to this parameter, use the following configuration.

Parameter	Description
Enable = 1	Sends the command every time through the command list.
Internal Address = 1000	Places data at address 1000 of the module memory. Based on the configuration in ModDef this will put the data at the tag **MCMR.DATA.READDATA[0]**.
Poll Interval = 0	No delay for this command.
Count = 2	Reads 2 consecutive registers from the Slave device. These 2 Modbus registers will make up the "Energy Consumption" floating-point value.
Swap = 0			
	Swap Code	Description	
	0	None - No Change is made in the byte ordering (1234 = 1234)	
	1	Words - The words are swapped (1234=3412)	
	2	Words & Bytes - The words are swapped then the bytes in each word are swapped (1234=4321)	
	3	Bytes - The bytes in each word are swapped (1234=2143)	
Node = 1	Sends the command to Modbus Slave Device Address 1.
Func = 3	Issues a Modbus Function Code 3 to "Read Holding registers."
MB Address in Device = 256	Along with the Function Code 3, MB Address in Device 256 will read Modbus address 40257 of the Slave device.

Along with the Function Code 3, MB Address in Device 256 will read Modbus address 40257 of the slave device. The above command will read 40257 and 40258 of the Modbus Slave #1 and place that data in **MCMR.DATA.READDATA[0]** and **[1].**

Within the controller tags section of the ControlLogix processor, it is necessary to configure a tag with the data type of "REAL" as shown in the following illustration.

[+]	Energy_Consumption	REAL[1]	Float

Copy data from the **MCMR.DATA.READDATA[0]** and **[1]** into the tag **ENERGY_CONSUMPTION** that has a data type of REAL. Use a **COP** statement within the ladder logic. Here is an example.
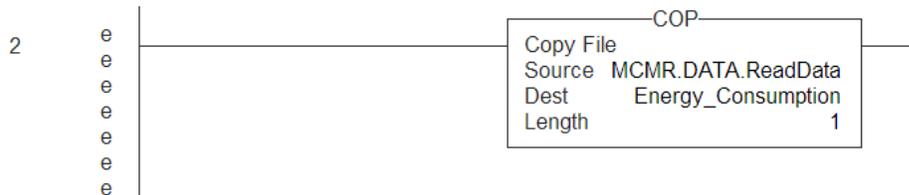
Because the tag **MCMR.DATA.READDATA[0]** should only be used within the above command, an unconditional COP statement can be used.

Notice the length of the COP statement is a value of 1. Within a Rockwell Automation processor, a COP statement will copy the required amount of "Source" values to fill the "Dest" tag for the Length specified.

Therefore, the above statement will copy ReadData[0] and [1] to fill the 32 bits required for the tag "Energy_Consumption".

Note: Do not use a MOV statement. A MOV will convert the data from the Source register to the destination register data type. This would **create** a data casting statement and will result in the loss or corruption of the original data.

Read Multiple Floating-Point Registers

The following table is an example to read Multiple Floating-Point values and device addresses. The table shows 7 consecutive floating-point values (14 Modbus addresses).

Value		Description	Type
40261	KW	Demand (power)	Float. upper 16 bits
40263	VAR	Reactive Power	Float. upper 16 bits
40265	VA	Apparent Power	Float. upper 16 bits
40267		Power Factor	Float. upper 16 bits
40269	VOLTS	Voltage, line to line	Float. upper 16 bits
40271	VOLTS	Voltage, line to neutral	Float. upper 16 bits
40273	AMPS	Current	Float. upper 16 bits

Configure the command to read these 7 floats as follows.
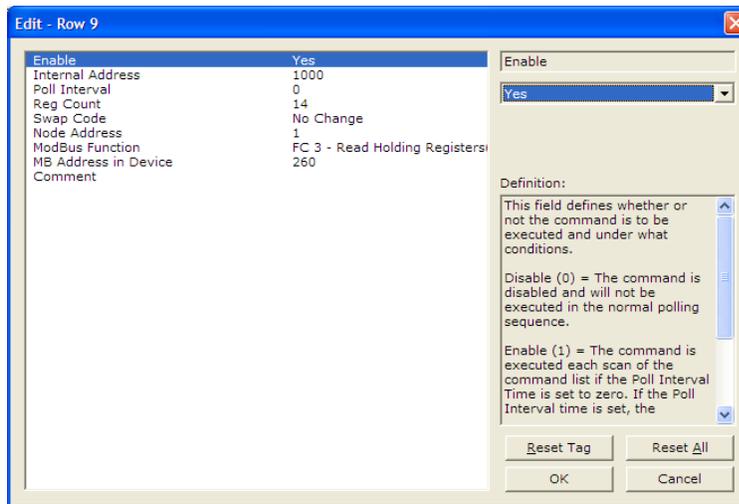
Configure an array of 7 floats within the ControlLogix processor as shown in the following illustration.

The following **COP** statement will copy the data from **MCMR.DATA.READDATA[0] TO [13]** into the array **MCM_FLOAT_DATA[0] TO [6].**
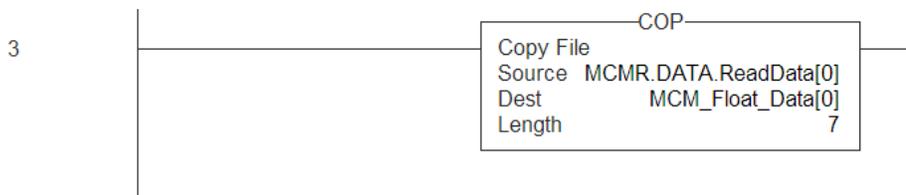
The "Length" parameter is set to the number of Floating-Point values that must be copied from the **MCMR.DATA.READDATA** array.

Write Floats to Slave Device

To issue a Write command to Floating-Point addresses, use the configuration in the following table. The table describes the Modbus Map for the slave device.

Value		Description	Type
40261	KW	Demand (power)	Float. upper 16 bits
40263	VAR	Reactive Power	Float. upper 16 bits
40265	VA	Apparent Power	Float. upper 16 bits
40267		Power Factor	Float. upper 16 bits
40269	VOLTS	Voltage, line to line	Float. upper 16 bits
40271	VOLTS	Voltage, line to neutral	Float. upper 16 bits
40273	AMPS	Current	Float. upper 16 bits

You must use a **COP** statement to copy the data from floating-point data tags within the ControlLogix processor, into the **MCMR.DATA.WRITEDATA** array used by the MVI56E-MCMR module. Below is an example.

```
                                    Main controller tag
                                    Data to write to
                                        module
                                         COP
  4                               Copy File
                                  Source        MCM_Float_Data[0]
                                  Dest   MCMR.DATA.WriteData[0]
                                  Length                     14
```
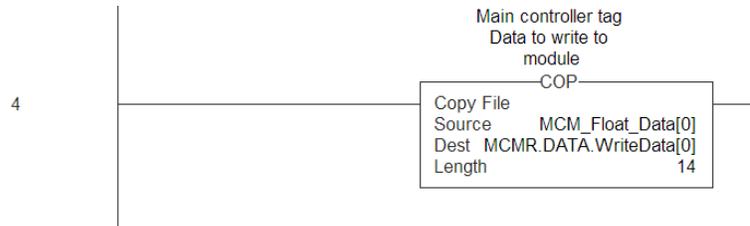
The length of this COP statement must now be 14. This will COP as many of the **MCM_FLOAT_DATA** values required to occupy the **MCMR.DATA.WRITEDATA** array for a length of 14. This will take 7 registers, **MCM_FLOAT_DATA[0] TO [6],** and place that data into **MCMR.DATA.WRITEDATA[0] TO [13].**

You must configure the command to write all 7 floats (14 Modbus addresses) as follows.
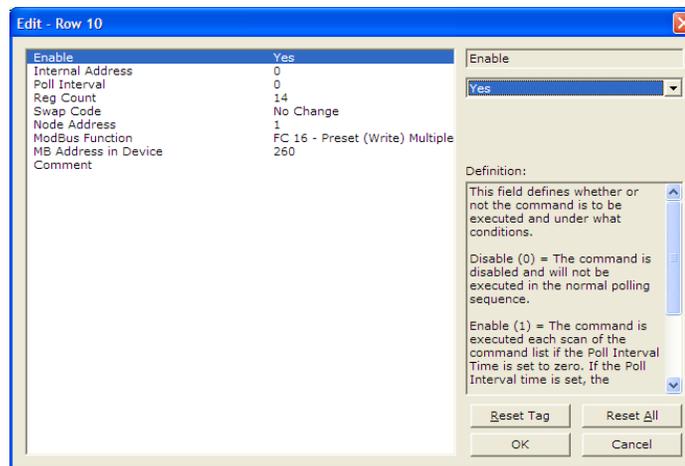
The above command will take the data from **MCMR.DATA.WRITEDATA[0] TO [13]** and write this information to Modbus Slave Device Address 1 at data addresses 40261 to 40274.

Read Floats with Single Modbus Register Address (Enron/Daniel Float)

Some Modbus slave devices use a single Modbus address to store 32 bits of data. This type of data is typically referred to as Enron or Daniel Floating-Point.

A device that uses this addressing method may have the following Modbus Memory Map.

Address	Data Type	Parameter
47001	32 bit REAL	Demand
47002	32 bit REAL	Reactive Power
47003	32 bit REAL	Apparent Power
47004	32 bit REAL	Power Factor
47005	32 bit REAL	Voltage: Line to Line
47006	32 bit REAL	Voltage: Line to Neutral
47007	32 bit REAL	Current

This type of device uses one Modbus address per floating-point register. To read these values from the Slave device, configure the following command within the module.
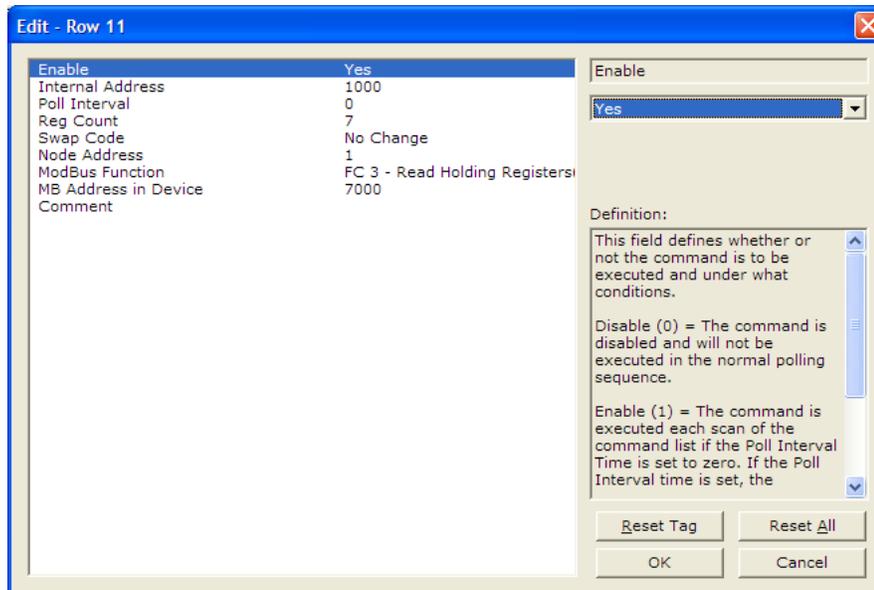
Notice that the count is now set to a value of 7. Because the Slave device utilizes only 7 Modbus addresses, a count of 7 will cause the Slave to respond with 14 registers (28 bytes) of information.

Important: This command will still occupy 14 register within the **MCMR.DATA.READDATA** array. You must not use addresses 1000 to 1013 in the Internal Address field for any other Modbus Master commands.

The **COP** statement for this type of data is the same as shown in Read Multiple Floating-Point Registers (page 76).

```
                                          ┌─COP────────────────────────┐
3                                         │ Copy File                  │
  ├──────────────────────────────────────┤ Source  MCMR.DATA.ReadData[0]│
                                          │ Dest          MCM_Float_Data[0]│
                                          │ Length                    7 │
                                          └────────────────────────────┘
```
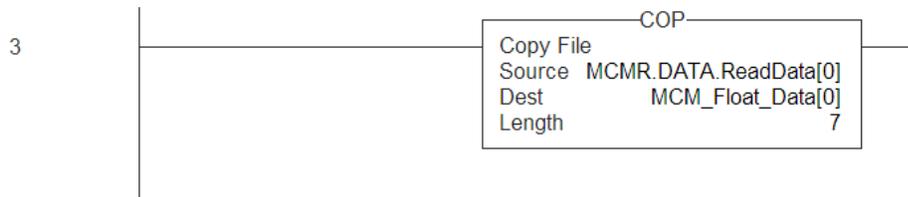
Write to Enron/Daniel Floats

To issue a Write command to Enron/Daniel Floats, use the Float Flag and Float Start parameters within the ModDef controller tags.

The following table describes the addresses that will be written to by the module.

Address	Data Type	Parameter
47001	32 bit REAL	Demand
47002	32 bit REAL	Reactive Power
47003	32 bit REAL	Apparent Power
47004	32 bit REAL	Power Factor
47005	32 bit REAL	Voltage: Line to Line
47006	32 bit REAL	Voltage: Line to Neutral
47007	32 bit REAL	Current

Configure the Float Start and Float Flag parameters as shown.

```
Float Flag                  Yes
Float Start                 7000
```

The Float Flag causes the module to use the Float Start parameter to determine which MB Address in Device requires a write command to issue double the number of bytes.

With the above configuration, any MB Address in Device > 7000 is known to be floating-point data. Therefore, a count of 1 will send 4 bytes of data, instead of the normal 2 bytes of data to a non Enron/Daniel floating-point register.

1 First, copy the floating-point data from the ControlLogix processor into the **MCMR.DATA.WRITEDATA** array used by the MVI56E-MCMR module. Below is an example.

```
                            Main controller tag
                            Data to write to
                                module
                          ┌─COP────────────────────────┐
4                         │ Copy File                  │
  ├───────────────────────┤ Source      MCM_Float_Data[0]│
                          │ Dest   MCMR.DATA.WriteData[0]│
                          │ Length                   14 │
                          └────────────────────────────┘
```
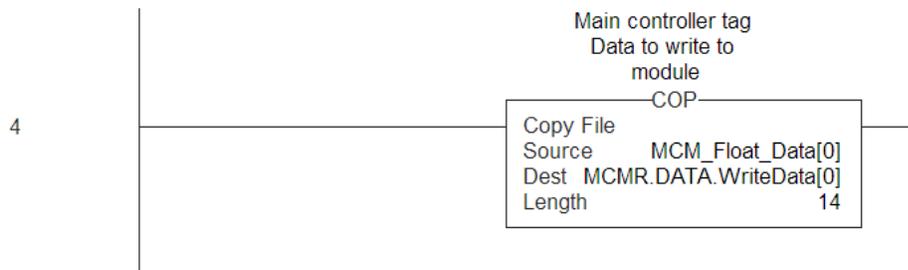
2 The length of this COP statement must now be 14. This will COP as many of the **MCM_FLOAT_DATA** values required to occupy the **MCMR.DATA.WRITEDATA** array for a length of 14. This will take 7 registers, **MCM_FLOAT_DATA[0] TO [6],** and place that data into **MCMR.DATA.WRITEDATA[0] TO [13].**

The following illustration shows the command required to write these 7 Floating-Point values.
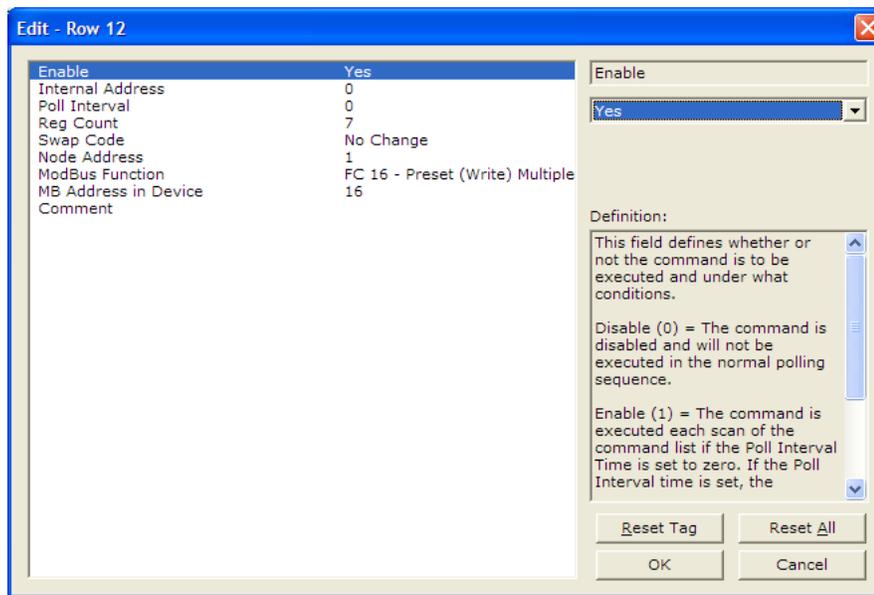
Based on the Internal Address and the configuration within the **BACKPLANE CONFIGURATION** section for Write Register Start and Write Register Count, the data from the tag **MCMR.DATA.WRITEDATA[0] TO [6]** will be written to Modbus addresses 47001 to 47007 of Modbus Slave Device Address 1.

Note: A swap code may be required to put the data in the proper format for the slave device.

2.4 Configuration as a Modbus Slave

2.4.1 Overview

When configuring the module as a slave, you will be providing a Modbus Memory Map to the person who is programming the Master side of the communications.

Note: If you are using the Sample Ladder Logic, the transfer of data is already done.

Information that is to be read by the Modbus Master device will be placed in the **MCMR.DATA.WRITEDATA** array as this will be pushed out to the module so that values from the ControlLogix processor can be read by the Modbus Master. Information that must be written to the ControlLogix processor from the Modbus Master device will be placed into the **MCMR.DATA.READDATA** array.

To configure module as a Modbus Slave, you must determine how much data you must transfer to and from the module, to the Modbus Master.

The sample ladder file is configured to transfer 600 16-bit registers in each direction. If more than that is required, please see Applications Requiring More Than 600 Registers of ReadData or WriteData.

2.4.2 Configuration File Settings

To configure Modbus slave mode, use the **BACKPLANE CONFIGURATION** settings.

This section specifies which of the MVI56E-MCMR module's 5000 registers of memory to send from the ControlLogix processor to the MVI56E-MCMR module (WriteData) and which registers to send from the MVI56E-MCMR module to the ControlLogix processor (ReadData).
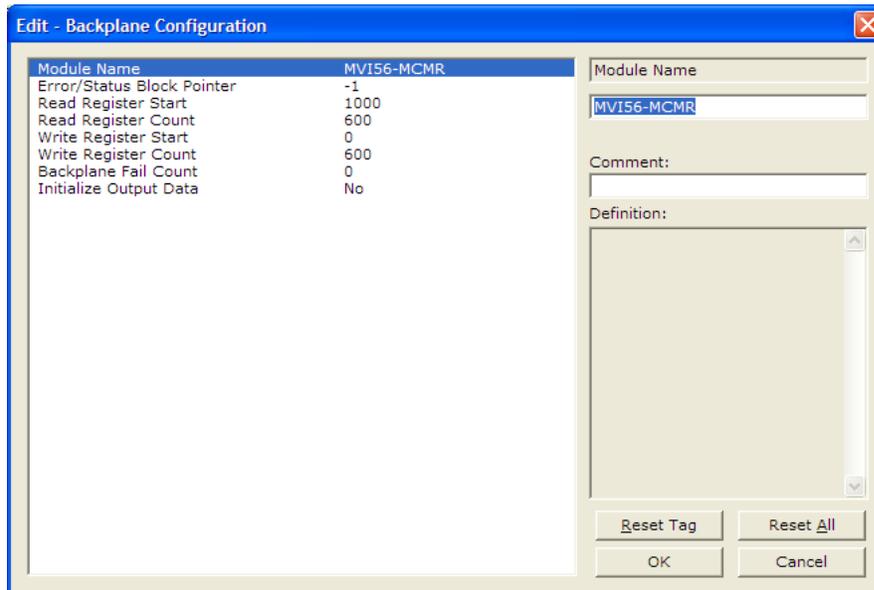
The **WRITE REGISTER START** determines the starting register location for **WRITEDATA [0 TO 599]** and the **WRITE REGISTER COUNT** determines how many of the 5000 registers to use for information to be written out to the module. The sample ladder file will configure 600 registers for Write Data, labeled **MCM.WRITEDATA[0 TO 599].**

Value	Description
Error/Status Block Pointer	This parameter places the STATUS data into the database of the module. This information can be read be the Modbus Master to know the status of the module.
Read Register Start	Determines where in the 5000 register module memory to begin obtaining data to present to the ControlLogix processor in the ReadData tags.
Read Register Count	Sets how many registers of data the MVI56E-MCMR module will send to the ControlLogix processor. This value should also be a multiple of 40.
Write Register Start	Determines where in the 5000 register module memory to place the data obtained from the ControlLogix processor from the WriteData tags.
Write Register Count	Sets how many registers of data the MVI56E-MCMR module will request from the ControlLogix processor. Because the module pages data in blocks of 40 words, this number must be evenly divisible by 40.
Backplane Fail Count	Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network.

With the sample configuration, the following is the layout of the tags and addressing.
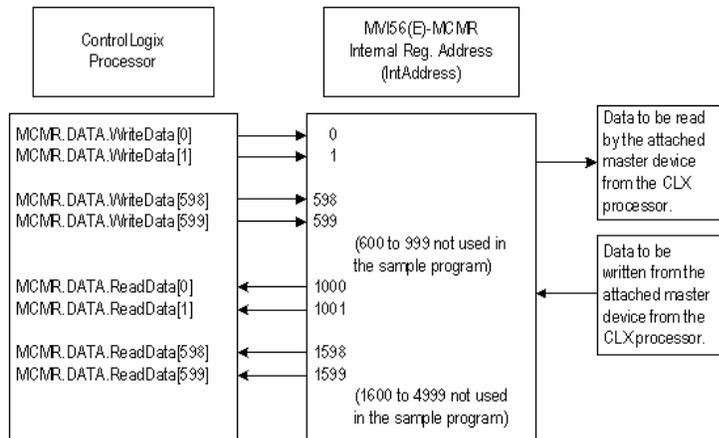
The sample configuration values configure the module database for **WRITEDATA[0 TO 599]** to be stored in the module memory at register 0 to 599, and **READDATA[0 TO 599]** to be stored in the module memory at registers 1000 to 1599 as shown above.

Modbus Memory Map

Based on the configuration described above, below is the default Modbus address for the module. Each register within the module can be accessed as a 0x bit address, 1x bit address, 3x register address, or 4x register address.

MVI Address	0x	1x	3x	4x	Tag Address
0	0001 to 0016	10001 to 10016	30001	40001	WriteData[0]
1	0017 to 0032	10017 to 10032	30002	40002	WriteData[1]
2	0033 to 0048	10033 to 10048	30003	40003	WriteData[2]
3	0049 to 0064	10049 to 10064	30004	40004	WriteData[3]
4	0065 to 0080	10065 to 10080	30005	40005	WriteData[4]
5	0081 to 0096	10081 to 10096	30006	40006	WriteData[5]
6	0097 to 0112	10097 to 10112	30007	40007	WriteData[6]
7	0113 to 0128	10113 to 10128	30008	40008	WriteData[7]
8	0129 to 0144	10129 to 10144	30009	40009	WriteData[8]
9	0145 to 0160	10145 to 10160	30010	40010	WriteData[9]
10	0161 to 0176	10161 to 10176	30011	40011	WriteData[10]
50	0801 to 0816	10801 to 10816	30051	40051	WriteData[50]
100	1601 to 1616	11601 to 11616	30101	40101	WriteData[100]
200	3201 to 3216	13201 to 13216	30201	40201	WriteData[200]
500	8001 to 8016	18001 to 18016	30501	40501	WriteData[500]
598	9569 to 9584	19569 to 19584	30599	40599	WriteData[598]
599	9585 to 9600	19585 to 19600	30600	40600	WriteData[599]
600 to 999	N/A	N/A	N/A	N/A	Reserved
1000			31001*	41001	ReadData[0]
1001			31002*	41002	ReadData[1]
1002			31003*	41003	ReadData[2]
1003			31004*	41004	ReadData[3]
1004			31005*	41005	ReadData[4]
1005			31006*	41006	ReadData[5]
1006			31007*	41007	ReadData[6]
1007			31008*	41008	ReadData[7]
1008			31009*	41009	ReadData[8]
1009			31010*	41010	ReadData[9]
1010			31011*	41011	ReadData[10]
1050			31051*	41051	ReadData[50]
1100			31101*	41101	ReadData[100]
1200			31201*	41201	ReadData[200]
1500			31501*	41501	ReadData[500]
1598			31599*	41599	ReadData[598]
1599			31600*	41600	ReadData[599]

The above addressing chart will work with many Modbus applications. Values listed in the ReadData array for 31001 to 31600 are shown with an * beside them.

Although these are valid addresses, they will not work in the application. The Master must issue a Write command to the addresses that correspond to the **READDATA** array. For Modbus addresses 3x, these are considered Input registers, and a Modbus Master does not have a function code for this type of data.

Customizing the Memory Map

In some cases, the above memory map will not work for the application. Sometimes a Master must read bits starting at address 0001, and must also read a register starting at 40001. With the memory map in this example (page 83), this is not possible, as **WRITEDATA[0]** is seen as both 0001 to 0016, and 40001. To accommodate this, you can customize the starting location within the module for each device using the parameters shown below.
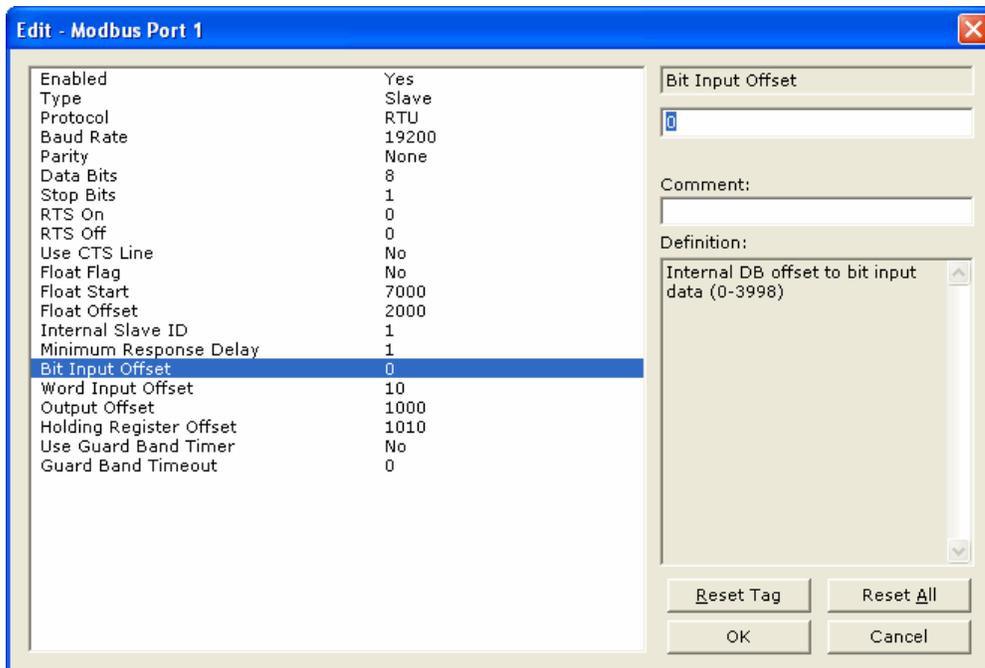
Parameter	Value	Description
Bit Input Offset	0	Defines the starting address within the module for 1x Modbus addressing. A value of 0 sets 10001 to 10016 as address 0 in the MVI56E-MCMR module.
Word Input Offset	10	Defines the starting address within the module memory for 3x registers.
Output Offset	1000	Defines the starting address within the module for 0x coils.
Holding Register Offset	1010	Defines the starting address within the module for 4x addressing.

Based on the configuration described above for the ModDef section of the module and the values specified for the offset parameters, below is the Modbus addressing map for the module.

MVI Address	0x	1x	3x	4x	Tag Address
0		10001 to 10016			WriteData[0]
1		10017 to 10032			WriteData[1]
9		10145 to 10160			WriteData[9]
10		10161 to 10176	30001		WriteData[10]
11		10177 to 10192	30002		WriteData[11]
100		11601 to 11616	30091		WriteData[100]
200		13201 to 13216	30191		WriteData[200]
500		18001 to 18016	30491		WriteData[500]
598		19569 to 19584	30489		WriteData[598]
599		19585 to 19600	30490		WriteData[599]
600 to 999	N/A	N/A	N/A	N/A	Reserved
1000	0001 to 0016				ReadData[0]
1001	0017 to 0032				ReadData[1]
1009	0145 to 0160				ReadData[9]
1010	0161 to 0176			40001	ReadData[10]
1011	0177 to 0192			40002	ReadData[11]
1050	0801 to 0816			40041	ReadData[50]
1100	1601 to 1616			40091	ReadData[100]
1200	3201 to 3216			40191	ReadData[200]
1500	8001 to 8016			40491	ReadData[500]
1598	9569 to 9584			40589	ReadData[598]
1599	9585 to 9600			40590	ReadData[599]

With the offset parameters listed above, the Modbus Master could read from coils 10001 to 10176 using the tags **MCMR.DATA.WRITEDATA[0] TO [9].** The Master could also read from address 30001 to 30490, and the data contained in those Modbus addresses would come from the tags **MCMR.DATA.WRITEDATA[10] TO [499]** within the ControlLogix program.

The Master could then write to coils addressing 0001 to 0160 and this data would reside within the ControlLogix program in tags **MCMR.DATA.READDATA[0] TO [9].** The Master could then write to registers using Modbus addresses 40001 to 40590, and this information would reside in addresses **MCMR.DATA.READDATA[10] TO [599].**

Note: The offset parameter only set the starting location for the data. As shown above, if the Master issues a Write command to address 40001, the data will go into the ControlLogix processor at address **MCMR.DATA.READDATA[10].**

Likewise, a Write To bit address 0161 will also change to address **MCMR.DATA.READDATA[10].0** within the program. Be careful not to overlap your data. You may want leave additional registers/bits unused to allow for future expansion in the program.

2.4.3 Slave Configuration

Value	Description
Enabled	**1** = enable port, **0** = disable port
Type	**1** = Modbus Slave Port
Protocol	**0** = Modbus RTU mode, **1** = Modbus ASCII mode
Baud Rate	Sets the baud rate for the port. Valid values for this field are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 384 or 3840 (for 38,400 baud), 576 or 5760 (for 57,600 baud) and 115,1152, or 11520 (for 115,200 baud)
Parity	**0** = None, **1** = Odd, **2** = Even
Data Bits	**8** = Modbus RTU mode, **8** or **7** = Modbus ASCII mode
Stop Bits	Valid values are **1** or **2**
RTS On	**0** to **65535** milliseconds to delay after RTS line is asserted on the port before data message transmission begins. This delay can be used to allow for radio keying or modem dialing before data transmission begins.
RTS Off	**0** to **65535** milliseconds to delay after data message is complete before RTS line is dropped on the port.
Use CTS Line	**No** or **Yes** This parameter is used to enable or disable hardware handshaking. The default setting is **No** hardware handshaking, CTS Line not used. Set to **No** if the connected devices do not need hardware handshaking. Set to **Yes** if the device(s) connected to the port require hardware handshaking (most modern devices do not). If you set this parameter to **Yes**, be sure to pay attention to the pinout and wiring requirements to ensure that the hardware handshaking signal lines are properly connected; otherwise communication will fail.
Float Flag	As a Slave, emulates Enron/Daniel style floats. See Floating Point Data Handling for more information (page 87).

Value	Description
Float Start	Register offset in message for floating data point. See Floating Point Data Handling for more information (page 87).
Float offset	Internal address for floats
Internal Slave ID	Valid values are **1** to **247**
Minimum Response Delay	**0** to **65535** milliseconds to delay before response
Bit Input Offset	Defines the starting address within the module for 1x Modbus addressing. A value of 0 sets 10001 to 10016 as address 0 in the MVI56E-MCMR module.
Word Input Offset	Defines the starting address within the module memory for 3x registers.
Output Offset	Defines the starting address within the module for 0x coils.
Holding Register Offset	Defines the starting address within the module for 4x addressing.
Use Guard Band Timer	**YES** or **NO** Packet gap timeout for messages
Guard Band Timeout	**0** to **65535** A value of 0 uses the default baud rate, or you can set a timeout value in milliseconds.

2.4.4 Floating-Point Data Handling (Modbus Slave)

In most applications, the use of floating-point data requires no special handling.

1 Copy the data to and from the MVI56E-MCMR module with a tag configured as a data type REAL in the ControlLogix processor.

Each floating-point value will occupy 2 registers on the Modbus network. Some Master devices use Enron or Daniel Float data. These types of floats require one Modbus register for each float in the module memory. If your Master requires this addressing, refer to the following section.
For standard floating-point data handling, the following is an example of copying 10 floats to the module.

2 First, configure a tag within the ControlLogix processor.

| ⊞-MCMR_Write_Floats | REAL[10] |

3 Then configure a COP statement within the main routine to copy this tag to the module's **MCMR.DATA.WRITEDATA** array.

```
                        COP
        Copy File
        Source    MCMR_Write_Floats[0]
        Dest   MCMR.DATA.WriteData[0]
        Length                      20
```
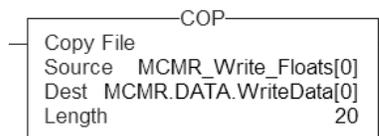
The length of the copy statement is determined by the Dest file size. To copy 10 floats from the MCM_Write_Floats array to the **MCMR.DATA.WRITEDATA** array, the length of the COP statement must be set to a value of 20.

To copy data from the MVI56E-MCMR module to a floating-point tag within the ControlLogix processor

1   Configure a tag within the ControlLogix processor as shown.

⊞-MCMR_ReadFloats	REAL[10]

2   Then configure the COP statement to move data from the **MCMR.DATA.READDATA** array, and over to the new tag **MCM_READ_FLOATS** tag as shown here.

```
                      -COP-
      Copy File
      Source   MCMR.DATA.ReadData[0]
      Dest          MCMR_ReadFloats[0]
      Length                       10
```
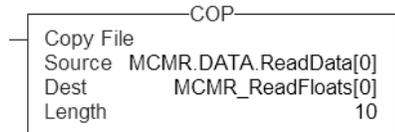
Once again, the COP statement will take as many of the Source elements required to fill the Dest tag for the length specified. Therefore, the COP statement will take **MCMR.DATA.READDATA[0] TO [19]** to fill the **MCM_READ_FLOATS[0] TO [9].**

Enron/Daniel Float Configuration

Sometimes it is necessary for the module to emulate Enron or Daniel floating-point addressing.

Copying the data to the **MCMR.DATA.WRITEDATA** array and from the **MCMR.DATA.READDATA** array is the same as described in the section above. The main difference is the addressing of the module.

For example, an Enron Float device is required to access address 47001 for floating-point data, and each Modbus register would emulate a single float value (does not require 2 Modbus addresses for 1 float value).

A Master device requiring this type of addressing, would require that for every count of 1, the MVI56E-MCMR module responds to the request message with 4 bytes (one 32-bit REAL) value.

To emulate this addressing, the module has the parameters **FLOAT FLAG, FLOAT START,** and **FLOAT OFFSET.**

Value	Description
Float Flag	Tells the module to use the Float Start and Float Offset parameters listed below
Float Start	Determines what starting address on the Modbus network to treat as floating-point data. A value of 7000 will signal the module that address 47001 on the Modbus network is the starting location for Modbus floating-point data. Every address will occupy 2 registers within the module's database.
Float Offset	Determines the address within the module to which to associate the data from the Float Start section.

Here is a sample configuration for the module.

```
Edit - Modbus Port 1

Enabled                    Yes              Float Flag
Type                       Slave
Protocol                   RTU              Yes
Baud Rate                  19200
Parity                     None             Comment:
Data Bits                  8
Stop Bits                  1
RTS On                     0
RTS Off                    0
Use CTS Line               No               Definition:
Float Flag                 Yes              Use floating data type (Yes/No)
Float Start                7000
Float Offset               2000
Internal Slave ID          1
Minimum Response Delay     1
Bit Input Offset           0
Word Input Offset          10
Output Offset              1000
Holding Register Offset    1010
Use Guard Band Timer       No
Guard Band Timeout         0

                                            Reset Tag        Reset All

                                              OK             Cancel
```

With the above configuration, this would be the addressing for the module.

Module Address	Modbus Address	Tag Address
100	47001	MCMR.DATA.WriteData[100]
102	47002	MCMR.DATA.WriteData[102]
104	47003	MCMR.DATA.WriteData[104]
110	47006	MCMR.DATA.WriteData[110]
120	47011	MCMR.DATA.WriteData[120]
200	47051	MCMR.DATA.WriteData[200]
300	47101	MCMR.DATA.WriteData[300]
500	47201	MCMR.DATA.WriteData[500]

2.5 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:

- o IP address (fixed IP required) _____ . _____ . _____ . _____
- o Subnet mask   _____ . _____ . _____ . _____
- o 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.
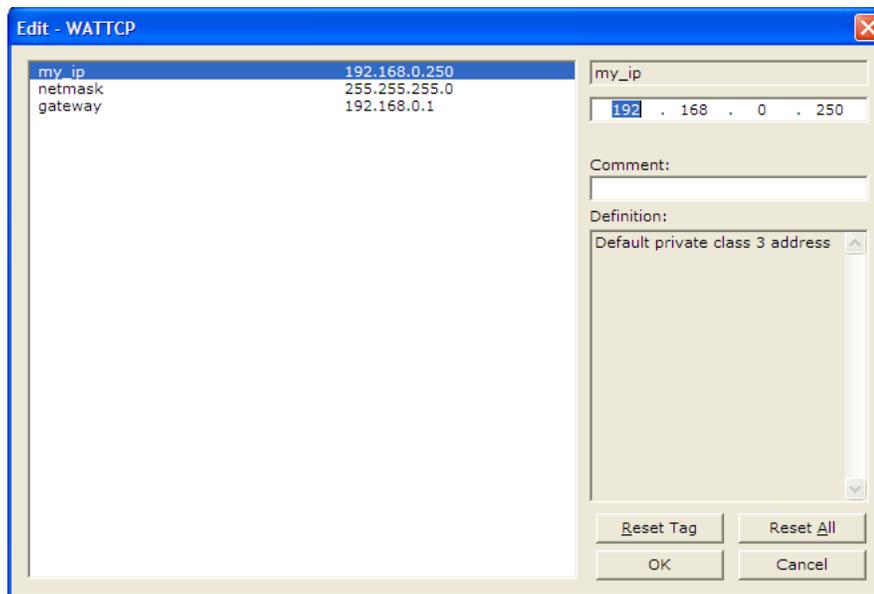
3 Edit the values for *my_ip*, *netmask* (subnet mask) and *gateway* (default gateway).
4 When you are finished editing, click **OK** to save your changes and return to the *ProSoft Configuration Builder* window.

2.6 Connecting Your PC to the Module's Ethernet Port

ith 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. Or, you can connect directly from the Ethernet
Port on your PC to the **CONFIG (E1)** Port on the module.
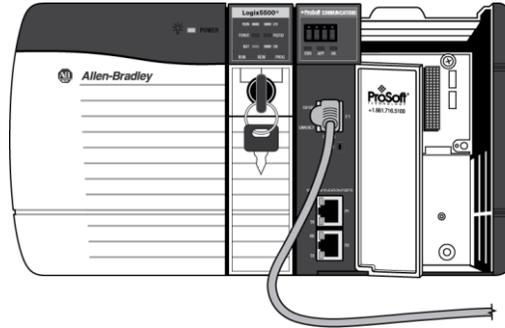
2.6.1 Setting Up a Temporary IP Address

Important: *ProSoft Configuration Builder* locates MVI56E-MCMR 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-MCMR**
module.
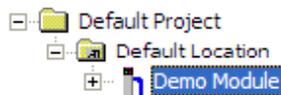
2 Click the right mouse button to open a shortcut menu. On the shortcut menu, choose **DIAGNOSTICS.**
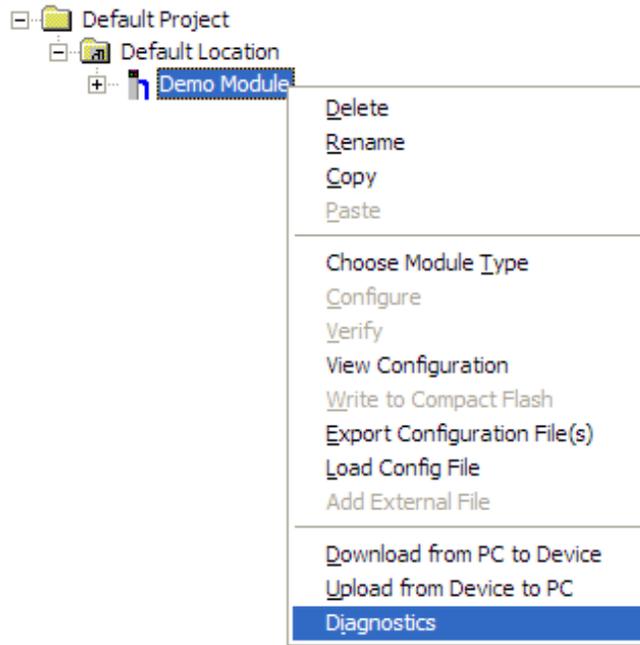
3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.

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**.
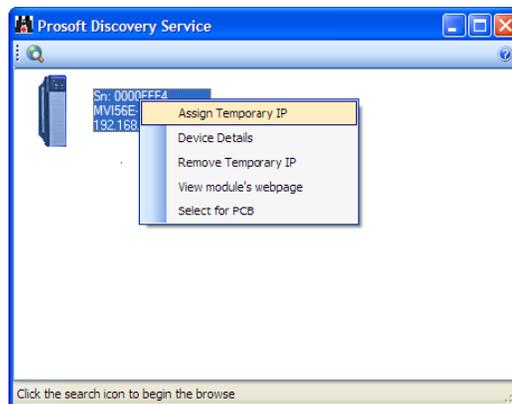
5 The module's default IP address is 192.168.0.250. Choose an unused IP within your subnet, and then click **OK**.
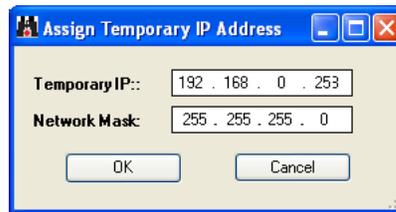
> **Important:** The temporary IP address is only valid until the next time the module is initialized. For information on how to set the module's permanent IP address, see Ethernet Configuration (page 90).

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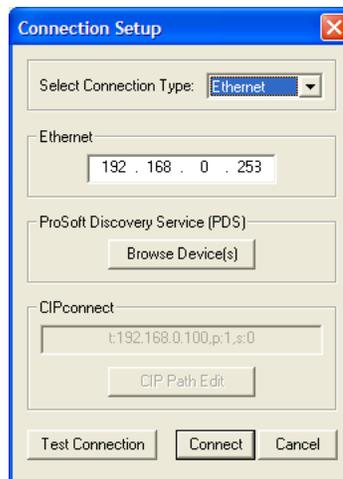
7   If the *Test Connection* is successful, click **CONNECT**. The *Diagnostics* menu will display in the *Diagnostics* window.

```
Diagnostics
Connection  Log  Module

                                          Time : 14.20.46

MODULE MENU
   ?=Display Menu
   B=Block Transfer Statistics
   C=Module Configuration
   D=Database View
   R=Transfer Configuration from PC to Unit
   S=Transfer Configuration from Unit to PC
   U=Reset diagnostic data
   V=Version Information
   W=Warm Boot Module
   @=Network Menu
   Esc=Exit Program


Path "Serial Com 1"
```
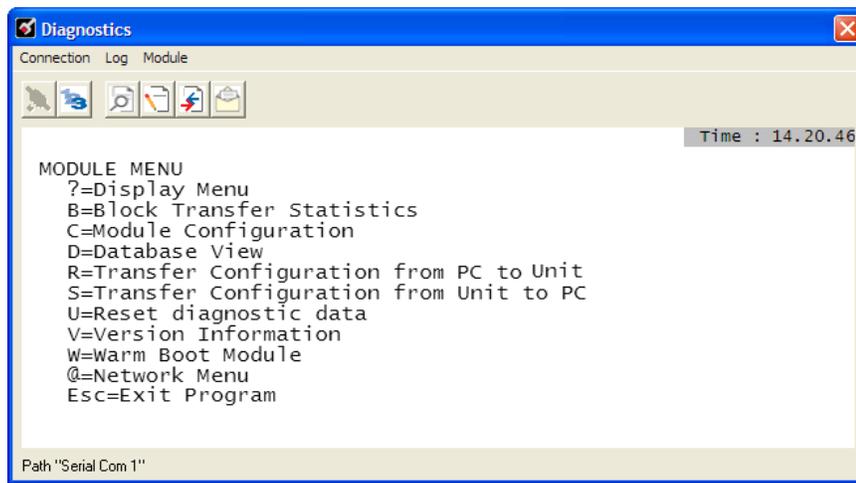
2.7   Downloading the Project to the Module

Note: For alternative methods of connecting to the module with your PC, refer to Using CIPconnect to Connect to the Module (page 96) or Using RSWho to Connect to the Module (page 106).

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-MCMR** 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.
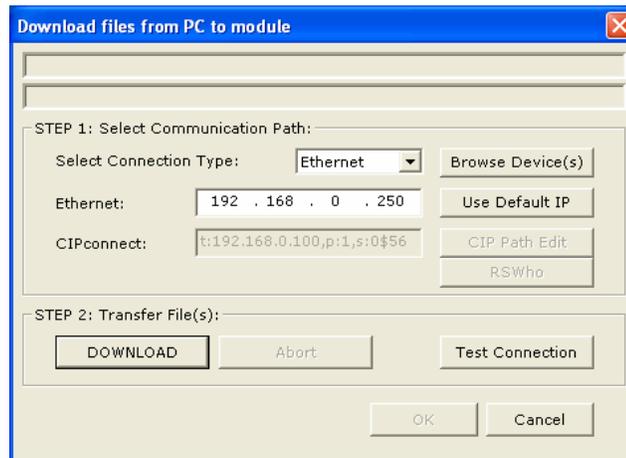
Click **TEST CONNECTION** to verify that the IP address allows access to the module.

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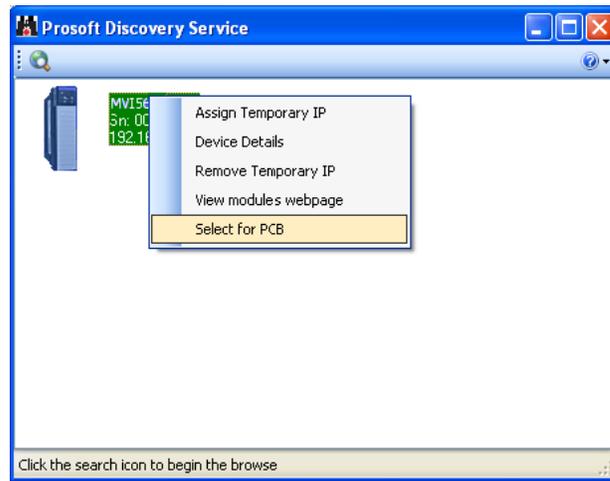
3 Select the module, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **SELECT FOR PCB**.
4 Close *ProSoft Discovery Service*.
5 Click **DOWNLOAD** to transfer the configuration to the module.

2.7.1 Using CIPconnect® to Connect to the Module

You can use CIPconnect® to connect a PC to the ProSoft Technology MVI56E-MCMR module over Ethernet using Rockwell Automation's 1756-ENBT EtherNet/IP® module. This allows you to configure the MVI56E-MCMR network settings and view 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 slot number of the MVI56E-MCMR in the destination ControlLogix chassis (the last ENBTx and chassis in the path).

To use CIPconnect, follow these steps.

1 In the *Select Port* dropdown list, choose **1756-ENBT**. The default path appears in the text box, as shown in the following illustration.
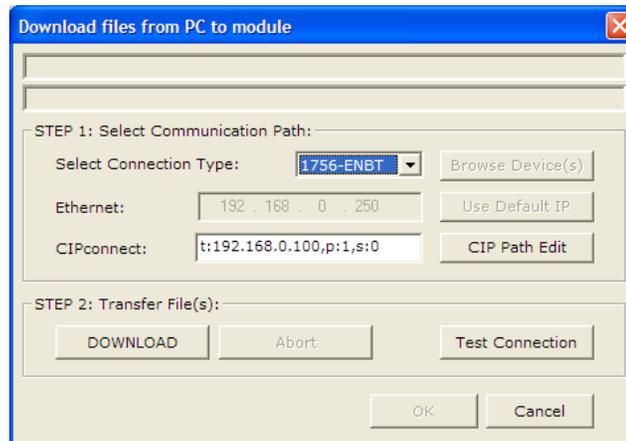
2 Click **CIP PATH EDIT** to open the *CIPconnect Path Editor* dialog box.
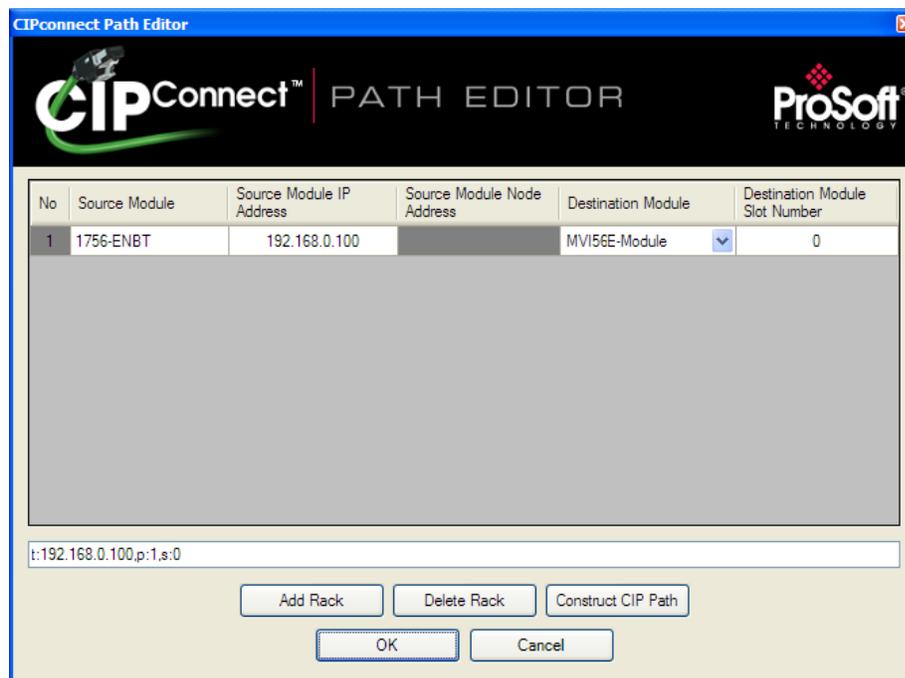
The *CIPconnect Path Editor* allows you to define the path between the PC and the MVI56E-MCMR module. The first connection from the PC is always a 1756-ENBT (Ethernet/IP) module.

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

- If the MVI56E-MCMR 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-MCMR 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-MCMR module.

   o If the module is located in a remote rack, add more racks to configure the full path.
   o The path can only contain ControlNet or Ethernet/IP networks.
   o 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.

The following examples should provide a better understanding on how to set up the path for your network.

Example 1: Local Rack Application

For this example, the MVI56E-MCMR module is located in the same rack as the 1756-ENBT that is connected to the PC.
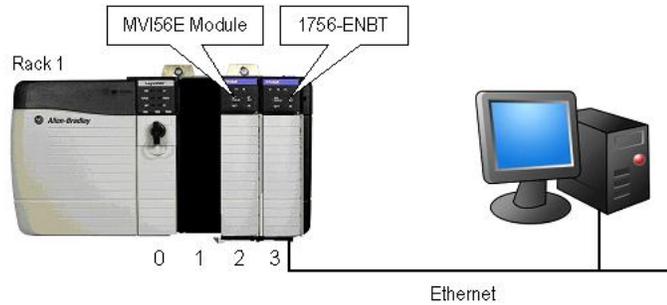
Rack 1

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

1 In the *Download* dialog box, click **CIP PATH EDIT.**
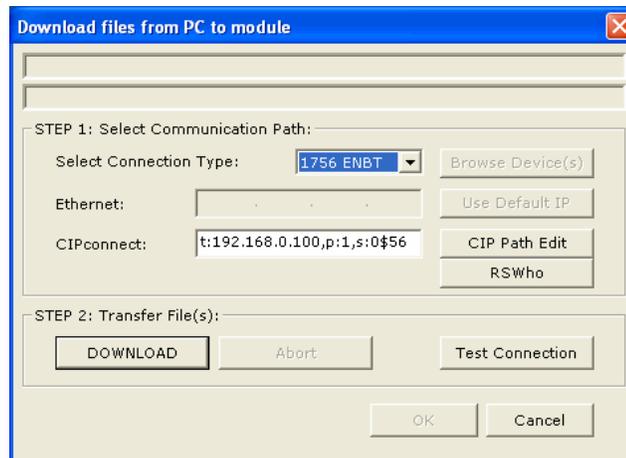
2 Configure the path as shown in the following illustration, and click
   CONSTRUCT CIP PATH to build the path in text format.
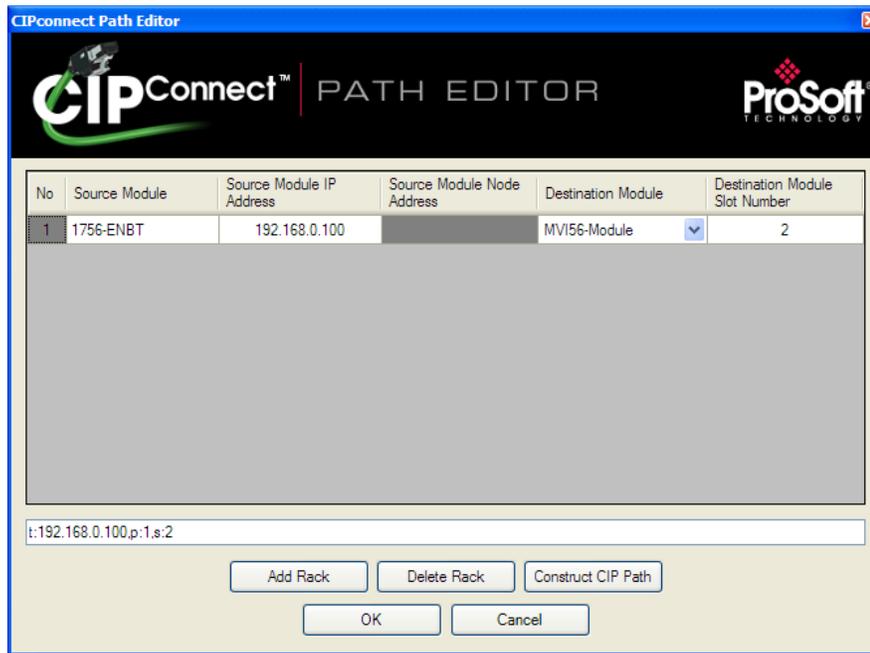
   Click **OK** to close the *CIPconnect Path Editor* and return to the *Download*
   dialog box.

3 Check the new path in the *Download* dialog box.
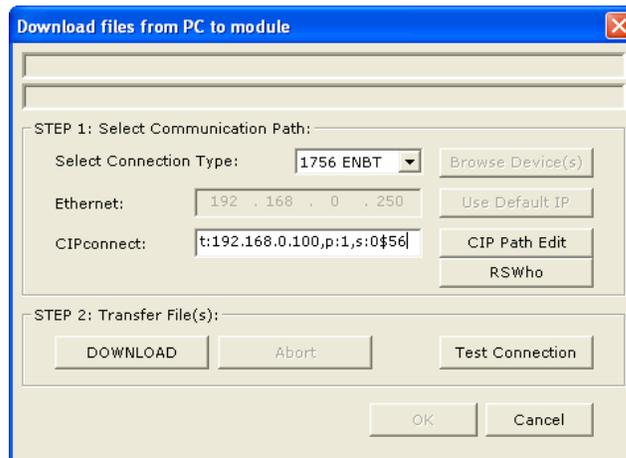
4   Click **TEST CONNECTION** to verify that the physical path is available. The
following message should be displayed upon success.
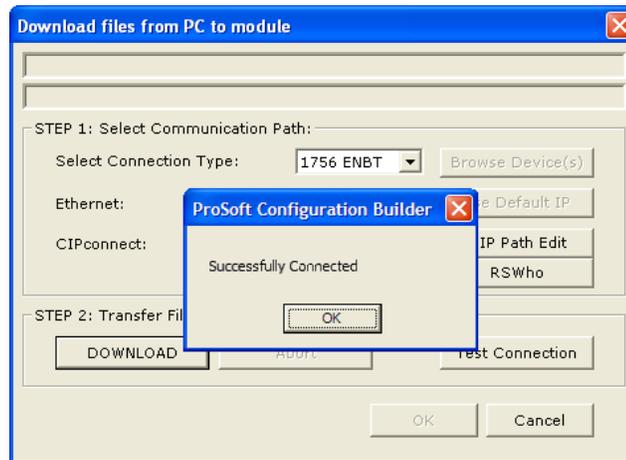
5   Click **OK** to close the Test Connection pop-up and then click **DOWNLOAD** to
download the configuration files to the module through the path.
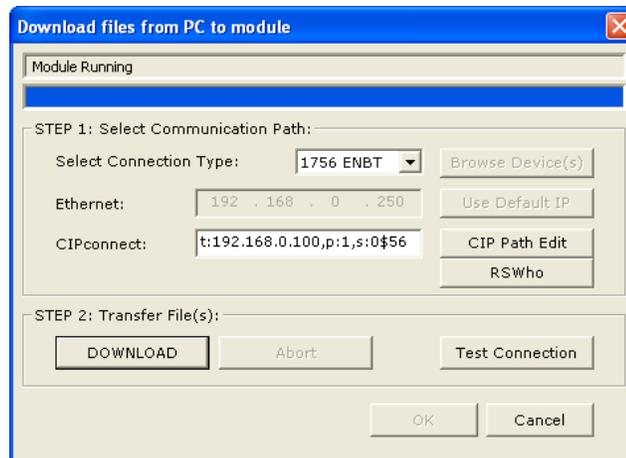
Example 2: Remote Rack Application

For this example, the MVI56E-MCMR module is located in a remote rack accessible through ControlNet, as shown in the following illustration.
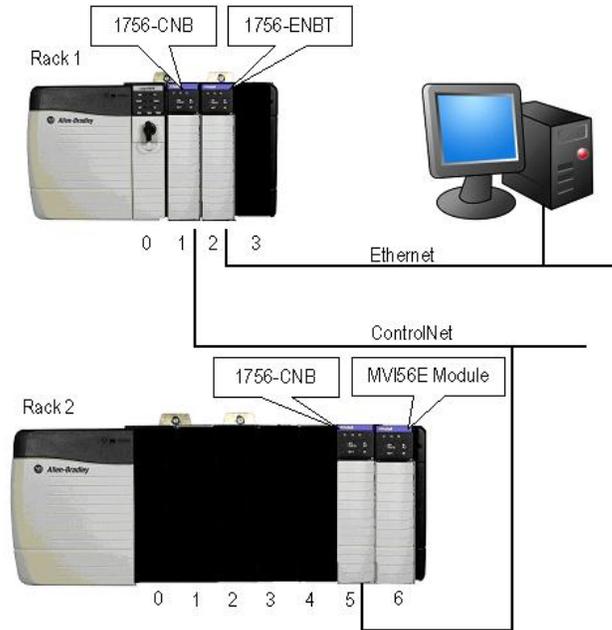
Rack 1

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-MCMR	-

1 In the *Download* dialog box, click **CIP PATH EDIT**.
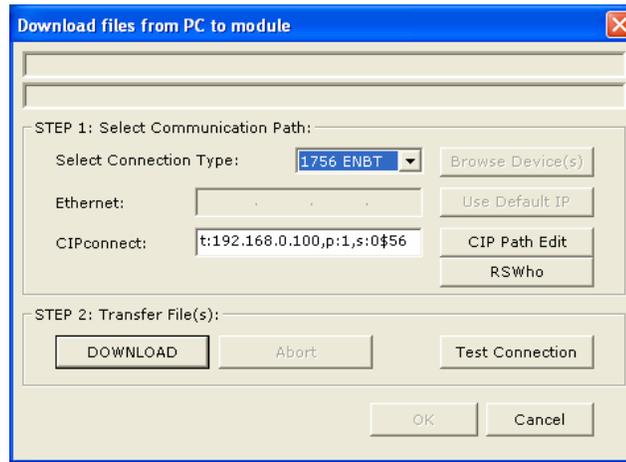
2 Configure the path as shown in the following illustration and click **CONSTRUCT CIP PATH** to build the path in text format.
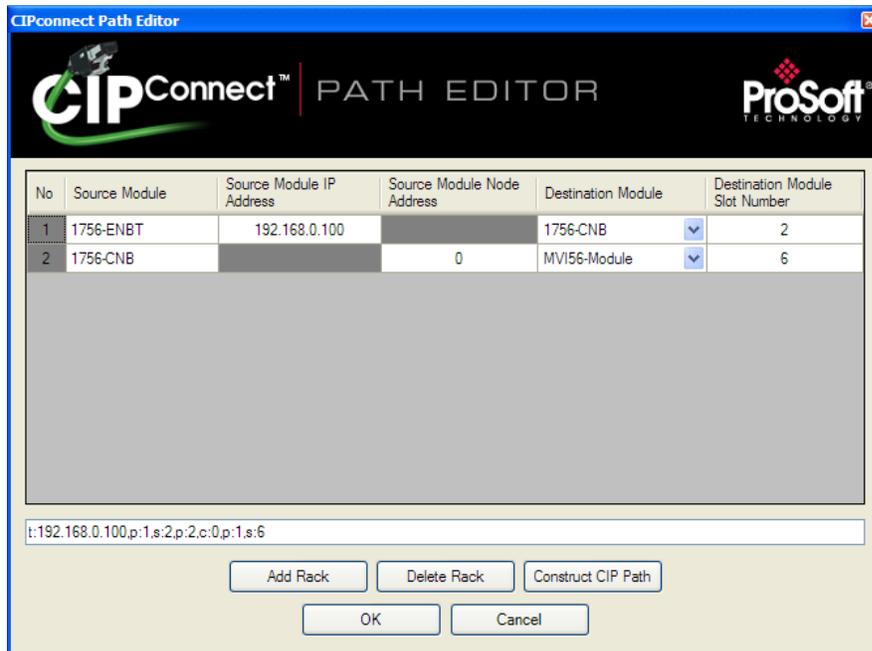
Click **OK** to close the *CIPconnect Path Editor* and return to the *Download* dialog box.

3 Check the new path in the *Download* dialog box.
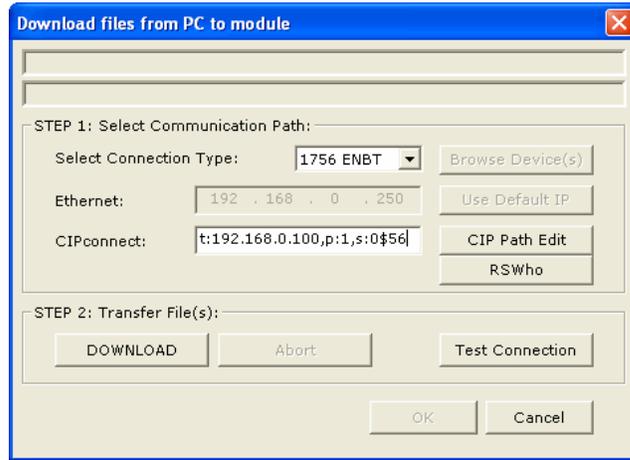
4 Click **TEST CONNECTION** to verify that the physical path is available. The following message should be displayed upon success.
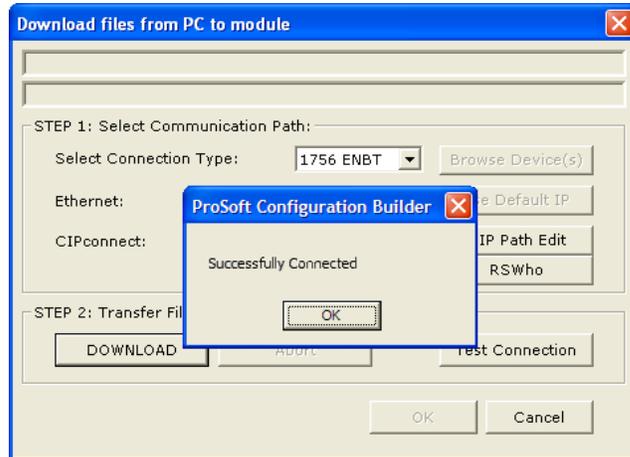
5 Click **DOWNLOAD** to download the configuration files to the module through the path.
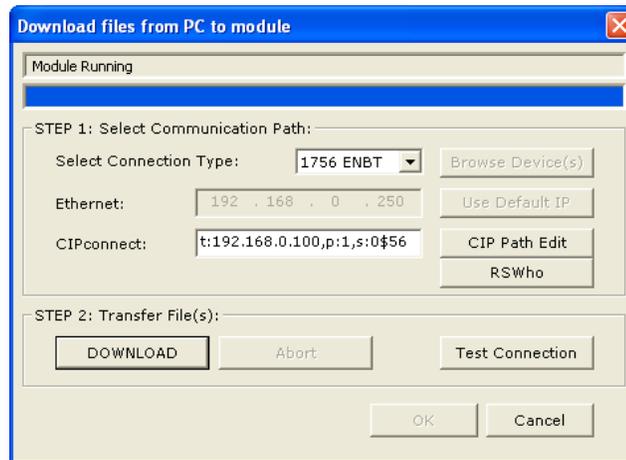
2.7.2 Using RSWho to Connect to the Module

You need to have RSLinx installed on your PC to use this feature. You also need an ENBT module set up in the rack. For information on setting up the ENBT module, see Using CIPconnect to Connect to the Module (page 96).

1 In the tree view in *ProSoft Configuration Builder*, right-click the **MVI56E-MCMR** 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.
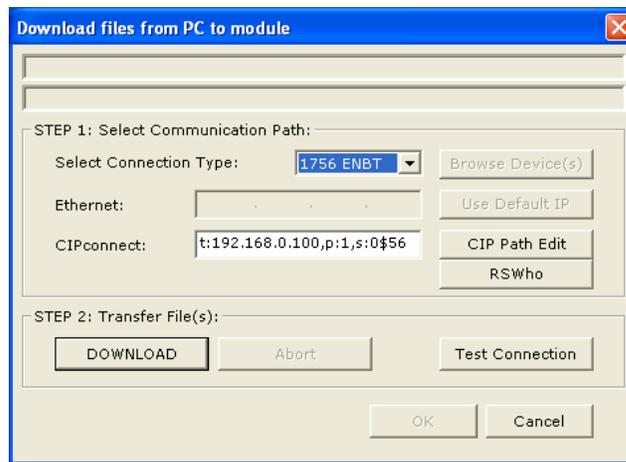
4 Click **RSWHO** to display modules on the network. The MVI56E-MCMR module will automatically be identified on the network.
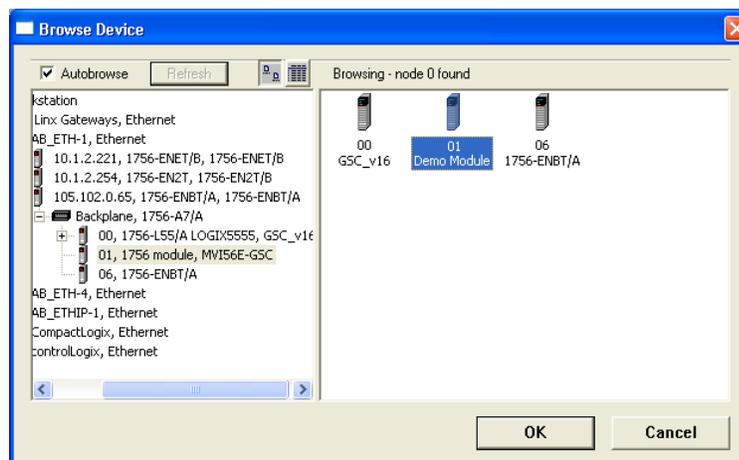
5 Select the module, and then click **OK**.
6 In the *Download* dialog box, click **DOWNLOAD**.

3   Verify Communication

In This Chapter

There are several ways to verify that the MVI56E-MCMR module is communicating with the processor and with the Modbus network.

- View the LED Status Indicators
- View the Module Status in the RSLogix 5000 Controller Tags
- View Diagnostics in ProSoft Configuration Builder

3.1   Verify Master Communications

Within the MVI56E-MCMR module, there are several ways to verify that the Modbus Master commands are working correctly.

The most common, and detailed method of checking the communications is using the **MODBUS PORT X COMMAND ERROR POINTER** parameter. This parameter will tell you the individual status of each command that is issued by the module.

For example, with the Modbus Port 1 Command Error Pointer set to 1100 to 1101 for Modbus Master Commands 1 and 2, using the default **READ START 600** and **READ COUNT 600** in the Backplane Configuration, that data is mapped to ReadData[500] and ReadData[501].

Another method is to check the **MCMR.STATUS.PRTXERRS** location for a running count of commands issued, responses received, errors, and so on.

For example, to check command status for Port 1, toggle the value of the controller tag **MCMR.CONTROL.CMDCONTROLP1.CMDERRTRIGGER.** The status data for that command populates the controller tag **MCMR.CONTROL.CMDCONTROLP1.CMDERRDATA[X].**

3.1.1   Status Data Definition as a Master

This section contains a description of the members present in the **MCMR.STATUS** object. This data is transferred from the module to the processor as part of each read block using the module's Input Image. Sample Ladder Logic will copy this information from the **LOCAL: X: I.DATA {OFFSET}** tag into the **MCMR.STATUS** array.

Offset	Content	Description
0	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
1 to 2	Product Code	These two registers contain the product code of "MCM".
3 to 4	Product Version	These two registers contain the product version for the current running software.
5 to 6	Operating System	These two registers contain the month and year values for the program operating system.
7 to 8	Run Number	These two registers contain the run number value for the currently running software.
9	Port 1 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
10	Port 1 Command List Response	This field contains the number of Slave response messages received on the port.
11	Port 1 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
12	Port 1 Requests	This field contains the total number of messages sent out of the port.
13	Port 1 Responses	This field contains the total number of messages received on the port.
14	Port 1 Errors Sent	This field contains the total number of message errors sent out of the port.
15	Port 1 Errors Received	This field contains the total number of message errors received on the port.

Offset	Content	Description
16	Port 2 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
17	Port 2 Command List Response	This field contains the number of Slave response messages received on the port.
18	Port 2 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
19	Port 2 Requests	This field contains the total number of messages sent out the port.
20	Port 2 Responses	This field contains the total number of messages received on the port.
21	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
22	Port 2 Errors Received	This field contains the total number of message errors received on the port.
23	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
24	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
25	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
26	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
27	Command Block Count	This field contains the total number of command blocks received from the processor.
28	Error Block Count	This field contains the total number of block errors recognized by the module.
29	Port 1 Current Error	For a Slave port, this field contains the value of the current error code returned. For a Master port, this field contains the index of the currently executing command.
30	Port 1 Last Error	For a Slave port, this field contains the value of the last error code returned. For a Master port, this field contains the index of the command with the error.
31	Port 2 Current Error	For a Slave port, this field contains the value of the current error code returned. For a Master port, this field contains the index of the currently executing command.
32	Port 2 Last Error	For a Slave port, this field contains the value of the last error code returned. For a Master port, this field contains the index of the command with an error.

3.1.2 Command Error Codes

The MVI56E-MCMR module will return an individual error code for every command configured within the **MODBUS PORT X COMMANDS** section. The location of these error codes are determined by the parameter **MODBUS PORT X COMMAND ERROR POINTER.** This parameter determines where in the module's 5000-register database the error codes for each command will be placed. The number of error codes returned into the database is determined by the number of commands configured in the Modbus Port x Commands section of the configuration. For 10 commands, 10 registers will be used; for 100 commands, 100 registers will be used.

To be useful in the application, these error codes must be placed within the **MCMR.DATA.READDATA** array.

Once again, the configuration in the **BACKPLANE CONFIGURATION** section for **READ REGISTER START** and **READ REGISTER COUNT** determine which of the 5000 registers will be presented to the ControlLogix processor and placed in the tag **MCMR.DATA.READDATA** array.

Based on the sample configuration values for **READ REGISTER START** and **READ REGISTER COUNT,** this will be addresses 1000 to 1599 of the module memory. The following illustration shows the sample configuration values.
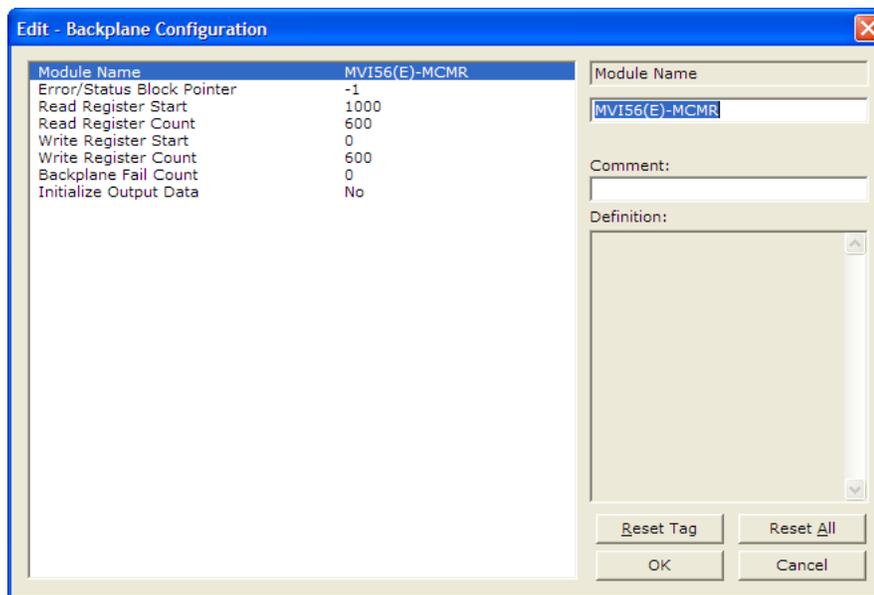
Based on these values shown above, a good place for the **MODBUS PORT X COMMAND ERROR POINTER** is address 1500, as shown.
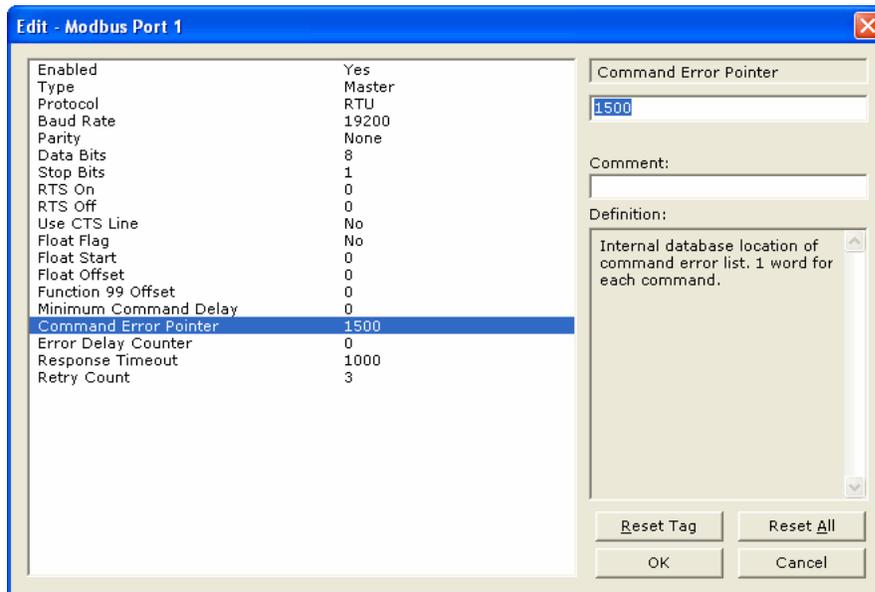
With the **COMMAND ERROR POINTER** set to address 1500, this will place your Command Error Data at addresses starting at 1500 of the module memory, and because of the before mentioned configuration of the **BACKPLANE CONFIGURATION READ REGISTER START** and **READ REGISTER COUNT** parameters, the command error data will be placed into the tags beginning at **MCMR.DATA.READDATA[500].**

Each command configured in the **MODBUS PORT X COMMANDS** will occupy one register within the **READDATA** array. For a command list consisting of 100 commands, the following table is true.

Error Code for Command	ReadData Location
1	MCMR.DATA.ReadData[500]
2	MCMR.DATA.ReadData[501]
3	MCMR.DATA.ReadData[502]
4	MCMR.DATA.ReadData[503]
5	MCMR.DATA.ReadData[504]
90	MCMR.DATA.ReadData[598]
99	MCMR.DATA.ReadData[599]

Standard Modbus Protocol Errors

Code	Description
1	Illegal Function
2	Illegal Data Address
3	Illegal Data Value
4	Failure in Associated Device
5	Acknowledge
6	Busy, Rejected Message

The "Standard Modbus Protocol Errors" are error codes returned by the device itself. This means that the Slave device understood the command, but replied with an Exception Response, which indicates that the command could not be executed. These responses typically do not indicate a problem with port settings or wiring.

The most common values are Error Code 2 and Error Code 3.

Error Code 2 means that the module is trying to read an address in the device that the Slave does not recognize as a valid address. This is typically caused by the Slave device skipping some registers. If you have a Slave device that has address 40001 to 40005, and 40007 to 40010, you cannot issue a read command for addresses 40001 to 40010 (function code 3, MB Address in Device 0, Count 10) because address 40006 is not a valid address for this Slave.

Instead, try reading just one register, and see if the error code goes away. You can also try adjusting your MB Address in Device -1, as some devices have a 1 offset.

An Error Code of 3 is common on Modbus Write Commands (Function Codes 5,6,15, or 16). Typically, this is because you are trying to write to a parameter that is configured as read only in the Slave device, or the range of the data you are writing does not match the valid range for that device.

Refer to the documentation for your Slave device, or contact ProSoft Technical Support for more help with these types of error codes.

Module Communication Error Codes

Code	Description
-1	CTS modem control line not set before transmit
-2	Timeout while transmitting message
-11	Timeout waiting for response after request
253	Incorrect Slave address in response
254	Incorrect function code in response
255	Invalid CRC/LRC value in response

"Module Communication Errors" are generated by the MVI56E-MCMR module, and indicate communication errors with the Slave device.

Error Code -11 indicates that the module is transmitting a message on the communications wire. However, it is not receiving a response from the addressed Slave. This error is typically caused by one or more of the following conditions.

▪ Parameter mismatch, for example the module is set for 9600 baud, Slave is set for 19,200, parity is set to none, Slave is expecting even, and so on.
▪ Wiring problem, for example the port jumper on the module is set incorrectly, or + and - lines on RS485 are switched)
▪ The Slave device is not set to the correct address, for example the Master is sending a command to Slave 1 and the Slave device is configured as device 10.

With a -11 error code, check all of the above parameters, wiring, and settings on the Slave device. Also, make sure that you cycle power to the module, or toggle the **MCMR.CONTROL.WARMBOOT** or **COLDBOOT** bit.

Error codes of 253 to 255 typically indicate noise on RS485 lines. Make sure that you are using the proper RS485 cable, with termination resistors installed properly on the line. If termination resistors are installed, try removing them, as they are usually only required on cable lengths of more than 1000 feet.

Command List Entry Errors

Code	Description
-41	Invalid enable code
-42	Internal address > maximum address
-43	Invalid Modbus Slave Device Address (< 0 or > 255)
-44	Count parameter set to 0
-45	Invalid function code
-46	Invalid swap code

The above error codes indicate that the module has detected an error when parsing the command.

For all commands that have not been configured (all parameters set to a value of 0) you will receive an error code of -44. To remove this error code, you can change your **MODBUS PORT X REG COUNT** parameter to the number of registers to send, and download the updated configuration to the module.

Transferring the Command Error List to the Processor

You can transfer the command error list to the processor from the module database. To place the table in the database, set the Command Error Pointer **(MCMR.PORT1.CMDERRPTR)** parameter to the database location desired.

In the sample ladder, the **MCMR.PORT1.CMDERRPTR** tag is set to a value of 1100. This will cause the error value of command 0 to be placed at database address 1100. Each command error value occupies one database word. The error value for command 1 will be in location 1101 and the remaining values in consecutive database locations.

To transfer this table to the processor, refer to Command Error Codes (page 110). Make sure that the Command Error table is in the database area covered by the Read Data **(MCMR.MODDEF.READSTARTREG** and **MCMR.MODDEF.READREGCNT).**

3.1.3 MCM Status Data

Status information can also be obtained from the MVI56E-MCMR module by checking the **MCMR.STATUS.PRTXERRS** location. Below is a sample.

MCMR.STATUS.Port1Stats	{...}
MCMR.STATUS.Port1Stats.PortTrigger	0
MCMR.STATUS.Port1Stats.CmdReq	-23865
MCMR.STATUS.Port1Stats.CmdResp	-23892
MCMR.STATUS.Port1Stats.CmdErr	105
MCMR.STATUS.Port1Stats.Requests	-23786
MCMR.STATUS.Port1Stats.Responses	-23872
MCMR.STATUS.Port1Stats.ErrSent	0
MCMR.STATUS.Port1Stats.ErrRec	0
MCMR.STATUS.Port1Stats.SlaveStats	{...}

If your system is working correctly, you will see **CMDREQ, CMDRESP, REQUESTS,** and **RESPONSES** all incrementing together. If you see that **CMDERR** is incrementing, determine what command is causing the error (using the error code defined in the previous section (page 110)) and correct the issue causing the error.

Note: This information is not as detailed as the individual error codes, but they can help to troubleshoot your application.

Also within the **MCMR.STATUS** location is the parameters for Last Error and Previous Error, shown below.

MCMR.STATUS.Port1CurrentErr	1	Decimal
MCMR.STATUS.Port1LastErr	0	Decimal

This indicates the command index that last generated an error and does not indicate a command currently in error. In the above example, a value of 0 in **PORT1LASTERR** indicates that the last error was generated by **MODBUS PORT 1 COMMAND 0.** This does not indicate that this command is currently in error. The value in **MCMR.STATUS.PORT1PREVIOUSERR** indicates that before **MASTER COMMAND 0** generated an error, **MODBUS PORT 1 COMMAND 1** posted an error.

3.2   Verify Slave Communications

For verifying the communications to the module as a Slave, you can monitor the **STATUS** tags under the **PRTXERRS** section.

Below is an example.

⊞ MCMR.STATUS.Port2Stats.Requests	0
⊞ MCMR.STATUS.Port2Stats.Responses	0

The **REQUESTS** field shows the number of request messages sent to the module as a Slave. The **RESPONSES** field shows how many times the module has responded to a request message from the Modbus Master.

3.2.1 Status Data Definition as a Slave

This section contains a description of the members present in the
MCMR.STATUS object. This data is transferred from the module to the
processor as part of each read block using the module's Input Image. Sample
Ladder Logic will copy this information from the **LOCAL: X: I.DATA {OFFSET}** tag
into the **MCMR.STATUS** array.

Offset	Content	Description
0	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
1 to 2	Product Code	These two registers contain the product code of "MCM".
3 to 4	Product Version	These two registers contain the product version for the current running software.
5 to 6	Operating System	These two registers contain the month and year values for the program operating system.
7 to 8	Run Number	These two registers contain the run number value for the currently running software.
12	Port 1 Requests	This field is the total number of messages sent out of the port.
13	Port 1 Responses	This field contains the total number of messages received on the port.
14	Port 1 Errors Sent	This field contains the total number of message errors sent out of the port.
15	Port 1 Errors Received	This field contains the total number of message errors received on the port.
19	Port 2 Requests	This field contains the total number of messages sent out the port.
20	Port 2 Responses	This field contains the total number of messages received on the port.
21	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
22	Port 2 Errors Received	This field contains the total number of message errors received on the port.
23	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
24	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
25	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
26	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
27	Command Block Count	This field contains the total number of command blocks received from the processor.
28	Error Block Count	This field contains the total number of block errors recognized by the module.
29	Port 1 Current Error	For a Slave port, this field contains the value of the current error code returned.
30	Port 1 Last Error	For a Slave port, this field contains the value of the last error code returned.
31	Port 2 Current Error	For a Slave port, this field contains the value of the current error code returned.
32	Port 2 Last Error	For a Slave port, this field contains the value of the last error code returned.

4   Ladder Logic

In This Chapter

Ladder logic is required for application of the MVI56E-MCMR module. Tasks that must be handled by the ladder logic are module data transfer, special block handling, and status data receipt. Additionally, a power-up handler may be needed to handle the initialization of the module's data and to clear any processor fault conditions.

The sample ladder logic, on the *ProSoft Solutions DVD*, is extensively commented, to provide information on the purpose and function of each rung. For most applications, the sample ladder will work without modification.

4.1   MVI56E-MCMR User Defined Data Types

This section describes the controller tags that are defined in the example logic to interface with the module. The user can extend these tags to meet the specifications required for their application, If additional data transfer is required.

4.1.1   Module Status Data and Variables (MCMRModuleDef)

All status and variable data related to the MVI56E-MCMR is stored in a user defined data type. An instance of the data type is required before the module can be used. This is done by declaring a variable of the data type in the Controller Tags Edit Tags dialog box.

The following table describes the structure of this object.

Name	Data Type	Description
DATA	MCMRDATA (page 118)	Read Data and Write Data
STATUS	MCMRSTATUS (page 119)	Status information in each read block
CONTROL	MCMRCONTROL (page 120)	Command Control Object
UTIL	MCMRUTIL (page 118)	Backplane Object

This object contains objects that define variables for the module and status data related to the module. Each of these object types is discussed in the following topics of the document.

Backplane Object (MCMRUTIL)

The **MCMRUTIL** object stores all the variables required for the data transfer operation between the module and the controller. The LastRead data member is used as the handshaking byte to indicate the arrival of new data from the module.

The following table describes the structure of this object.

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

The other members of the object are utilized in the ladder logic to assist in the data transfer operation.

Module Data Object (MCMRDATA)

Data for the module is stored in two controller tags for the example ladder logic. The read data (data transferred from the module to the processor) is stored in the controller tag **MCMR.READDATA[ ]**. The write data (data transferred from the processor to the module) is stored in the controller tag **MCMR.WRITEDATA[ ]**. Separate tags can be constructed for each data type used by the controlled devices and for each device.

Name	Data Type	Description
ReadData	INT[600]	Data read from module

Name	Data Type	Description
WriteData	INT[600]	Data to write to module

Status Object (MCMRSTATUS)

This object stores the status data of the module. The **MCMRSTATUS** object shown below is updated each time a read block is received by the processor. Use this data to monitor the state of the module at a "real-time rate".

Name	Data Type	Description
PassCnt	INT	Program cycle counter
Product	INT[2]	Product Name
Rev	INT[2]	Revision Level Number
OP	INT[2]	Operating Level Number
Run	INT[2]	Run Number
Port1Stats	MCMRPortStats	Port error statistics for Port 1
Port2Stats	MCMRPortStats	Port error statistics for Port 2
Block	MCMRBlockStats	Block transfer statistics
Port1CurrentErr	INT	Current error/index for Port 1
Port1LastErr	INT	Last error/index for Port 1
Port2CurrentErr	INT	Current error/index for Port 2
Port2LastErr	INT	Last error/index for Port 2
StatusMsgData	INT[46]	This status data is returned when requested by a Status Message (MSG) and can be used to detect proper module operation.
StatusTrigger	BOOL	Triggers Status reading.

Within the **MCMRSTATUS** objects are objects containing the status information for each application port. Refer to MVI56E-MCMR Status Data Definition (page 177) for a complete listing of the data stored in this object. This data is acquired from the module using Message instructions (MSGs).

MCMRPortStats

The **MCMRPORTSTATS** object holds the status data related to a single Modbus port. The following table describes the structure of this object.

Name	Data Type	Description
PortTrigger	BOOL	Triggers port status reading
CmdReq	INT	Total number of command list requests sent
CmdResp	INT	Total number of command list responses received
CmdErr	INT	Total number of command list errors
Requests	INT	Total number of requests for port
Responses	INT	Total number of responses for port
ErrSent	INT	Total number of errors sent
ErrRec	INT	Total number of errors received
SlaveStats	SINT[250]	Port Slave status values

This information is passed to the controller from the module with each normal read block image.

MCMRBlockStats

The **MCMRBLOCKSTATS** object stores the block transfer statistics for the MVI56E-MCMR module.

Name	Data Type	Description
Read	INT	Total number of read block transfers
Write	INT	Total number of write block transfers
Parse	INT	Total number of blocks parsed
Event	INT	Total number of event blocks received
Cmd	INT	Total number of command blocks received
Err	INT	Total number of block transfer errors

Command Control Data Object (MCMRCONTROL)

Contains the data structure required for the processor to request special tasks from the module. The command control task allows the processor to dynamically enable commands configured in the port command list. The event command task allows the processor to dynamically build any commands to be sent by the module Master port to a remote Slave.

The following table describes the structure of this object.

Name	Data Type	Description
ColdBoot	BOOL	Triggers a Cold Boot Command
WarmBoot	BOOL	Triggers a Warm Boot Command
CmdControlP1	MCMRCmdControl	Command Control for Port 1
CmdControlP2	MCMRCmdControl	Command Control for Port 2
EventTriggerP1	BOOL	Triggers the Event Command.
EventTriggerP2	BOOL	Triggers the Event Command.
EventCmdP1	MCMREventCmd[100]	This object contains the attributes to define a Master command. An array of these objects is used for each port.
EventCmdP2	MCMREventCmd[100]	This object contains the attributes to define a Master command. An array of these objects is used for each port.
EventCmdRespP1	INT[5]	Event Command Response for Port 1
EventCmdRespP2	INT[5]	Event Command Response for Port 2

MCMREventCmd

When the command bit (**MCMR.CONTROL.EVENTTRIGGERPX**) is set in the example ladder logic, the module will build a block 9901 with the command contained in the first command of the **MCMR.CONTROL.EVENTCMDPX[ ]** array. The module will receive this block and build and send the command to the specified control device using a MSG block.

The following table describes the data for the command element in the **MCMREVENTCMD** array.

Name	Data Type	Description
Enable	INT	0=Disable, 1=Continuous, 2=Event Command
IntAddress	INT	Module's internal address associated with the command
PollInt	INT	Minimum number of seconds between issuance of command (0 to 65535 Sec)
Count	INT	Number of registers associated with the command
Swap	INT	Swap code used with command
Device	INT	Device index in Device Table to associate with the command
Func	INT	Function code for the command
DevAddress	INT	Address in device associated with the command

MCMRCmdControl

When the command bit (**MCMR.CONTROL.CMDCONTROLPX.CMDTRIGGER**) is set in the example ladder logic, the module will build a block 9901 with the number of commands set through:
MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[0].

The command indexes will be set through the controller tags starting from **MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[1]** to **MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[20]**

For example, in order to enable commands 0, 2 and 5 the following values would be set:

MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[0] = 3

MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[1] = 0

MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[2] = 2

MCMR.CONTROL.CMDCONTROLPX.CMDCONTROLDATA[3] = 5

The module will receive this block and build and send the command to the specified control device using a MSG block.

The following table describes the data for the command element in **MCMRCMDCONTROL**.

Name	Data Type	Description
CmdTrigger	BOOL	Command Trigger
CmdControlData	INT[21]	Command Control Data
CmdControlResp	INT[5]	Command Control Response
CmdErrTrigger	BOOL	Command Error Trigger
CmdErrData	INT[102]	Command Error Data

5   Diagnostics and Troubleshooting

In This Chapter

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.

5.1 Ethernet LED Indicators

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

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

5.1.1 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 Modbus ports
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 MVI56E Optional Add-On Instruction.
Config P1/P2 <Modbus mode> <Port type> <Baud> <Parity> <Data bits> <Stop Bits> <RS Interface> <ID (Slave)> <Cmds: (Master)>	After power up and every reconfiguration, the module will display the configuration of both ports. The information consists of: ▪ Modbus mode: RTU/ASCII ▪ Port type: Master/Slave ▪ Baud: 115200 / 57600 / 38400 / 19200 / 9600/ 4800 / 2400 / 1200 / 600 / 300 ▪ Parity: None / Even / Odd ▪ Data bits: 7 / 8 ▪ Stop bits: 1 / 2 ▪ RS Interface: RS-232 / RS-422 / RS-485 ▪ ID: Slave Modbus Address ▪ Cmds: Configured Modbus Master Commands

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
<Port Status>	OK: Port is communicating without error
	Master/Slave Communication Errors: port is having communication errors. Refer to PCB diagnostics (page 122, page 130) for further information about the error.

5.1.2 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-MCMR 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	OFF	The battery voltage is OK and functioning.
		ON	The battery voltage is low or battery is not present. Allow battery to charge by keeping module plugged into rack for 24 hours. If ERR LED still does not go off, contact ProSoft Technology, as the battery is not a user-serviceable item.

5.2 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.
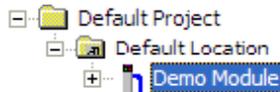
2 On the shortcut menu, choose **DIAGNOSTICS.**
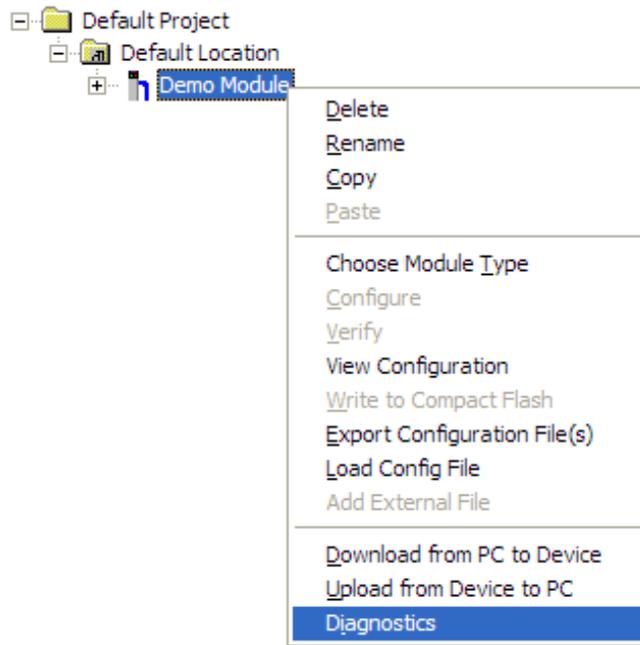
3 In the *Diagnostics* window, click the **SET UP CONNECTION** button to browse for the module's IP address.

4 In the *Connection Setup* dialog box, click the **TEST CONNECTION** button to verify that the module is accessible with the current settings.
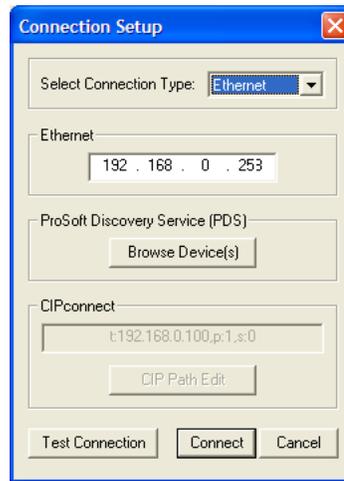
You can also use CIPconnect® to connect to the module through a 1756-ENBT card. Refer to Using CIPconnect to Connect to the Module (page 96) for information on how to construct a CIP path.
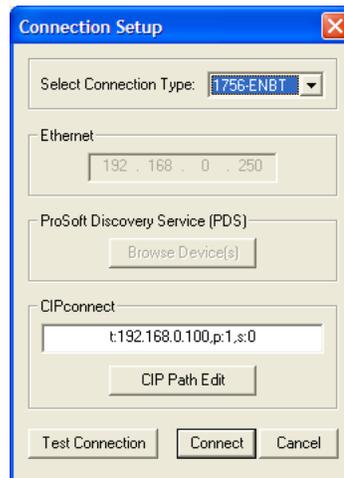
5 If the *Test Connection* is successful, click **CONNECT**.

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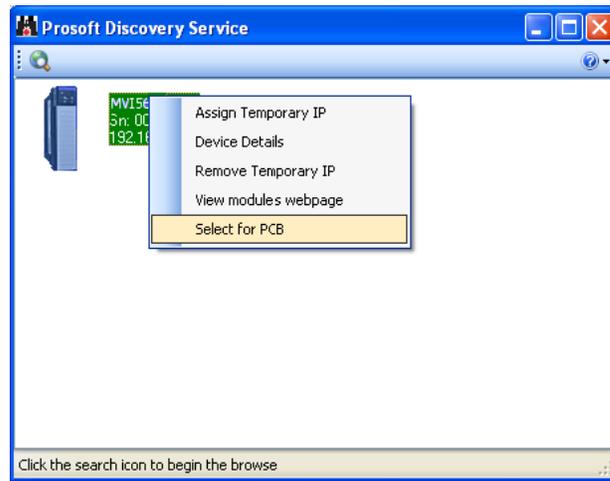
2 Close *ProSoft Discovery Service*, and click the **CONNECT** button again.
3 If these troubleshooting steps fail, verify that the Ethernet cable is connected properly between your computer and the module, either through a hub or switch (using the grey cable) or directly between your computer and the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

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

5.2.1 Connect to the Module's Web Page

The module's internal web server provides access to module status, diagnostics, and firmware updates.

1 In *ProSoft Discovery Service*, select the module to configure, and then click the right mouse button to open a shortcut menu.
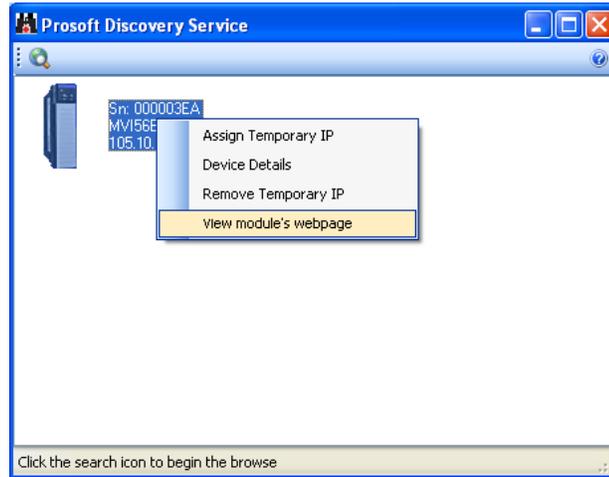
2 On the shortcut menu, choose **VIEW MODULE'S WEBPAGE**.
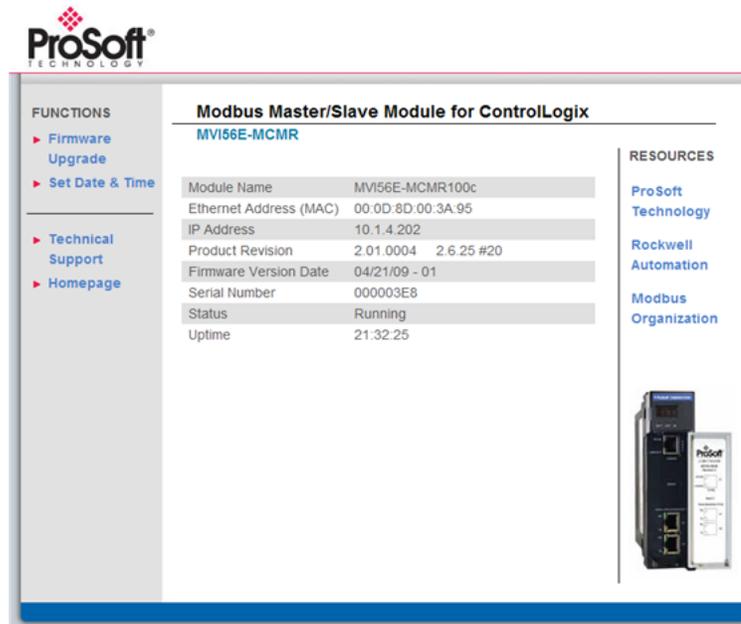
5.2.2 The Diagnostics Menu

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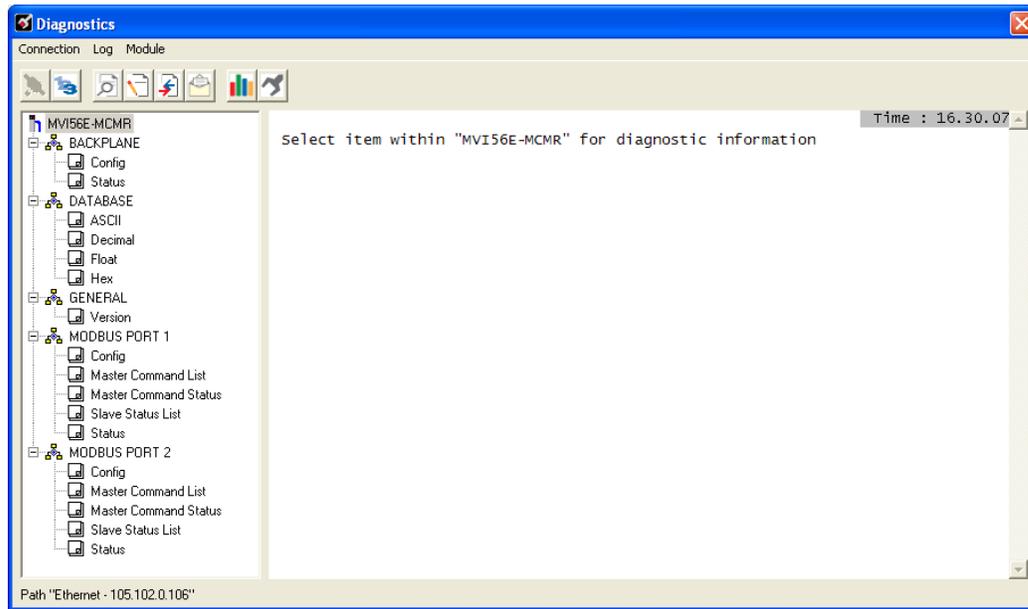
5.2.3 Monitoring Backplane Information

Use the *BACKPLANE* menu to view the backplane status information for the MVI56E-MCMR module.

Backplane Configuration

Click Config to view current backplane configuration settings, including

- Read Start
- Read Count
- Write Start
- Write Count
- Error Status Pointer

The settings on this menu correspond with the **[BACKPLANE CONFIGURATION]** section of the module configuration file.

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.

5.2.4 Monitoring Database Information

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

You can view data in the following formats:

ASCII

```
DATABASE DISPLAY 0 to 99 (ASCII) :

° □ M C E R 2 . 0 1 0 4 0 9 2 1 0 1
j □ ¦ □ § □ j □
¡ □ § □ ã S ã S á S
```

Decimal

```
DATABASE DISPLAY 0 to 99 (DECIMAL) :                    [Refresh Counter: 24]

      0    5520   17229   21061   11826   12592   13360   14640   12594   12592
    892       0    3566    3567       0       0       0       0     892       0
   3566    3567       0       0       0       0   28075   28074   28074       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
```

Float

```
DATABASE DISPLAY 0 to 49 (FLOAT) :                     [Refresh Counter: 8]

-1.42363105E+028  2.11809419E+011  2.56376298E-009  1.68041093E-004  2.56393351E-009
 1.25976732E-042  1.71398323E-030  0.00000000E+000  0.00000000E+000  1.25976732E-042
 1.71398323E-030  0.00000000E+000  0.00000000E+000  1.08282789E+034  4.30548953E-041
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
 0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000  0.00000000E+000
```

Hexadecimal

```
DATABASE DISPLAY 0 to 99 (HEXADECIMAL) :

0000 B164 434D 5245 2E32 3130 3430 3930 3132 3130
038A 0000 0E26 0E27 0000 0000 0000 0000 038A 0000
0E26 0E27 0000 0000 0000 0000 81EE 81ED 81ED 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

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

5.2.5 Monitoring General Information

Use the General Menu to view module version information.

```
MVI56E-MCMR > GENERAL > Version :

PRODUCT NAME CODE            :MCER
SOFTWARE REVISION LEVEL      :2.01
OPERATING SYSTEM REVISION    :0409
RUN NUMBER                   :2101
PROGRAM SCAN COUNTER         :57061
IP ADDRESS                   :10.1.4.202
ETHERNET ADDRESS (MAC)       :00:0d:8d:00:3a:95
BACKPLANE DRIVER VERSION     :2.6
BACKPLANE API VERSION        :1.1
MODULE NAME                  :MVI56E-MCMR
VENDOR ID                    :309
DEVICE TYPE                  :12
PRODUCT CODE                 :5002
SERIAL NUMBER                :000003E8
REVISION                     :2.1
```

The values on this menu correspond with the contents of the module's Miscellaneous Status registers.

5.2.6 Monitoring Modbus Port Information

Use the Modbus Port 1 and Modbus Port 2 menus to view the information for each of the MVI56E-MCMR module's Modbus application ports.

Port Configuration

Use the Port Configuration menu to view configuration settings for Modbus Port 1 and Modbus Port 2.

Master Command List

Use the Master Command List menu to view the command list settings for Modbus Port 1 and Modbus Port 2.

Use the scroll bar on the right edge of the window to view each Modbus Master command.

Note: The Master Command List is available only if the port is configured as a Modbus Master.

Master Command Status

Use the Master Command Status menu to view Master command status for Modbus Port 1 and Modbus Port 2.

A zero indicates no error.

A non-zero value indicates an error. Refer to Error Codes (page 110) for an explanation of each value.

Slave Status List

Use the Slave Status List menu to view the status of each Slave connected to the Modbus Master port.

Slaves attached to the Master Port can have one of the following states:

0	The Slave is inactive and not defined in the command list for the Master Port.
1	The Slave is actively being polled or controlled by the Master Port. This does not indicate that the Slave has responded to this message.
2	The Master Port has failed to communicate with the Slave device. Communications with the Slave is suspended for a user defined period based on the scanning of the command list.
3	Communications with the Slave has been disabled by the ladder logic. No communication will occur with the Slave until this state is cleared by the ladder logic.

Refer to Slave Status Blocks (page 177) for more information.

Port Status

Use the Port Status menu to view status for Modbus Port 1 and Modbus Port 2. During normal operation, the number of requests and responses should increment, while the number of errors should not change.

5.2.7 Data Analyzer

The Data Analyzer mode allows you to view all bytes of data transferred on each port. Both the transmitted and received data bytes are displayed. Use of this feature is limited without a thorough understanding of the protocol.

Configuring the Data Analyzer

Select Timing Interval

Time Ticks help you visualize how much data is transmitted on the port for a specified interval. Select the interval to display, or choose No Ticks to turn off timing marks.

Select the Communication Port to Analyze

You can view incoming and outgoing data for one application port at a time. Choose the application port to analyze.

Select the Data Format

You can view incoming and outgoing data in Hexadecimal (HEX) or Alphanumeric (ASCII) format.

Starting the Data Analyzer

The following illustration shows an example of the Data Analyzer output.
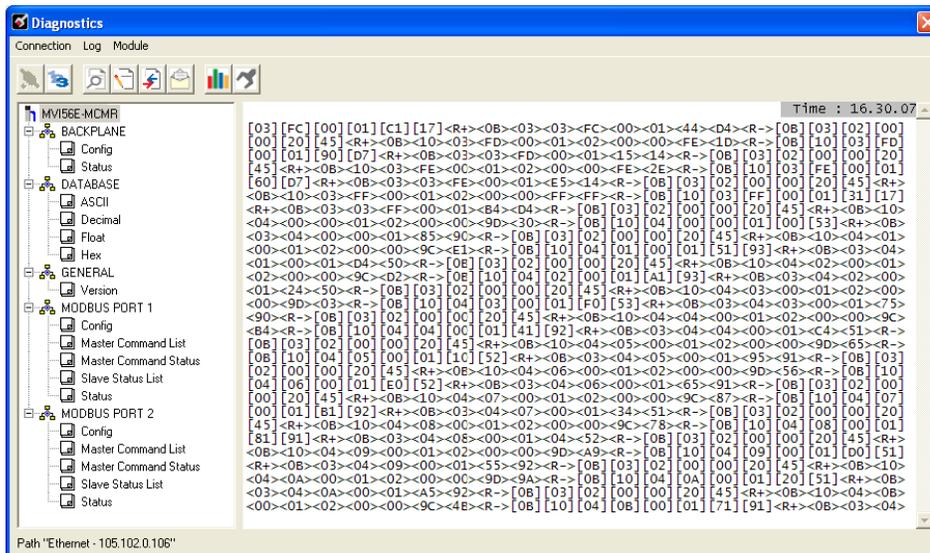
The Data Analyzer can display the following special characters.

Character	Definition
[ ]	Data enclosed in these characters represent data received on the port.
< >	Data enclosed in these characters represent data transmitted on the port.
<R+>	These characters are inserted when the RTS line is driven high on the port.
<R->	These characters are inserted when the RTS line is dropped low on the port.
<CS>	These characters are displayed when the CTS line is recognized high.
_TT_	These characters are displayed when the "Time Tick" is set to any value other than "No Ticks".

Stopping the Data Analyzer

Click to stop Data Analyzer

> **Important:** When in analyzer mode, program execution will slow down. Only use this tool during a troubleshooting session. Before disconnecting from the Config/Debug port, please stop the data analyzer. This action will allow the module to resume its normal high speed operating mode.

Data Analyzer Tips

For most applications, HEX is the best format to view the data, and this does include ASCII based messages (because some characters will not display in the Diagnostics window, and by capturing the data in HEX, we can figure out what the corresponding ASCII characters are supposed to be).

The Tick value is a timing mark. The module will print a _TT for every xx milliseconds of no data on the line. Usually 10milliseconds is the best value to start with.

To save a capture file of your Diagnostics session

1 After you have selected the Port, Format, and Tick, we are now ready to start a capture of this data.

Click to capture the Diagnostics session to a log file

2 When you have captured the data you want to save, click again to stop capturing data.

Click to stop capturing

You have now captured, and saved the file to your PC. This file can now be used in analyzing the communications traffic on the line, and assist in determining communication errors. The log file name is PCB-Log.txt, located in the root directory of your hard drive (normally Drive C).

Now you have everything that shows up on the Diagnostics screen being logged to a file called PCB-Log.txt. You can email this file to ProSoft Technical Support for help with issues on the communications network.

To begin the display of the communications data, start the Data Analyzer. When the Data Analyzer is running, you should see something like this.
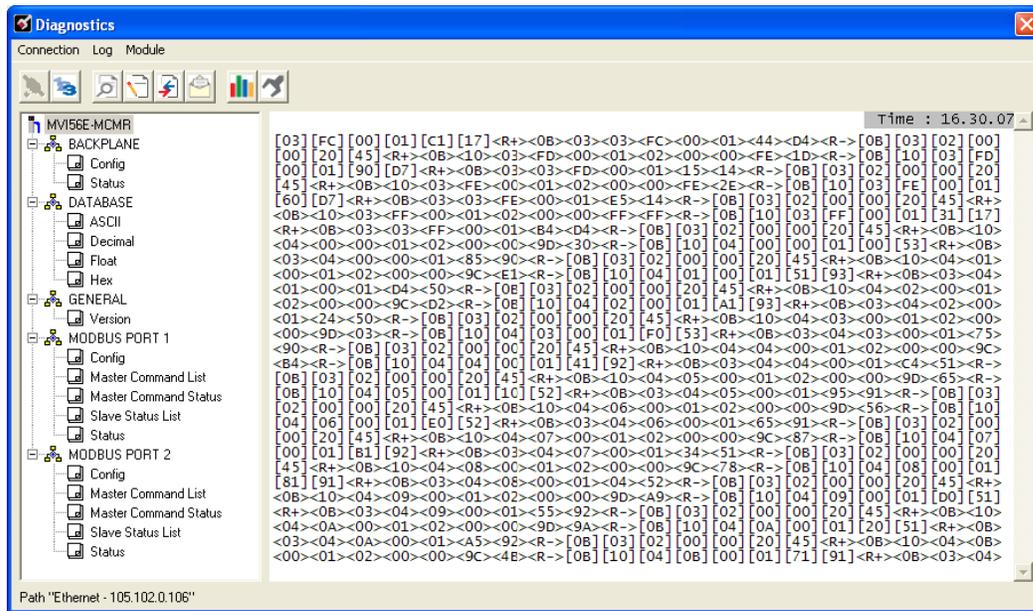
The <R+> means that the module is transitioning the communications line to a transmit state.

All characters shown in <> brackets are characters being sent out by the module.

The <R-> shows when the module is done transmitting data, and is now ready to receive information back.

And finally, all characters shown in the [ ] brackets is information being received from another device by the module.

After taking a minute or two of traffic capture, stop the Data Analyzer.

Click to stop Data Analyzer

5.3   Reading Status Data from the Module

The MVI56E-MCMR module returns a 33-word Status Data block that can be used to determine the module's operating status. This data is transferred to the ControlLogix processor with an MSG instruction. For a complete listing of the status data object, refer to MVI56E-MCMR Status Data Definition (page 177).

5.3.1   Required Hardware

You can view configuration information, perform maintenance, and send (upload) or receive (download) configuration files through the module's Ethernet port.

ProSoft Technology recommends the following minimum hardware to connect your computer to the module:

- 80486 based processor (Pentium preferred)
- 1 megabyte of memory
- Use the included Ethernet cable to connect the module to an Ethernet hub or a 10/100 Base-T Ethernet switch, or directly to the Ethernet port on your PC.

5.3.2   Viewing the Error Status Table

Command execution status and error codes for each individual command are stored in a Master Command Status/Error List, held in the module's internal memory. There are several ways to view this data.

- View Command Status, Slave Status and Port Status in the Diagnostics dialog box in ProSoft Configuration Builder (page 132).
- Configure the Command Error Pointer parameter (<CmdErrPtr>) to copy the status/error values into the User Database area of module memory.
- Copy this table to a section of the ReadData area, where you can view it in the **<READDATAARRAY>** tag array in the ControlLogix controller tag database. You can use these values for communications status monitoring and alarming.
  - o   **COMMAND ERROR POINTER = "MCMR.CONFIG.PORTX.CMDERRPTR"**
  - o   **<READDATAARRAY> = "MCMR.DATA.READDATA[X]"**

These variables would hold the literal tag names in the sample program or Add-On Instruction. Use these variables to accommodate future ladder or tag changes while maintaining backward compatibility.

5.4 Communication Error Codes

During module configuration download, the OK and APP LEDs will cycle through various states. If the OK LED remains RED and the APP LED remains OFF or RED for a long period of time, look at the configuration error words in the configuration request block. The structure of the block is shown in the following table.

Offset	Description	Length
0	Reserved	1
1	9000	1
2	Module Configuration Errors	1
3	Port 1 Configuration Errors	1
4	Port 2 Configuration Errors	1
5 to 38	Spare	34
39	-2 or -3	1

The bits in each configuration word are shown in the following table. The module configuration error word has the following definition:

Bit	Description	Value
0	Read block start value is greater than the database size.	0x0001
1	Read block start value is less than zero.	0x0002
2	Read block count value is less than zero.	0x0004
3	Read block count + start is greater than the database size.	0x0008
4	Write block start value is greater than the database size.	0x0010
5	Write block start value is less than zero.	0x0020
6	Write block count value is less than zero.	0x0040
7	Write block count + start is greater than the database size.	0x0080
8		0x0100
9		0x0200
10		0x0400
11		0x0800
12		0x1000
13		0x2000
14		0x4000
15		0x8000

The port configuration error words have the following definitions:

Bit	Description	Value
0	Type code is not valid. Enter a value from 0 (Master) to 1 (Slave).	0x0001
1	The float flag parameter is not valid.	0x0002
2	The float start parameter is not valid.	0x0004
3	The float offset parameter is not valid.	0x0008
4	Protocol parameter is not valid.	0x0010
5	Baud rate parameter is not valid.	0x0020
6	Parity parameter is not valid.	0x0040
7	Data bits parameter is not valid.	0x0080
8	Stop bits parameter is not valid.	0x0100
9	Slave ID is not valid.	0x0200
10	Input bit or word, output word and/or holding register offset(s) are not valid.	0x0400
11	Command count parameter is not valid.	0x0800
12	Spare	0x1000
13	Spare	0x2000
14	Spare	0x4000
15	Spare	0x8000

Correct any invalid data in the configuration for proper module operation. When the configuration contains a valid parameter set, all the bits in the configuration words will be clear. This does not indicate that the configuration is valid for the user application. Make sure each parameter is set correctly for the specific application.

Note: If the APP, BP ACT and OK LEDs blink at a rate of every one-second, this indicates a serious problem with the module. Call ProSoft Technology Support to arrange for repairs.

5.4.1 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.

5.4.2 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-MCMR. Verify that all modules in the rack are correctly configured in the ladder logic.

Module Errors

Problem description	Steps to take
BP ACT LED (not present on MVI56E modules) remains OFF or blinks slowly 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 it, and then restore power to the rack.

6 Reference

In This Chapter

6.1   About the Modbus Protocol

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

The original Modbus specification uses a serial connection to communicate commands and data between Master and Slave devices on a network. Later enhancements to the protocol allow communication over other types of networks.

Modbus is a Master/Slave protocol. The Master establishes a connection to the remote Slave. When the connection is established, the Master sends the Modbus commands to the Slave. The MVI56E-MCMR module can work as a Master and as a Slave.

The MVI56E-MCMR module also works as an input/output module between itself and the Rockwell Automation backplane and processor. The module uses an internal database to pass data and commands between the processor and Master and Slave devices on Modbus networks.

6.2 Specifications

The MVI56E Modbus Master/Slave Communication Module with Reduced Data Block allows users to integrate Modbus devices and networks into the Rockwell Automation® ControlLogix® architecture.

Compatible devices include not only Modicon® PLCs (almost all support the Modbus protocol) but also a wide assortment of processors, HMI displays, SCADA systems and field devices made by a variety of manufacturers. The module acts as an input/output module between the Modbus network and the ControlLogix processor. The data transfers between the module and the processor are asynchronous from communications on the Modbus network. A 5000-word register space in the module hold the data to exchanged between the processor and the Modbus network.

6.2.1 General Specifications

- Backward-compatible with previous MVI56-MCMR versions
- Single-Slot, 1756 ControlLogix® backplane compatible
- 10/100 Mbps Ethernet port with Auto Cable Crossover Detection
- User-definable module data memory mapping of up to 5000, 16-bit registers
- CIPconnect®-enabled network configuration and diagnostics monitoring using ControlLogix 1756-ENxT modules and EtherNet/IP® pass-thru communications
- Reduced I/O image size designed specifically to optimize remote rack implementations
- ProSoft Configuration Builder (PCB) software supported, a Windows-based graphical user interface providing simple product and network configuration
- Sample Ladder Logic and Add-On Instruction (AOI) used for data transfers between module and processor
- 4-character, scrolling LED display of status and diagnostic data in plain English
- ProSoft Discovery Service (PDS) software finds the module on the network and assigns a temporary IP address to facilitate module access
- Personality Module (non-volatile CF card) to store network configuration allowing quick in-the-field product replacement by transferring the CF card

6.2.2 Hardware Specifications

Specification	Description
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 to 150 Hz
Relative Humidity	5% to 95% (without condensation)
LED Indicators	Battery Status (ERR) Application Status (APP) Module Status (OK)
4-Character, Scrolling, Alpha-Numeric LED Display	Shows Module, Version, IP, Port Status, P1 and P2 Settings, and Error Information
Debug/Configuration Ethernet port (E1 - Config)	
Ethernet Port	10/100 Base-T, RJ45 Connector, for CAT5 cable Link and Activity LED indicators Auto-crossover cable detection
Serial Application ports (P1 & P2)	
Software configurable communication parameters	Baud rate: 110 baud to 115.2 kbps RS-232, RS-485, and RS-422 Parity: none, odd or even Data bits: 5, 6, 7, or 8 Stop bits: 1 or 2 RTS on/off delay: 0 to 65535 milliseconds Full hardware handshaking control (optional) Radio and modem support
App Ports (P1, P2)	RJ45 (DB-9M with supplied adapter cable) Configurable RS-232 hardware handshaking 500V Optical isolation from backplane RS-232, RS-422, RS-485 jumper-select, per port RX (Receive) and TX (Transmit) LEDs, each port
Shipped with Unit	RJ45 to DB-9M cables for each serial port 5 foot Ethernet Straight-Thru Cable (Gray)

6.2.3 General Specifications - Modbus Master/Slave

Communication parameters	Baud Rate: 110 to 115K baud Stop Bits: 1 or 2 Data Size: 7 or 8 bits Parity: None, Even, Odd RTS Timing delays: 0 to 65535 milliseconds
Modbus Modes	RTU mode (binary) with CRC-16 ASCII mode with LRC error checking
Floating Point Data	Floating point data movement supported, including configurable support for Enron, Daniel®, and other implementations
Modbus Function Codes Supported	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 Holding Register 8: Diagnostics (Slave Only, Responds to Subfunction 00) 15: Force(Write) Multiple Coils 16: Preset (Write) Multiple Holding Registers 17: Report Slave ID (Slave Only) 22: Mask Write Holding Register (Slave Only) 23: Read/Write Holding Registers (Slave Only)

6.2.4 Functional Specifications

Modbus Master

A port configured as a virtual Modbus Master actively issues Modbus commands to other nodes on the Modbus network, supporting up to 100 commands on each port. The Master ports have an optimized polling characteristic that polls slaves with communication problems less frequently.

Command List	Up to 100 command per Master port, each fully configurable for function, slave address, register to/from addressing and word/bit count.
Polling of command list	Configurable polling of command list, including continuous and on change of data, and dynamically user or automatic enabled.
Status Data	Error codes available on an individual command basis. In addition, a slave status list is maintained per active Modbus Master port.

Modbus Slave

A port configured as a Modbus slave permits a remote Master to interact with all data contained in the module. This data can be derived from other Modbus slave devices on the network, through a Master port, or from the ControlLogix processor.

Node address	1 to 247 (software selectable)
Status Data	Error codes, counters and port status available per configured slave port

6.3   Functional Overview

6.3.1   Processor/Module Data Transfers

The MVI56E-MCMR module communicates directly over the ControlLogix backplane or across any supported 1756 network bridge (the most common being 1756-CNBx Control Net Bridge and 1756-ENxT EtherNet/IP Bridge). Data travels between the module and the ControlLogix processor across the backplane or network using the module's Input and Output Images and MSG instructions.

The data update frequency is determined by the Requested Packet Interval (RPI) defined in the module's I/O configuration and the communication load and speed on the Modbus, ControlNet, or EtherNet/IP networks. When the module is installed in the ControlLogix chassis, typical backplane update rates range from 1 to 10 milliseconds. Execution time for MSG instruction data transfers are dependant on the amount of unscheduled time available on the backplane or network and how frequently the MSG instruction is executed.

Data received by the Modbus driver is placed in the module's internal memory in an area designated to receive it. The data in this area is then transferred to the processor in the I/O Input Image. This data is processed by ladder logic to fill the **MCMR.DATA.READDATA** array controller tags. The Input Image size is 42 words per image block, 40 of which are user data, along with two control words. Larger amounts of user data can be moved from the module to the processor by using multiple sequential block transfers. The module will automatically sequence multiple 40-word blocks until the total amount of user data has been moved. The module calculates the required number of read data blocks by dividing the **READ REGISTER COUNT** parameter in the configuration file by 40 and rounding up to the next higher whole integer value.

The processor inserts data in the module's Output Image to transfer to the module. The module's program extracts the data and stores it in the internal module database, so that it may be transmitted by the Master driver to Slaves on serial network. Additionally, the ControlLogix processor can send special function blocks to the module to instruct it to perform special tasks. The Output Image size is 42 words per image block, 40 of which are user data, along with one control word. Larger amounts of user data can be moved from the processor to the module by using multiple sequential block transfers. The module will automatically sequence multiple 40-word blocks until the total amount of user data has been moved. The module calculates the required number of write data blocks by dividing the Write Register Count parameter in the configuration file by 40 and rounding up to the next higher whole integer value.

Special function blocks are also passed between the module and the processor using MSG instructions initiated under ladder logic control. These blocks are transferred between the processor and the module only when triggered by user-programmed logic.

The following illustration shows the data transfer method used to move data between the ControlLogix processor, the MVI56E-MCMR module, and the serial network. This applies only for the scheduled I/O data.
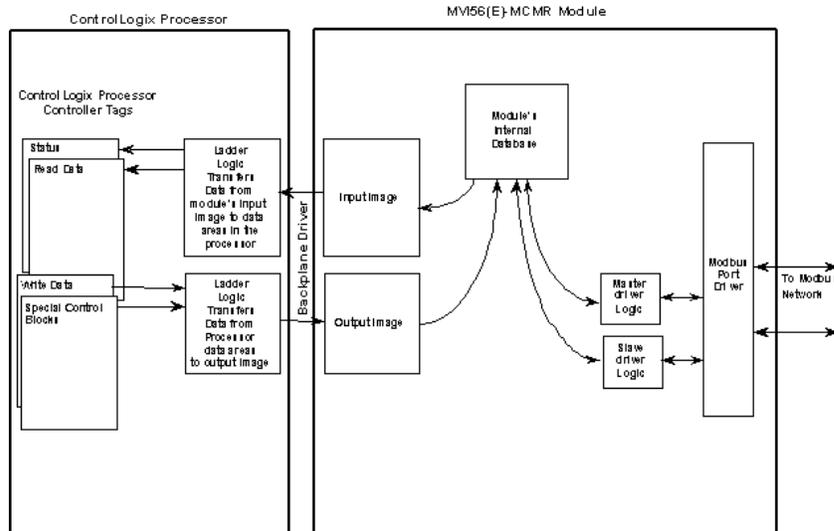
As shown in the diagram, all data transferred between the module and the processor over the backplane is through the Input and Output Images. Ladder logic must be written in the ControlLogix processor to interface the Input and Output Image data defined in the controller tags. The user is responsible for handling and interpreting all data received on the application ports and transferred to the Input Image.

Using Data Blocks

Each block transferred between the module and the processor contains block identification codes that define the content or function of the block of data transferred.

Blocks -1 and 0 contain no data when transferred from the processor to the module. Blocks 1 to 125 transfer data stored or to be stored in the module's database 40-words of data per block. These data blocks send data from module to the processor (monitored data received from the devices on the serial network) and to send data from the processor to the module (control data to send to the end devices). Block identification codes 9901 to 9999 are used for special function blocks to control the module.

The following table describes the block identification codes used by the module.

Type	Block # Range	Block Descriptions	Available When Port:	
			Is Master?	Is Slave?
I/O	-1 and 0	Null (Used when Read or Write Register Count = 0)	Yes	Yes
I/O	1 to 125	Read or Write Data Blocks	Yes	Yes
I/O	1000 to 1125	Initialize Output Data Blocks	Yes	Yes
MSG	9250	Module Error/Status Data Block	Yes	Yes
MSG	9901	Event Command Block for Port 1	Yes	No
MSG	9911	Event Command Block for Port 2	Yes	No
MSG	9902	Command Control Block for Port 1	Yes	No
MSG	9912	Command Control Block for Port 2	Yes	No
MSG	9950	Get Command Error List for Port 1 Block	Yes	No
MSG	9951	Get Command Error List for Port 2 Block	Yes	No
MSG	9960	Get Slave Enable/Disable Data for Port 1 Block	Yes	No
MSG	9961	Get Slave Enable/Disable Data for Port 2 Block	Yes	No
I/O	9998	Warm Boot Request Block	Yes	Yes
I/O	9999	Cold Boot Request Block	Yes	Yes

As you can see, some data is transferred between the module and the ControlLogix processor using the Input and Output Images (Type=I/O), and some is transferred using MSG blocks (Type=MSG). Data transferred using the Input and Output Images is used for high-speed, deterministic delivery time data, controlled by the Requested Packet Interval (RPI) assigned to the module in the I/O configuration in RSLogix 5000. The MSG data is used for lower priority data and is transferred using MSG instructions under ladder logic control. MSG data is handled when there is time available in the unscheduled bandwidth of the network.

6.3.2 Normal Data Transfer Blocks

Normal data transfer includes the transferring of data received by, or to be transmitted to, the Modbus drivers and the status data. This data is transferred through read (Input Image) and write (Output Image) blocks. Refer to Module Data (page 118) for a description of the data objects used with the blocks and the ladder logic required. The following topics discuss the structure and function of each block.

Read Block

Read Blocks transfer data from the module to the ControlLogix processor's **MCMR.DATA.READDATA** controller tag array. 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 to 125)	1
1 to 40	Read Data	40
41	Read Block ID (1 to 125)	1

The *Read Block Identification Code* (word 41) is used to signal to the ControlLogix processor that a new block is ready for processing. It also tells the processor where in the **MCMR.DATA.READDATA** controller tag array to place the data contained in the block.

If the value of the code is set to 1, the block contains the 40 words of data from the module database starting at the address specified in the configuration file parameter, **READ START REGISTER**. This data will be put into the ReadData array, starting at **READDATA[0]** up to **READDATA[39]**.

Read Block ID 2 would contain the next consecutive 40 words from the module database to be placed in **READDATA[40]** up to **READDATA[79]** and so on, up to the total amount of data words specified in the configuration parameter, **READ REGISTER COUNT**.

The block also contains the *Write Block Identification Code* the module expects to receive from the processor. Under normal data transfer conditions, the ladder logic should use the *Write Block Identification Code* to build the appropriate Output Image data block, unless a special function block is required. The special function blocks will be discussed in the next section.

Write Block

Write Blocks transfer data from the ControlLogix processor's **MCMR.DATA.WRITEDATA** controller tag array to the module. The following table describes the structure of the Output Image used to transfer this data.

Write Block from Processor to Module

Word Offset	Description	Length
0	Write Block ID (1 to 125)	1
1 to 40	Write Data	40
41	Spare	1

The *Write Block Identification Code* specifies the index to the 40 words that are currently being transferred from the **MCMR.DATA.WRITEDATA** array to the module. If the code is set to -1 or 0, the Write Block contains no valid data, as would be the case if the configuration parameter, **WRITE REGISTER COUNT**, was set to 0, indicating the user did not have any data to move from the processor to the module.

If the word contains a value from 1 to 125, the data contained in the block will be placed in the appropriate position of the module's database. Data from MCMR.DATA.WriteData[0] through [39] will be transferred using Write Block ID1 and will be placed in the module's user database area beginning at the address specified in the configuration file parameter, **WRITE START REGISTER**. Write Block ID2 will contain data from MCMR.DATA.WriteData[40} through [79] and will be placed in the next consecutive 40-word block of the module's user database. Data will continue being transferred in 40-word blocks for the total amount of data words specified in the parameter, **WRITE REGISTER COUNT**.

Under normal data transfer conditions, the value used for the Write Block Identification Code should be the same as that received in Read Block (Input Image) Word 0, unless some special function block is required. The special function blocks will be discussed in the next section.

6.3.3 Special Function Blocks

Write Block Identification Codes greater than 125 cause the module to perform special functions. Some of these blocks are high-priority and are moved between the processor and the module through the Input and Output Images. Others are of low-priority and are moved using Message (MSG) instructions. Each *Special Function Write Block Code* has a corresponding *Special Function Read Block Code*, which will be returned to the processor in the next Input Image, to confirm the module received and processed the Special Function Write request. The *Special Function Block Codes* recognized and used by the module are defined in the following topics.

Module Status Block (9250)

The *General Module Status* block contains some basic information about the module itself and diagnostic counters to help monitor activity on each port and across the ControlLogix backplane. The block can be requested as needed and is available regardless of whether the module's ports are configured as Masters or Slaves. Use block identification code 9250 to request this *General Module Status* block.

This block of data is requested using the "Get Attribute Single" service type MSG instruction.
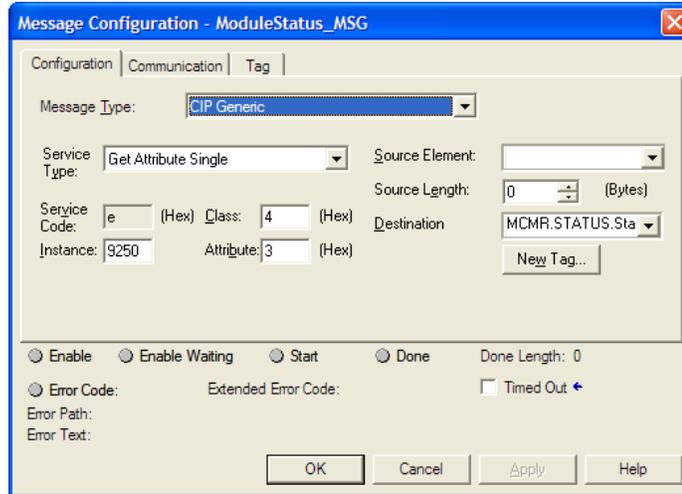
The following table describes the format of the 33-word data block returned to the processor by this MSG instruction.

Block Response from Module to Processor

Word Offset	Content	Description
0	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
1 to 2	Product Code	The four bytes of these two words contain the ASCII code values of the 4-letter product code.
3 to 4	Product Version	These two registers contain the product version of the current running module firmware.
5 to 6	Operating System	These two registers contain the month and year values for the program operating system.
7 to 8	Run Number	These two registers contain the run number value for the currently running software.
9	Port 1 Command List Requests	When Port 1 is a Master, this field contains the number of requests made from Port 1 to Slave devices on the network.
10	Port 1 Command List Response	When Port 1 is a Master, this field contains the number of Slave response messages received on the port.

Word Offset	Content	Description
11	Port 1 Command List Errors	When Port 1 is a Master, this field contains the number of command errors processed on the port. These errors could be due to a bad response or bad command.
12	Port 1 Requests	This field contains the total number of messages sent from the port when it is a Master and the total number or messages received when it is a Slave.
13	Port 1 Responses	This field contains the total number of messages sent from the port when it is a Slave and the total number or messages received when it is a Master.
14	Port 1 Errors Sent	When Port 1 is a Slave, this field contains the total number of message errors sent out of the port. The Slave will send error messages when a command received is invalid.
15	Port 1 Errors Received	When Port 1 is a Master, this field contains the total number of message errors received on the port from Slaves on the network. Slaves send error responses when they think they have received an invalid command or a command with invalid parameters.
16	Port 2 Command List Requests	When Port 2 is a Master, this field contains the number of requests made from Port 2 to Slave devices on the network.
17	Port 2 Command List Response	When Port 2 is a Master, this field contains the number of Slave response messages received on the port.
18	Port 2 Command List Errors	When Port 2 is a Master, this field contains the number of command errors processed on the port. These errors could be due to a bad response or bad command.
19	Port 2 Requests	This field contains the total number of messages sent from the port when it is a Master and the total number or messages received when it is a Slave.
20	Port 2 Responses	This field contains the total number of messages sent from the port when it is a Slave and the total number or messages received when it is a Master.
21	Port 2 Errors Sent	When Port 2 is a Slave, this field contains the total number of message errors sent out of the port. The Slave will send error messages when a command received is invalid.
22	Port 2 Errors Received	When Port 2 is a Master, this field contains the total number of message errors received on the port from Slaves on the network. Slaves send error responses when they think they have received an invalid command or a command with invalid parameters.
23	Read Block Count	This field contains the total number of Input Image blocks transferred across the backplane from the module to the processor.
24	Write Block Count	This field contains the total number of Output Image blocks transferred across the backplane from the processor to the module.

Word Offset	Content	Description
25	Parse Block Count	This field contains the total number of Output Image blocks received from the processor that were considered valid by the module firmware (successfully parsed or understood). If backplane communications are normal, this value will be equal or nearly equal to the *Write Block Count* value. If this counter does not increment along with the *Write Block Count* counter, you have a serious backplane communication problem between the processor and the module. Check the module configuration in the I/O Configuration section of your process logic for possible additional error information.
26	*Event Command* Block Count	This field contains the total number of *Event Command* blocks received from the processor.
27	*Command Control* Block Count	This field contains the total number of *Command Control* blocks received from the processor.
28	Backplane Communication Error Block Count	This field contains the total number of block errors recognized by the module. If this counter is incrementing, you have a serious backplane communication problem between the processor and the module. Check the module configuration in the I/O Configuration section of your process logic for possible additional error information.
29	Port 1 Current Error	For a Slave port, this field contains the value of the current error code returned to a remote Master. For a Master port, this field contains the *Command List* index of the currently executing command that is receiving an error from a Slave.
30	Port 1 Last Error	For a Slave port, this field contains the value of the most recent previous error code returned to a remote Master. For a Master port, this field contains the *Command List* index of the command which received the most recent previous error from a Slave.
31	Port 2 Current Error	For a Slave port, this field contains the value of the current error code returned to a remote Master. For a Master port, this field contains the *Command List* index of the currently executing command that is receiving an error from a Slave.
32	Port 2 Last Error	For a Slave port, this field contains the value of the most recent previous error code returned to a remote Master. For a Master port, this field contains the *Command List* index of the command which received the most recent previous error from a Slave.

Event Command Blocks (9901, 9911)

The *Event Command* special function is applicable only when the module's port is configured as a Modbus Master. *Event Commands* are best used to send commands based on special process conditions, such as emergency shutdowns or device-specific resets. Whenever an *Event Command* data block is received by the module, it will insert the requested command into the beginning of the Command Queue, so that the special command is sent before the next regular polling command.

Sending a message (MSG instruction) containing *Event Command Block Identification Code* **9901** for Port 1 or **9911** for Port 2 will cause the module to issue one user-constructed command. All the data required for one Modbus command must be included in the MSG instruction using the *Event Command Block ID Code*.

If you use the provided sample ladder logic or Add-On Instruction (AOI), the Modbus Command parameter data required for this special function block will be placed in the controller tag array, **MCMR.CONTROL.EVENTCMDP1[0]**. for Port 1 or **MCMR.CONTROL.EVENTCMDP2[0]** for Port 2. Once the command parameters have been properly loaded into this array element, the *Event Command* special function can be executed by setting a value of one (1) into the controller tag, **MCMR.CONTROL.EVENTTRIGGERP1** for Port 1 or **MCMR.CONTROL.EVENTTRIGGERP2** for Port 2.

You will notice that **MCMR.CONTROL.EVENTCMDP1[X]** and **MCMR.CONTROL.EVENTCMDP2[X]** are actually 100-element arrays, capable of holding up to 100 pre-configured command parameter sets. However, at this time, only the first element of each array, **MCMR.CONTROL.EVENTCMDP1[0]** or **MCMR.CONTROL.EVENTCMDP2[0]** is used in the MSG instructions of the sample ladder logic or AOI. If you wish to use the other 99 elements of this array to hold potential *Event Commands* that you might want to execute, you will need to create additional logic to use them. You could:

1 Create logic to COPY the parameter data from any array element, 1-99, into element 0 before triggering the *Event Command* MSG instruction.
2 Create logic that duplicates the sample MSG instruction, modify it for a specific array element, create a unique trigger tag for this logic, and use these to send the specific pre-configured *Event Command* contained in that array element.

The following table lists the parameters required for a user-constructed *Event Command* and shows the order in which the parameters will be passed by the MSG instruction. You will notice, these are the same parameters and are in the same order as in any normal polling command you create in the configuration file for the Master port.

Block Request from Processor to Module

Word Offset	Definitions
0	Enable (must be set to 1)
1	Internal DB Address (0-4999)
2	Poll Interval (set to 0)
3	Count (1-125, or maximum supported by the target Slave device)
4	Swap (0, 1, 2, or 3)
5	Device (Modbus Slave Device Address Number of target Slave)
6	Function (Modbus Function Code: 1, 2, 3, 4, 5, 6, 15, or 16)
7	Device Address (0-9999, address offset in target Slave database)
8	Reserved (set to 0)
9	Reserved (set to 0)

Refer to Master Command Configuration (page 60) for a detailed definition of the fields contained in this block. They are the same as those used in constructing the commands in ProSoft Configuration Builder (PCB) in the **MODBUS PORT 1 COMMANDS** or **MODBUS PORT 2 COMMANDS** lists.

The *Send Event Command* message uses the following parameters in the MSG configuration:
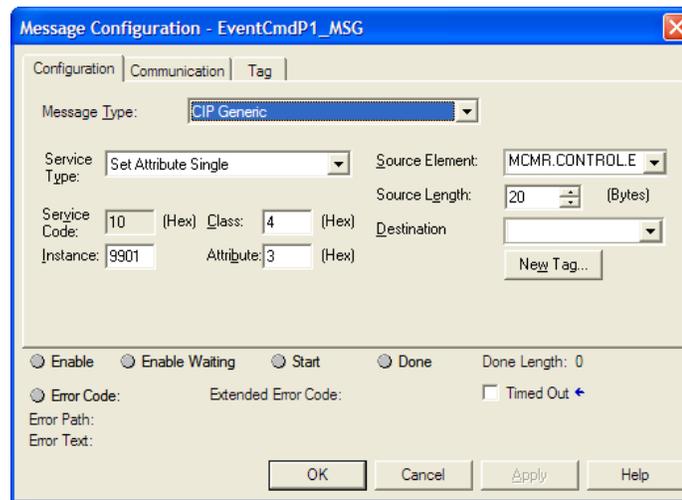
Status Data will be returned to the processor by using a "Get Attribute Single" service type MSG instruction. If you follow the provided sample ladder logic or use the provided Add-On Instruction (AOI), the *Get Event Command Status MSG* will be triggered at the same time as the *Send Event Command* message by the same controller tag, **MCMR.CONTROL.EVENTTRIGGERP1** for Port 1 or **MCMR.CONTROL.EVENTTRIGGERP2** for Port 2. The *Get Event Command Status* message uses the following parameters in the MSG configuration:
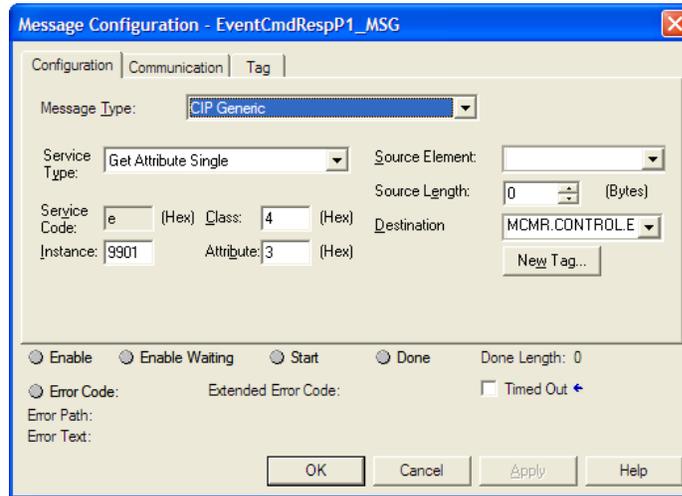
A 5-word response will be passed back through the MSG instruction to the controller tag array, **MCMR.CONTROL.EVENTCMDRESPP1**. or **MCMR.CONTROL.EVENTCMDRESPP2**, depending on whether the Event Command was sent for Port 1 (**9901**) or Port 2 (**9911**). The following table lists the 5-word response data received:

Block Response from Module to Processor

Word Offset	Definitions
0	9901 or 9911 Event Command ID number
1	0 = Fail-command not added to the command queue,
	1 = Success-command added to the command queue.
2	Reserved for future use (will always be zero)
3	Reserved for future use (will always be zero)
4	Reserved for future use (will always be zero)

Please note that the status returned in Word 1 indicates only that the command received from the *Send Event Command* MSG was considered a valid command and was successfully added to the top of the Command Queue as the next command to be sent. A "Success" result in this data block does not indicate:

- Whether the command was successfully sent on the Modbus Network
- Whether the Slave received or responded to the command
- Whether the Slave's response (in any) was valid

There are many potential reasons why a command might fail after having been successfully added to the Command Queue. For more details, see Standard Modbus Protocol Errors or Module Communication Error Codes.

Command Control Blocks (9902 or 9912)

The *Command Control* special function is applicable only when the module's port is configured as a Modbus Master. *Command Control* is best used to send commands based on special process conditions, such as emergency shutdowns, device-specific resets or any conditions which might require priority polling of a specific Slave Device. Whenever a *Command Control* data block is received by the module, it will insert the requested command or commands into the beginning of the Command Queue, so that the special command or commands will be sent before the next regular polling command.

Sending a message (MSG instruction) containing *Command Control Block Identification Code* **9902** for Port 1 or **9912** for Port 2 will cause a list of from one (1) to twenty (20) commands to be placed in the Command Queue using commands from the user-created PCB Command Lists. These lists are the ones created in the **MODBUS PORT 1 COMMANDS** or **MODBUS PORT 2 COMMANDS** sections of the PCB configuration file that was downloaded to the module and that are used by the module for normal, automatic, repetitive polling.

Any command in the Command List may be given execution priority using a *Command Control* special function block MSG, regardless of the value set in the **ENABLE** parameter for that command. However, commands placed in the Command List intended for exclusive use with *Command Control* will most often have their **ENABLE** parameter set to zero(0) and will not be executed as part of a regular polling routine. Commands enabled using *Command Control* will be added to the Command Queue for a one-time execution each time the MSG instruction is activated.

Command Control will not enable commands for normal, repeated polling if they are not already enabled in the Command List. But *Command Control* may be used to force commands that are enabled for normal polling to the top of the Command Queue so that they will be executed out of their normal polling sequence, as well as being executed in the normal polling order.

If you use the provided sample ladder logic or Add-On Instruction (AOI), the Modbus Command parameter data required for this special function block will be placed in the controller tag array,
MCMR.CONTROL.CMDCONTROLP1.CMDCONTROLDATA for Port 1 or
MCMR.CONTROL.CMDCONTROLP2.CMDCONTROLDATA for Port 2. You will notice that these are 21-element arrays. The structure of the *Command Control* special function data block and the elements in the associated arrays is shown in the following table.

Block Request from Processor to Module

Word Offset or Controller Tag Array Element	Data Field(s)	Description
0	Command Count	This field contains the number of commands to add to the Command Queue. Valid values for this field are 1 to 20.
1 to 20	Command Index or Indexes of the Command or Commands to be added to the Command Queue	These 20 words of data contain the index numbers to commands in the Command List that need to be added to the Command Queue. The commands in the list will be placed in the command queue for immediate processing by the module. The command indexes may be listed in any order and the same command index may be repeated in the list. Valid values for these 20 fields are 0 to 99.

Once the command count and index or indexes have been properly loaded into this array, the *Command Control* special function can be executed by setting a value of one (1) into the controller tag,
MCMR.CONTROL.CMDCONTROLP1.CMDTRIGGER for Port 1 or
MCMR.CONTROL.CMDCONTROLP2CMDTRIGGER for Port 2.

The *Send Command Control* message uses the following parameters in the MSG configuration:
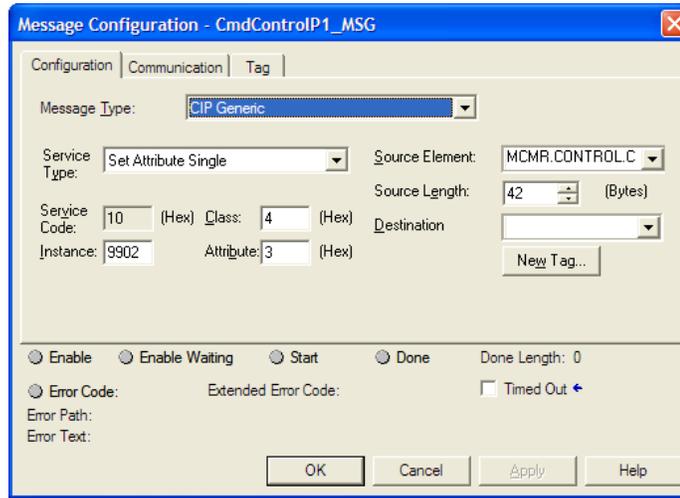
Status Data can be returned to the processor using a "Get Attribute Single" service type MSG instruction. If you follow the provided sample ladder logic or use the provided Add-On Instruction (AOI), the *Get Command Control Status* MSG will be triggered at the same time as the *Send Command Control* message by the same controller tag, **MCMR.CONTROL.CMDCONTROLP1.CMDTRIGGER** for Port 1 or **MCMR.CONTROL.CMDCONTROLP2CMDTRIGGER** for Port 2.

The *Get Command Control Status* message uses the following parameters in the MSG configuration:
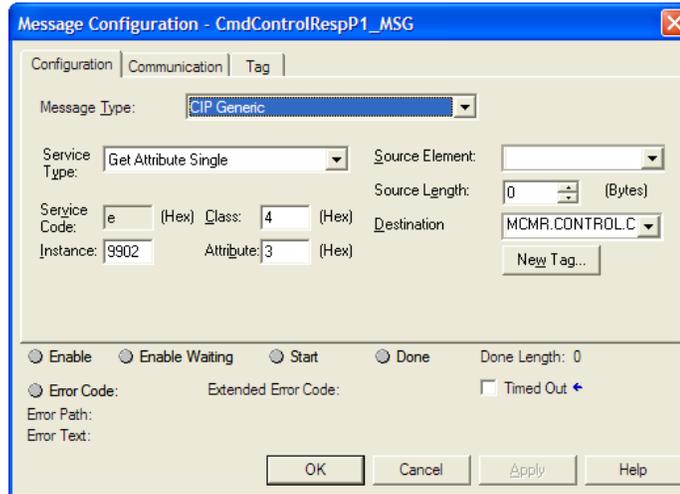
A 5-word response will be passed back through the MSG instruction to the controller tag array, **MCMR.CONTROL.CMDCONTROLP1.CMDCONTROLRESP** or **MCMR.CONTROL.CMDCONTROLP1.CMDCONTROLRESP**, depending on whether the Event Command was sent for Port 1 (**9901**) or Port 2 (**9911**). The following table lists the 5-word response data received.

Block Response from Module to Processor

Word Offset	Definitions
0	9902 or 9912 Command Control ID Number
1	0 = Fail-No Special Commands were added to the Command Queue
	>0 = Success - Indicates the number of command control successfully added to the Command Queue.
	Note: This number should match the Command Count (Word 0 of the *Send Command Control* MSG)
2	Reserved for future use (will always be zero)
3	Reserved for future use (will always be zero)
5	Reserved for future use (will always be zero)

Please note that the status returned in Word 1 indicates only that the command or commands from the *Send Command Control* MSG was or were successfully added to the top of the Command Queue as the next command or commands to be sent. A "Success" result in this data block does not indicate:

▪ Whether the command was successfully sent on the Modbus Network
▪ Whether the Slave received or responded to the command
▪ Whether the Slave's response (in any) was valid

There are many potential reasons why a command might fail after having been successfully added to the Command Queue. For more details, see Standard Modbus Protocol Errors or Module Communication Error Codes. But, unlike *Event Commands*, which do not provide any status feedback on actual command execution, the execution status of commands sent using *Command Control* can be obtained.

Since these commands are part of the Command List, the execution status of these commands is available in the *Master Command Error List*. The status shown in this list will be updated every time the command is executed. Therefore, once the Command Control MSG was been sent and after a short delay, you can request the command execution status using a *Get Command Error List* MSG. See the next section for details.

Command Error List Blocks (9950, 9951)

The *Get Command Error List* special function is applicable only when the module's port is configured as a Modbus Master. Use block identification codes 9950 for Port 1 or 9951for Port 2 anytime you want to request the *Command Error List* for the 100 user-configurable commands that may be sent by that port.

There is a one-to-one correspondence between the position of a command in the *Command List* and its corresponding execution status in the *Command Error List*. Each word in the *Command Error List* holds the most recent execution status of each corresponding command in the **MODBUS PORT 1 COMMANDS** or **MODBUS PORT 2 COMMANDS** lists, which were configured in the PCB module configuration file and downloaded to the module. As each command is executed, whether as part of normal, repetitive polling or if triggered by *Command Control*, the status of the most recent execution will be placed in the appropriate word of the 100-word *Command Error List*.

The *Command Error List* is requested using a "Get Attribute Single" service type MSG instruction. The following illustration shows the MSG instruction configuration for a *Get Command Error List* MSG.
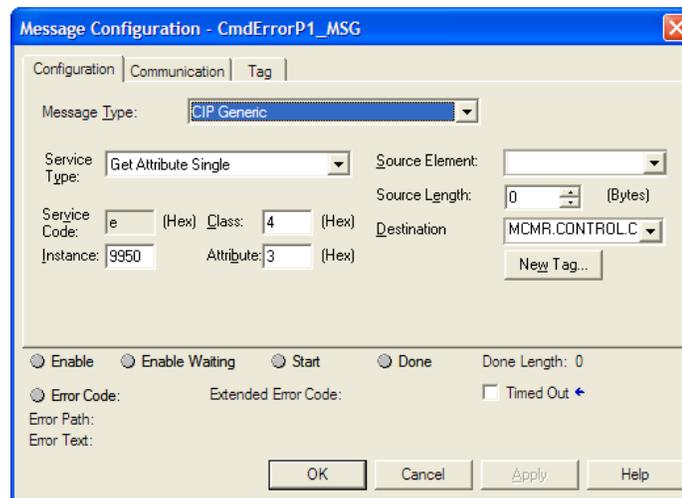
The format of the data returned by the *Get Command Error List* MSG is shown in the following table.

Block Response from Module to Processor

Word Offset	Data Field(s)	Description
0	Number of Commands to report	This field contains the number of commands to report in the response message. The value is always 100.
1	Start Index of First Command	This field always contains a 0. The status of all 100 possible commands will be returned, starting with Command Index 0, the first command in the *Command List*.
2 to 102	Command Error List	Each word of this area contains the last execution status value recorded for the command. The order of status words is the same as the order of commands in the *Command List*

A status value of zero (0) in the *Command Error List* indicates either that the corresponding command is not used or that it has been executed successfully. Any non-zero value found in this list indicates that some type of error was encountered while trying to execute the corresponding command.

There are many potential reasons why a command might fail. For more details, see Standard Modbus Protocol Errors or Module Communication Error Codes.

Slave Status Blocks (9960, 9961)

The *Get Slave Status List* special function is available only when the module's port is configured as a Modbus Master. Use block identification codes 9960 for Port 1 or 9961 for Port 2 to request the current polling state of each Slave device that could be polled by a Master port.

The results returned in this list have a one-to-one correspondence with the 248 possible Modbus Slave Device Address values, 0-247. Each word in the Slave Status List corresponds to a single Modbus Slave Device Address.

This list is updated each time the Master driver attempts to poll a specific Modbus Slave Device; but gives very little information regarding the success or failure of that poll attempt. Therefore, this list is not very useful for determining general communications health, for creating communication failure alarms, or for troubleshooting network problems. A better tool for those needs is the *Command Error List*.

The *Get Slave Status List* message uses the "Get Attribute Single" service type MSG instruction. The format of the data returned by the MSG is shown in the following table.

Block Response from Module to Processor

Word Offset	Data Field(s)	Description	
0	Number of Slaves to report	This field always receives a value of 248 (-8 as a signed SINT, 16#F8). A status word for all potential Modbus Slave Device Addresses will be included in the returned data.	
1	Start index of first Slave	This field is always 0. All 248 potential Modbus Slave Device Addresses will have a corresponding Status Word in the returned data.	
2 to 39	Slave Status List data	0	The Slave is inactive, not currently the slave being actively polled, waiting for its turn to be polled, or not a Modbus Slave Device Address used in the *Command List* for the Master port.
		1	The Slave at this specific address is the one currently being polled or controlled by the Master port. This does not indicate that the Slave has responded to a poll request; only that the Master is currently trying to poll this Slave.
		2	The may be thought of as a "Slow Poll" mode, whereby a Slave with communications errors will be polled at a lower-than-normal frequency. This status is set by the Master port whenever it has failed to communicate with the Slave device and the Port configuration parameter, **ERROR DELAY COUNTER**, has been set greater than 0.

The following illustration shows the MSG instruction configuration for a block of this type.
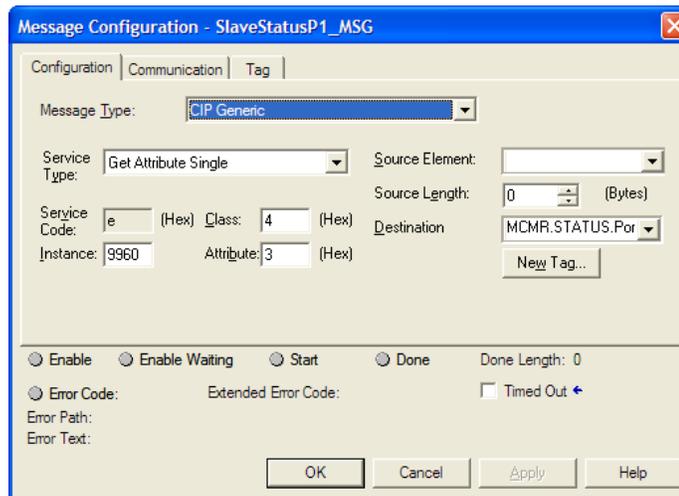
Warm Boot Block (9998)

The *Warm Boot* special function is used to restart the module application from within processor logic. This allows the module to be restarted without removing it from the chassis or removing power from the chassis. Restarting or "rebooting" the module will momentarily interrupt normal module operation, such as Modbus polling and backplane data transfers. It will also clear and reset all module diagnostic counters and user database memory.

This special function is one of the few that is sent in the Output Image Write Data block instead of a MSG instruction. Therefore, the effect of triggering a *Warm Boot* is almost immediate.

If you follow the sample ladder logic or AOI, the block identification code 9998 for the *Warm Boot* special function is embedded in the logic that formulates the Write Data block to be sent to the module in the Output Image. You can activate this special block logic by setting the controller tag, **MCMR.CONTROL.WARMBOOT**, to a value of 1. This will force an immediate module reboot.

The following table describes the format of the *Warm Boot* data block constructed by the processor.

Block Request from Processor to Module

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

Warm Boot and *Cold Boot* special functions are almost identical in what they do to reboot the module. The main difference between the two is that the *Warm Boot* restarts the internal firmware application without interrupting backplane power to the module. Therefore, a *Warm Boot* will complete and the module will return to normal operation a few seconds faster than when a *Cold Boot* is used. However, if the module is not operating correctly and a *Warm Boot* does not completely restore normal operation, a *Cold Boot* may be required and may be more effective at clearing errors and restarting the application.

Cold Boot Block (9999)

The *Cold Boot* special function is used to restart the module application from within processor logic. This allows the module to be restarted without removing it from the chassis or removing power from the chassis. Restarting or "rebooting" the module will momentarily interrupt normal module operation, such as Modbus polling and backplane data transfers. It will also clear and reset all module diagnostic counters and user database memory.

This special function is one of the few that is sent in the Output Image Write Data block instead of a MSG instruction. Therefore, the effect of triggering a *Cold Boot* is almost immediate.

If you follow the sample ladder logic or AOI, the block identification code 9999 for the *Cold Boot* special function is embedded in the logic that formulates the Write Data block to be sent to the module in the Output Image. You can activate this special block logic by setting the controller tag, **MCMR.CONTROL.COLDBOOT**, to a value of 1. This will force an immediate module reboot.

The following table describes the format of the *Cold Boot* data block constructed by the processor.

Block Request from Processor to Module

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

Warm Boot and *Cold Boot* special functions are almost identical in what they do to reboot the module. The main difference between the two is that the *Cold Boot* restarts the internal firmware application by interrupting backplane power to the module. Therefore, a *Cold Boot* will take a few extra seconds to complete before the module will return to normal operation than it would if a *Warm* Boot were used. However, if the module is not operating correctly, and a *Cold Boot* may be more effective at clearing errors and restarting the application than a *Warm Boot* might be.

6.3.4 Master Driver

In the Master mode, the MVI56E-MCMR module is responsible for issuing read or write commands to Slave devices on the Modbus network. These commands are user configured in the module via the Master Command List received from the ControlLogix processor or issued directly from the ControlLogix processor (event command control). Command status is returned to the processor for each individual command in the command list status block. The location of this status block in the module's internal database is user defined. The following flow chart and associated table describe the flow of data into and out of the module.
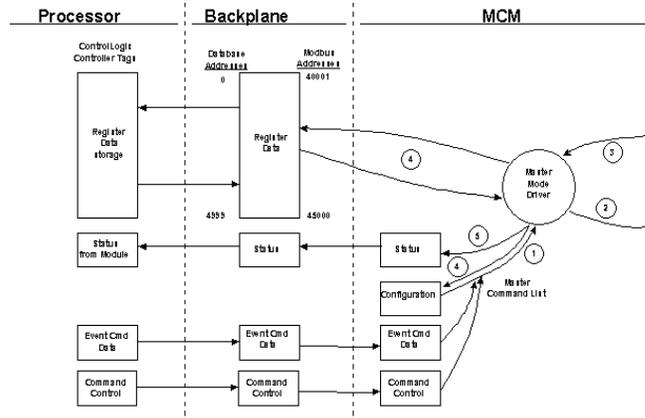
1   The Master driver obtains configuration data from the Compact Flash Disk. The configuration data obtained includes general module configuration data as well as the Master Command List. These values are used by the Master driver to determine the type of commands to be issued to Modbus Slave Devices on the Modbus network

2   After configuration, the Master driver begins transmitting read and/or write commands to the Modbus Slave Devices on the network. If writing data to a Modbus Slave Device, the data to send in the write command is obtained from the module's internal database.

3   Presuming successful processing by the Modbus Slave Device specified in the command, a response message is received into the Master driver for processing.

4   If the command was a command to read data, the data received from the Modbus Slave Device is passed into the module's internal database.

5   Status is returned to the ControlLogix processor for each command in the *Master Command List* (page 169).

Refer to Configuration as a Modbus Master (page 55) for a description of the parameters required to define the virtual Modbus Master port. Command Control Blocks describes the structure and content of each command.

Important: Take care when constructing each command in the list for predictable operation of the module. If two commands write to the same internal database address of the module, the results will not be as desired. All commands containing invalid data are ignored by the module.

Master Command List

In order to function in the Master Mode, you must define the module's Master Command List. This list contains up to 100 individual entries, with each entry containing the information required to construct a valid command. A valid command includes the following items:

- Command enable mode: (0) disabled, (1) continuous or (2) conditional
- Slave Node Address
- Command Type: Read or Write up to 125 words (16000 bits) per command
- Database Source and Destination Register Address: The addresses where data will be written or read.
- Count: The number of words to be transferred - 1 to 125 on FC 3, 4, or 16. Select the number of bits on FC 1, 2, 15.

As the list is read in from the processor and as the commands are processed, an error value is maintained in the module for each command. This error list can be transferred to the processor. The following tables describe the error codes generated by the module.

Note: 125 words is the maximum count allowed by the MODBUS protocol. Some field devices may support less than the full 125 words. Check with your device manufacturer for the maximum count supported by your particular slave.

6.3.5 Slave Driver

The Slave Driver Mode allows the MVI56E-MCMR module to respond to data read and write commands issued by a remote Master on the Modbus network. The following flow chart and associated table describe the flow of data into and out of the module.
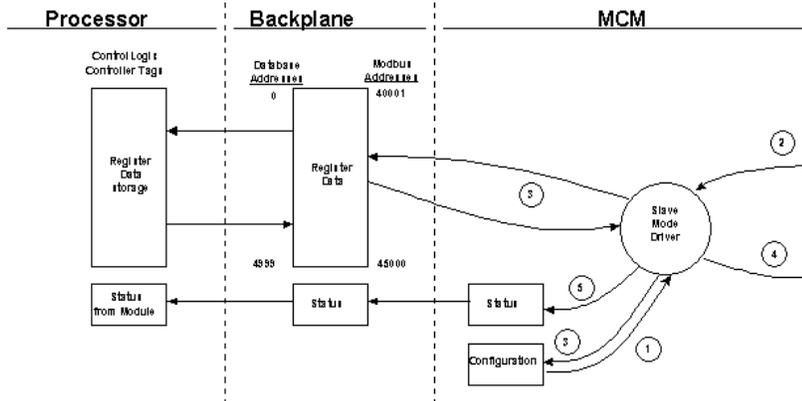
1 The Modbus Slave port driver receives the configuration information from the Compact Flash Disk. This information configures the backplane exchanges, user memory Read and Write Data areas, the serial ports, and Modbus Slave Device characteristics. Additionally, the configuration information contains parameters that can be used to offset data in the database to addresses different from those requested in messages received from Master units.

2 A Remote Master Device, such as a Modicon PLC or an HMI application, issues a read or write command to the module's Modbus Slave Device Address. The port driver qualifies the message before accepting it into the module.

3 After the module accepts the command, the data is immediately transferred to or from the internal database in the module. 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.

4 After the data processing has been completed in Step 3, the response is issued to the originating Master Device.

5 Counters are available in the *General Module Status* (page 153) Block that permit the ladder logic program to determine the level of activity of the Slave Driver.

Refer to Configuration as a Modbus Slave (page 81) for a list of the parameters that must be defined for a Slave port.

6.4   Cable Connections

The application ports on the MVI56E-MCMR module support RS-232, RS-422, and RS-485 interfaces. Please inspect the module to ensure that the jumpers are set correctly to correspond with the type of interface you are using.

Note: When using RS-232 with radio modem applications, some radios or modems require hardware handshaking (control and monitoring of modem signal lines). Enable this in the configuration of the module by setting the UseCTS parameter to 1.

6.4.1 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 172) for a diagram of how to configure Ethernet cable.

6.4.2 Ethernet Cable Configuration

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

Crossover cable			Straight- through cable	
RJ-45 PIN	**RJ-45 PIN**		**RJ-45 PIN**	**RJ-45 PIN**
1 Rx+	3 Tx+		1 Rx+	1 Tx+
2 Rx-	6 Tx-		2 Rx-	2 Tx-
3 Tx+	1 Rx+		3 Tx+	3 Rx+
6 Tx-	2 Rx-		6 Tx-	6 Rx-
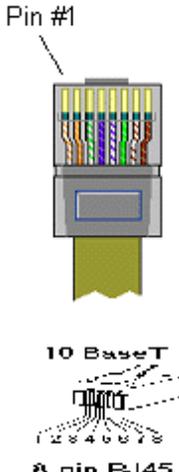
6.4.3 Ethernet Performance

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

- Accessing the web interface (refreshing the page, downloading files, and so on) may affect MCMR performance
- High Ethernet traffic may impact MCMR performance (consider CIPconnect (page 96) for these applications and disconnect the module Ethernet port from the network).

6.4.4 RS-232 Application Port(s)

When the RS-232 interface is selected, the use of hardware handshaking (control and monitoring of modem signal lines) is user definable. If no hardware handshaking will be used, here are the cable pinouts to connect to the port.
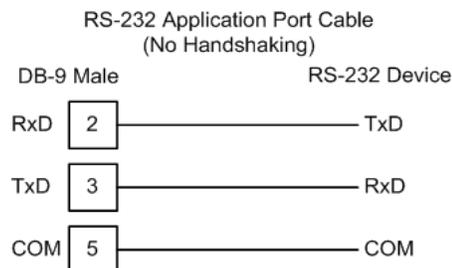
RS-232: Modem Connection (Hardware Handshaking Required)

This type of connection is required between the module and a modem or other communication device.

RS-232 Application Port Cable
(Modem Connection)
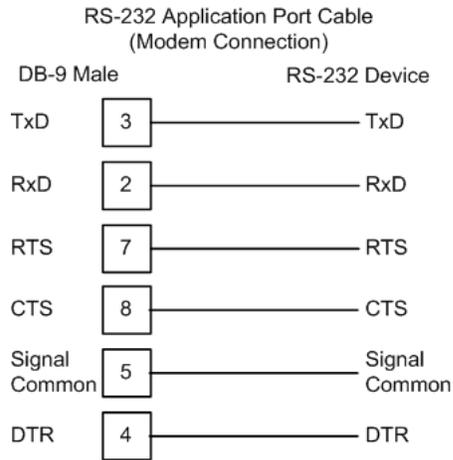
The "Use CTS Line" parameter for the port configuration should be set to 'Y' for most modem applications.

RS-232: Null Modem Connection (Hardware Handshaking)

This type of connection is used when the device connected to the module requires hardware handshaking (control and monitoring of modem signal lines).

RS-232 Application Port Cable
(Hardware Handshaking)
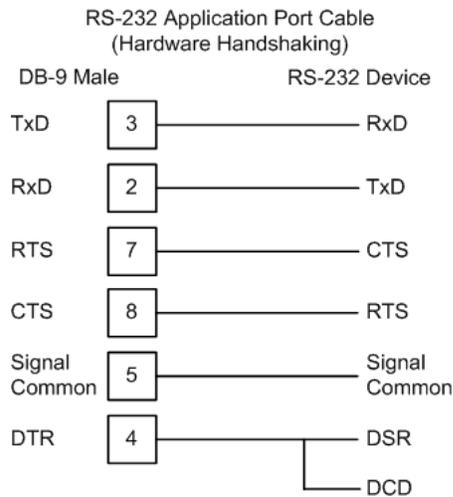
RS-232: Null Modem Connection (No Hardware Handshaking)

This type of connection can be used to connect the module to a computer or field device communication port.

RS-232 Application Port Cable
(No Handshaking)

DB-9 Male   RS-232 Device

RxD   2 ─────────────── TxD

TxD   3 ─────────────── RxD

COM   5 ─────────────── COM
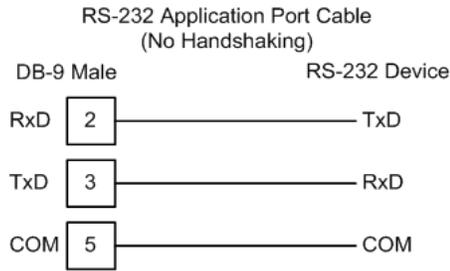
Note: For most null modem connections where hardware handshaking is not required, the *Use CTS Line* parameter should be set to **N** and no jumper will be required between Pins 7 (RTS) and 8 (CTS) on the connector. If the port is configured with the *Use CTS Line* set to **Y**, then a jumper is required between the RTS and the CTS lines on the port connection.

RS-232 Application Port Cable
(No Handshaking)

DB-9 Male   RS-232 Device

TxD   3 ─────────────── RxD

RxD   2 ─────────────── TxD

RTS   7 ┐
         │ RTS-CTS jumper must
         │ be installed if CTS line
         │ monitoring enabled.
CTS   8 ┘

Signal   Signal
Common 5 ─────────────── Common

DTR   4
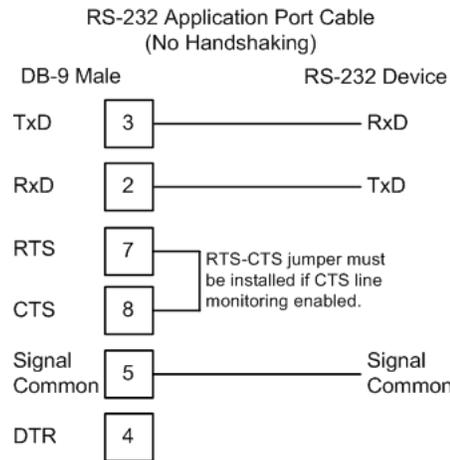
6.4.5 RS-422

The RS-422 interface requires a single four or five wire cable. The Common connection is optional, depending on the RS-422 network devices used. The cable required for this interface is shown below:
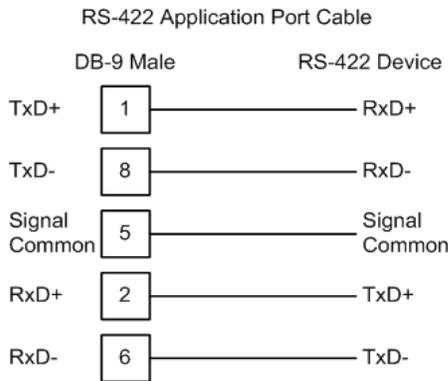
6.4.6 RS-485 Application Port(s)

The RS-485 interface requires a single two or three wire cable. The Common connection is optional, depending on the RS-485 network devices used. The cable required for this interface is shown below:
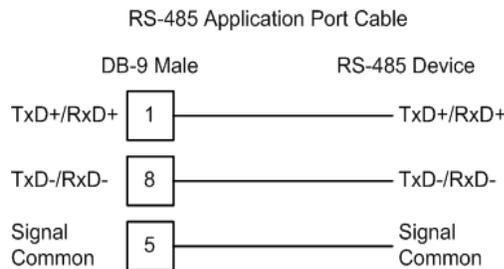
Note: Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In these cases, installing a 120-ohm terminating resistor between pins 1 and 8 on the module connector end of the RS-485 line may improve communication quality.

RS-485 and RS-422 Tip

If communication in the RS-422 or RS-485 mode does not work at first, despite all attempts, try switching termination polarities. Some manufacturers interpret + and -, or A and B, polarities differently.

6.4.7 DB9 to RJ45 Adaptor (Cable 14)
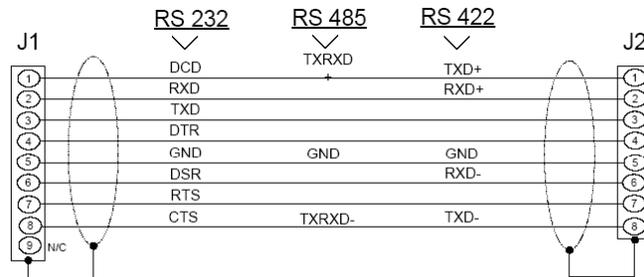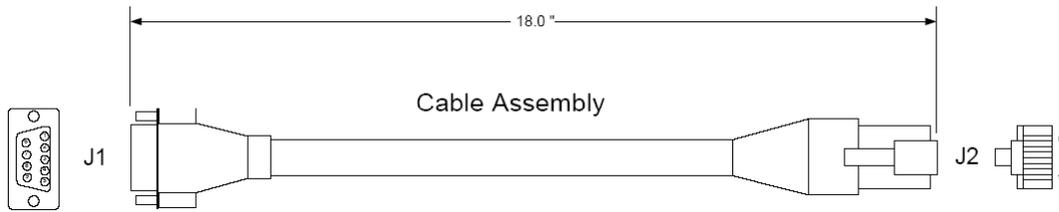
Wiring Diagram

6.5 MVI56E-MCMR Status Data Definition

This section contains a description of the members present in the MCMR.STATUS object. This data is transferred from the module to the processor in a message block.

Offset	Content	Description
0	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
1 to 2	Product Code	These two registers contain the product code of "MCM".
3 to 4	Product Version	These two registers contain the product version for the current running software.
5 to 6	Operating System	These two registers contain the month and year values for the program operating system.
7 to 8	Run Number	These two registers contain the run number value for the currently running software.
9	Port 1 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
10	Port 1 Command List Response	This field contains the number of Slave response messages received on the port.
11	Port 1 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
12	Port 1 Requests	This field contains the total number of messages sent out of the port.
13	Port 1 Responses	This field contains the total number of messages received on the port.
14	Port 1 Errors Sent	This field contains the total number of message errors sent out of the port.
15	Port 1 Errors Received	This field contains the total number of message errors received on the port.
16	Port 2 Command List Requests	This field contains the number of requests made from this port to Slave devices on the network.
17	Port 2 Command List Response	This field contains the number of Slave response messages received on the port.
18	Port 2 Command List Errors	This field contains the number of command errors processed on the port. These errors could be due to a bad response or command.
19	Port 2 Requests	This field contains the total number of messages sent out the port.
20	Port 2 Responses	This field contains the total number of messages received on the port.
21	Port 2 Errors Sent	This field contains the total number of message errors sent out the port.
22	Port 2 Errors Received	This field contains the total number of message errors received on the port.
23	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
24	Write Block Count	This field contains the total number of write blocks transferred from the module to the processor.
25	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
26	Command Event Block Count	This field contains the total number of command event blocks received from the processor.

Offset	Content	Description
27	Command Block Count	This field contains the total number of command blocks received from the processor.
28	Error Block Count	This field contains the total number of block errors recognized by the module.
29	Port 1 Current Error	For a Slave port, this field contains the value of the current error code returned. For a Master port, this field contains the index of the currently executing command.
30	Port 1 Last Error	For a Slave port, this field contains the value of the last error code returned. For a Master port, this field contains the index of the command with the error.
31	Port 2 Current Error	For a Slave port, this field contains the value of the current error code returned. For a Master port, this field contains the index of the currently executing command.
32	Port 2 Last Error	For a Slave port, this field contains the value of the last error code returned. For a Master port, this field contains the index of the command with an error.

6.6 Modbus Protocol Specification

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

6.6.1 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 Master	Supported in Slave
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
8	Diagnostics		X
15	Multiple Coil Write	X	X
16	Multiple Register Write	X	X
17	Report Slave ID		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 slave device.

6.6.2 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 Slave only. Broadcast mode is not supported with this function code. In addition to the Slave 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 Slave 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 Slave 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 Slave 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 Slave interface device is serviced at the end of a controller's scan, data will reflect coil status at the end of the scan. Some Slaves 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.

Adr	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.

6.6.3 Read Input Status (Function Code 02)

Query

This function allows the user to obtain the ON/OFF status of discrete inputs in the addressed Slave PC Broadcast mode is not supported with this function code. In addition to the Slave 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 Slave 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 Slave number 11.

Adr	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 Slave 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 Slave interface device is serviced at the end of a controller's scan, data will reflect input status at the end of the scan. Some Slaves 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.

Adr	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.

6.6.4 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 Slave. 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 Slave 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 Slave 584 number 11.

Adr	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 Slave 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 Slave 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 Slaves 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.

Adr	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

6.6.5 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 Slave 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 Slave number 11.

Adr	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 Slave 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 Slave 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.

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

6.6.6 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 Slave address 00 (Broadcast Mode) will force all attached Slaves 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 Slave number 11 to turn ON coil 0173.

Adr	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.

Adr	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".

6.6.7 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 Slave address zero (Broadcast mode) all Slave 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.

Adr	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.

Adr	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

6.6.8 Diagnostics (Function Code 08)

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

The function uses a two-byte sub-function code field in the query to define the type of test to be performed. The slave echoes both the function code and sub-function code in a normal response. Some of the diagnostics commands 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-MCMR 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.
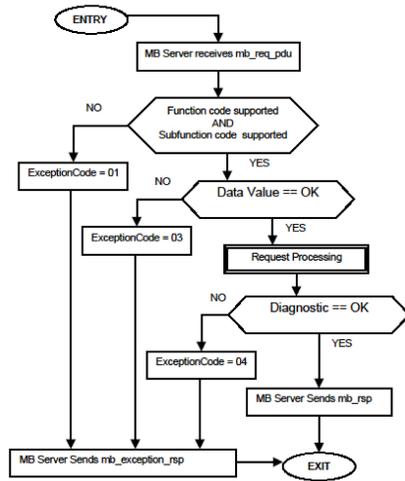
6.6.9 Force Multiple Coils (Function Code 15)

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 Slave address 0 (Broadcast Mode) will force all attached Slaves 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.

Adr	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 Slave address, function code, starting address, and quantity of coils forced.

Adr	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.

6.6.10 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.

Adr	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.

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

6.6.11 Modbus Exception Responses

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

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

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

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

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

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

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

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

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

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

Modbus Exception Codes

Code	Name	Meaning
01	Illegal Function	The function code received in the query is not an allowable action for the Slave. This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the Slave is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
02	Illegal Data Address	The data address received in the query is not an allowable address for the Slave. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed; a request with offset 96 and length 5 will generate exception 02.
03	Illegal Data Value	A value contained in the query data field is not an allowable value for Slave. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expectation of the application program, 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 Slave was attempting to perform the requested action.
05	Acknowledge	Specialized use in conjunction with programming commands. The Slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the Master. The Master can next issue a poll program complete message to determine if processing is completed.
06	Slave Device Busy	Specialized use in conjunction with programming commands. The Slave is engaged in processing a long-duration program command. The Master should retransmit the message later when the Slave 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 Slave attempted to read record file, but detected a parity error in the memory. The Master can retry the request, but service may be required on the Slave 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.

6.7   Using the Optional Add-On Instruction Rung Import

6.7.1 Before You Begin

- Make sure that you have installed RSLogix 5000 version 16 (or later).
- Download the Optional Add-On file
  MVI56(E)MCMR_Optional_AddOn_Rung_v1_1.L5X from
  www.prosoft-technology.com or copy it from the *ProSoft Solutions DVD*.
- Save a copy in a folder in your PC.

6.7.2 Overview

The Optional Add-On Instruction Rung Import contains optional logic for MVI56E-MCMR 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 MCMR data.

6.7.3 Installing the Rung Import with Optional Add-On Instruction

1   Right-click on an empty rung in the *MainRoutine* of your existing ladder logic
    and choose **IMPORT RUNG**.
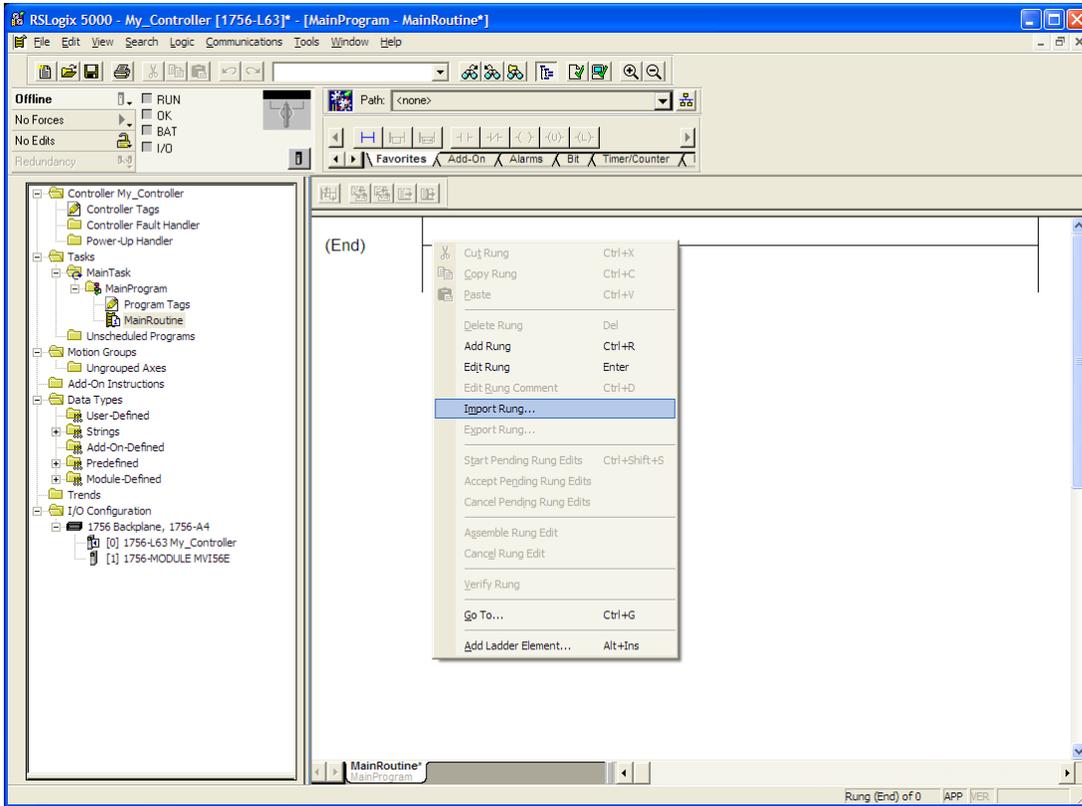
2 Navigate to the folder where you saved
MVI56(E)MCMR_Optional_AddOn_Rung_v1_1.L5X and select the file. Click
the **IMPORT** button.
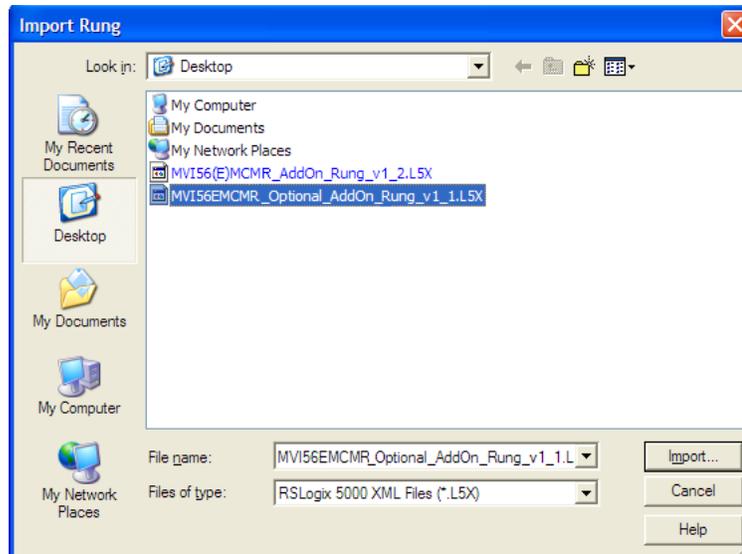
3 In the **IMPORT CONFIGURATION** window, click **OK.**
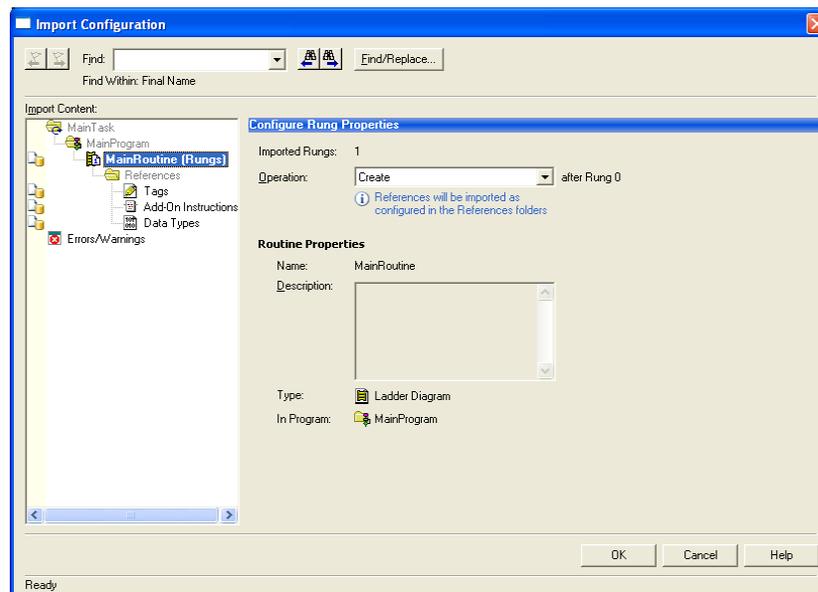
The Optional Add-On instruction will be now visible in the ladder logic. Observe that the procedure has also imported data types and controller tags associated to the Optional Add-On instruction.
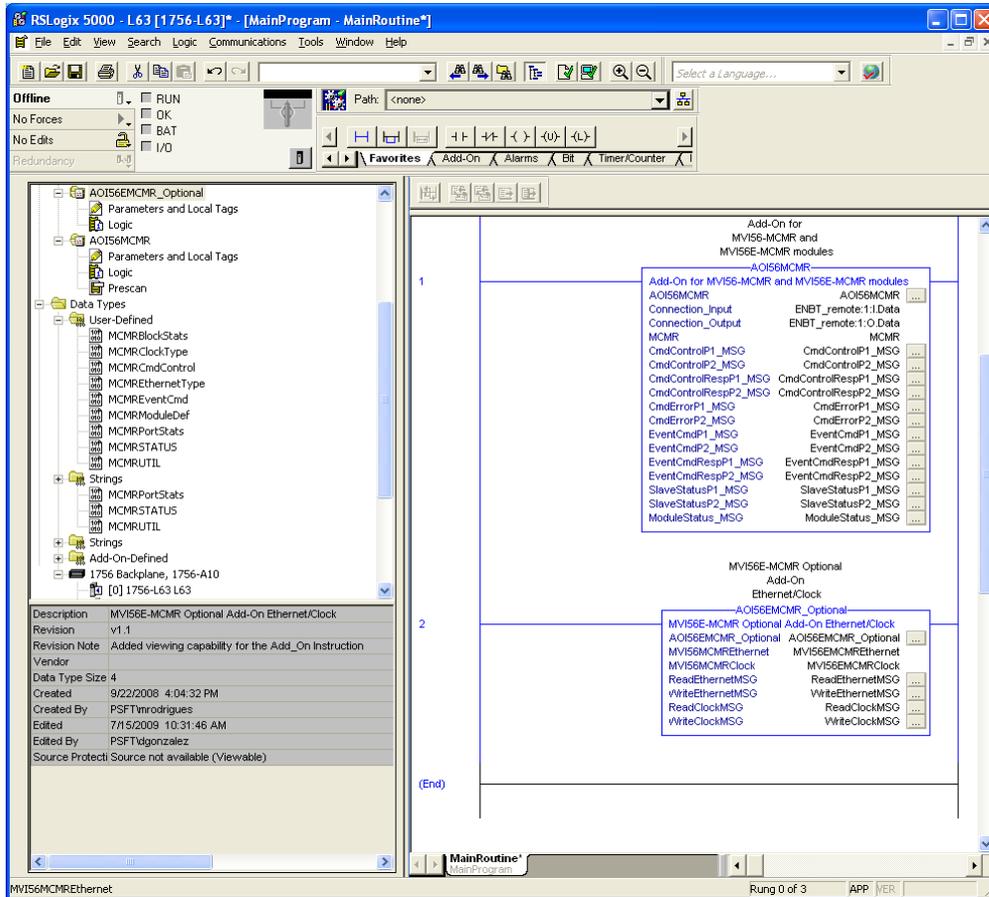
You will notice that new tags have been imported: four **MESSAGE** tags,
MVI56EMCMRCLOCK and **MVI56EMCMRETHERNET** tags.
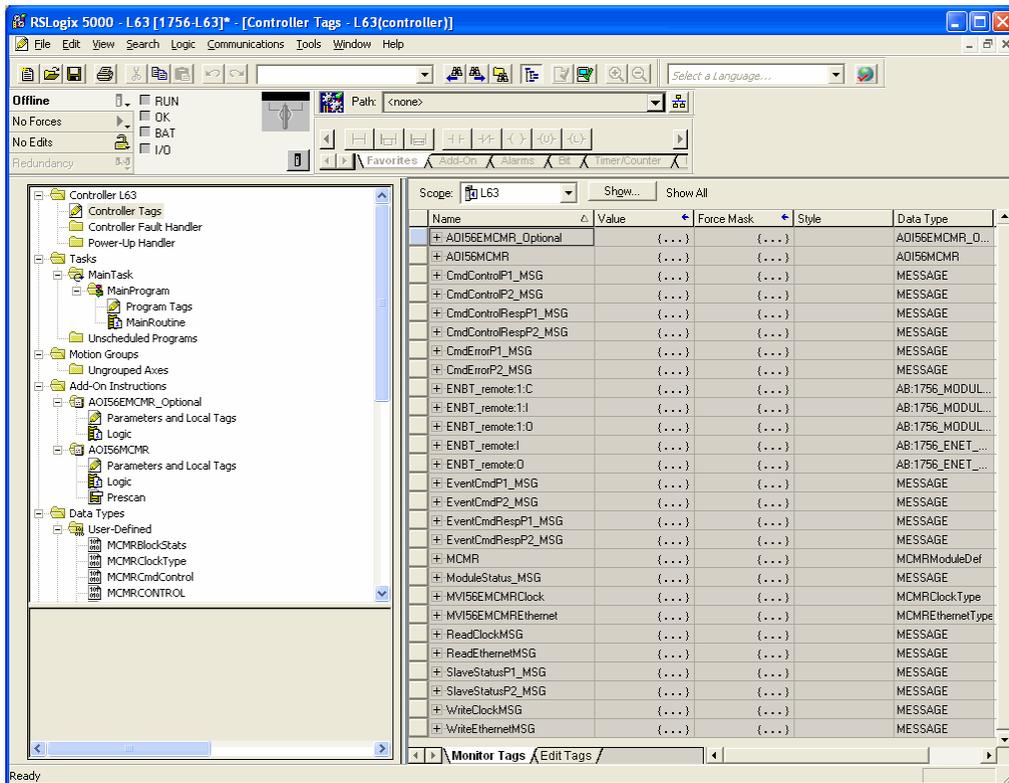
4   In the Optional Add-On instruction, click the **[...]** button next to each **MSG** tag
to open the **MESSAGE CONFIGURATION TAG**.

5   Click the **COMMUNICATION** tab and click the **BROWSE** button as follows.
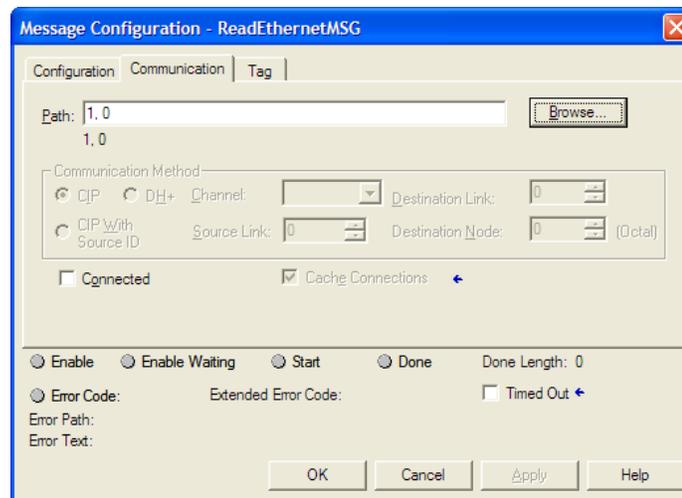
6 Select the module to configure the message path.
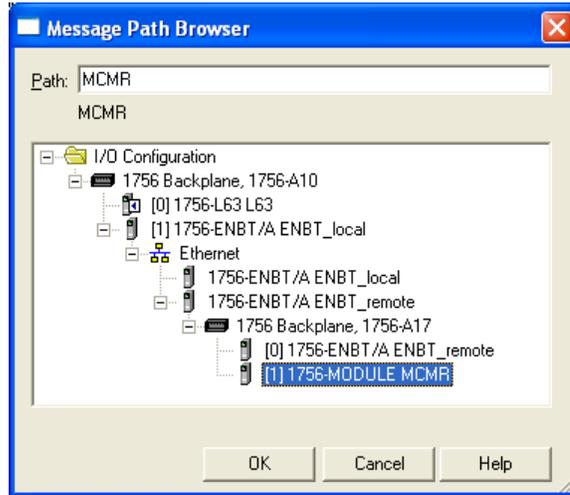
6.7.4 Reading the Ethernet Settings from the Module

Expand the **MVI56MCMRE**THERNET controller tag and move a value of 1 to
MVI56MCMRETHERNET.R**EAD.**

The bit will be automatically reset and the current Ethernet settings will be copied to **MVI56MCMRETHERNET** controller tag as follows.

⊟ MVI56EMCMREthernet	{...}
— MVI56EMCMREthernet.Read	0
— MVI56EMCMREthernet.Write	0
⊟ MVI56EMCMREthernet.Config	{...}
⊟ MVI56EMCMREthernet.Config.IP	{...}
⊞ MVI56EMCMREthernet.Config.IP[0]	105
⊞ MVI56EMCMREthernet.Config.IP[1]	102
⊞ MVI56EMCMREthernet.Config.IP[2]	0
⊞ MVI56EMCMREthernet.Config.IP[3]	32
⊟ MVI56EMCMREthernet.Config.Netmask	{...}
⊞ MVI56EMCMREthernet.Config.Netmask[0]	255
⊞ MVI56EMCMREthernet.Config.Netmask[1]	255
⊞ MVI56EMCMREthernet.Config.Netmask[2]	255
⊞ MVI56EMCMREthernet.Config.Netmask[3]	0
⊟ MVI56EMCMREthernet.Config.Gateway	{...}
⊞ MVI56EMCMREthernet.Config.Gateway[0]	192
⊞ MVI56EMCMREthernet.Config.Gateway[1]	168
⊞ MVI56EMCMREthernet.Config.Gateway[2]	0
⊞ MVI56EMCMREthernet.Config.Gateway[3]	1

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

6.7.5 Writing the Ethernet Settings to the Module

Expand the **MVI56EMCMRETHERNET** controller tag.

Set the new Ethernet configuration in **MVI56EMCMRETHERNET.CONFIG**

Move a value of 1 to **MVI56MCMRETHERNET.WRITE**

⊟·MVI56EMCMREthernet	{...}
─MVI56EMCMREthernet.Read	0
─MVI56EMCMREthernet.Write	1
⊟·MVI56EMCMREthernet.Config	{...}
⊟·MVI56EMCMREthernet.Config.IP	{...}
⊞·MVI56EMCMREthernet.Config.IP[0]	105
⊞·MVI56EMCMREthernet.Config.IP[1]	102
⊞·MVI56EMCMREthernet.Config.IP[2]	0
⊞·MVI56EMCMREthernet.Config.IP[3]	132
⊟·MVI56EMCMREthernet.Config.Netmask	{...}
⊞·MVI56EMCMREthernet.Config.Netmask[0]	255
⊞·MVI56EMCMREthernet.Config.Netmask[1]	255
⊞·MVI56EMCMREthernet.Config.Netmask[2]	255
⊞·MVI56EMCMREthernet.Config.Netmask[3]	0
⊟·MVI56EMCMREthernet.Config.Gateway	{...}
⊞·MVI56EMCMREthernet.Config.Gateway[0]	192
⊞·MVI56EMCMREthernet.Config.Gateway[1]	168
⊞·MVI56EMCMREthernet.Config.Gateway[2]	0
⊞·MVI56EMCMREthernet.Config.Gateway[3]	1

After the message is executed, the **MVI56MCMRETHERNET.WRITE** bit resets to 0.

⊟·MVI56EMCMREthernet	{...}
─MVI56EMCMREthernet.Read	0
─MVI56EMCMREthernet.Write	0
⊟·MVI56EMCMREthernet.Config	{...}
⊟·MVI56EMCMREthernet.Config.IP	{...}
⊞·MVI56EMCMREthernet.Config.IP[0]	105
⊞·MVI56EMCMREthernet.Config.IP[1]	102
⊞·MVI56EMCMREthernet.Config.IP[2]	0
⊞·MVI56EMCMREthernet.Config.IP[3]	132
⊟·MVI56EMCMREthernet.Config.Netmask	{...}
⊞·MVI56EMCMREthernet.Config.Netmask[0]	255
⊞·MVI56EMCMREthernet.Config.Netmask[1]	255
⊞·MVI56EMCMREthernet.Config.Netmask[2]	255
⊞·MVI56EMCMREthernet.Config.Netmask[3]	0
⊟·MVI56EMCMREthernet.Config.Gateway	{...}
⊞·MVI56EMCMREthernet.Config.Gateway[0]	192
⊞·MVI56EMCMREthernet.Config.Gateway[1]	168
⊞·MVI56EMCMREthernet.Config.Gateway[2]	0
⊞·MVI56EMCMREthernet.Config.Gateway[3]	1

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

6.7.6 Reading the Clock Value from the Module

Expand the **MVI56MCMRCLOCK** controller tag and move a value of 1 to **MVI56MCMRCLOCK.READ**

⊟ MVI56EMCMRClock	{...}
── MVI56EMCMRClock.Read	1
── MVI56EMCMRClock.Write	0
⊟ MVI56EMCMRClock.Config	{...}
⊞ MVI56EMCMRClock.Config.Year	0
⊞ MVI56EMCMRClock.Config.Month	0
⊞ MVI56EMCMRClock.Config.Day	0
⊞ MVI56EMCMRClock.Config.Hour	0
⊞ MVI56EMCMRClock.Config.Minute	0
⊞ MVI56EMCMRClock.Config.Seconds	0

The bit will be automatically reset and the current clock value will be copied to **MVI56MCMRCLOCK.CONFIG** controller tag as follows.

⊟ MVI56EMCMRClock	{...}
── MVI56EMCMRClock.Read	0
── MVI56EMCMRClock.Write	0
⊟ MVI56EMCMRClock.Config	{...}
⊞ MVI56EMCMRClock.Config.Year	2009
⊞ MVI56EMCMRClock.Config.Month	5
⊞ MVI56EMCMRClock.Config.Day	4
⊞ MVI56EMCMRClock.Config.Hour	15
⊞ MVI56EMCMRClock.Config.Minute	38
⊞ MVI56EMCMRClock.Config.Seconds	9

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

6.7.7 Writing the Clock Value to the Module

Expand the **MVI56MCMRCLOCK** controller tag.
Set the new Clock value in **MVI56MCMRCLOCK.CONFIG**

Move a value of 1 to **MVI56MCMRCLOCK.WRITE**

MVI56EMCMRClock	{...}
—MVI56EMCMRClock.Read	0
—MVI56EMCMRClock.Write	1
—MVI56EMCMRClock.Config	{...}
⊞ MVI56EMCMRClock.Config.Year	2009
⊞ MVI56EMCMRClock.Config.Month	5
⊞ MVI56EMCMRClock.Config.Day	4
⊞ MVI56EMCMRClock.Config.Hour	15
⊞ MVI56EMCMRClock.Config.Minute	38
⊞ MVI56EMCMRClock.Config.Seconds	9

The bit will be automatically reset to 0.

MVI56EMCMRClock	{...}
—MVI56EMCMRClock.Read	0
—MVI56EMCMRClock.Write	0
—MVI56EMCMRClock.Config	{...}
⊞ MVI56EMCMRClock.Config.Year	2009
⊞ MVI56EMCMRClock.Config.Month	5
⊞ MVI56EMCMRClock.Config.Day	4
⊞ MVI56EMCMRClock.Config.Hour	15
⊞ MVI56EMCMRClock.Config.Minute	38
⊞ MVI56EMCMRClock.Config.Seconds	9

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

6.8 Using the Sample Program - RSLogix 5000 Version 15 and earlier

he sample program included with your MVI56E-MCMR module contains predefined controller tags, configuration information, data types, and ladder logic that allow the module to communicate between the ControlLogix processor and a network of MCMR devices. For most applications, the sample program will work without modification.

6.8.1 Adding the Sample Ladder to an Existing Application

1   Copy the Controller Tags (page 118) from the sample program.
2   Copy the User-Defined Data Types (page 118) from the sample program.
3   Copy the Ladder Rungs from the sample program.
4   Save and Download (page 43) the new application to the controller and place the processor in run mode.

If all the configuration parameters are set correctly and the module is attached to the Modbus network, the module's Application LED (APP LED) should remain off and the backplane activity LED (BP ACT) should blink rapidly. If you encounter errors, refer to Diagnostics and Troubleshooting (page 122, page 130).

6.8.2 Add the Module to the Project

Important: The following steps describe how to install and configure the MVI56E-MCMR module with RSLogix 5000 version 15 or older. If you are using RSLogix 5000 version 16, please refer to Sample Add-On Instruction Import Procedure.
Note: The RSLogix software must be in "offline" mode to add the module to a project.

1   In the **CONTROLLER ORGANIZATION** window, select **I/O CONFIGURATION**, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE.**
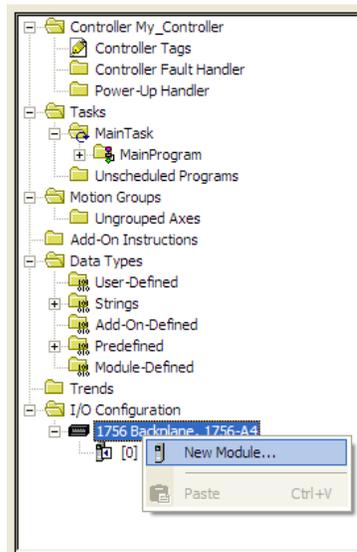
This action opens the **SELECT MODULE** dialog box.
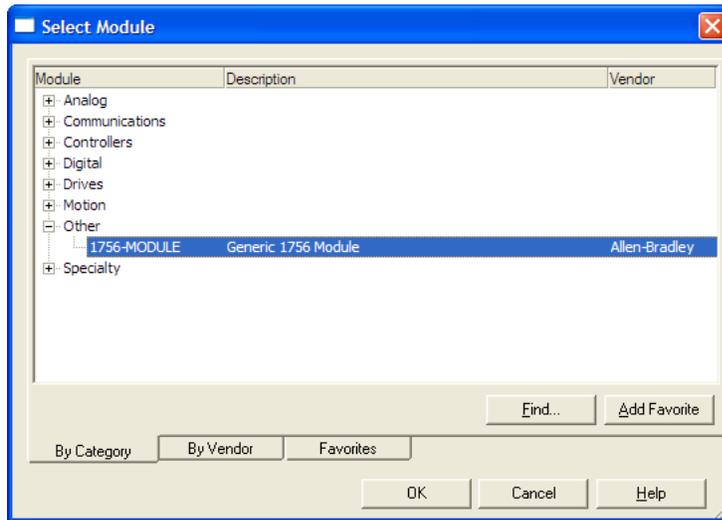
2   Select **1756-MODULE (GENERIC 1756 MODULE)** from the list, and then click
     OK. This action opens the **NEW MODULE** dialog box.
3   In the **NEW MODULE** dialog box, you must select **DATA - INT** as the Comm
     Format. Configure the Assembly Instance and Size parameters as shown in
     the following illustration.
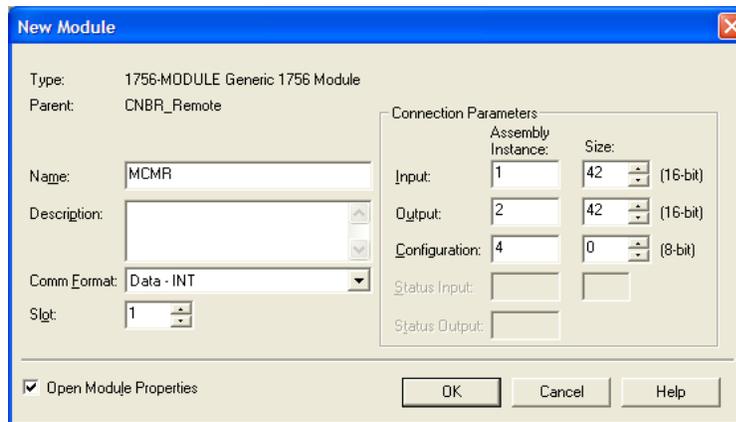
4 Click **OK** to save your module settings. This action opens the **MODULE PROPERTIES** dialog box.
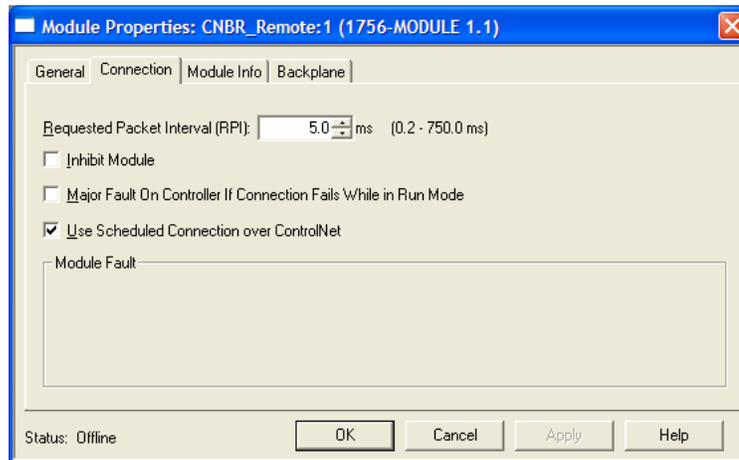
5 In the **CONNECTION** tab, the Requested Packet Interval (RPI) value represents the time interval at which the module will attempt backplane communication with the processor. This value should not be set to less than 1 millisecond. Values between 1 and 10 milliseconds should work with most applications. If the module is installed in a Remote chassis and accessed via ControlNet, the RPI must not be set to a value lower than the ControlNet Network Update Time (NUT). Doing so will cause scheduling errors in RSNetworx for ControlNet.

6 Click **OK** to save your settings. Notice that the module now appears in the **CONTROLLER ORGANIZATION** window.
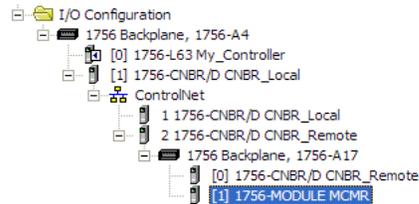
6.8.3 Copying the User Defined Data Types

Next, copy the User Defined Data Types from the sample program to your existing program. These data types contain configuration information, status, commands and other functions used by the program.

1 Arrange the two RSLogix 5000 windows on your desktop so that they are side-by-side.
2 In the **CONTROLLER ORGANIZATION** pane in the Sample Program, expand the **DATA TYPES** folder until the list of User-Defined data types is visible.
3 In the Sample Program window, select one data type at a time, and then drag the data type to the User-Defined data types folder in your existing program.
4 Repeat these steps until you have copied all of the data types from the sample program into your existing application.

> **Note:** Data types prefixed with an underscore **[_]** are used in optional routines, and need not be copied unless your application requires them. Refer to MVI56E-MCMR User Defined Data Types for a description of the usage for each data type.

6.8.4 Copy Sample Controller Tags

The sample program contains the following controller tag arrays:

- **MCMR.DATA.READDATA** and **MCMR.DATA.WRITEDATA** tags hold all data related to the database.
- **MCMR.STATUS** holds all status data related to the module (type MCMRModuleDef).
- **MCMR.CONTROL** holds all the tags needed to support the Special Function Blocks
- **MCMR.UTIL** holds module logic control 'scratchpad' tags.

The sample ladder logic also includes controller tags for the MSG instructions to pass data between the module and the ControlLogix processor.

Name	△	Value	←	Force Mask	←	Style	Data Type	Description
⊞ CmdControlP1_MSG		{...}		{...}			MESSAGE	
⊞ CmdControlP2_MSG		{...}		{...}			MESSAGE	
⊞ CmdControlRespP1_MSG		{...}		{...}			MESSAGE	
⊞ CmdControlRespP2_MSG		{...}		{...}			MESSAGE	
⊞ CmdErrorP1_MSG		{...}		{...}			MESSAGE	
⊞ CmdErrorP2_MSG		{...}		{...}			MESSAGE	
⊞ ENBT_slot6:2:C		{...}		{...}			AB:1756_MODUL...	
⊞ ENBT_slot6:2:I		{...}		{...}			AB:1756_MODUL...	
⊞ ENBT_slot6:2:O		{...}		{...}			AB:1756_MODUL...	
⊞ ENBT_slot6:I		{...}		{...}			AB:1756_ENET_...	
⊞ ENBT_slot6:O		{...}		{...}			AB:1756_ENET_...	
⊞ EventCmdP1_MSG		{...}		{...}			MESSAGE	
⊞ EventCmdP2_MSG		{...}		{...}			MESSAGE	
⊞ EventCmdRespP1_MSG		{...}		{...}			MESSAGE	
⊞ EventCmdRespP2_MSG		{...}		{...}			MESSAGE	
⊞ MCMR		{...}		{...}			MCMRModuleDef	
⊞ MJFAULTS		{...}		{...}	Decimal	DINT[12]	
⊞ ModuleStatus_MSG		{...}		{...}			MESSAGE	
⊞ SlaveStatusP1_MSG		{...}		{...}			MESSAGE	
⊞ SlaveStatusP2_MSG		{...}		{...}			MESSAGE	

6.8.5 Add the Ladder Logic

If you are creating your own ladder logic, copy the rungs shown in the following illustration from the sample program to your application.
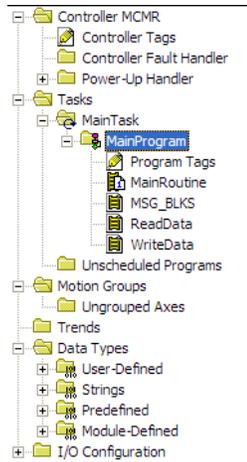
6.8.6 Ladder Logic - RSLogix Version 15 and Lower

MSG_BLKS

The MSG_BLKS routine passes data between the module and the ControlLogix processor using MSG instructions. Data transferred using these blocks is of low-priority and completely under control of the ladder logic.

The first rung requests the module's error/status data:
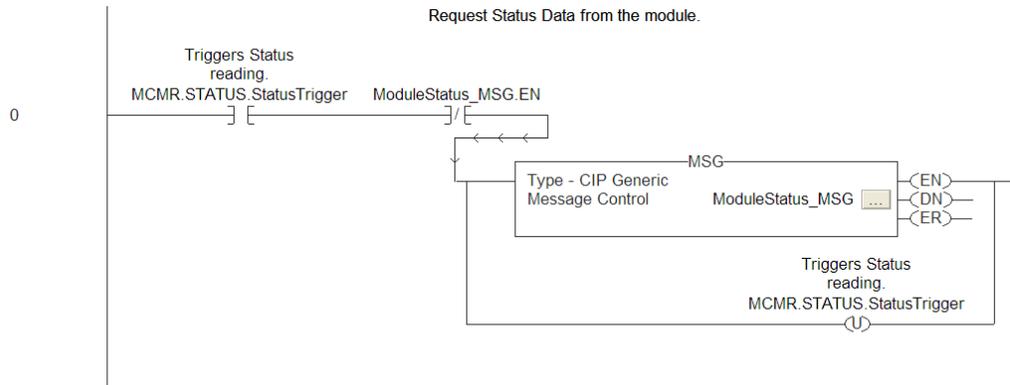
The following illustration shows the format of the MSG block.
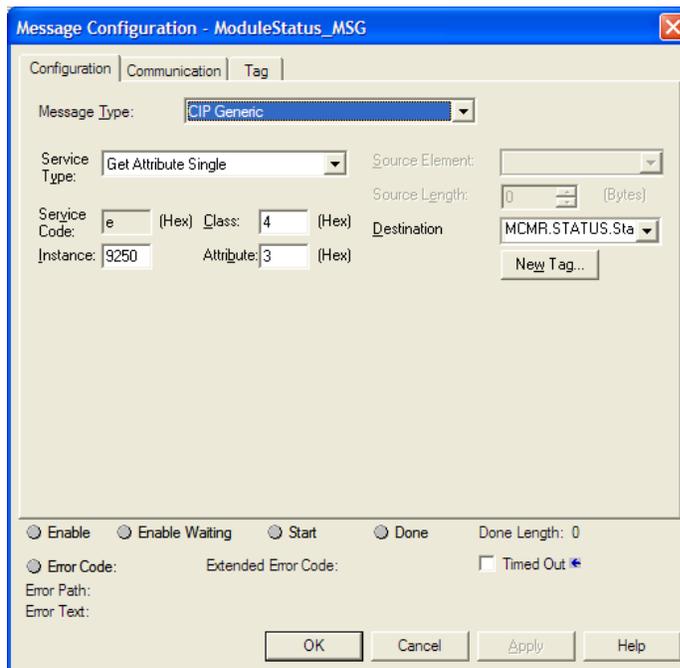
After the block is read from the module, the data received should be copied to the module's status controller tag area. The following illustration shows the ladder logic to accomplish this task.

Copy Status Data received from the module to the status tag structure.

```
                                                    Program cycle
                                                       counter
   ModuleStatus_MSG.DN                                 ─COP─
1  ──┐┌──                                     Copy File
     ┘└                                       Source   MCMR.STATUS.StatusMsgData[0]
                                              Dest             MCMR.STATUS.PassCnt
                                              Length                             1

                                                    Product Name
                                                     ─COP─
                                              Copy File
                                              Source   MCMR.STATUS.StatusMsgData[1]
                                              Dest             MCMR.STATUS.Product[0]
                                              Length                             8

                                                  Total number of
                                                  command list
                                                  requests sent
                                                     ─COP─
                                              Copy File
                                              Source   MCMR.STATUS.StatusMsgData[9]
                                              Dest    MCMR.STATUS.Port1Stats.CmdReq
                                              Length                             7

                                                  Total number of
                                                  command list
                                                  requests sent
                                                     ─COP─
                                              Copy File
                                              Source   MCMR.STATUS.StatusMsgData[16]
                                              Dest     MCMR.STATUS.Port2Stats.CmdReq
                                              Length                             7

                                                  Total number of
                                                  read block transfers
                                                     ─COP─
                                              Copy File
                                              Source   MCMR.STATUS.StatusMsgData[23]
                                              Dest             MCMR.STATUS.Block.Read
                                              Length                             6

                                                  Current error/index
                                                    for Port 1
                                                     ─COP─
                                              Copy File
                                              Source   MCMR.STATUS.StatusMsgData[29]
                                              Dest          MCMR.STATUS.Port1CurrentErr
                                              Length                             4

                                                  ModuleStatus_MSG.DN
                                                        ─(U)─
```

The next rung passes a block 9901 from the processor to the module:

Send Event Command data for port 1 to module.

Send Event Command data for port 2 to module.
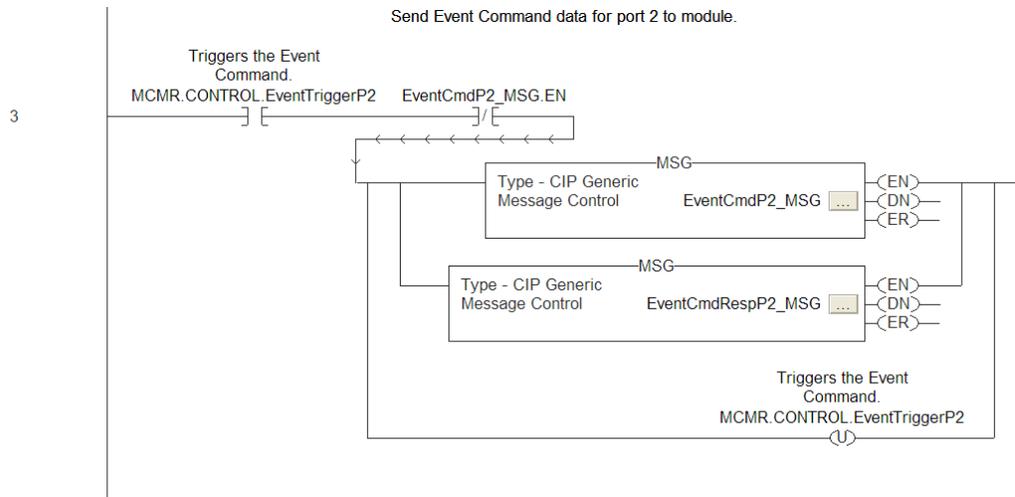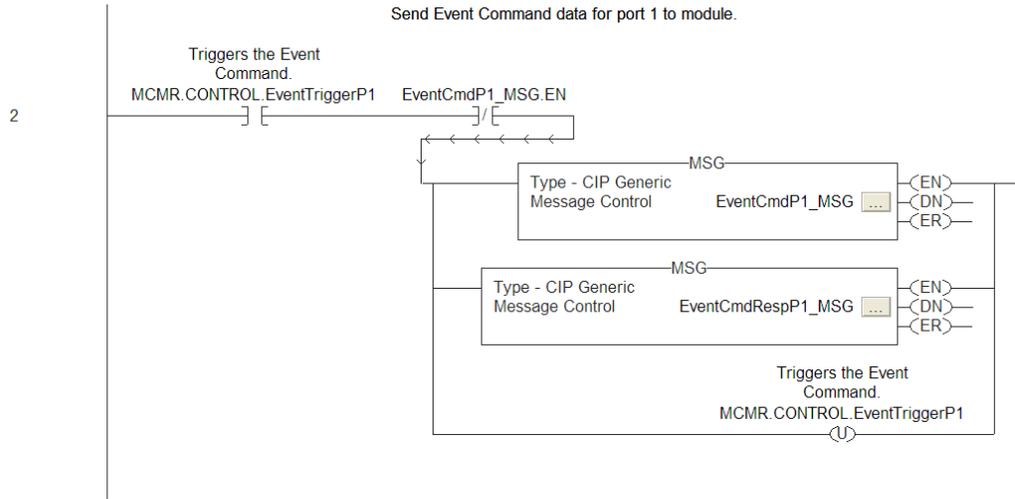
The following illustrations show the format of the MSG block.
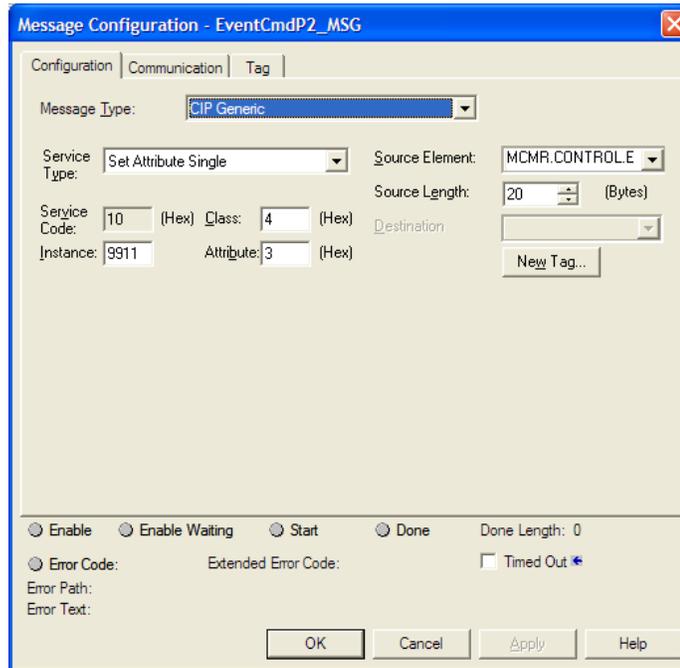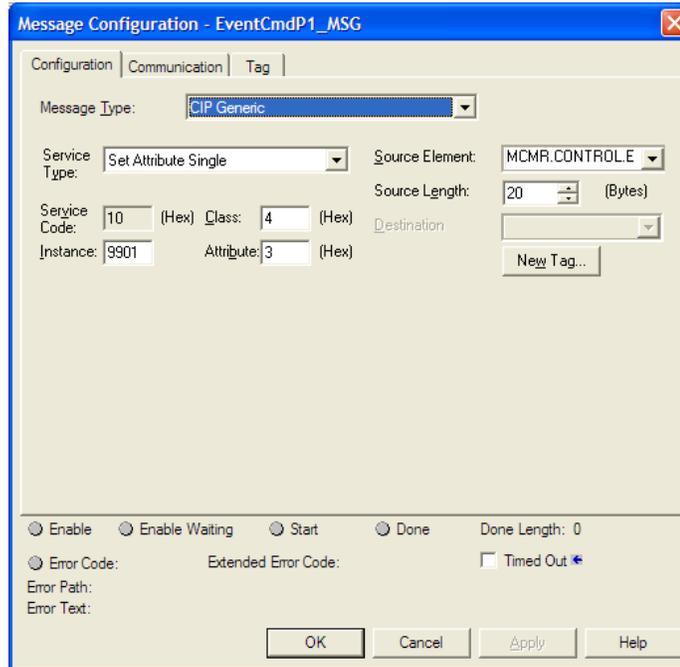
The format of the MSG block for this rung is as shown in the previous illustration except it used the MCMRCmds[0] for the source and destination tags.

The next rung performs the functions of a 9902 block:

Send Command Control data for port 1 to module.

Send Command Control data for port 2 to module.
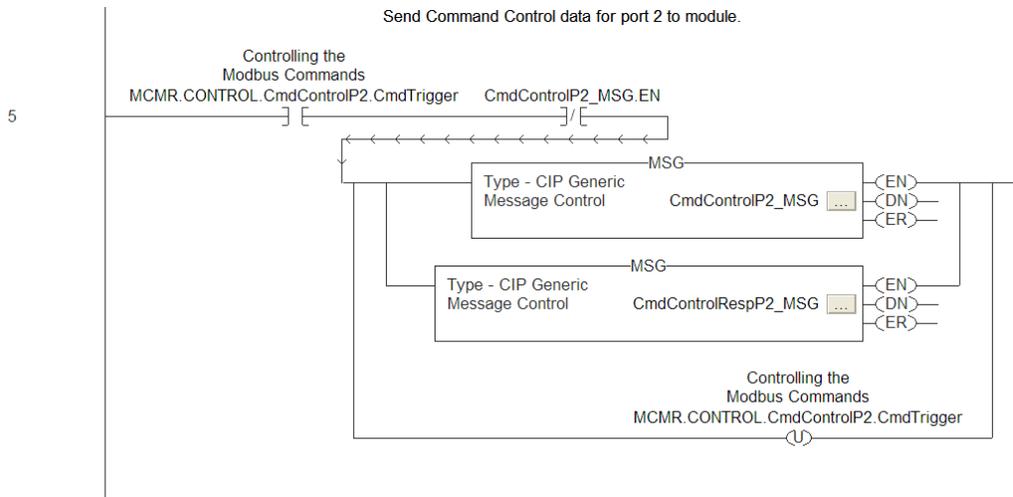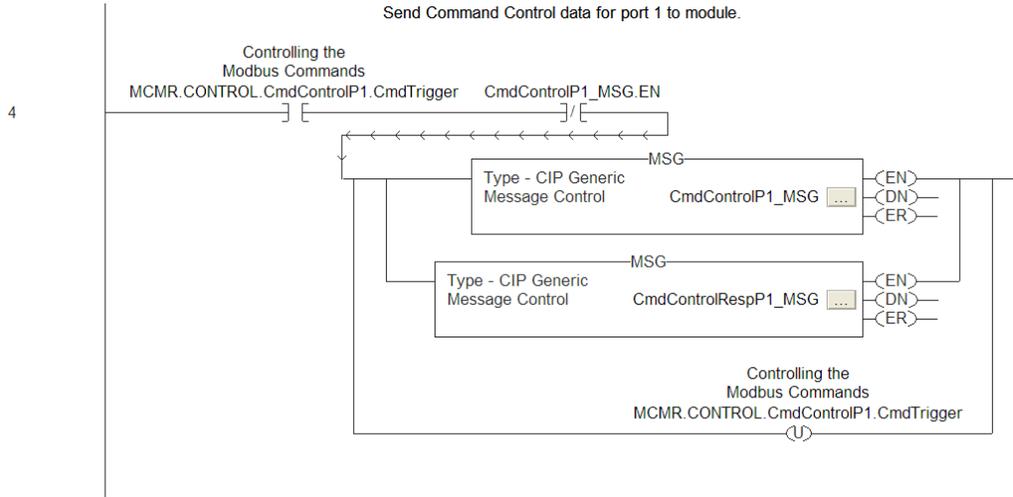
The following illustrations show the format of the MSG block.
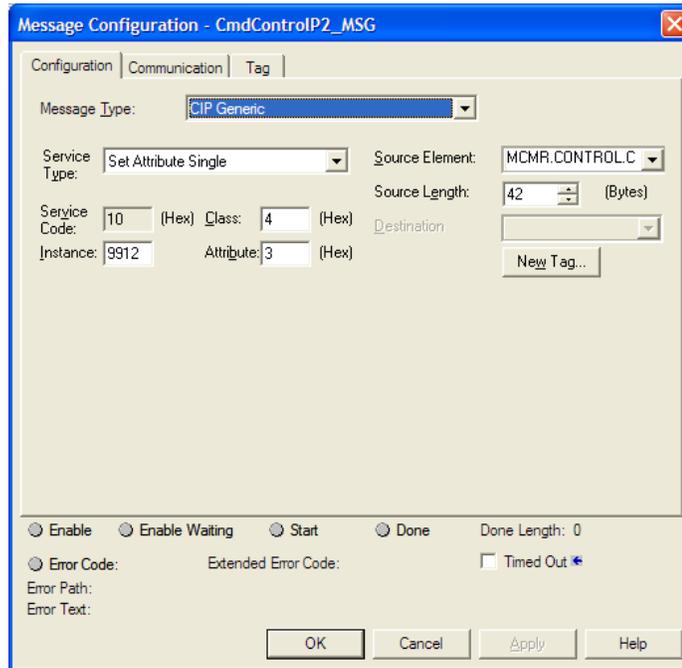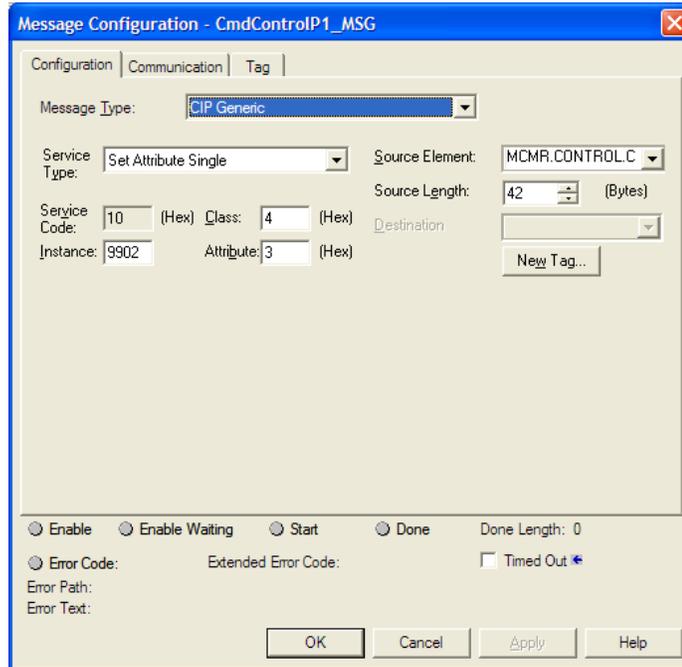
The next rung requests a set of command list errors:

The following illustration shows the format of the MSG block.

The data contained in the CmdErrData controller tag determines the set of errors returned.

Request Slave Status from port 1 from module.

8

Triggers port
status reading
MCMR.STATUS.Port1Stats.PortTrigger   SlaveStatusP1_MSG.EN

MSG
Type - CIP Generic
Message Control   SlaveStatusP1_MSG ...   —(EN)—
                                                      —(DN)—
                                                      —(ER)—

Triggers port
status reading
MCMR.STATUS.Port1Stats.PortTrigger
—(U)—

Request Slave Status from port 2 from module.

9

Triggers port
status reading
MCMR.STATUS.Port2Stats.PortTrigger   SlaveStatusP2_MSG.EN

MSG
Type - CIP Generic
Message Control   SlaveStatusP2_MSG ...   —(EN)—
                                                      —(DN)—
                                                      —(ER)—

Triggers port
status reading
MCMR.STATUS.Port2Stats.PortTrigger
—(U)—

The following illustration shows the format of the MSG block.

7 Support, Service & Warranty

Contacting Technical Support

ProSoft Technology, Inc. (ProSoft) 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 serial, Ethernet or fieldbus devices interfaced to the module, if any.

Note: *For technical support calls within the United States, an after-hours answering system allows 24-hour/7-days-a-week pager access to one of our qualified Technical and/or Application Support Engineers. Detailed contact information for all our worldwide locations is available on the following page.*

Internet	Web Site: www.prosoft-technology.com/support E-mail address: support@prosoft-technology.com
Asia Pacific (location in Malaysia)	Tel: +603.7724.2080, E-mail: asiapc@prosoft-technology.com Languages spoken include: Chinese, English
Asia Pacific (location in China)	Tel: +86.21.5187.7337 x888, E-mail: asiapc@prosoft-technology.com Languages spoken include: Chinese, English
Europe (location in Toulouse, France)	Tel: +33 (0) 5.34.36.87.20, E-mail: support.EMEA@prosoft-technology.com Languages spoken include: French, English
Europe (location in Dubai, UAE)	Tel: +971-4-214-6911, E-mail: mea@prosoft-technology.com Languages spoken include: English, Hindi
North America (location in California)	Tel: +1.661.716.5100, E-mail: support@prosoft-technology.com Languages spoken include: English, Spanish
Latin America (Oficina Regional)	Tel: +1-281-2989109, E-Mail: latinam@prosoft-technology.com Languages spoken include: Spanish, English
Latin America (location in Puebla, Mexico)	Tel: +52-222-3-99-6565, E-mail: soporte@prosoft-technology.com Languages spoken include: Spanish
Brasil (location in Sao Paulo)	Tel: +55-11-5083-3776, E-mail: brasil@prosoft-technology.com Languages spoken include: Portuguese, English

7.1 Warranty Information

For complete details regarding ProSoft Technology's TERMS & CONDITIONS OF SALE, WARRANTY, SUPPORT, SERVICE AND RETURN MATERIAL AUTHORIZATION INSTRUCTIONS please see the documents on the Product DVD or go to www.prosoft-technology/warranty.

Documentation is subject to change without notice.

Index