



Where Automation Connects.



**inRAx<sup>®</sup>**

**MVI69-MCM**

**CompactLogix or MicroLogix  
Platform**

Modbus Communication Module

March 22, 2011

**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](http://www.prosoft-technology.com)

[support@prosoft-technology.com](mailto:support@prosoft-technology.com)

**Copyright © 2011 ProSoft Technology, Inc., all rights reserved.**

MVI69-MCM User Manual

March 22, 2011

ProSoft Technology<sup>®</sup>, ProLinx<sup>®</sup>, inRAx<sup>®</sup>, ProTalk<sup>®</sup>, and RadioLinx<sup>®</sup> are Registered Trademarks of ProSoft Technology, Inc. All other brand or product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

## **ProSoft Technology<sup>®</sup> 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 CD-ROM, and are available at no charge from our web site: [www.prosoft-technology.com](http://www.prosoft-technology.com)

## Important Installation Instructions

Power, Input, and Output (I/O) wiring must be in accordance with Class I, Division 2 wiring methods, Article 501-4 (b) of the National Electrical Code, NFPA 70 for installation in the U.S., or as specified in Section 18-1J2 of the Canadian Electrical Code for installations in Canada, and in accordance with the authority having jurisdiction. The following warnings must be heeded:

- A** WARNING - EXPLOSION HAZARD - SUBSTITUTION OF COMPONENTS MAY IMPAIR SUITABILITY FOR CLASS I, DIV. 2;
- B** WARNING - EXPLOSION HAZARD - WHEN IN HAZARDOUS LOCATIONS, TURN OFF POWER BEFORE REPLACING OR WIRING MODULES
- C** WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.
- D** THIS DEVICE SHALL BE POWERED BY CLASS 2 OUTPUTS ONLY.

## 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.

## Warnings

### 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 non-hazardous.
- C** Suitable for use in Class I, Division 2 Groups A, B, C and D 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.

**Warning: This module is not hot-swappable!** Always remove power from the rack before inserting or removing this module, or damage may result to the module, the processor, or other connected devices.

## Battery Life Advisory

The MVI46, MVI56, MVI56E, MVI69, and MVI71 modules use a rechargeable Lithium Vanadium Pentoxide battery to backup the real-time clock and CMOS. The battery should last for the life of the module. The module must be powered for approximately twenty hours before the battery becomes fully charged. After it is fully charged, the battery provides backup power for the CMOS setup and the real-time clock for approximately 21 days. When the battery is fully discharged, the module will revert to the default BIOS and clock settings.

**Note:** The battery is not user replaceable.

## Markings

### Electrical Ratings

- Backplane Current Load: 800 mA @ 5.1 Vdc
- Power Supply Distance Rating: 2
- Operating Temperature: 0°C to 60°C (32°F to 140°F)
- Storage Temperature: -40°C to 85°C (-40°F to 185°F)
- 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

Class I, Division 2 Groups A, B, C, D

II 3 G

Ex nA IIC X

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	Applicable Standard(s)
ATEX	EN 60079-0:2006, EN 60079-15:2005
DNV	DET NORSKE VERITAS Test 2.4
CE	EMC-EN61326-1:2006; EN61000-6-4:2007
CB Safety	CA/10533/CSA, IEC 61010-1 Ed. 2, CB 243333-2056722 (2090408)
GOST-R	EN 61010



ME06

# Contents

Your Feedback Please .....	2
How to Contact Us .....	2
ProSoft Technology® Product Documentation .....	2
Important Installation Instructions .....	3
MVI (Multi Vendor Interface) Modules .....	3
Warnings .....	3
Battery Life Advisory .....	3
Markings.....	4

## Guide to the MVI69-MCM User Manual 9

### 1 Start Here 11

1.1	System Requirements .....	12
1.2	Package Contents .....	13
1.3	Installing ProSoft Configuration Builder Software .....	14
1.4	Setting Jumpers .....	15
1.5	Install the Module in the Rack .....	16

### 2 Configuring the MVI69-MCM Module 19

2.1	MVI69-MCM Sample Add-On Instruction Import Procedure .....	20
2.1.1	Create a new RSLogix5000 project .....	20
2.1.2	Create the Module .....	21
2.1.3	Import the Ladder Rung .....	23
2.1.4	Set the Read/Write Data Lengths .....	27
2.1.5	Set the Block Transfer Parameter Size .....	29
2.1.6	Set the Connection Input Size Values .....	30
2.1.7	Adding Multiple Modules (Optional) .....	31
2.1.8	Connecting Your PC to the Processor .....	39
2.1.9	Download the Sample Program to the Processor .....	39
2.1.10	Connect your PC to the Module .....	45
2.2	Using ProSoft Configuration Builder .....	46
2.2.1	Setting Up the Project .....	46
2.2.2	Renaming PCB Objects .....	48
2.3	Downloading the Project to the Module Using a Serial COM port .....	49
2.4	Module Configuration .....	50
2.4.1	[Module].....	50
2.4.2	[Backplane 69].....	50
2.4.3	[MCM Port x] .....	53
2.4.4	[Modbus Port x Commands].....	59

### 3 Ladder Logic 69

3.1	Ladder Logic and Firmware Compatibility Note .....	70
3.2	Module Data Object (MCM1ModuleDef) .....	71
3.2.1	Status Object (MCM1Status).....	72
3.2.2	User Data Objects .....	73

3.2.3	Slave Polling Control and Status .....	73
3.2.4	MODBUS Message Data .....	74
3.3	Adding the Module to an Existing CompactLogix Project.....	75
3.4	Adding the Module to an Existing MicroLogix Project .....	79
<b>4</b>	<b>Diagnostics and Troubleshooting</b> .....	<b>81</b>
4.1	LED Status Indicators .....	82
4.1.1	Clearing a Fault Condition .....	83
4.1.2	Troubleshooting .....	84
4.2	Using ProSoft Configuration Builder (PCB) for Diagnostics .....	85
4.2.1	Using the Diagnostic Window in ProSoft Configuration Builder .....	85
4.2.2	Navigation .....	87
4.2.3	Main Menu .....	88
4.2.4	Database View Menu.....	90
4.2.5	Backplane Menu .....	92
4.2.6	Protocol Serial MCM Menu.....	93
4.2.7	Master Command Error List Menu.....	94
4.2.8	Serial Port Menu .....	95
4.2.9	Data Analyzer .....	96
4.3	Reading Status Data from the Module .....	99
<b>5</b>	<b>Reference</b> .....	<b>101</b>
5.1	Product Specifications .....	102
5.1.1	General Specifications .....	102
5.1.2	Hardware Specifications .....	103
5.1.3	General Specifications - Modbus Master/Slave.....	104
5.1.4	Functional Specifications .....	105
5.2	Functional Overview .....	106
5.2.1	About the MODBUS Protocol .....	106
5.2.2	Module Power Up .....	106
5.2.3	Main Logic Loop .....	107
5.2.4	Backplane Data Transfer .....	107
5.3	Data Flow between MVI69-MCM Module and CompactLogix or MicroLogix Processor	110
5.3.1	Slave Driver .....	110
5.3.2	Master Driver Mode .....	112
5.4	Normal Data Transfer .....	115
5.4.1	Block Request from the Processor to the Module .....	115
5.4.2	Block Response from the Module to the Processor .....	115
5.4.3	Read Block and Write Block Transfer Sequences.....	116
5.4.4	If Block Transfer Size = 60 .....	117
5.4.5	If Block Transfer Size = 120 .....	118
5.4.6	If Block Transfer Size = 240 .....	119
5.4.7	Status Data Block (Read Block ID = 0).....	119
5.5	Special Control and Status Blocks.....	121
5.5.1	Slave Disable and Enable Control Blocks .....	121
5.5.2	Slave Status Blocks .....	124
5.5.3	Event Command .....	125
5.5.4	Command Control.....	127
5.5.5	Pass-Through Control Blocks.....	129
5.5.6	Initialize Output Data .....	133

5.5.7	Warm Boot Block (9998) .....	133
5.5.8	Cold Boot Block (9999) .....	133
5.6	Modbus Protocol Specification .....	134
5.6.1	Commands Supported by the Module.....	134
5.6.2	Read Coil Status (Function Code 01) .....	134
5.6.3	Read Input Status (Function Code 02).....	135
5.6.4	Read Holding Registers (Function Code 03) .....	136
5.6.5	Read Input Registers (Function Code 04).....	137
5.6.6	Force Single Coil (Function Code 05) .....	138
5.6.7	Preset Single Register (Function Code 06).....	139
5.6.8	Diagnostics (Function Code 08).....	140
5.6.9	Force Multiple Coils (Function Code 15).....	142
5.6.10	Preset Multiple Registers (Function Code 16) .....	143
5.6.11	MODBUS Exception Responses.....	144
5.7	Cable Connections .....	146
5.7.1	RS-232 Configuration/Debug Port .....	146
5.7.2	RS-232 Application Port(s).....	146
5.7.3	RS-422 .....	149
5.7.4	RS-485 Application Port(s).....	149
5.7.5	DB9 to RJ45 Adaptor (Cable 14) .....	150
5.8	MCM Database Definition .....	151
5.9	Status Data Definition.....	152

**6 Support, Service & Warranty 155**

	Contacting Technical Support.....	155
6.1	Return Material Authorization (RMA) Policies and Conditions.....	157
6.1.1	Returning Any Product .....	157
6.1.2	Returning Units Under Warranty .....	158
6.1.3	Returning Units Out of Warranty .....	158
6.2	LIMITED WARRANTY.....	159
6.2.1	What Is Covered By This Warranty .....	159
6.2.2	What Is Not Covered By This Warranty .....	160
6.2.3	Disclaimer Regarding High Risk Activities .....	160
6.2.4	Intellectual Property Indemnity.....	161
6.2.5	Disclaimer of all Other Warranties .....	161
6.2.6	Limitation of Remedies ** .....	162
6.2.7	Time Limit for Bringing Suit .....	162
6.2.8	No Other Warranties .....	162
6.2.9	Allocation of Risks.....	162
6.2.10	Controlling Law and Severability.....	163

**Index 165**





## Guide to the MVI69-MCM 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 81)	This section describes Diagnostic and Troubleshooting procedures.
Reference Product Specifications Functional Overview	→	Reference (page 101) Product Specifications (page 102) Functional Overview (page 106, page 91)	These sections contain general references associated with this product, Specifications, and the Functional Overview.
Support, Service, and Warranty Index	→	Support, Service and Warranty (page 155) Index	This section contains Support, Service and Warranty information.  Index of chapters.



# 1 Start Here

## *In This Chapter*

❖ System Requirements .....	12
❖ Package Contents .....	13
❖ Installing ProSoft Configuration Builder Software.....	14
❖ Setting Jumpers .....	15
❖ Install the Module in the Rack .....	16

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

- **Rockwell Automation® RSLogix™ software:** launch the program, configure ladder logic, and transfer the ladder logic to the processor
- **Microsoft Windows:** install and launch programs, execute menu commands, navigate dialog boxes, and enter data
- **Hardware installation and wiring:** install the module, and safely connect MODBUS and CompactLogix or MicroLogix devices to a power source and to the MVI69-MCM module's application port(s)

## 1.1 System Requirements

The MVI69-MCM module requires the following minimum hardware and software components:

- *Rockwell Automation CompactLogix or MicroLogix* processor, with compatible power supply and one free slot in the rack, for the MVI69-MCM module. The module requires 800 mA of available power.

**Important:** The MVI69-MCM module has a power supply distance rating of 2 (L43 and L45 installations on first 2 slots of 1769 bus).

**Important:** For 1769-L23x processors, please make note of the following limitations.

- 1769-L23-QBFC1B = 800 mA at 5 Vdc (One MVI69-MCM will use all 800 mA of available power. No other modules can be used with an MVI69 module connected to this processor.)
- 1769-L23E-QB1B = 1000 mA at 5 Vdc (One MVI69-MCM will use 800 mA of available power. One other module can be used on this rack provided it consumes less than 200 mA at 5 Vdc.)
- 1769-L23E-QBFC1B = 450 mA at 5 Vdc (No MVI69 module can be used with this processor.)

- *Rockwell Automation RSLogix 5000 (CompactLogix) or RSLogix 500 (MicroLogix)* programming software
- *Rockwell Automation RSLinx* communication software
- *Pentium® II* 450 MHz minimum. *Pentium III* 733 MHz (or better) recommended
- Supported operating systems:
  - *Microsoft Windows XP Professional with Service Pack 1 or 2*
  - *Microsoft Windows 2000 Professional with Service Pack 1, 2, or 3*
  - *Microsoft Windows Server 2003*
- 128 Mbytes of RAM minimum, 256 Mbytes of RAM recommended
- 100 Mbytes of free hard disk space (or more based on application requirements)
- 256-color VGA graphics adapter, 800 x 600 minimum resolution (True Color 1024 × 768 recommended)
- CD-ROM drive
- HyperTerminal or other terminal emulator program capable of file transfers using Ymodem protocol.

## 1.2 Package Contents

The following components are included with your MVI69-MCM 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	MVI69-MCM Module	MVI69-MCM	Modbus Communication Module
1	Cable	Cable #15, RS232 Null Modem	For RS232 Connection to the CFG Port
3	Cable	Cable #14, RJ45 to DB9 Male Adapter cable	For DB9 Connection to Module's 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 CD		Contains sample programs, utilities and documentation for the MVI69-MCM module.

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

### 1.3 Installing ProSoft Configuration Builder Software

You must install the *ProSoft Configuration Builder (PCB)* software to configure the module. You can always get the newest version of *ProSoft Configuration Builder* from the ProSoft Technology website.

#### **Installing ProSoft Configuration Builder from the ProSoft website**

- 1 Open your web browser and navigate to <http://www.prosoft-technology.com/pcb>
- 2 Click the **DOWNLOAD HERE** link to download the latest version of *ProSoft Configuration Builder*.
- 3 Choose **SAVE** or **SAVE FILE** when prompted.
- 4 Save the file to your *Windows Desktop*, so that you can find it easily when you have finished downloading.
- 5 When the download is complete, locate and open the file, and then follow the instructions on your screen to install the program.

If you do not have access to the Internet, you can install *ProSoft Configuration Builder* from the *ProSoft Solutions Product CD-ROM*, included in the package with your module.

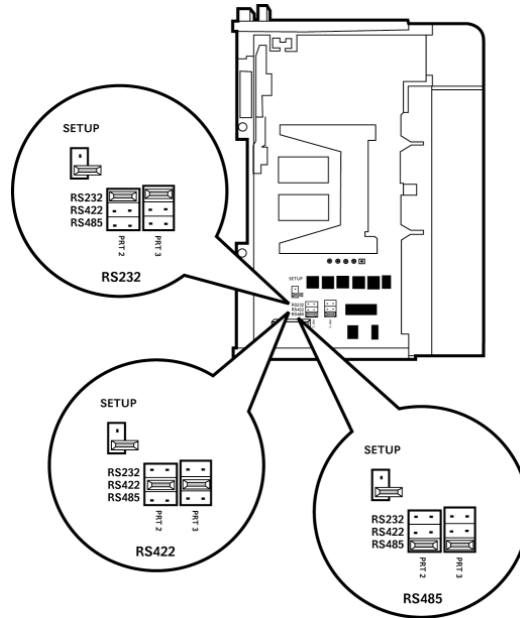
#### **Installing ProSoft Configuration Builder from the Product CD-ROM**

- 1 Insert the *ProSoft Solutions Product CD-ROM* into the CD-ROM drive of your PC. Wait for the startup screen to appear.
- 2 On the startup screen, click **PRODUCT DOCUMENTATION**. This action opens a *Windows Explorer* file tree window.
- 3 Click to open the **UTILITIES** folder. This folder contains all of the applications and files you will need to set up and configure your module.
- 4 Double-click the **SETUP CONFIGURATION TOOL** folder, double-click the **PCB\_\*.EXE** file and follow the instructions on your screen to install the software on your PC. The information represented by the "\*" character in the file name is the *PCB* version number and, therefore, subject to change as new versions of *PCB* are released.

**Note:** Many of the configuration and maintenance procedures use files and other utilities on the CD-ROM. You may wish to copy the files from the Utilities folder on the CD-ROM to a convenient location on your hard drive.

## 1.4 Setting Jumpers

When the module is manufactured, the port selection jumpers are set to RS-232. To use RS-422 or RS-485, you must set the jumpers to the correct position. The following diagram describes the jumper settings.



The Setup Jumper acts as "write protection" for the module's flash memory. In "write protected" mode, the Setup pins are not connected, and the module's firmware cannot be overwritten. Do not jumper the Setup pins together unless you are directed to do so by ProSoft Technical Support.

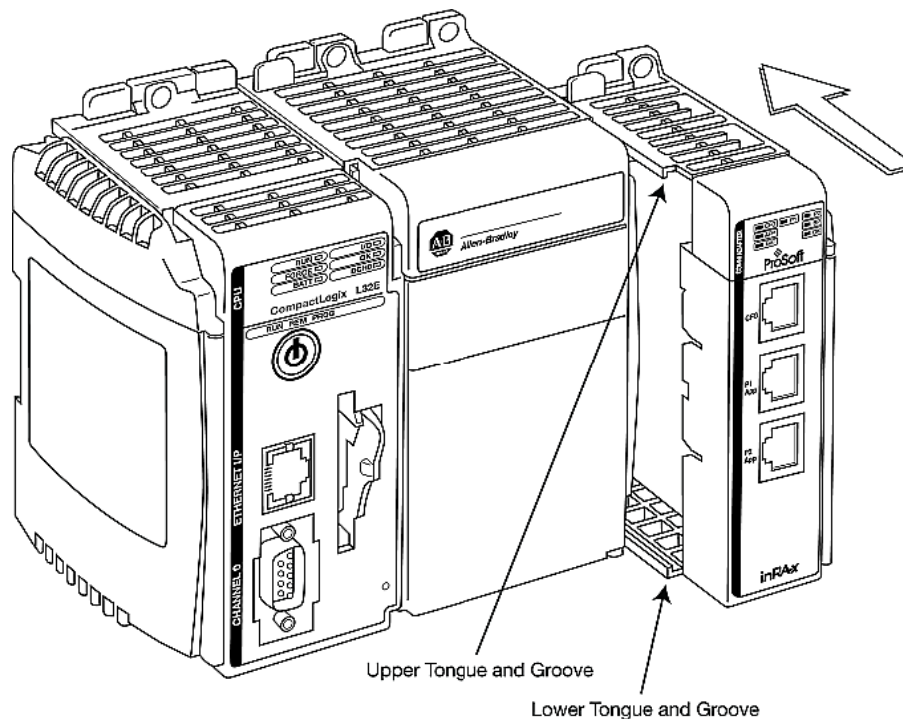
## 1.5 Install the Module in the Rack

This section describes how to install the module into a CompactLogix or MicroLogix rack

Before you attempt to install the module, make sure that the bus lever of the adjacent module is in the unlocked (fully right) position.

**Warning: This module is not hot-swappable!** Always remove power from the rack before inserting or removing this module, or damage may result to the module, the processor, or other connected devices.

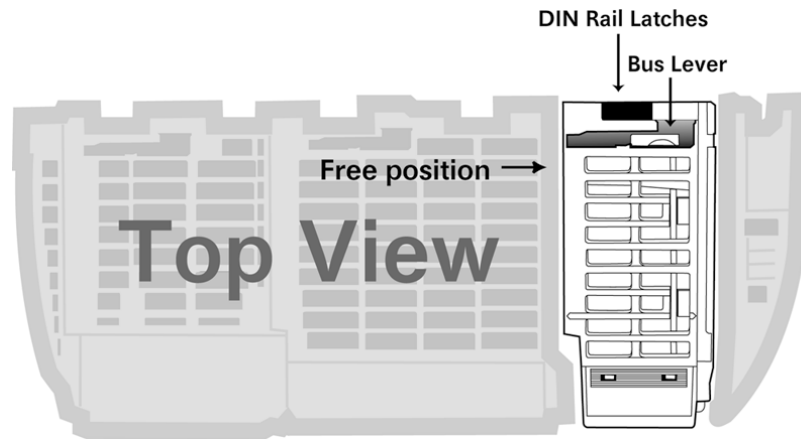
- 1 Align the module using the upper and lower tongue-and-groove slots with the adjacent module and slide forward in the direction of the arrow.



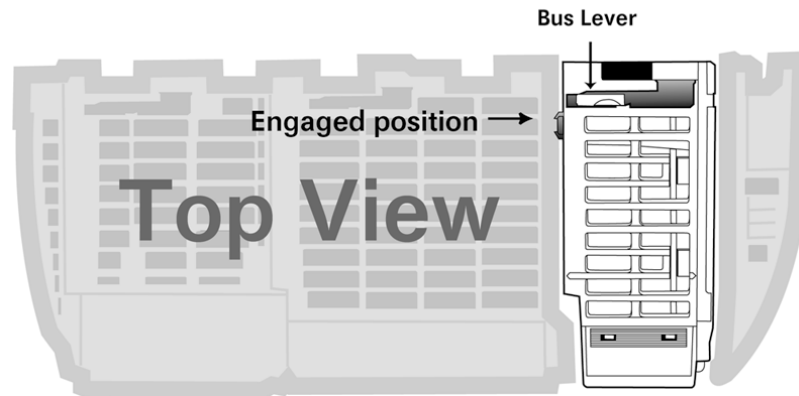
- 2 Move the module back along the tongue-and-groove slots until the bus connectors on the MVI69 module and the adjacent module line up with each other.



- 3 Push the module's bus lever back slightly to clear the positioning tab and move it firmly to the left until it clicks. Ensure that it is locked firmly in place.

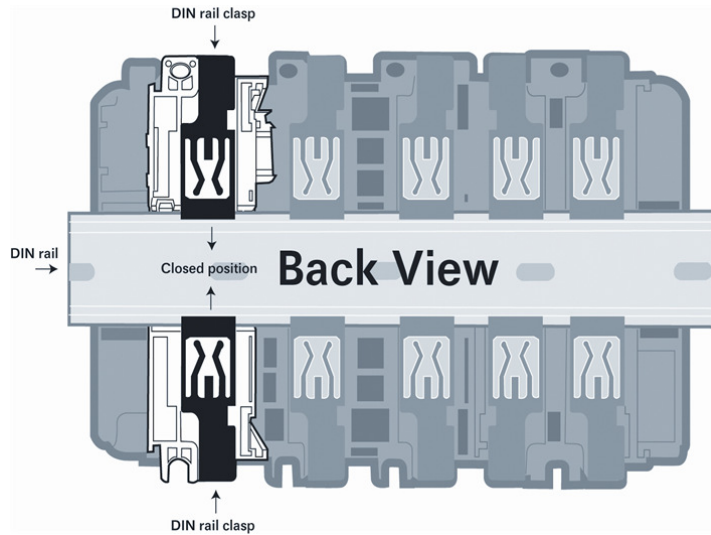
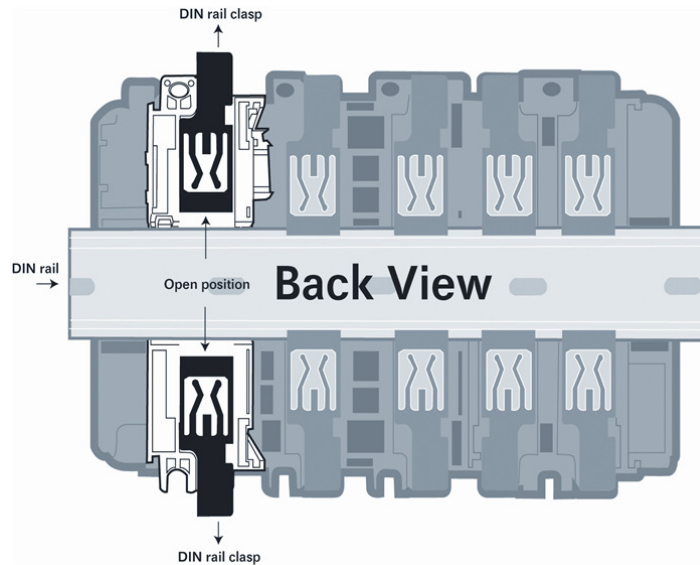


Move the Bus Lever to the left  
until it clicks



- 4 Close all DIN-rail latches.

- 5 Press the DIN-rail mounting area of the controller against the DIN-rail. The latches will momentarily open and lock into place.



## 2 Configuring the MVI69-MCM Module

### *In This Chapter*

- ❖ MVI69-MCM Sample Add-On Instruction Import Procedure .....20
- ❖ Using ProSoft Configuration Builder..... 46
- ❖ Downloading the Project to the Module Using a Serial COM port .....49
- ❖ Module Configuration ..... 50

## 2.1 MVI69-MCM Sample Add-On Instruction Import Procedure

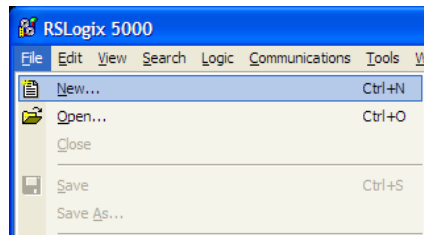
**Note:** this section only applies if you are using RSLogix 5000 version 16 or higher. If you are configuring the MVI69-MCM module with an earlier version of RSLogix 5000, please refer to Installing and Configuring the Module with a CompactLogix Processor (page 75).

The following file is required before you start this procedure. Copy the file from the ProSoft Solutions CD-ROM, or download it from [www.prosoft-technology.com](http://www.prosoft-technology.com).

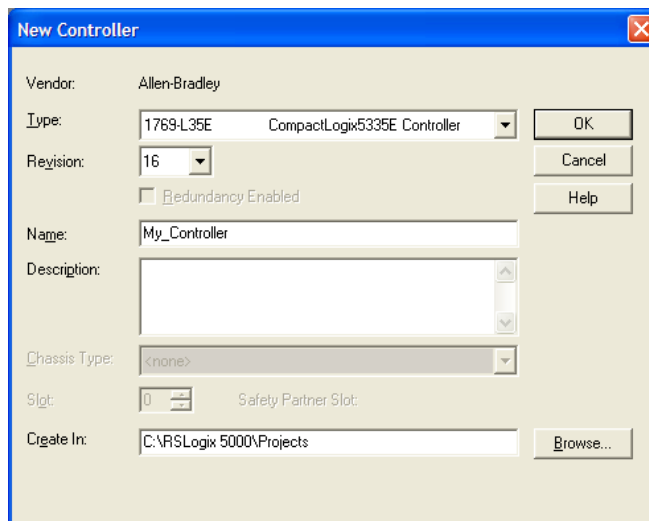
File Name	Description
MVI69MCM_AddOn_Rung_v1_4.L	L5X file contains the Add-On instruction, the user defined data types, data objects and ladder logic required to set up the MVI69-MCM module

### 2.1.1 Create a new RSLogix5000 project

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

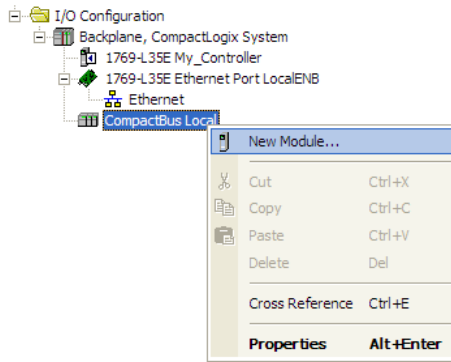


- 2 Select **REVISION 16**

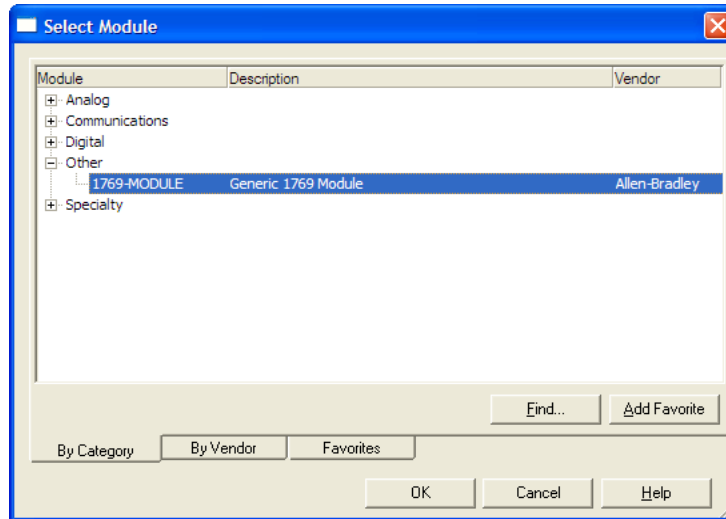


### 2.1.2 Create the Module

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



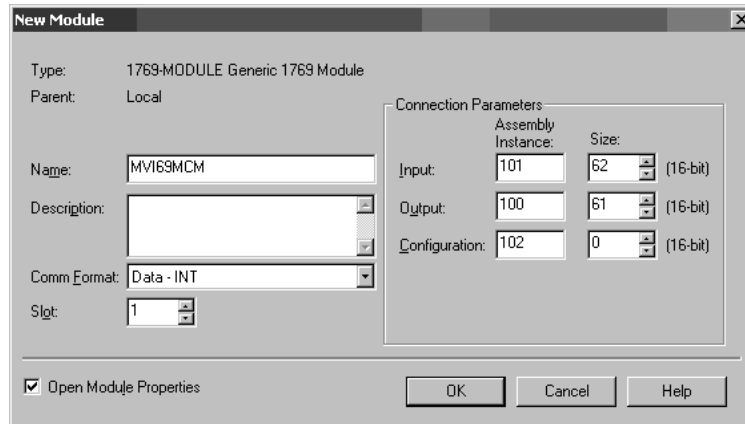
- 2 Select **1769-MODULE**



- 3 Set the Module Properties values as follows:

Parameter	Value
Name	Enter a module identification string. Example: MVI69MCM
Description	Enter a description for the module. Example: ProSoft communication module for Serial Modbus communications.
Comm Format	Select Data-INT
Slot	Enter the slot number in the rack where the MV69-MCM module will be installed.
Input Assembly Instance	101
Input Size	62 / 122 / 242
Output Assembly Instance	100
Output Size	61 / 121 / 241
Configuration Assembly Instance	102
Configuration Size	0

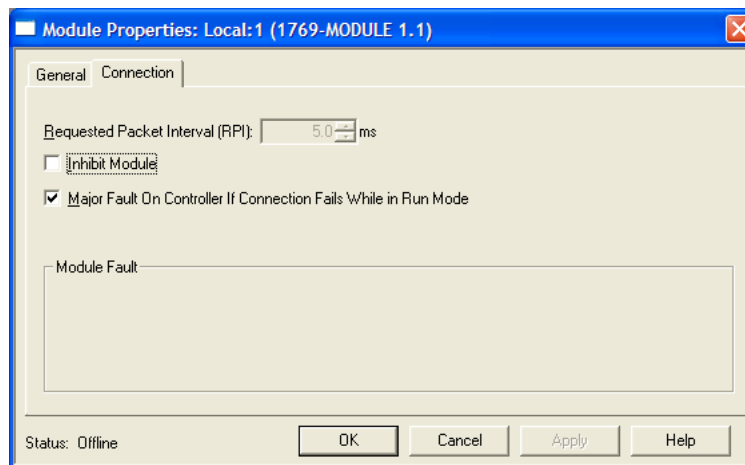
The following illustration shows an example where the module was configured for a block transfer size of 60 words (input block size = 62 words, output block size = 61 words):



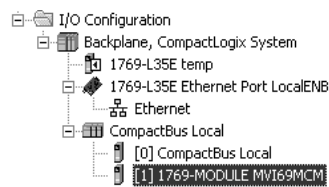
The following options are available:

Block Transfer Size	Input Block Size	Output Block Size
60	62	61
120	122	121
240	242	241

- 4 On the Connection tab, set the RPI value for your project. Click OK to confirm.

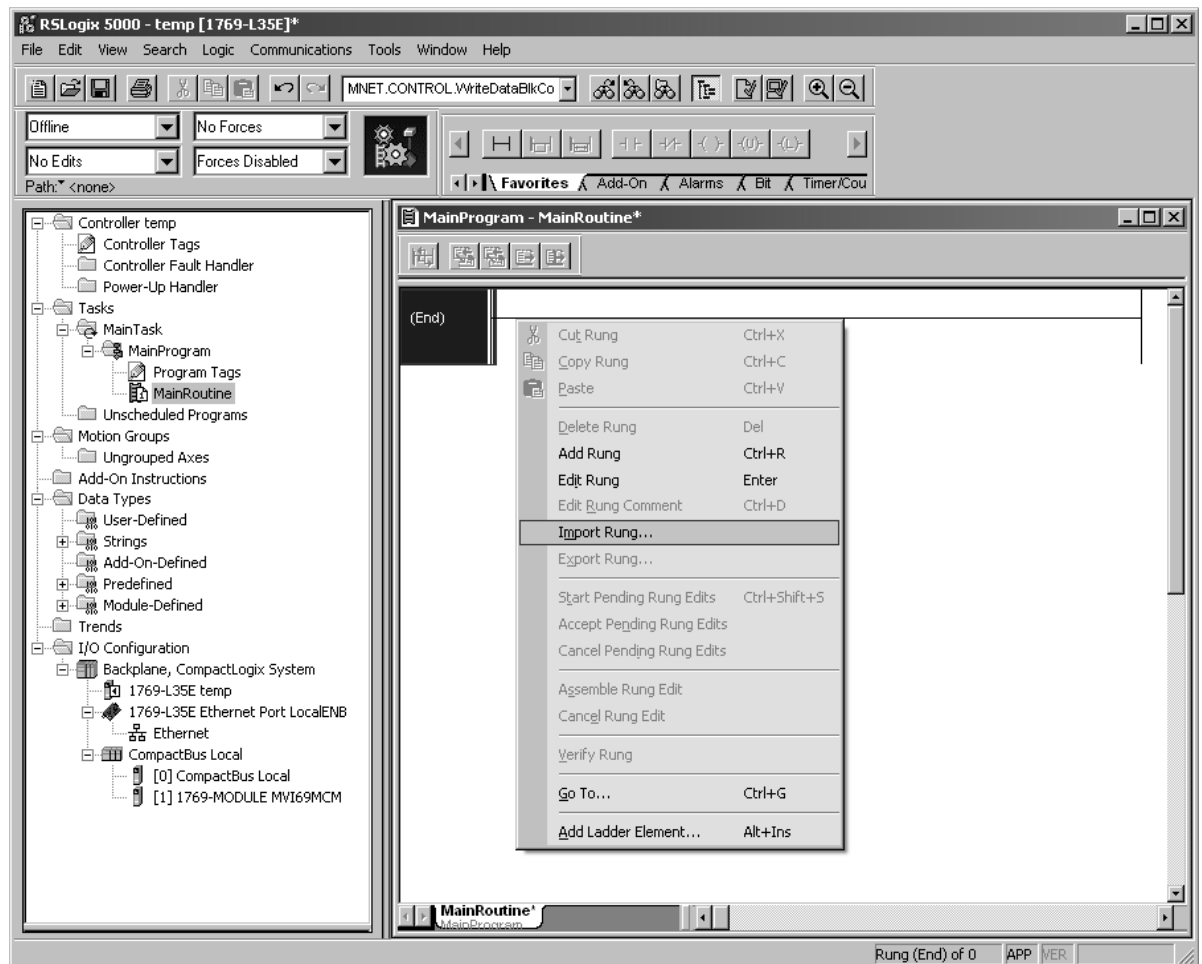


Now the MVI69-MCM module will be visible at the I/O Configuration section.

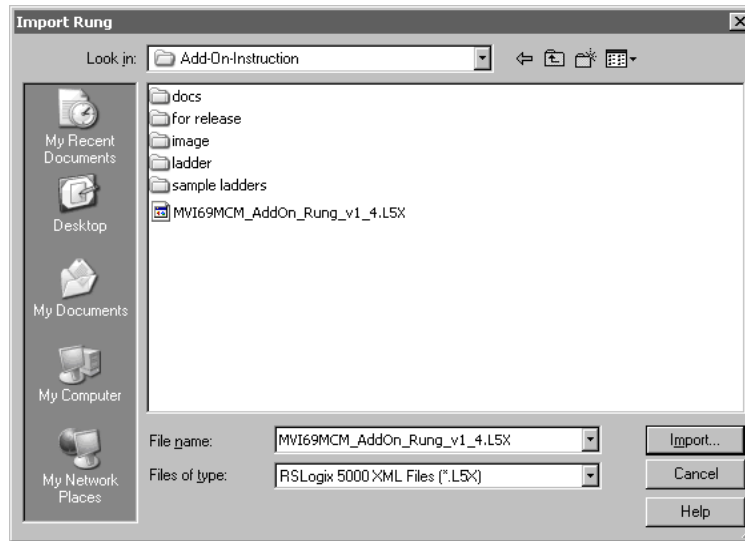


### 2.1.3 Import the Ladder Rung

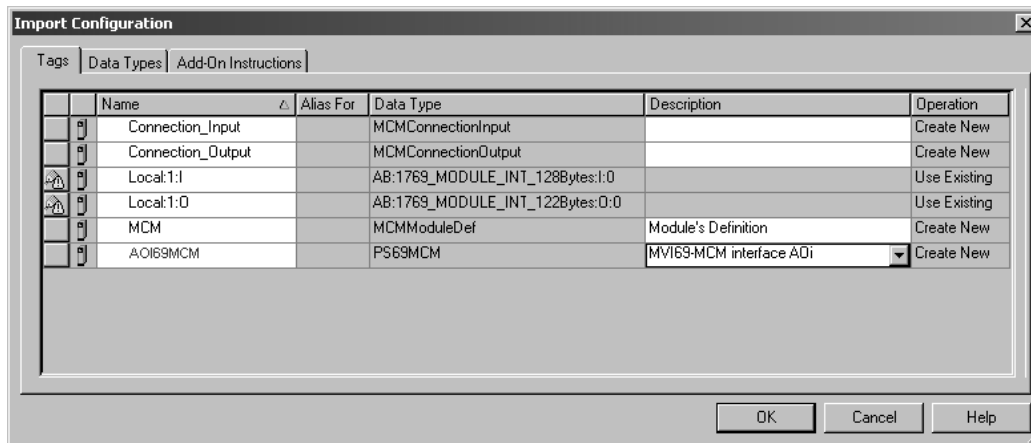
- 1 Open your application in RSLogix 5000.
- 2 To create a new routine, expand the **TASKS** folder, and then expand the **MAIN TASK** folder.
- 3 On the **MAIN PROGRAM** folder, click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW ROUTINE**.
- 4 In the **NEW ROUTINE** dialog box, enter the name and description of your routine, and then click **OK**. In this example we are demonstrating the importing of the ladder rung using the default MainRoutine. In the case where you create a routine by an other name for placing the Add-On instruction, then in your original routine where your other ladder logic is located you need to add a rung with a jump instruction to the new routine holding the Add-On instruction.
- 5 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...**".



6 Select the **MVI69MCM\_ADDON\_RUNG\_v1\_4.L5X** file

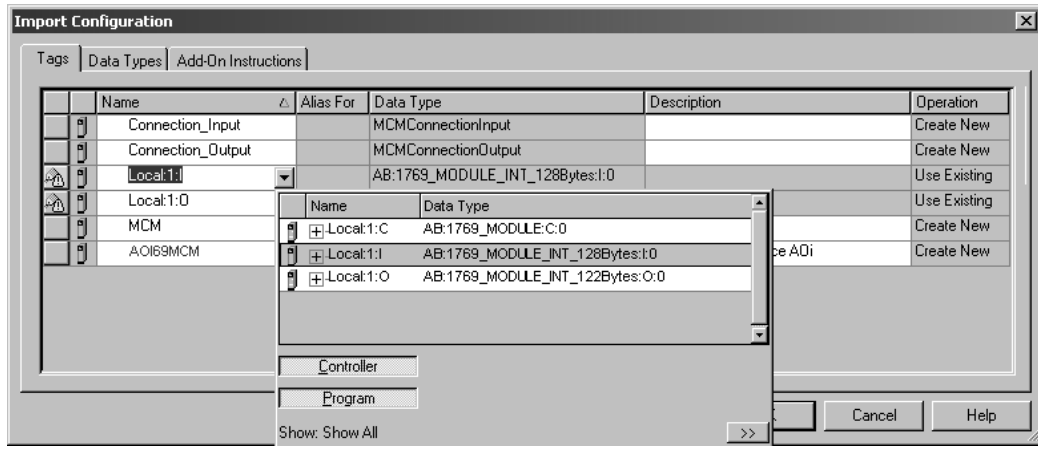


7 The following window will be displayed showing the controller tags to be created during the import procedure: If desired, the description, "MVI69-MCM Interface AOI" may be typed into the description field for MVI69MCM\_AddOn\_Rung\_v1\_4.L5x file.

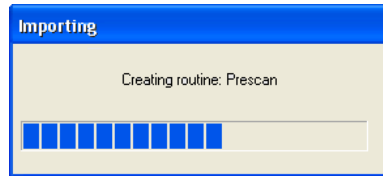




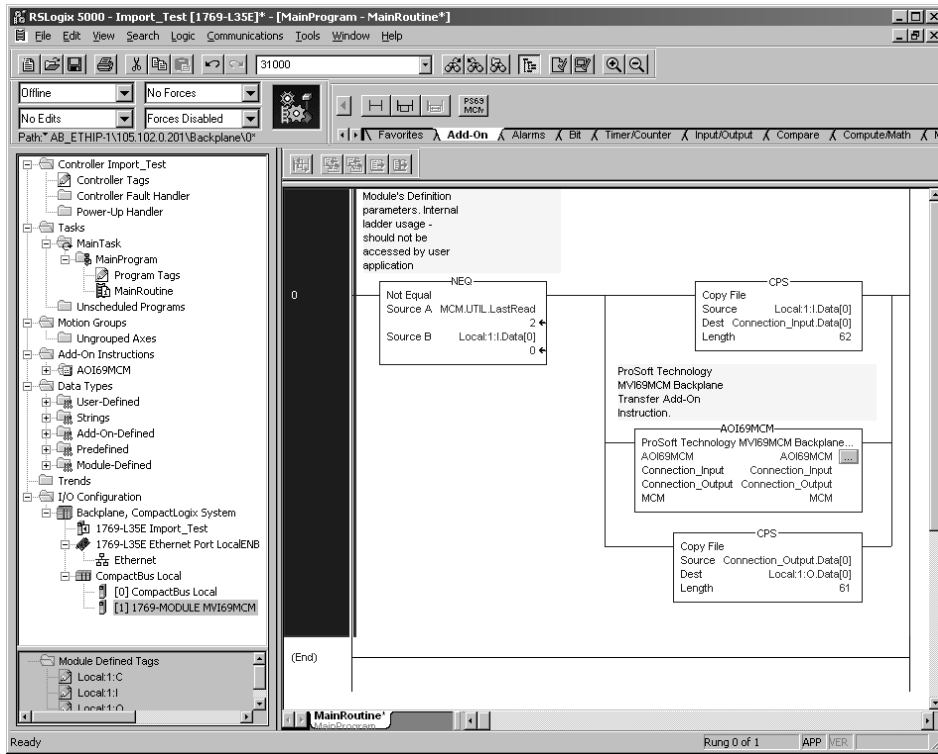
- 8 If you are using the module in a different slot (or remote rack) select the correct connection input and output variables associated to the module. If your module is located in slot 1 of the local rack this step is not required.



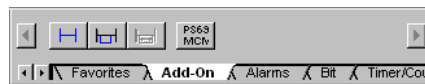
- 9 Click **OK** to confirm the import. RSLogix will indicate that the import is under progress:



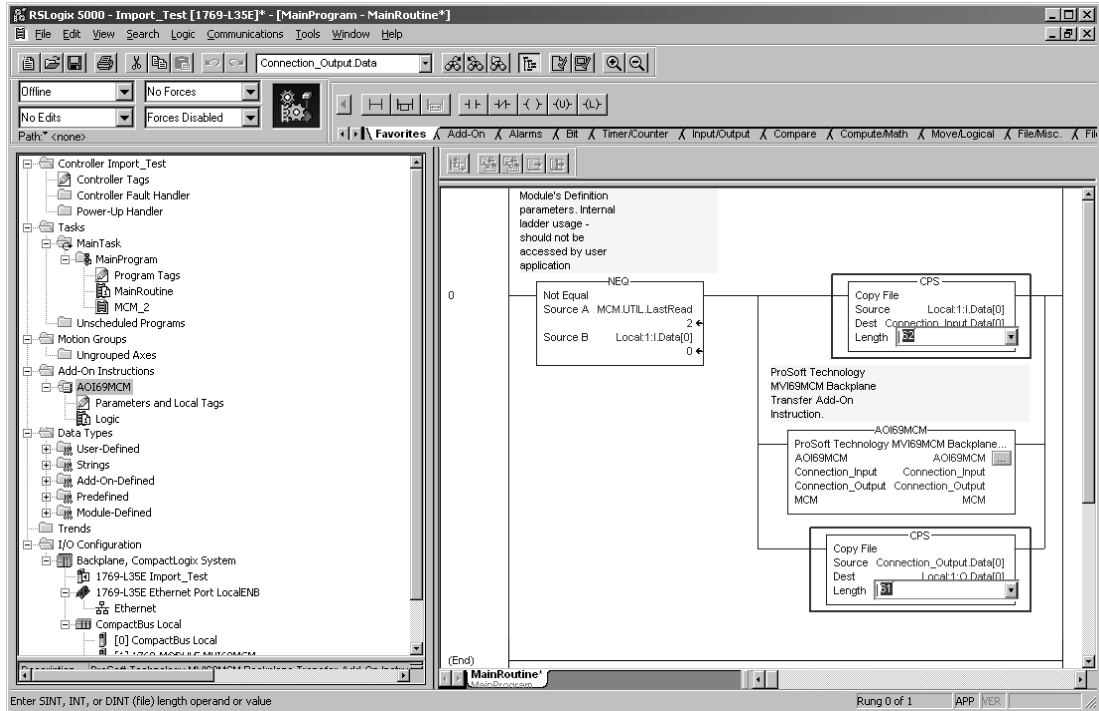
When the import is completed, the new rung with the Add-On instruction will be visible as shown in the following illustration.



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



**10** The imported rung will contain the Add-On instruction with two CPS instructions as follows below. The CPS instructions are set by default for a length of 62/61 words as follows:



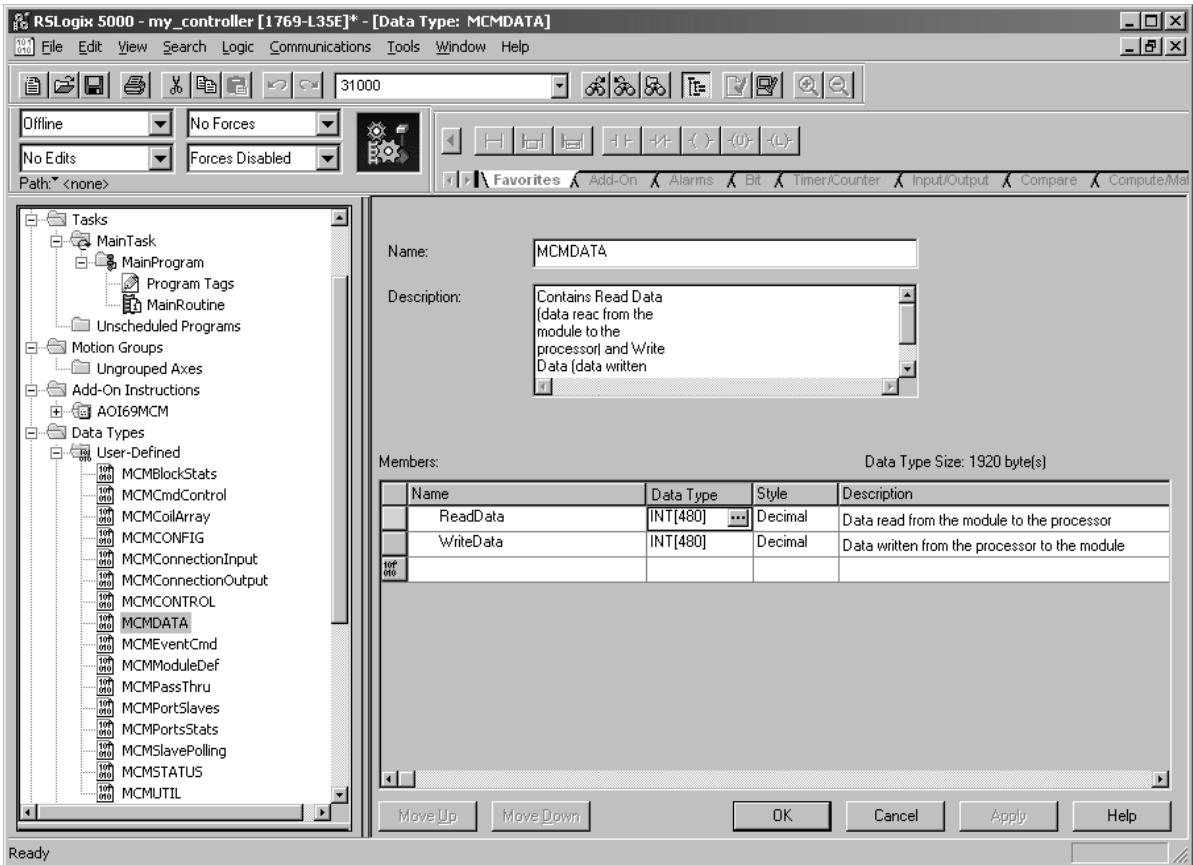
Edit the above CPS instructions Length field values according to the following table.

"Block Transfer Size Parameter" – 60/120/240 options)		Ladder Routine window:	
Connection Parameters:		CPS instructions Length field values:	
Input Size:	Output Size:		
62	61	62	61
122	121	122	121
242	241	242	241

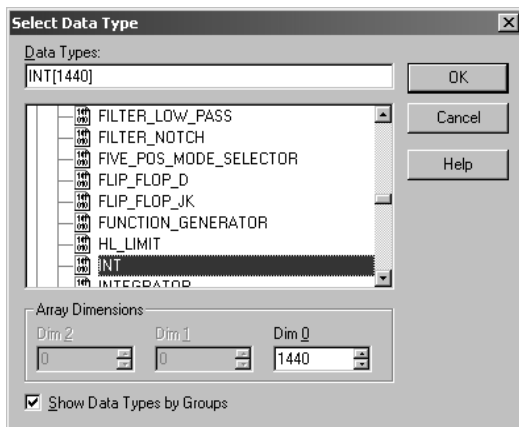
### 2.1.4 Set the Read/Write Data Lengths

- The imported rung contains the **MCMDATA** object Tag arrays **READDATA** and **WRITEDATA** set to the factory default values of 480. These tags will contain:
  - READDATA** - data area copied from the module to the processor
  - WRITEDATA** - data area copied from the processor to the module

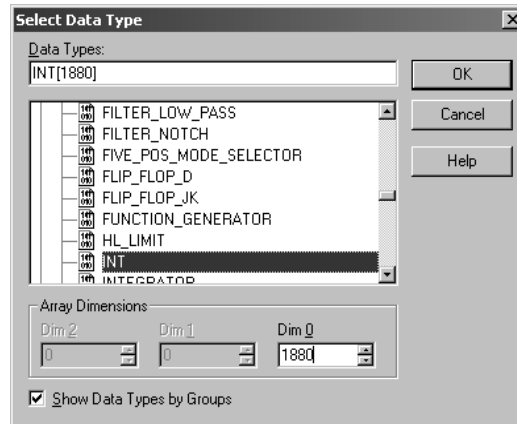
- If you have changed the **READ REGISTER COUNT** and **WRITE REGISTER COUNT** values in the **[BACKPLANE 69]** section of the module's configuration file, you must adjust these array sizes to match those values.



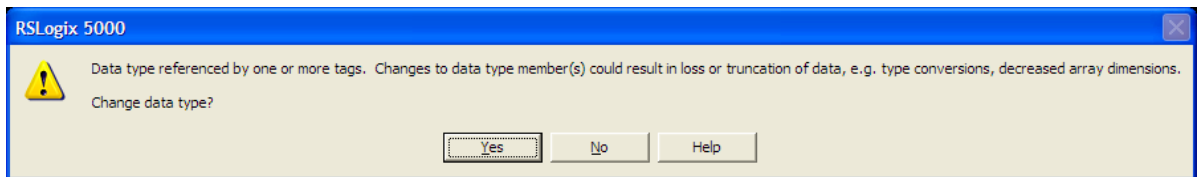
**Example:** If in the configuration file section [Backplane 69] the parameter setting is "Read Register Count : 1440" then set ReadData tag array size to INT[1440].



**Example:** If in the configuration file section [Backplane 69] the parameter setting is "Write Register Count : 1880" then set WriteData tag array size to INT[1880].

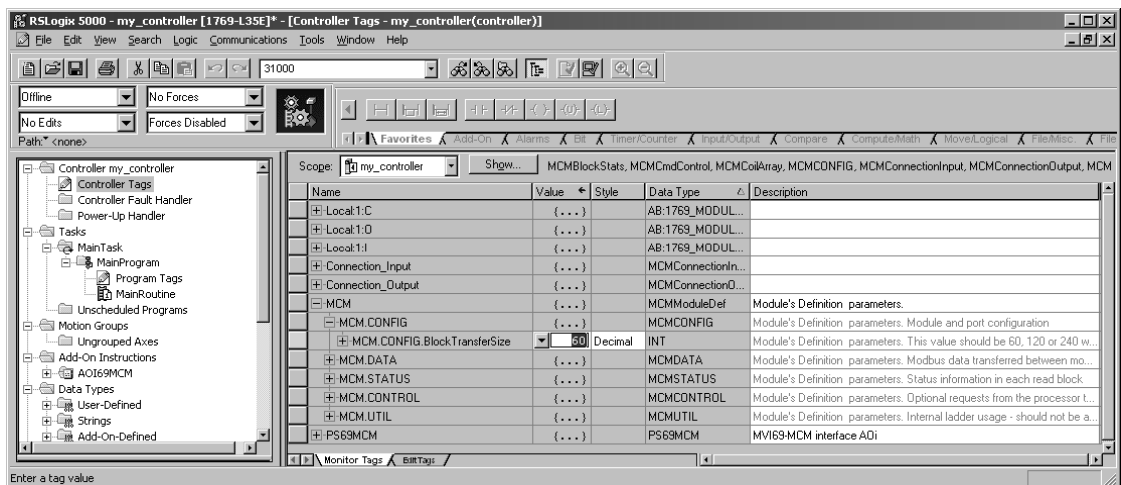


You will be prompted to confirm the changes. Click Yes to continue.



### 2.1.5 Set the Block Transfer Parameter Size

The **MCM.BLOCKTRANSFERSIZE** controller tag is set to 60 in the Add-On Instruction. If you have configured a different block transfer size in the module's configuration file, you must change this value to match.



Edit the tag values according to the following table.

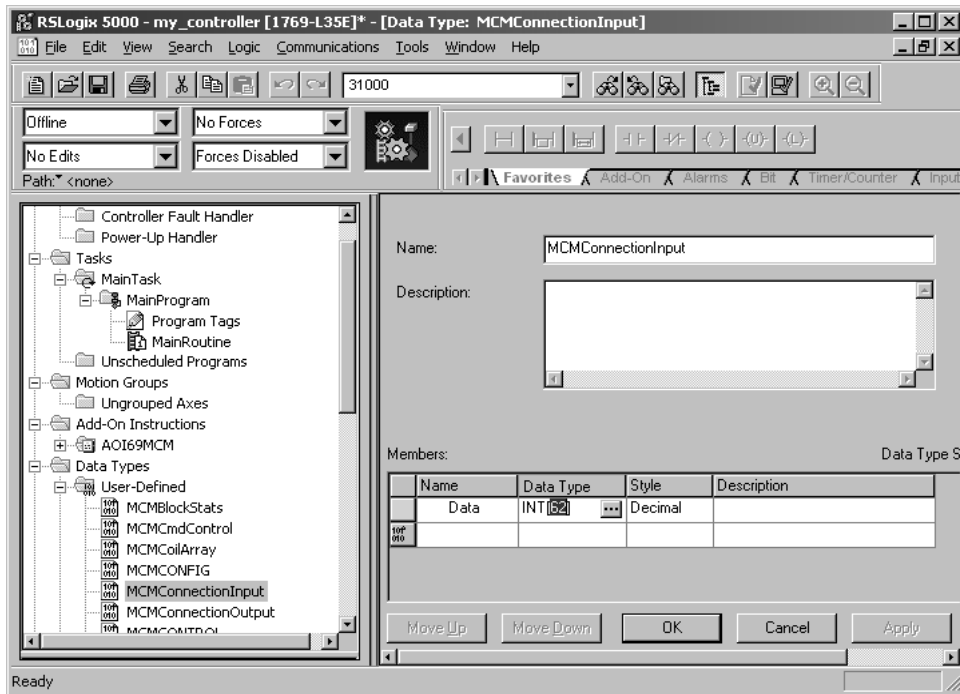
<b>Module Properties dialog box:</b>		<b>Controller Organizer's Controller Tags folder:</b>
<b>Connection Parameters:</b>		<b>MCM.BlockTransferSize tag value:</b>
Input Size:	Output Size:	
62	61	60
122	121	120
242	241	240

### 2.1.6 Set the Connection Input Size Values

If you change the block transfer size, you must also change the following data types:

- **MCMCONNECTIONINPUT** – Data type used for the Connection Input pin in the Add-On instruction.
- **MCMCONNECTIONOUTPUT** – Data type used for the Connection Output pin in the Add-On instruction.

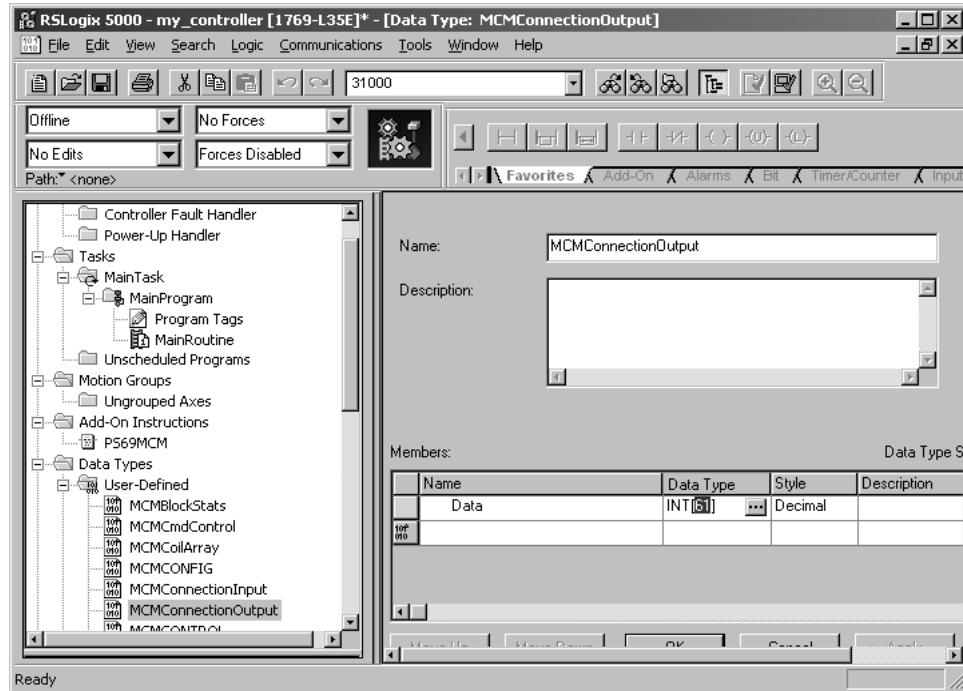
Access the user data type definition **MCMCONNECTIONINPUT** as follows



Edit the tag values according to the following table.

<b>Module Properties dialog box:</b>		<b>Controller Organizer's Controller Tags folder:</b>
<b>Connection Parameters:</b>		<b>MCMConnectionInput.Data tag value:</b>
<b>Input Size:</b>		
62		62
122		122
242		242

Access the user data type definition **MCMCONNECTIONOUTPUT** as follows:



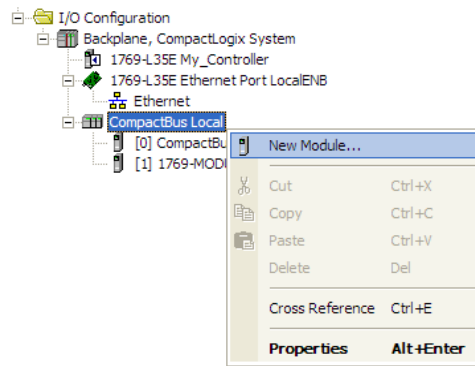
Edit the tag values according to the following table.

Module Properties dialog box:	Controller Organizer's Controller Tags folder:
Connection Parameters:	MCMConnectionOutput.Data tag value:
<b>Output Size:</b>	
61	61
121	121
241	241

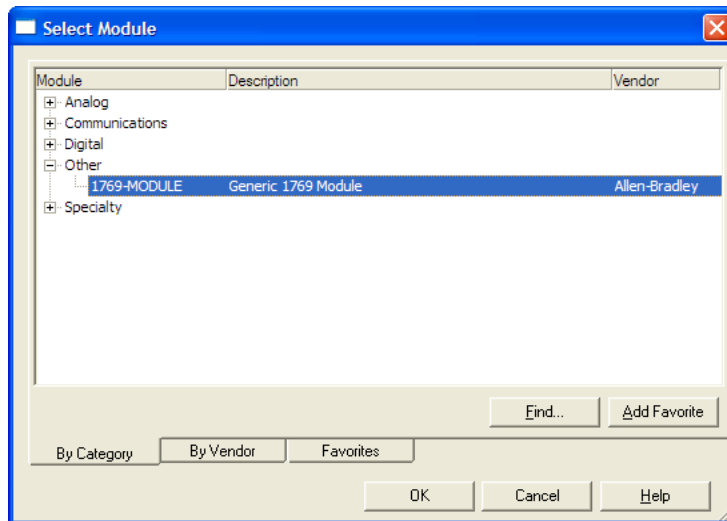
### 2.1.7 Adding Multiple Modules (Optional)

**Important:** If your application requires more than one MVI69-MCM module into the same project, follow the steps below and make certain that both modules are assigned identical Block Transfer Sizes.

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



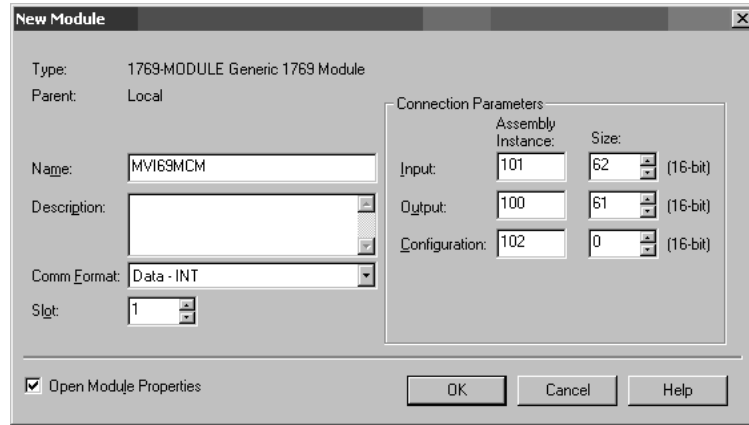
- 2 Select **1769-MODULE**



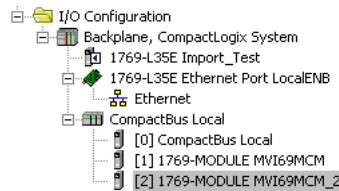
- 3 Fill the module properties as follows:

Parameter	Value
Name	Enter a module identification string. Example: MVI69MCM_2
Description	Enter a description for the module. Example: ProSoft communication module for Serial Modbus communications.
Comm Format	Select Data-INT
Slot	Enter the slot number in the rack where the MV69-MCM module will be installed.
Input Assembly Instance	101
Input Size	62 / 122 / 242
Output Assembly Instance	100
Output Size	61 / 121 / 241
Configuration Assembly Instance	102
Configuration Size	0



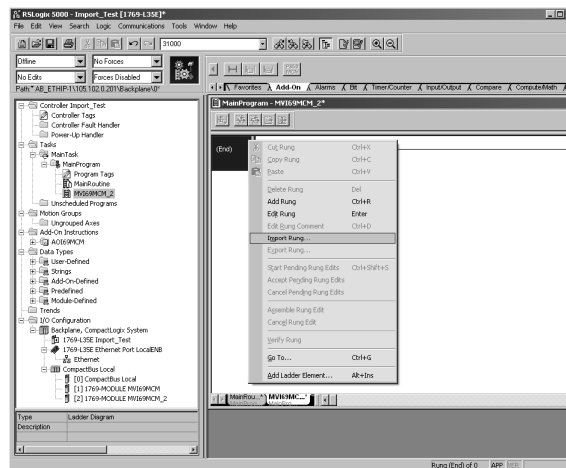


4 Click **OK** to confirm. The new module is now visible:

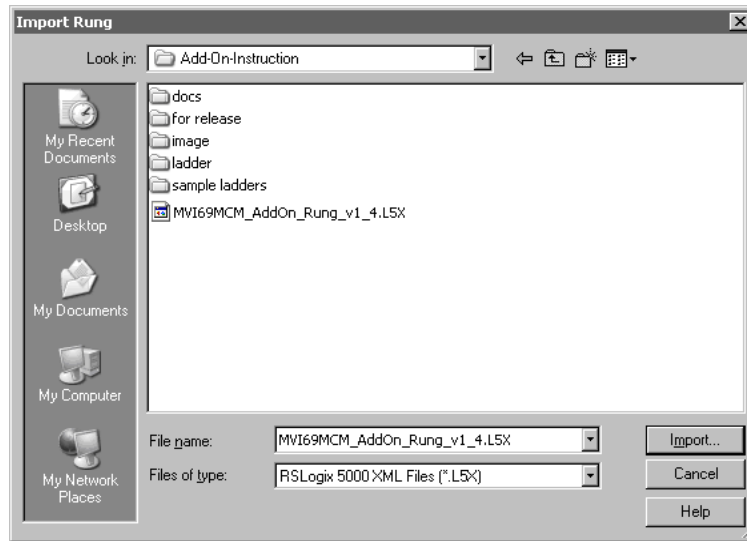


- 5 Expand the **TASKS** folder, and then expand the **MAINTASK** folder.
- 6 On the **MAINPROGRAM** folder, click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW ROUTINE**.
- 7 In the New Routine dialog box, enter the name and description of your routine, and then click **OK**.
- 8 Select an empty rung in the new routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **"IMPORT RUNG..."**.

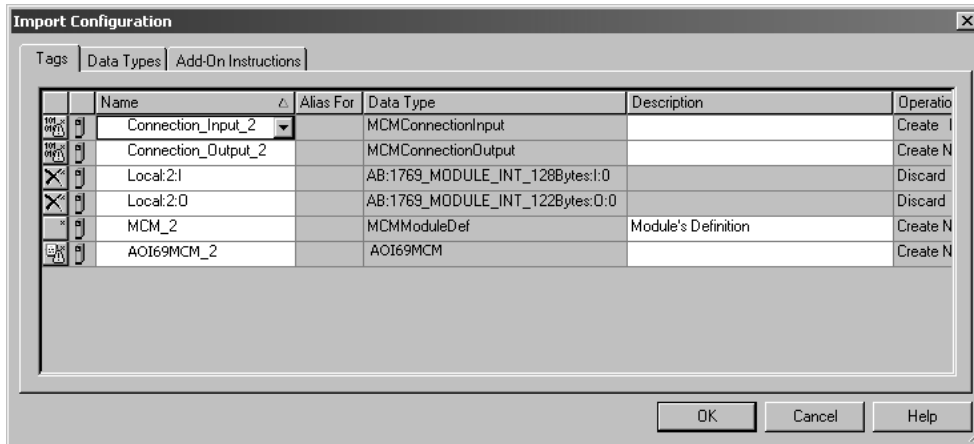
**Note:** It is not necessary to create a completely new routine. It is possible to add the MVI69-MCM\_2 module in the previously created routine. If it is desired to create a new routine the user needs to also create a rung with a jump instruction in the previous routine to the new routine.



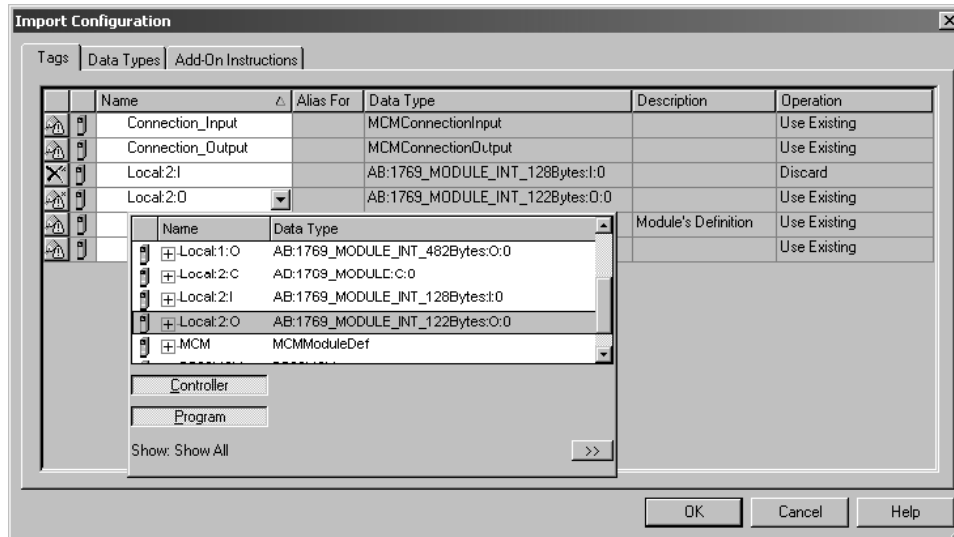
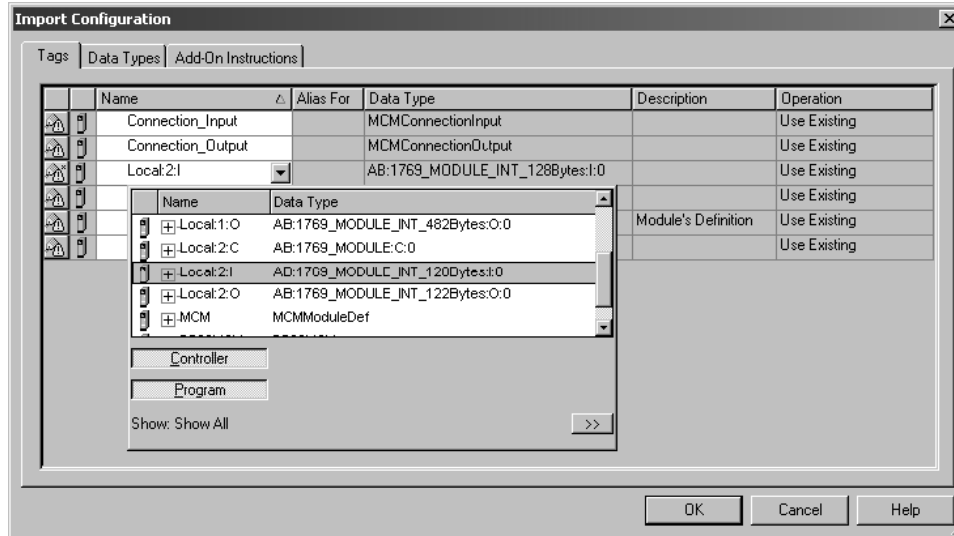
9 Select the file **MVI69MCM\_ADDON\_RUNG\_v1\_4.L5X**



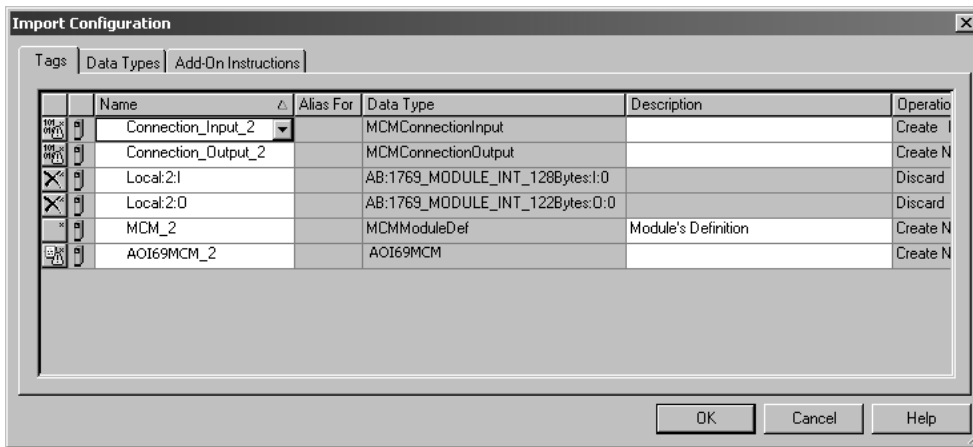
10 The following window will be displayed showing the tags to be imported:



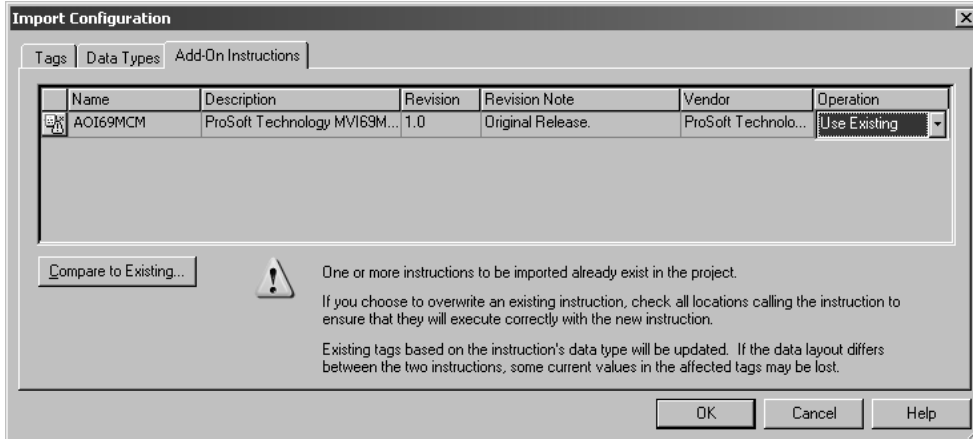
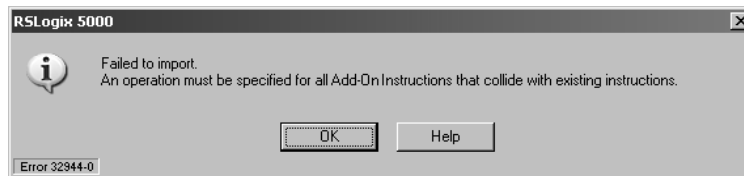
- Associate the I/O connection variables to the correct module. The default values are **LOCAL:1:I** and **LOCAL:1:O**. These require re-assignment to the new module's location.



12 Change the default tags **MCM** and **AOI69MCM** to avoid conflict with existing tags. This example procedure will append the string "\_2" as follows:



13 You will be prompted to confirm your change. Click **OK** to continue.



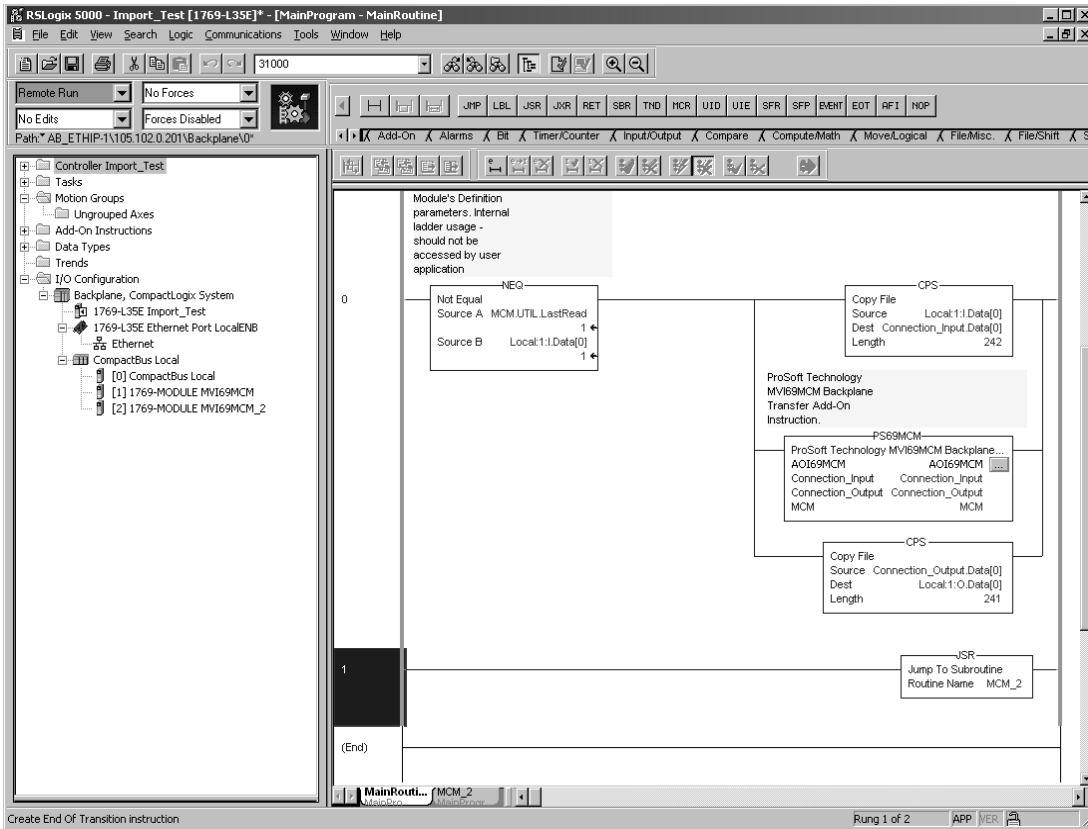
14 Click OK to confirm.

The screenshot displays the RSLogix 5000 software interface. On the left, the project tree shows the hierarchy: Controller Import\_Test > MainProgram > MainRoutine > MCM\_2. The central area shows a ladder logic diagram with a 'Not Equal' instruction. Source A is 'MCM\_2.UTIL.LastRead' and Source B is 'Local:2:1.Data[0]'. To the right, there are 'Copy File' instructions for 'Local:2:1.Data[0]' and 'Local:2:0.Data[0]'. The console window at the bottom shows the following text:

```

Importing C:\Work Files\TESTING\Products\inRax\MVI69\MCM\Add-On-Instruction\AOI69MCM.L5X...
Identical Data Type: Data Type "MCMEventCmd" not imported (existing Data Type used)
Warning: Name collision: Data Type "MCMConnectionInput" not imported (discarded)
Identical Data Type: Data Type "MCMSTATUS" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMPassThru" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMPortSlaves" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMModuleDef" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMDATA" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMBlockStats" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMUTIL" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMCoilArray" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMPortsStats" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMSlavePolling" not imported (existing Data Type used)
Identical Data Type: Data Type "MCMCmdControl" not imported (existing Data Type used)
Warning: Name collision: Data Type "MCMConnectionOutput" not imported (discarded)
Identical Data Type: Data Type "MCMCONFIG" not imported (existing Data Type used)
Warning: Name collision: Add-On Instruction AOI69MCM not imported (existing Add-On Instruction used)
Creating tag: Connection_Output_2
Creating tag: MCM_2
Creating tag: AOI69MCM_2
Creating tag: Connection_Input_2
Complete - 0 error(s), 3 warning(s)
    
```

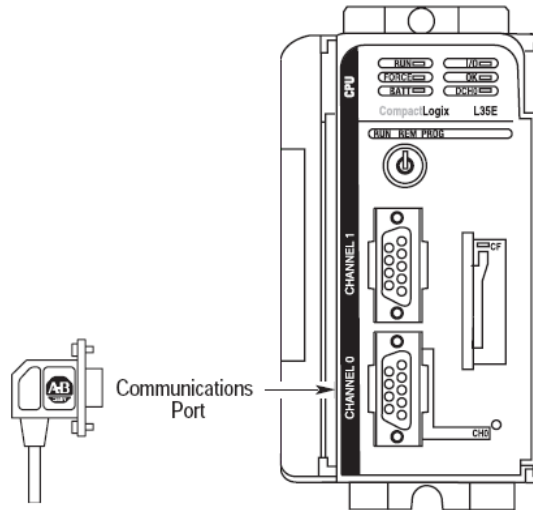
- 15 Because the second module's logic was created in a new routine, enter a rung in the Main routine with a **JSR** instruction to the new routine to enable the PLC logic to communicate with both modules.



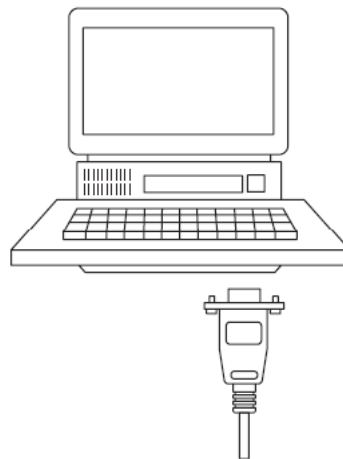
The setup procedure is now complete. Save the project and download the application to your CompactLogix processor.

### 2.1.8 Connecting Your PC to the Processor

- 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.



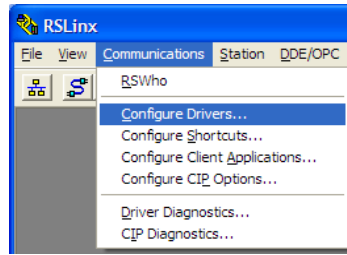
### 2.1.9 Download the Sample Program to the Processor

#### Configuring the RSLinx Driver for the PC COM Port

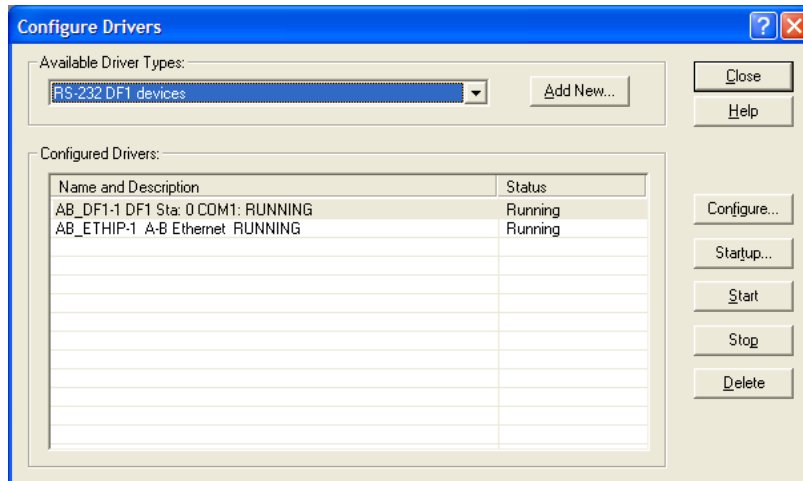
If RSLogix is unable to establish communication with the processor, follow these steps.

- 1 Open *RSLinx*.

- 2 Open the **COMMUNICATIONS** menu, and choose **CONFIGURE DRIVERS**.



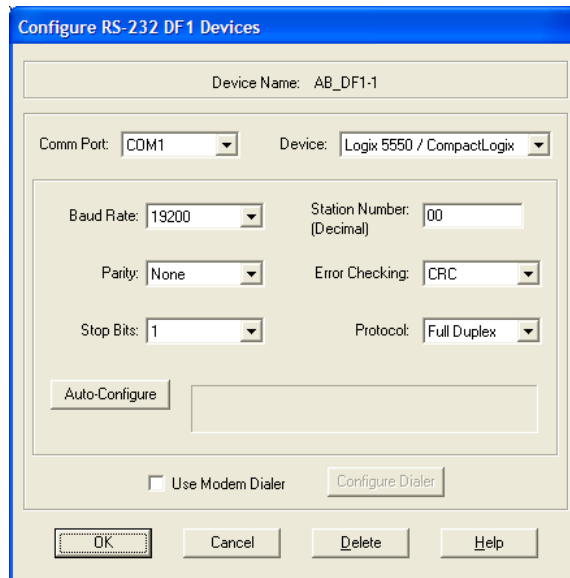
This action opens the *Configure Drivers* dialog box.



**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*.



- 1 Click to select the driver, and then click **CONFIGURE**. This action opens the *Configure RS-232 DF1 Devices* dialog box.



- 2 Click the **AUTO-CONFIGURE** button. *RSLogix* will attempt to configure your serial port to work with the selected driver.
- 3 When you see the message *Auto Configuration Successful*, click the **OK** button to dismiss the dialog box.

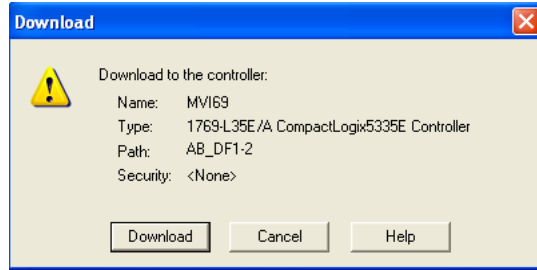
**Note:** If the auto-configuration procedure fails, verify that the cables are connected correctly between the processor and the serial port on your computer, and then try again. If you are still unable to auto-configure the port, refer to your *RSLogix* documentation for further troubleshooting steps.

### Downloading to the Processor

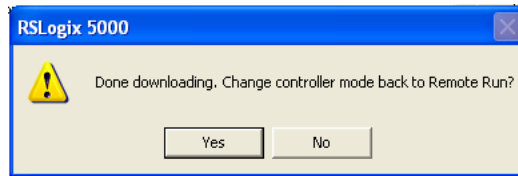
**Note:** The key switch on the front of the CompactLogix processor must be in the **REM OR PROG** position.

- 1 If you are not already online to the processor, open the **COMMUNICATIONS** menu, and then choose **DOWNLOAD**. *RSLogix* will establish communication with the processor.

- 2 When communication is established, RSLogix will open a confirmation dialog box. Click the **DOWNLOAD** button to transfer the sample program to the processor.



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

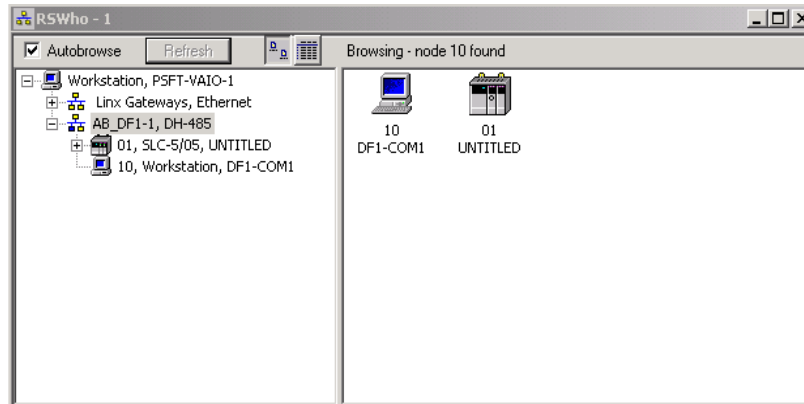


**Note:** If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

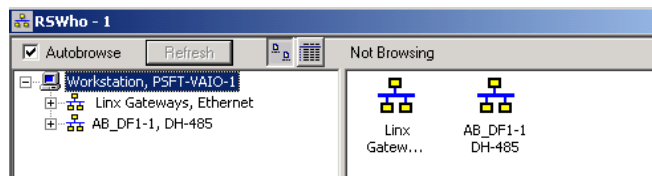
Disabling the RSLinx Driver for the Com Port on the PC

The communication port driver in *RSLinx* can occasionally prevent other applications from using the PC's COM port. If you are not able to connect to the module's configuration/debug port using *ProSoft Configuration Builder (PCB)*, *HyperTerminal* or another terminal emulator, follow these steps to disable the *RSLinx* Driver.

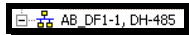
- 1 Open *RSLinx* and go to **COMMUNICATIONS>RSWHO**
- 2 Make sure that you are not actively browsing using the driver that you wish to stop. The following shows an actively browsed network:



- 3 Notice how the DF1 driver is opened, and the driver is looking for a processor on node 1. If the network is being browsed, then you will not be able to stop this driver. To stop the driver your *RSWho* screen should look like this:

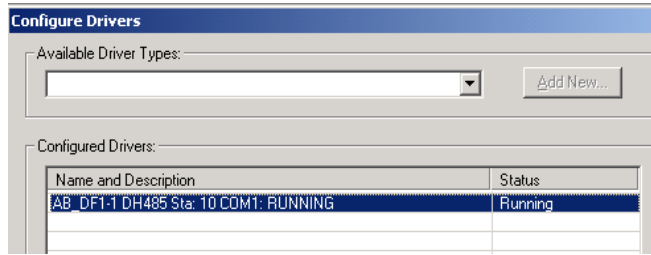


Branches are displayed or hidden by clicking on the  or the  icons.

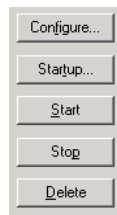


- 4 When you have verified that the driver is not being browsed, go to **COMMUNICATIONS>CONFIGURE DRIVERS**

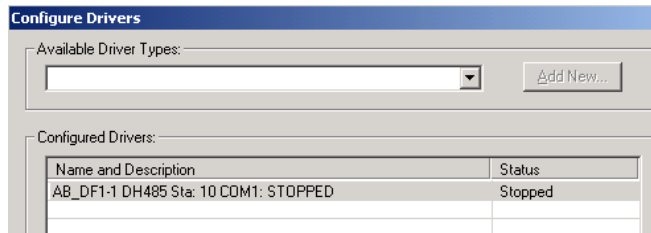
You may see something like this:



If you see the status as running, you will not be able to use this com port for anything other than communication to the processor. To stop the driver press the **STOP** button on the side of the window:



5 After you have stopped the driver you will see the following:



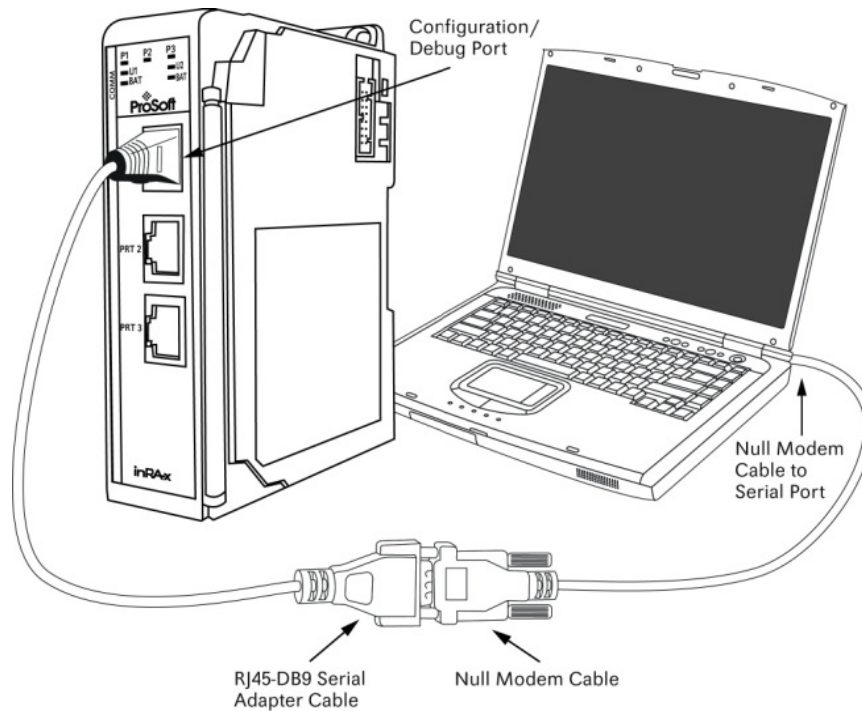
6 You may now use the com port to connect to the debug port of the module.

**Note:** You may need to shut down and restart your PC before it will allow you to stop the driver (usually only on *Windows NT* machines). If you have followed all of the above steps, and it will not stop the driver, then make sure you do not have *RSLogix* open. If *RSLogix* is not open, and you still cannot stop the driver, then reboot your PC.

### 2.1.10 Connect your PC to the Module

With the module securely mounted, connect your PC to the Configuration/Debug port using an RJ45-DB-9 Serial Adapter Cable and a Null Modem Cable.

- 1 Attach both cables as shown.
- 2 Insert the RJ45 cable connector into the Configuration/Debug port of the module.
- 3 Attach the other end to the serial port on your PC.

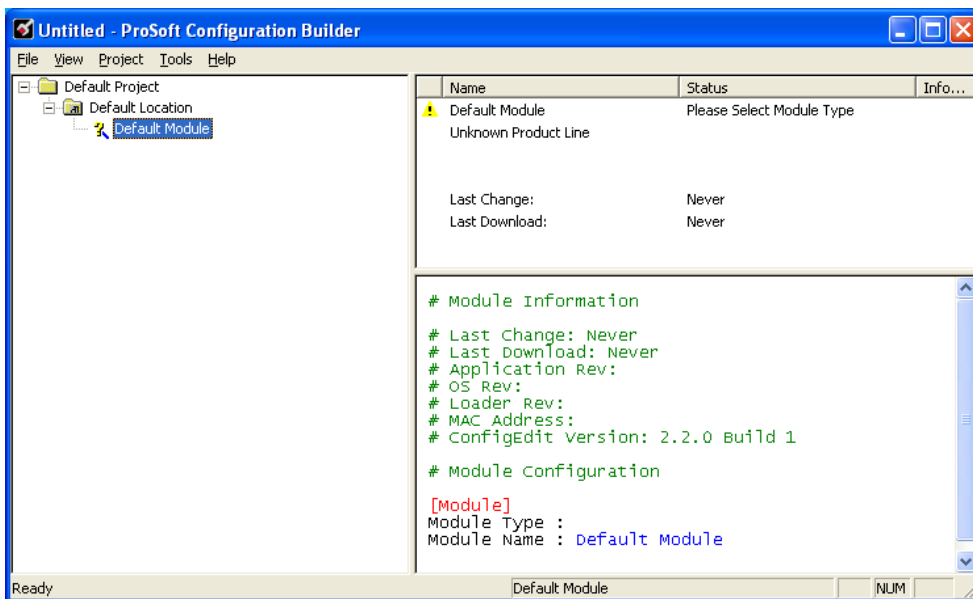


## 2.2 Using ProSoft Configuration Builder

*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.

### 2.2.1 Setting Up the Project

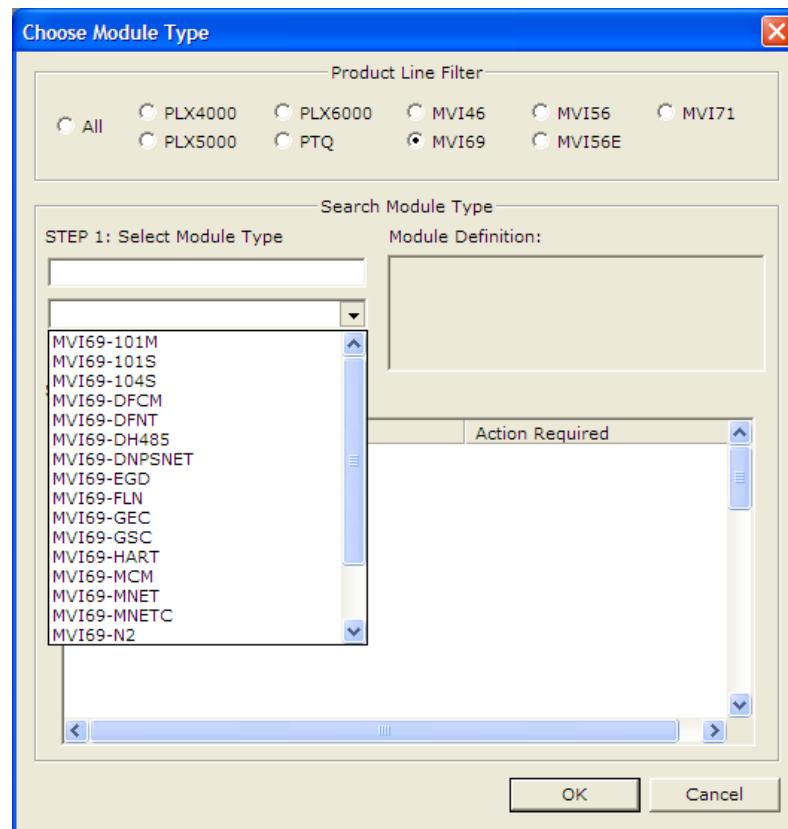
To begin, start *ProSoft Configuration Builder*. If you have used other *Windows* configuration tools before, you will find the screen layout familiar. *ProSoft Configuration Builder's* window consists of a tree view on the left, an information pane and a configuration pane on the right side of the window. When you first start *ProSoft Configuration Builder*, 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 *ProSoft Configuration Builder* window with a new project.



Your first task is to add the MVI69-MCM 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.

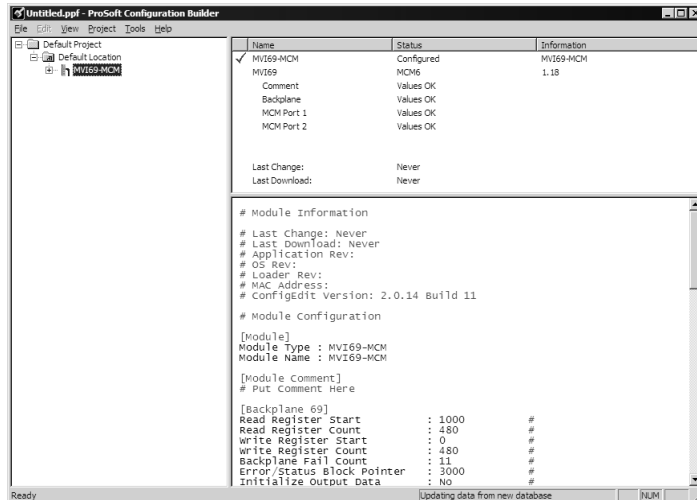
- 2 On the shortcut menu, select **CHOOSE MODULE TYPE**. This action opens the *Choose Module Type* dialog box.



- 3 In the *Product Line Filter* area of the dialog box, select **MVI69**. In the *Select Module Type* dropdown list, select **MVI69-MCM**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window.

## 2.2.2 Renaming PCB Objects



Notice that the contents of the information pane and the configuration pane changed when you added the module to the project.



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

- 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.

### Printing a Configuration File

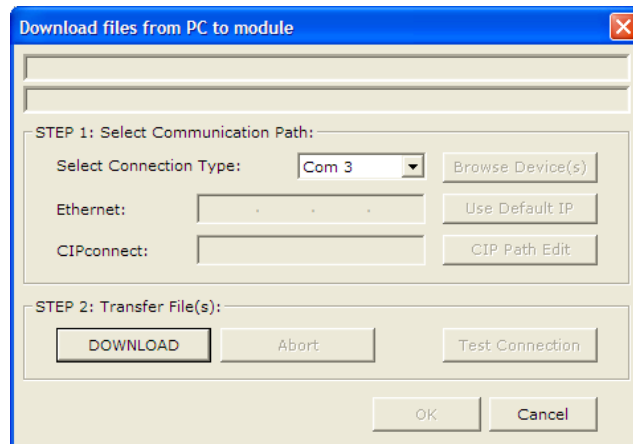
- 1 Select the module icon, and then click the right mouse button to open a shortcut menu.
- 2 On the shortcut menu, choose **VIEW CONFIGURATION**. This action opens the *View Configuration* window.
- 3 In the *View Configuration* window, open the **FILE** menu, and choose **PRINT**. This action opens 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 Downloading the Project to the Module Using a Serial COM port

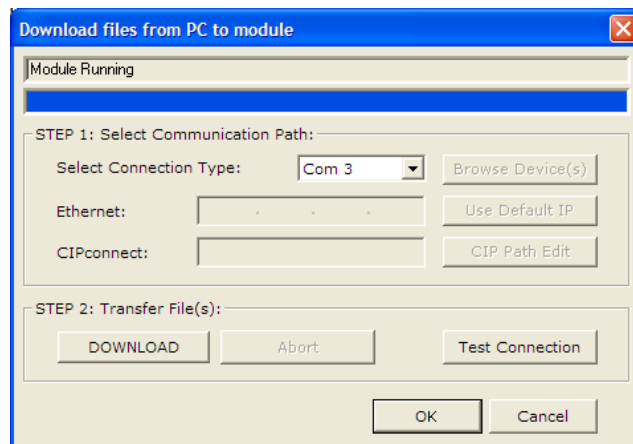
For the module to use the settings you configured, you must download (copy) the updated *Project* file from your PC to the module.

- 1 In the tree view in *ProSoft Configuration Builder*, click once to select the module.
- 2 Open the *Project* menu, and then choose **MODULE/DOWNLOAD**. The program will scan your PC for a valid com port (this may take a few seconds). When *PCB* has found a valid COM port, the *Download* dialog box will open.



- 3 Choose the COM port to use from the dropdown list, and then click the **DOWNLOAD** button.

The module will perform a platform check to read and load its new settings. When the platform check is complete, the status bar in the *Download* dialog box will display the message *Module Running*.



## 2.4 Module Configuration

### 2.4.1 [Module]

This section defines the configuration for the Module level data.

#### Module Name

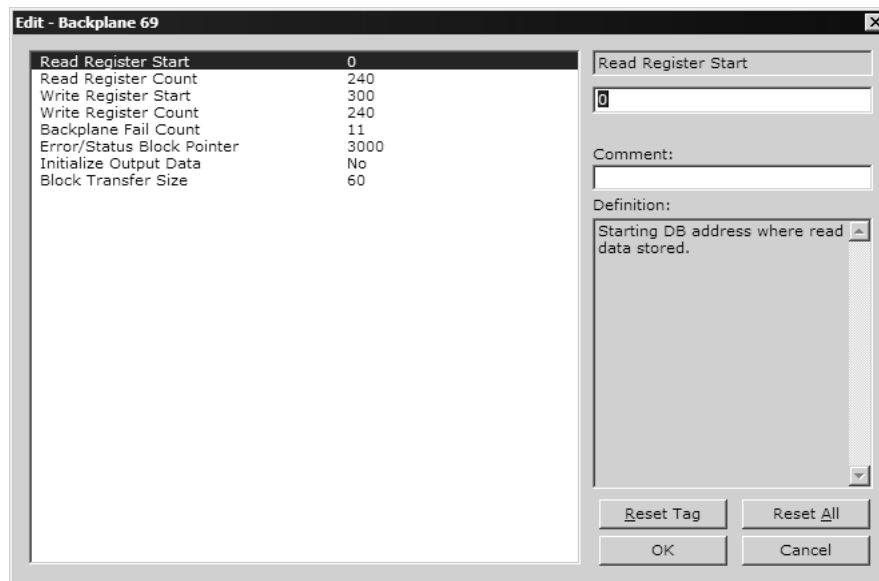
0 to 80 characters

This parameter assigns a name to the module that can be viewed using the configuration/debug port. Use this parameter to identify the module and the configuration file.

### 2.4.2 [Backplane 69]

This section provides the module with a unique name, identifies the method of failure for the communications for the module if the processor is not in run, and describes how to initialize the module upon startup.

The following example shows a sample [Backplane Configuration] section:



Modify each of the parameters based on the needs of your application.

### Read Register Start

**0 to 4999**

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

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

### Read Register Count

**0 to 5000**

The *Read Register Count* parameter specifies the size of the Read Data area of module memory and the number of registers to transfer from this area to the processor, up to a maximum of 5000 words.

**Note:** Total *Read Register Count* and *Write Register Count* cannot exceed 5000 total registers. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

### Write Register Start

**0 to 4999**

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

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

### Write Register Count

**0 to 5000**

The *Write Register Count* parameter specifies the size of the Write Data area of module memory and the number of registers to transfer from the processor to this memory area, up to a maximum value of 5000 words.

**Note:** Total *Read Register Count* and *Write Register Count* cannot exceed 5000 total registers. Read Data and Write Data Areas must be configured to occupy separate address ranges in module memory and should not be allowed to overlap.

### Backplane Fail Count

**0 to 65535**

This parameter specifies the number of consecutive backplane transfer failures that can occur before communications should be halted.

### Error/Status Block Pointer

**-1 to 4939**

Starting register location in virtual Modbus database for the error/status table. If a value of -1 is entered, the error/status data will not be placed in the database. All other valid values determine the starting location of the data. This data area includes the module version information and all server error/status data. Refer to Status Data Definition for more information.

### Initializing Output Data

**YES or NO**

This parameter determines if the output data for the module should be initialized with values from the processor. If the value is set to **NO** (0), the output data will be initialized to 0. If the value is set to **YES** (1), the data will be initialized with data from the processor. Use of this option requires associated ladder logic to pass the data from the processor to the module.

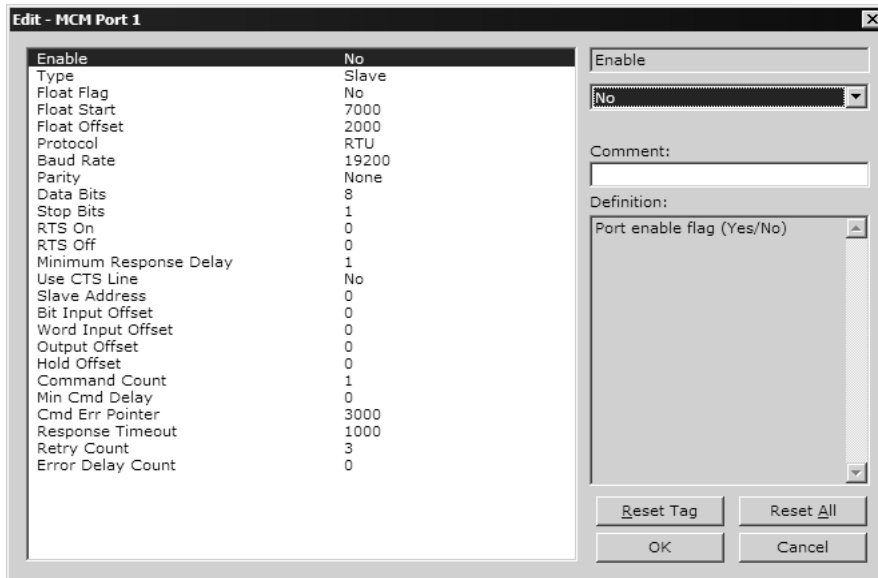
### Block Transfer Size

**60, 120 or 240**

This read-only parameter specifies the number of words in each block transferred between the module and processor.

### 2.4.3 [MCM Port x]

The information in this section applies to both Port 1 and Port 2.



#### Enable

Yes or No

This parameter specifies whether to enable or disable the port. No = Port Disabled, Yes = Port Enabled.

#### Type

0=Master, 1=Slave, 2=PT Formatted, 3=PT Formatted Swap

This parameter specifies which device type the port will emulate. Refer to Pass-Through Control Blocks (page 129) for information on using port types 2 or 3.

### Float Flag

#### **YES or NO**

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

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

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

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

### Float Start

0 to 32767

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

### Float Offset

0 to 4999

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

### Protocol

RTU or ASCII

This parameter specifies the Modbus protocol to be used on the port. Valid protocols are: *rtu* = Modbus RTU and *ascii* = Modbus ASCII.

**Baud Rate**

This is the baud rate to be used on the port. Enter the baud rate as a value. For example, to select **19K** baud, enter **19200**.

<b>Baud Rate</b>	<b>Parameter Value Options</b>
110	110
150	150
300	300
600	600
1200	12 or 1200
2400	24 or 2400
4800	48 or 4800
9600	96 or 9600
19,200	19, 192 or 19200
38,400	38, 384 or 38400
57,600	57 or 576
115,200	115 or 1152

**Parity**

*None, Odd, Even*

Parity is a simple error checking algorithm used in serial communication. This parameter specifies the type of parity checking to use.

All devices communicating through this port must use the same parity setting.

**Data Bits**

**7 or 8**

This parameter sets the number of data bits for each word used by the protocol. All devices communicating through this port must use the same number of data bits.

**Stop Bits**

**1 or 2**

Stop bits signal the end of a character in the data stream. For most applications, use one stop bit. For slower devices that require more time to re-synchronize, use two stop bits.

All devices communicating through this port must use the same number of stop bits.

**RTS On**

**0 to 65535** milliseconds

This parameter sets the number of milliseconds to delay after *Ready To Send* (RTS) is asserted before data will be transmitted.

RTS Off

**0** to **65535** milliseconds

This parameter sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal will be set low.

Minimum Response Delay

0 to 65535

This parameter is used only when the port is configured as a slave. It sets the number of milliseconds to wait before responding to a command received on the port from a remote Master. This delay is sometimes required to accommodate slower Master devices.

Use CTS Line

**YES** or **NO**

This parameter specifies if the Clear To Send (CTS) modem control line is to be used or not. If the parameter is set to **No**, the CTS line will not be monitored. If the parameter is set to **YES**, the CTS line will be monitored and must be high before the module will send data. Normally, this parameter is required when half-duplex modems are used for communication (2-wire). This procedure is commonly referred to as *hardware handshaking*.

Slave Address

1 to 255

This parameter defines the Slave Node Address for the internal database. All requests received by the port with this address are processed by the module. Verify that each device has a unique address on a network. Valid range for this parameter is 1 to 255.

Bit Input Offset

0 to 4999

This parameter specifies the offset address into the internal Modbus database for network requests for Modbus function 2 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.

Word Input Offset

0 to 4999

This parameter specifies the offset address into the internal Modbus database for network requests for Modbus function 4 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.



Output Offset

0 to 4999

This parameter specifies the offset address into the internal Modbus database for network requests for Modbus function 1, 5 or 15 commands. For example, if the value is set to 100, an address request of 0 will correspond to register 100 in the database.

Hold Offset

0 to 4999

This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 3, 6, or 16 commands. For example, if a value of 50 is entered, a request for address 0 will correspond to the register 50 in the database.

Command Count

0 to 100

This parameter specifies the number of commands to be processed by the Modbus Master port.

Minimum Command Delay

0 to 65535

This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries.

Command Error Pointer

-1 to 4899

This parameter sets the address in the internal Modbus database where the command error will be placed. If the value is set to -1, the data will not be transferred to the database. The valid range of values for this parameter is -1 to 4899. For example, if this parameter is configured for 1000, the command errors will be copied to the database as follows:

1000: error code for command 0

1001: error code for command 1

and so on.

An error code of 0 means that the command was successfully sent (no error). Refer to Status Data Definition for the command error code listings.

### Response Timeout

**0 TO 65535** milliseconds

This parameter sets the command response timeout period in 1 millisecond increments. This is the time that a port configured as a Master will wait for a response from the addressed slave before re-transmitting the command (Retries) or skipping to the next command in the Command List. The value to set depends on the communication network used and the expected response time (plus a little extra) of the slowest device on the network.

### Retry Count

**0 to 10**

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

### Error Delay Counter

**0 to 65535**

This parameter specifies the number of polls to skip on the slave before trying to re-establish communications. After the slave fails to respond, the Master will skip commands to be sent to the slave the number of times entered in this parameter.

```
MODBUS MASTER/SLAVE COMMUNICATION MODULE (MVI56-MCMR) MENU
?=Display Menu
A=Data Analyzer
B=Block Transfer Statistics
C=Module Configuration
D=Modbus Database View
Master Command Errors : E=Port 1 F=Port 2
Master Command List : I=Port 1 J=Port 2
Slave Status List : O=Port 1 P=Port 2
R=Receive Configuration from Remote
S=Send Configuration to Remote
V=Version Information
W=Warm Boot Module
Communication Status : 1=Port 1 2=Port 2
Port Configuration : 6=Port 1 7=Port 2

Esc=Exit Program

PORT 1 MODBUS STATUS:
Enabled : Y
Retries : 2 Cur Cmd : 0 State : 3
ComState: 0 Cur Err : 1 Last Err: 0

Number of Command Requests: 18
Number of Cmd Responses : 0
Number of Command Errors : 70
Number of Requests : 71
Number of Responses : 0
Number of Errors Received : 0
Number of Errors Sent : 0

PORT 1 CONFIGURATION:
Enabled : Y Port Type : (0) - MASTER
SLAVE SETUP:
Modbus Slave ID: 0 Pass-Through = DISABLED
Offsets:
BitIn: 0 wordIn: 0 output: 0 Holding: 0
Floating-point Data:
Flag: N Start: 0 offset: 0
Route Count : 0
Function 99 Offset : 0
Use Packet Gap : N Packet Gap Delay : 0
MASTER SETUP:
Command Count : 2 Cmd Delay: 0 Cmd Offset : 0
Response Timeout: 1000 Retries : 3 delay count: 0
COMMUNICATION PARAMETERS:
Protocol: 0 (Modbus RTU)
Baud: 4800 Parity: NONE Databits: 8 Stopbits: 1
RTS on: 0 RTS Off: 0 Use CTS Line: N
```

### 2.4.4 [Modbus Port x Commands]

The [Modbus Port x Commands] section of the configuration file defines the command list specifications for the Master port. The information in this section applies to both Port 1 and Port 2.

	Enable	Internal Address	Poll Interval	Reg Count	Swap Code	Node Address	ModBus Function	MB Address in Device	Comment
✓ 1	Yes	600	0	10	No Change	1	Read Holding Registers (4X)	0	Read from address 40001 in slave
✓ 2	Yes	610	0	10	No Change	1	Read Input Registers (3X)	10	Read from address 30011 in slave
✓ 3	Yes	9920	0	160	No Change	1	Read Coil (0X)	0	Read coil address 0001 in slave
✓ 4	Yes	10800	0	160	No Change	1	Read Input (1X)	480	Read coil address 10481 in slave
✓ 5	Conditional	0	0	1	No Change	1	Preset Single Register (4X)	620	Conditional write from address 0
✓ 6	Conditional	10	0	10	No Change	1	Preset Multiple Registers (4X)	630	Conditional write from address 1
✓ 7	Conditional	320	0	1	No Change	1	Read Input Registers (3X)	9600	Conditional write from address 2
✓ 8	Conditional	480	0	160	No Change	1	Force Multiple Coil (0X)	9760	Conditional write from address 3

Enable Value Status - OK

Buttons: Set to Defaults, Add Row, Insert Row, Delete Row, Move Up, Move Down, Edit Row, Copy Row, Paste Row, OK, Cancel

#### Command List Overview

In order to interface the MVI69-MCM module with MODBUS slave devices, you must construct a command list. The commands in the list specify the slave device to be addressed, the function to be performed (read or write), the data area in the device to interface with and the registers in the internal database to be associated with the device data. The Master command list supports up to 100 commands.

The command list is processed from top (command #0) to bottom. A poll interval parameter is associated with each command to specify a minimum delay time in number of seconds between the issuance of a command. If the user specifies a value of 10 for the parameter, the command will be executed no more frequently than every 10 seconds.

Write commands have a special feature, as they can be set to execute only if the data in the write command changes, which can improve network performance. If the register data values in the command have not changed since the command was last issued, the command will not be executed. To enable this feature, set the enable code for the command to a value of 2.

#### Modbus Command Configuration

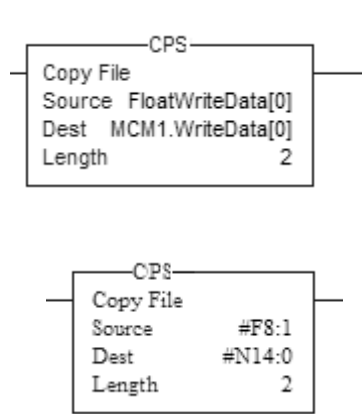
The ProSoft Technology MCM Modbus Master and Slave communication drivers support several data read and write commands. When configuring a Master port, the decision on which command to use is made depending on the type of data being addressed, and the level of Modbus support in the slave equipment. When configuring as a slave, it may be important to understand how the Modbus commands function in order to determine how to structure the application data.

### Floating Point Support

The movement of floating point data between the MCM module and other devices is easily accomplished as long as the device supports IEEE 754 Floating Point format. This IEEE format is a 32-bit single precision floating point format.

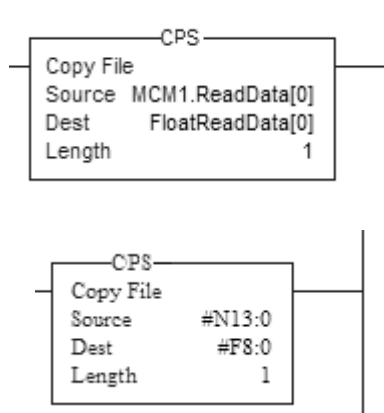
The programming necessary to move the floating point data takes advantage of the CPS command that exists in the Compact Logix and SLC processors. The CPS command is unique for CPX/SLC data movement commands in that it is an untyped function, meaning that no data conversion is done when moving data between file types (that is, it is an image copy not a value copy).

The structure of the CPS command to move data from a Floating Point file into an integer file (something you would do to move floating point values to the module) is as follows:



This command will move one floating point value in two 16 bit integer images to the integer file. For multiple floating point values increase the count field by a factor of 2 per floating point value.

The structure of the COP command to move data from an Integer file to a Floating Point file (something you would do to receive floating point values from the module) is as follows:



This command will move two 16 bit integer registers containing one floating point value image to the floating point file. For multiple values increase the count field.

ENRON Floating Point Support

Many manufacturers have implemented special support in their drivers to support what is commonly called the Enron version of the MODBUS protocol. In this implementation, register addresses > 7000 are presumed to be floating point values. The significance to this is that the count field now becomes a 'number of values' field. In floating point format, each value represents two words.

Configuring the Floating Point Data Transfer

A common question when using the module as a Modbus Master is how floating point data is handled. This really depends on the slave device and how it addresses this application.

Just because your application is reading/writing floating point data, does not mean that you must configure the Float Flag, Float Start, and Float Offset parameters within the module.

These parameters are only used to support what is typically referred to as Enron or Daniel Modbus, where one register address must have 32 bits, or one floating point value. Below is an example:

**Example #1**

Modbus Address	Data Type	Parameter
47101	32 bit REAL	TEMP Pump #1
47102	32 bit REAL	Pressure Pump #1
47103	32 bit REAL	TEMP Pump #2
47104	32 bit REAL	Pressure Pump #2

With the module configured as a Master, you only need to enable these parameters to support a write to this type of addressing (Modbus FC 6 or 16).

If the slave device shows addressing as shown in Example #2, then you need not do anything with the Float Flag, Float Start parameters, as they use two Modbus addresses to represent one floating point value:

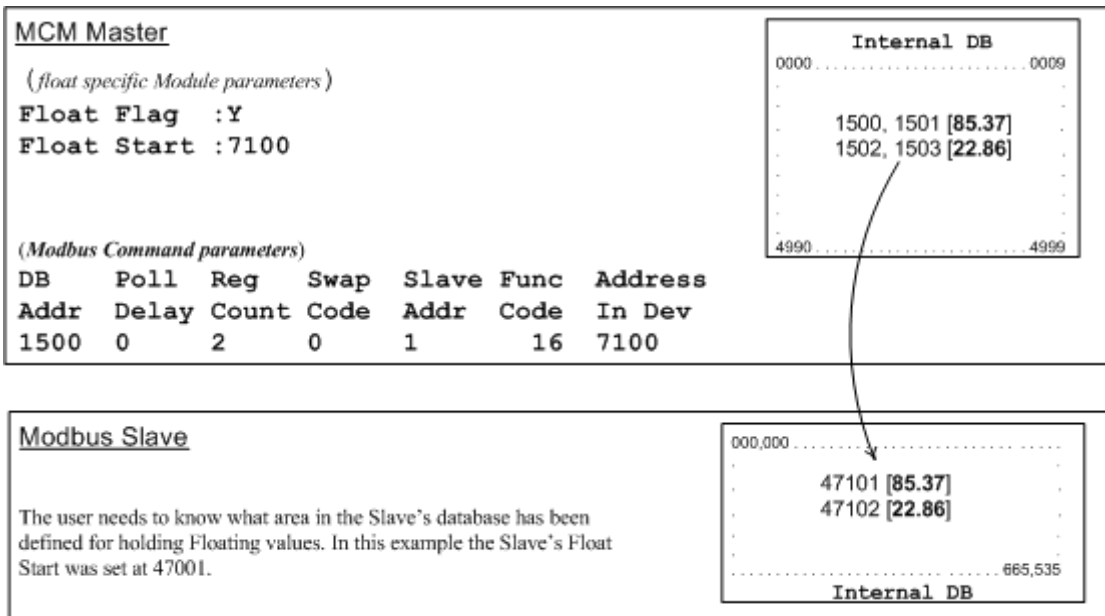
**Example #2**

Modbus Address	Data Type	Parameter
47101	32 bit REAL	TEMP Pump #1
47103	32 bit REAL	Pressure Pump #1
47105	32 bit REAL	TEMP Pump #2
47107	32 bit REAL	Pressure Pump #2

Because each 32 bit REAL value is represented by two Modbus Addresses (example 47101 and 47102 represent TEMP Pump #1), then you need not set the Float Flag, or Float Start for the module for Modbus FC 6 or 16 commands being written to the slave.

The next few pages show three specific examples:

**Specific Example #1: Master is issuing Modbus command with FC 16 (with Float Flag: Yes) to transfer Float data to Slave.**



**(Float specific module parameters)**

**Float Flag: "Y"** tells the Master to consider the data values that need to be sent to the Slave as floating point data where each data value is composed of 2 words (4 bytes or 32 bits).

**Float Start:** Tells the Master that if this address number is  $\leq$  the address number in "Addr in Dev" parameter to double the byte count quantity to be included in the Command FC6 or FC16 to be issued to the Slave. Otherwise the Master will ignore the "Float Flag: Y" and treat data as composed of 1 word, 2 bytes.

**(Modbus Command parameters)**

**DB Addr** - Tells the Master where in its data memory is the beginning of data to obtain and write out to the Slave (slave) device.

**Reg Count** - Tells the Master how many data points to send to the Slave. Two counts will mean two floating points with Float Flag: Y and the "Addr in Dev" => the "Float Start" Parameter.

**Swap Code** - Tells the Master how to orient the Byte and Word structure of the data value. This is device dependent. Check Command Entry formats Section.

**Func Code** - Tells the Master to write the float values to the Slave. FC16.

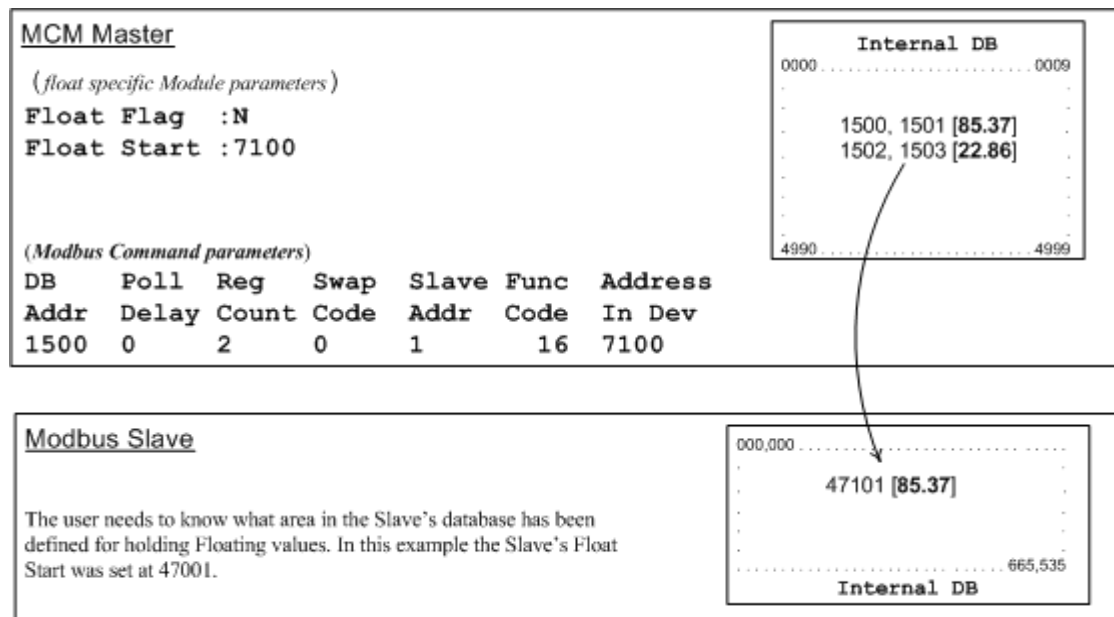
**Addr in Dev** - Tells the Master where in the Slave's database to locate the data.

In the above example, the Master's Modbus command to transmit inside the Modbus packet will be as follows.

	Slave address	Function Code	Address in Device	Reg count	Byte Count	Data
DEC	01	16	7100	2	8	85.37 22.86
HEX	01	10	1B BC	00 02	08	BD 71 42 AA E1 48 41 B6

In this example, the Master's Modbus packet contains the data byte and data word counts that have been doubled from the amount specified by Reg Count due to the Float flag set to Y. Some Slaves look for the byte count in the data packet to know the length of the data to read from the wire. Other slaves know at which byte the data begins and read from the wire the remaining bytes in the packet as the data the Master is sending.

**Specific Example#2: Master is issuing Modbus command with FC 16 (with Float Flag: No) to transfer Float data.**



**Float Flag: "N"** tells the Master to ignore the floating values and treat each register data as a data point composed of 1 word, 2 bytes or 16 bits.

**Float Start:** Ignored.

**DB Addr** - same as when Float Flag: Y.

**Reg Count** - Tells the Master how many data points to send to the Slave.

**Swap Code** - same as when Float Flag: Y.

**Func Code** - same as when Float Flag: Y.

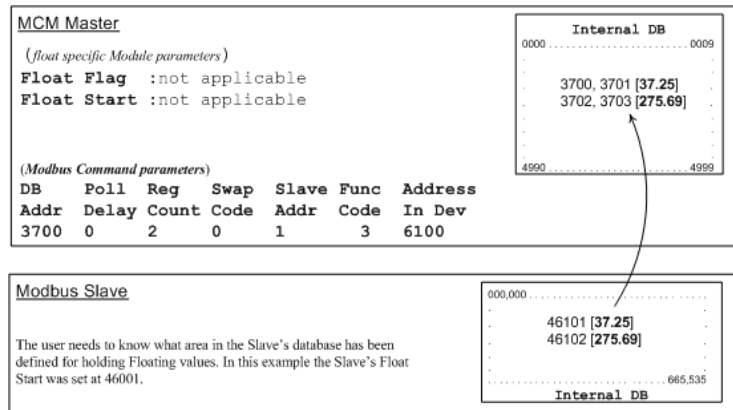
**Addr in Dev** - same as when Float Flag: Y as long as the Slave's Float Flag = Y.

In the above example, the Master's Modbus command to transmit inside the Modbus packet will be as follows.

	Slave address	Function Code	Address in Device	Reg Count	Byte Count	Data
DEC	01	16	7100	2	4	85.37
HEX	01	10	1B BC	00 02	04	BD 71 42 AA

In this example, the Master's Modbus packet contains the data byte and data word counts that have NOT been doubled from the amount specified by Reg Count due to the Float Flag set to N. The Slave looks for the byte count in the data packet to know the length of the data to read from the wire. Because of insufficient byte count, some slaves will read only half the data from the Master's transmission. Other slaves will read all 8 bytes in this example because they will know where in the packet the data starts and ignore the byte count parameter inside the Modbus packet.

**Specific Example#3: Master is issuing Modbus command with FC 3 to transfer Float data from Slave.**



**Float Flag:** Not applicable with Modbus Function Code 3.

**Float Start:** Not applicable with Modbus Function Code 3.

**DB Addr** - Tells the Master where in its data memory to store the data obtained from the Slave.

**Reg Count** - Tells the Master how many registers to request from the Slave.

**Swap Code** - same as above.



**Func Code** - Tells the Master to read the register values from the Slave. FC3.

**Addr in Dev** - Tells the Master where in the Slave's database to obtain the data.

In the above example, the Master's Modbus command to transmit inside the Modbus packet will be as follows.

	Slave address	Function Code	Address in Device	Reg count
DEC	01	3	6100	2
HEX	01	03	17 D4	00 02

In the above example the (Enron/Daniel supporting) Slave's Modbus command to transmit inside the Modbus packet will be as follows.

	Slave address	Function Code	Byte Count	Data
DEC	01	3	8	32.75 275.69
HEX	01	03	08	00 00 42 03 D8 52 43 89

In the above example the (a NON-Enron/Daniel supporting) Slave's Modbus command that will be transmitted inside the Modbus packet will be as follows.

	Slave address	Function Code	Byte Count	Data
DEC	01	3	4	32.75
HEX	01	03	04	00 00 42 03

Enable

0 to 4

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

If the parameter is set to 0, the command is disabled and will not be executed in the normal polling sequence.

Setting the parameter to a value of 1 for the command causes the command to be executed each scan of the command list if the Poll Interval Time is set to zero. If the Poll Interval time is set, the command will be executed, when the interval timer expires.

If the parameter is set to 2, the command will execute only if the internal data associated with the command changes. This value is valid only for write commands.

A value of 3 can be used to enable a write command continuously (similar to enable code of 1), but if the float flag and float start parameters are being used, the module will only send out a standard 16 bit write. Only valid for FC 6, and 16 when the MB Address in Device >= Float Start Parameter and Float Flag is set to Yes.

A value of 4 will provide the same disabling of floating point writes as the enable code of 3, but utilizing conditional writes (similar to enable code 2 functionality).

**Important:** Not all revisions of the MCM driver support this parameter. To determine if your module supports this parameter, contact ProSoft Technical Support.

Internal Address

0 to 4999 for Modbus Function 3, 4, 6, or 16

or

0 to 65535 for Modbus Function Code 1, 2, 5, or 15.

This field specifies the internal database register to be associated with the command.

If the command is a read function, the data read from the slave device will be placed starting at the register value entered in this field.

If the command is a write function, the data written to the slave device will be sourced from the address specified.

Poll Interval

**0 TO 65535**

This parameter specifies the minimum interval between executions of a continuous commands (*Enable* code of 1). The value is in seconds. Therefore, if a value of 10 is entered, the command will execute no more frequently than once every 10 seconds.

Reg Count

**Regs**

1 to 125

**Coils**

1 to 2000

The Reg Count parameter specifies the number of registers or digital points to be associated with the command. Functions 5 and 6 ignore this field as they only apply to a single data point.

For functions 1, 2 and 15, this parameter sets the number of digital points (inputs or coils) to be associated with the command.

For functions 3, 4 and 16, this parameter sets the number of registers to be associated with the command.

Swap Code

0, 1, 2, 3

This parameter defines if the data received from the Modbus slave is to be ordered differently than received from the slave device. This parameter is helpful when dealing with floating-point or other multi-register values, as there is no standard method of storage of these data types in slave devices. This parameter can be set to order the register data received in an order useful by other applications. The following table defines the values and their associated operations:

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 Address

1 to 255

(0 = broadcast)

This parameter specifies the Modbus slave node address on the network to be considered. Values of 1 to 255 are permitted. Most Modbus devices only accept an address in the range of 1 to 247 so be careful. If the value is set to zero, the command will be a broadcast message on the network. The Modbus protocol permits broadcast commands for write operations. Do not use this node address for read operations.

Modbus Func

1, 2, 3, 4, 5, 6, 15, 16

This parameter specifies the Modbus function to be executed by the command. These function codes are defined in the Modbus protocol. The following table defines the purpose of each function supported by the module.

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

*MB Address in Device*

This parameter specifies the starting Modbus register or digital point address to be considered by the command in the Modbus slave device. Refer to the documentation of each Modbus slave device on the network for their register and digital point address assignments. For example, in an Omni Flow Computer, points 1901 to 1936 are digital points used for alarm flags. These points can be read using Modbus Function Code 1 and placed in the module's internal database.

## 3 Ladder Logic

### In This Chapter

- ❖ Ladder Logic and Firmware Compatibility Note ..... 70
- ❖ Module Data Object (MCM1ModuleDef)..... 71
- ❖ Adding the Module to an Existing CompactLogix Project ..... 75
- ❖ Adding the Module to an Existing MicroLogix Project..... 79

Ladder logic is required for application of the MVI69-MCM 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 CD-ROM*, is extensively commented, to provide information on the purpose and function of each rung. For most applications, the sample ladder will work without modification.

### 3.1 Ladder Logic and Firmware Compatibility Note

**Important:** MVI69-MCM modules with firmware version 1.21 and newer cannot use ladder logic written for earlier firmware versions. Please use the ladder logic or Add-On Instruction specifically labeled for your MVI69-MCM module's firmware version. Firmware version 1.21 includes the following changes to the pass-through control blocks:

- Mutual exclusion on Pass-Through Block IDs 9956, 9957, 9958, and 9959 from both ports - If both ports are configured as slave ports, when both of the slave ports receive write commands with the same Function Code, which would need to use the same block identifier from the above list, the module will process the command from the port which first received the command and will return an Exception Code error code 6 (node is busy - retry command later error) from the other port that received the command last. The Master will retry the command on the busy port after a short delay. This prevents Pass-Through blocks on both ports from overwriting each other.
- The Pass-Through Block ID is now written by the module into the first word, the (0) offset, of the processor's backplane input image. Previously this location contained a 0 (zero) value. Ladder logic for earlier firmware versions will not work with MVI69-MCM firmware version 1.21 or later.

### 3.2 Module Data Object (MCM1ModuleDef)

All data related to the MVI69-MCM 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
BlockTransferSize	INT	
ReadData	INT[480]	Data read from module
WriteData	INT[480]	Data written to module
BP	MCM1Backplane	
ModuleStatus	MCM1_STATUS	
BlockRequest	MCM1BlockRequest	
CommandControl	MCM1CommandControlPorts	
EventCommand	MCM1EventCommandPorts	
SlavePollingControl	MCM1SlavePollingControlPorts	
SlaveStatus	MCM1SlaveStatusPorts	
MBCoil	MCM1Coil_Array	

This object contains objects that define the configuration, user data, status and command control data related to the module.

This object reads and write data between the module and the processor. Values entered determine the ladder logic and data size required in the application. The ReadData and WriteData arrays must be sized to or larger than the count values entered. The ladder logic must process the number of blocks of data to be transferred. The number of blocks is computed as follows:

$$\text{BlockCnt} = \text{INT}(\text{RegCnt}/n) + \text{if}(\text{MOD}(\text{RegCnt}, n), 1, 0)$$

Where  $n$  is the block transfer size, and equals 60, 120 or 240.

If the register count is evenly divisible by  $n$ , the number of blocks is easy to compute and the ladder is much simpler to write. If the number is not evenly divisible by  $n$ , special handling of the last block of data must be developed, as it must transfer less than  $n$  words.

**Important:** It is recommended that the count values always be set to values evenly divisible by  $n$ .

The Backplane Fail Count (page 52) parameter determines if the module should continue communicating on the MODBUS network when the backplane transfer operation fails. A value of zero indicates that the module should continue communicating when the backplane is not operational. If the value is greater than zero, the backplane will be retried the entered number of times before a failure will be reported and communication will cease on the ports. When backplane communication is restored, the module will start communicating on the network. For example, if you enter a value of 10 for the parameter, the module will stop all MODBUS communications if 10 successive backplane errors are recognized. When a successful transfer is recognized, the module will resume communications on the network.

The Error/Status Block Pointer (page 52) parameter defines the location in the module's database where the error/status data will be stored. If the value is set to -1, the data will not be stored in the user data area. A value between 0 and 4939 will cause the module's program to store the data at the specified location.

### 3.2.1 Status Object (MCM1Status)

This object views the status of the module. The **MCM1Status** 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
Pass_Cnt	INT	
Prod	SINT[4]	
Rev	SINT[4]	
Op	SINT[4]	
Run	SINT[4]	
PortErr	MCM1_PORT_ERROR[2]	
BlkStats	MCM1_BLK_STATS	
Port1_CurErr	INT	
Port1_LastErr	INT	
Port2_CurErr	INT	
Port2_LastErr	INT	

Refer to Status Data for a complete listing of the data stored in this object.



### 3.2.2 User Data Objects

These objects hold data to be transferred between the processor and the MVI69-MCM module. The user data is the read and write data transferred between the processor and the module as "pages" of data up to 240 words long.

Name	Data Type	Description
ReadData	INT[480]	Data read from module
WriteData	INT[480]	Data written to module

The read data (**READDATA**) is an array set to match the value entered in the **READREGCNT** parameter of the **MCM1MODULEDEF** object. For ease of use, this array should be dimensioned as an even increment of  $n$  words, where  $n = 60, 120$  or  $240$  words. This data is paged up to  $n$  words at a time from the module to the processor. The ReadData task places the data received into the proper position in the read data array. Use this data for status and control in the ladder logic of the processor.

The write data (**WRITEDATA**) is an array set to match the value entered in the **WRITEREGCNT** parameter of the **MCM1MODULEDEF** object. For ease of use, this array should be dimensioned as even increments of  $n$  words. This data is paged up to  $n$  words at a time from the processor to the module. The WriteData task places the write data into the output image for transfer to the module. This data is passed from the processor to the module for status and control information for use in other nodes on the network. If this array is  $> 480$  registers, change the high LIM value in ReadData rung 1 and WriteData rung 21 of the ladder file.

### 3.2.3 Slave Polling Control and Status

Two arrays are allocated in the module's primary object to hold the polling status of each slave on the Master ports. This status data can be used to determine which slaves are currently active on the port, are in communication error or have their polling suspended and disabled. Ladder logic in the processor can be written to monitor and control the status of each slave on a Master port. The following table describes the structure of this object.

Name	Data Type	Description
SlaveStatus	Decimal	Slaves Status

Using special blocks, the processor can request the current data for the slaves. Through the use of other blocks, the processor can enable or disable the polling of selected slaves.

### **3.2.4 MODBUS Message Data**

This version of the module's program includes formatted pass-through (page 129) mode. In this mode, write messages sent to a slave port are passed directly through to the processor. It is the responsibility of the ladder logic to process the message received using this feature. Two data objects are required for this mode: a variable to hold the length of the message and a buffer to hold the message. This information is passed from the module to the processor using block identification codes of 9956 through 9959, depending on the Modbus function code. Word two of this block contains the length of the message and the message starts at word 3. Other controller tags are required to store the controlled values contained in these messages. The MODBUS protocol supports controller of binary output (coils - functions 5 and 15) and registers (functions 6 and 16).

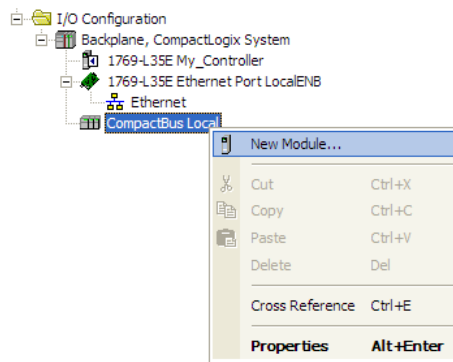
### 3.3 Adding the Module to an Existing CompactLogix Project

**Important:** The following steps describe how to install and configure the MVI69-MCM 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 (page 20).

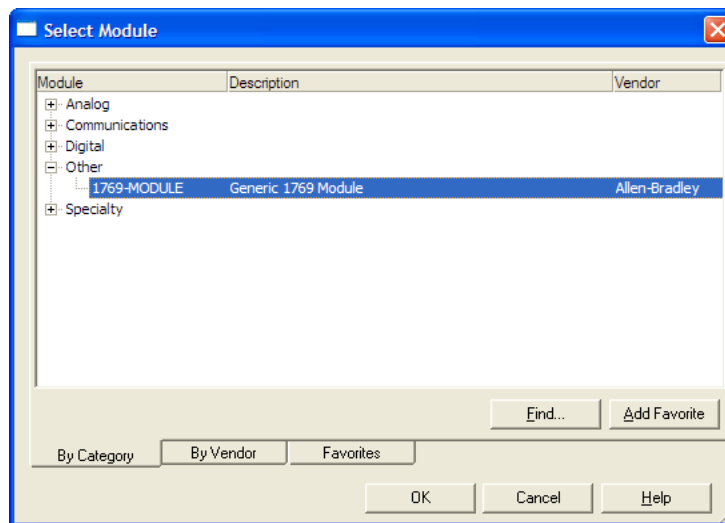
**Important:** The MVI69-MCM module has a power supply distance rating of 2 (L43 and L45 installations on first 2 slots of 1769 bus)

If you are installing and configuring the module with a CompactLogix processor, follow these steps. If you are using a MicroLogix processor, refer to the next section.

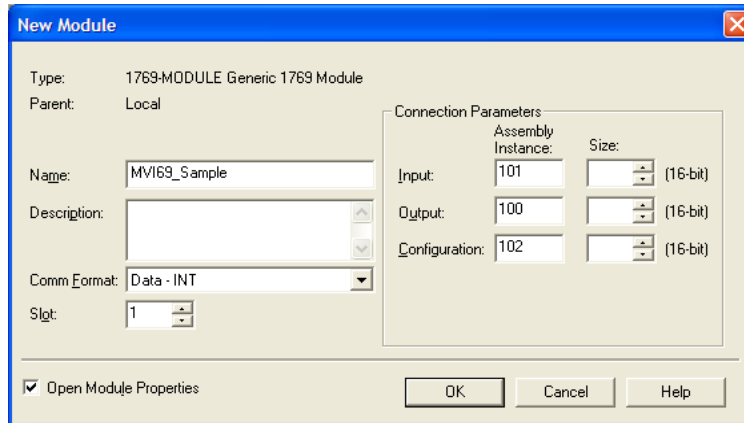
- 1 **Add the MVI69-MCM module to the project.** Right-click the mouse button on the **I/O CONFIGURATION** option in the **CONTROLLER ORGANIZATION** window to display a pop-up menu. Select the **NEW MODULE** option from the **I/O CONFIGURATION** menu.



This action opens the following dialog box:



- 2 Select the **1769-MODULE (GENERIC 1769 MODULE)** from the list and click OK.



- 3 Enter the Name, Description and Slot options for your application, using the values in the illustration above. You must select the **COMM FORMAT** as **DATA - INT** in the dialog box, otherwise the module will not communicate over the backplane of the CompactLogix rack.
- 4 Configure the Connection Parameters to match to the Block Transfer Size parameter in the configuration file. Use the values in the table corresponding with the block transfer size you configured.

**Block Transfer Size = 60**

Field	Recommended Value
Type	1769-MODULE Generic 1769 Module
Parent	Local
Name	MVI69
Description	MVI69 Application Module
Comm Format	Data - INT
Slot	The slot number in the rack where the module is installed
Input Assembly Instance	101
Input Size	62
Output Assembly Instance	100
Output Size	61
Configuration Assembly Instance	102
Configuration Size	0

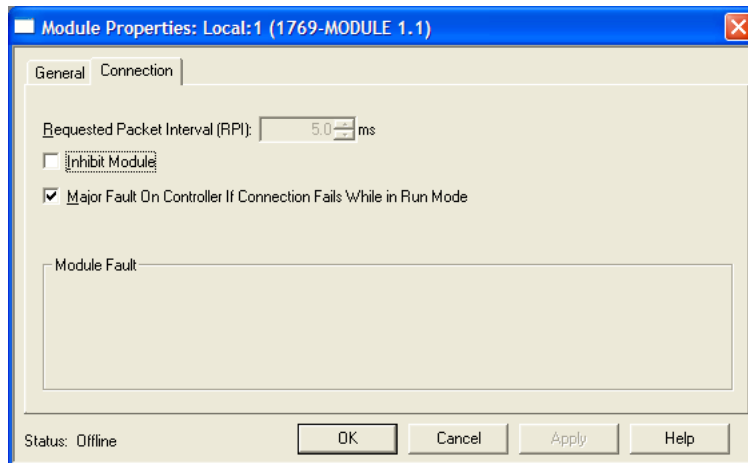
**Block Transfer Size = 120**

Field	Recommended Value
Type	1769-MODULE Generic 1769 Module
Parent	Local
Name	MVI69
Description	MVI69 Application Module

<b>Block Transfer Size = 120</b>	
Comm Format	Data - INT
Slot	The slot number in the rack where the module is installed
Input Assembly Instance	101
Input Size	122
Output Assembly Instance	100
Output Size	121
Configuration Assembly Instance	102
Configuration Size	0

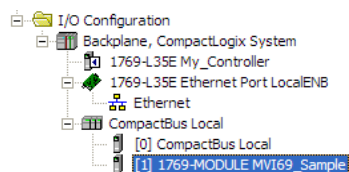
<b>Block Transfer Size = 240</b>	
<b>Field</b>	<b>Recommended Value</b>
Type	1769-MODULE Generic 1769 Module
Parent	Local
Name	MVI69
Description	MVI69 Application Module
Comm Format	Data - INT
Slot	The slot number in the rack where the module is installed
Input Assembly Instance	101
Input Size	242
Output Assembly Instance	100
Output Size	241
Configuration Assembly Instance	102
Configuration Size	0

5 Click **NEXT** to continue.



6 Select the **REQUEST PACKET INTERVAL** value for scanning the I/O on the module. This value represents the minimum frequency the module will handle scheduled events. This value should not be set to less than 1 millisecond. Values between 1 and 10 milliseconds should work with most applications.

- 7 Save the module. Click **OK** to dismiss the dialog box. **THE CONTROLLER ORGANIZATION** window now displays the module's presence.



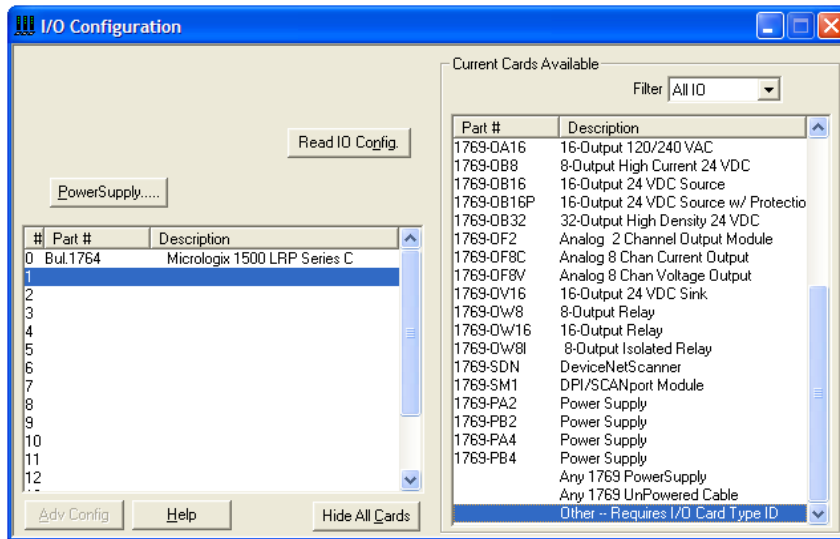
- 8 Copy the Controller Tags from the sample program.
- 9 Copy the User Defined Data Types from the sample program.
- 10 Copy the Ladder Rungs from the sample program.
- 11 Save and Download the new application to the controller and place the processor in run mode.

### 3.4 Adding the Module to an Existing MicroLogix Project

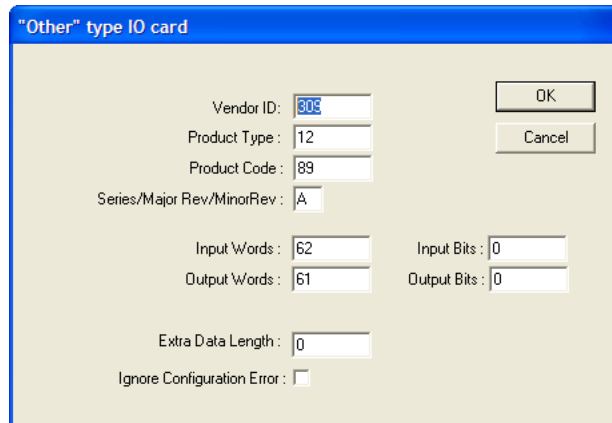
If you are installing and configuring the module with a MicroLogix controller, follow these steps. If you are using a CompactLogix controller, refer to the previous section.

The first step in setting up the processor ladder file is to define the I/O type module to the system. Start RSLogix 500, and follow these steps:

- 1 In RSLogix, open your existing application, or start a new application, depending on your requirements.
- 2 Double-click the **I/O CONFIGURATION** icon located in the *Controller* folder in the project tree. This action opens the *I/O Configuration* dialog box.



- 3 In the *I/O Configuration* dialog box, select **"OTHER - REQUIRES I/O CARD TYPE ID"** at the bottom of the list in the right pane, and then double-click to open the *"Other" type IO card* dialog box.
- 4 Enter the values shown in the following illustration to define the module correctly for the MicroLogix processor, and then click **OK** to save your configuration.



The *Input Words* and *Output Words* parameters will depend on the *Block Transfer Size* parameter you specify in the configuration file. Use the values from the following table.

Block Transfer Size	Input Words	Output Words
60	62	61
120	122	121
240	242	241

- 5 Click **OK** to continue.
- 6 After completing the module setup, the *I/O Configuration* dialog box will display the module's presence.

The last step is to add the ladder logic. If you are using the example ladder logic, adjust the ladder to fit your application. Refer to the example Ladder Logic section in this manual.

Download the new application to the controller and place the processor in RUN mode. If you encounter errors, refer to **Diagnostics and Troubleshooting** (page 81) for information on how to connect to the module's Config/Debug port to use its troubleshooting features.



## 4 Diagnostics and Troubleshooting

### In This Chapter

- ❖ LED Status Indicators..... 82
- ❖ Using ProSoft Configuration Builder (PCB) for Diagnostics..... 85
- ❖ Reading Status Data from the Module ..... 99

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

- LED status indicators on the front of the module provide general information on the module's status.
- Status data contained in the module can be viewed through the Configuration/Debug port, using the troubleshooting and diagnostic capabilities of *ProSoft Configuration Builder (PCB)*.
- Status data values can be transferred from the module to processor memory and can be monitored there manually or by customer-created logic.

## 4.1 LED Status Indicators

The LEDs indicate the module’s operating status as follows:

LED	Color	Status	Indication
CFG	Green	On	Data is being transferred between the module and a remote terminal using the Configuration/Debug port.
		Off	No data is being transferred on the Configuration/Debug port.
P1	Green	On	Data is being transferred between the module and the MODBUS network on Port 1.
		Off	No data is being transferred on the port.
P2	Green	On	Data is being transferred between the module and the MODBUS network on Port 2.
		Off	No data is being transferred on the port.
APP	Amber	On	The MVI69-MCM is functioning normally.
		Off	The MVI69-MCM module program has recognized a communication error between the module and the processor.
BP ACT	Amber	On	The LED is on when the module is performing a write operation on the backplane.
		Off	The LED is off when the module is performing a read operation on the backplane. Under normal operation, the LED should blink rapidly on and off.
OK	Red/ Green	Off	The card is not receiving any power and is not securely plugged into the rack.
		Green	The module is operating normally.
		Red	The program has detected an error or is being configured. If the LED remains red for over 10 seconds, the program has probably halted. Remove the card from the rack and re-insert the card to restart the module’s program.
BAT	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 BAT LED still does not go off, contact ProSoft Technology, as this is not a user serviceable item.

During module configuration, the OK LED will be red and the BP ACT LED will be on.

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.

### **4.1.1 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 CompactLogix or MicroLogix controller.

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

### 4.1.2 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 MVI69-MCM. 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: <ul style="list-style-type: none"> <li>▪ The processor is in RUN or REM RUN mode.</li> <li>▪ The backplane driver is loaded in the module.</li> <li>▪ The module is configured for read and write data block transfer.</li> <li>▪ The ladder logic handles all read and write block situations.</li> <li>▪ The module is properly configured in the processor I/O configuration and ladder logic.</li> </ul>
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.

## 4.2 Using ProSoft Configuration Builder (PCB) for Diagnostics

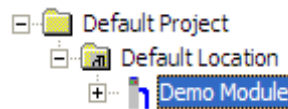
The *Configuration and Debug* menu for this module is arranged as a tree structure, with the *Main* menu at the top of the tree, and one or more submenus for each menu command. The first menu you see when you connect to the module is the *Main* menu.

Because this is a text-based menu system, you enter commands by typing the [command letter] from your computer keyboard in the *Diagnostic* window in *ProSoft Configuration Builder (PCB)*. The module does not respond to mouse movements or clicks. The command executes as soon as you press the [COMMAND LETTER] — you do not need to press [ENTER]. When you type a [COMMAND LETTER], a new screen will be displayed in your terminal application.

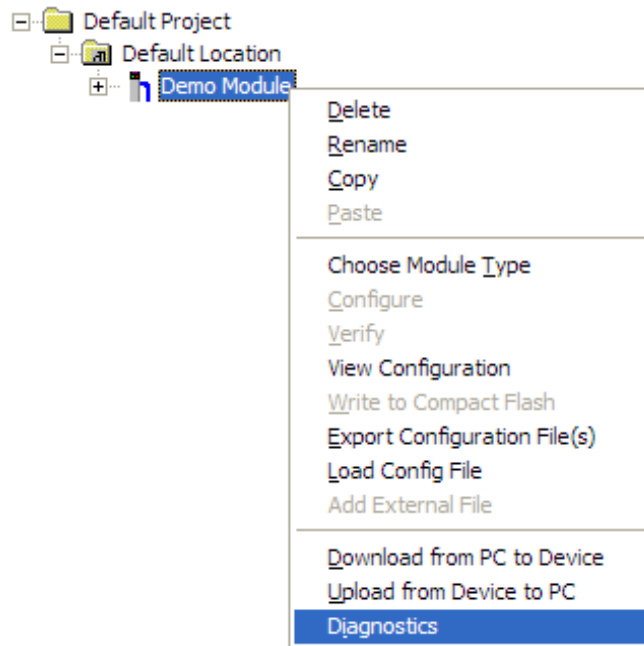
### 4.2.1 Using the Diagnostic Window in ProSoft Configuration Builder

To connect to the module's Configuration/Debug serial port

- 1 Start *PCB*, and then select the module to test. Click the right mouse button to open a shortcut menu.

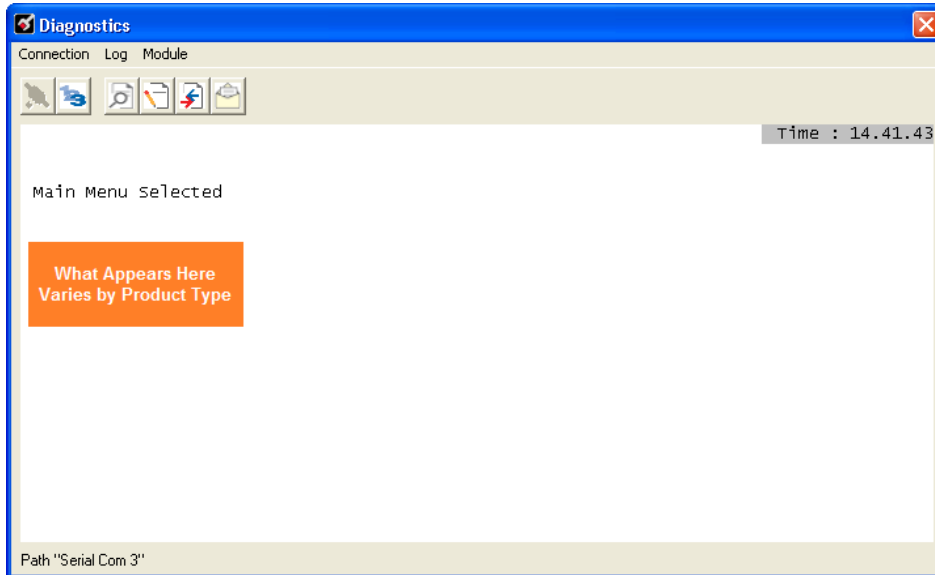


- 2 On the shortcut menu, choose **DIAGNOSTICS**.



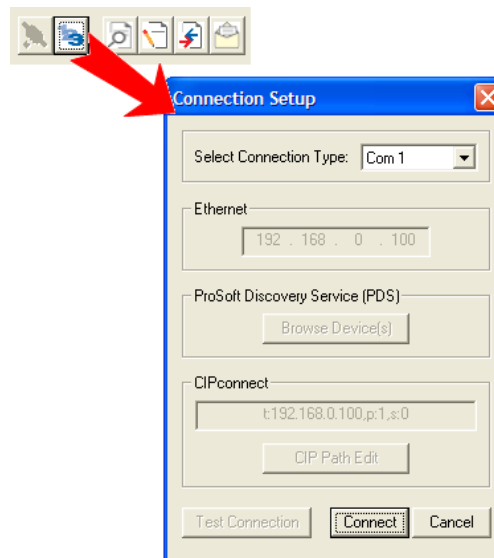
This action opens the *Diagnostics* dialog box.

- 3 Press [?] to open the *Main* menu.



If there is no response from the module, follow these steps:

- 1 Click to configure the connection. On the *Connection Setup* dialog box, select a valid com port or other connection type supported by the module.



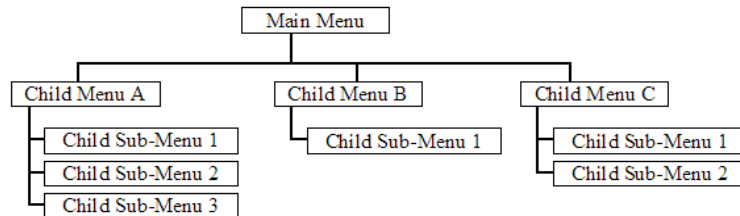
- 2 Verify that the null modem cable is connected properly between your computer's serial port and the module. A regular serial cable will not work.
- 3 On computers with more than one serial port, verify that your communication program is connected to the same port that is connected to the module.

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

## 4.2.2 Navigation

All of the submenus for this module contain commands to redisplay the menu or return to the previous menu. You can always return from a submenu to the next higher menu by pressing **[M]** on your keyboard.

The organization of the menu structure is represented in simplified form in the following illustration:



The remainder of this section shows the menus available for this module, and briefly discusses the commands available to you.

### Keystrokes

The keyboard commands on these menus are usually not case sensitive. You can enter most commands in lowercase or uppercase letters.

The menus use a few special characters (**?**, **-**, **+**, **@**) that must be entered exactly as shown. Some of these characters will require you to use the **SHIFT**, **CTRL**, or **ALT** keys to enter them correctly. For example, on US English keyboards, enter the **?** command as **SHIFT** and **/**.

Also, take care to distinguish the different uses for uppercase letter "eye" (**I**), lowercase letter "el" (**L**), and the number one (**1**). Likewise, uppercase letter "oh" (**O**) and the number zero (**0**) are not interchangeable. Although these characters look alike on the screen, they perform different actions on the module and may not be used interchangeably.

### 4.2.3 Main Menu

When you first connect to the module from your computer, your terminal screen will be blank. To activate the main menu, press the **[?]** key on your computer's keyboard. If the module is connected properly, the following menu will appear.

```
MVI69-MCM MENU
?=Display Menu
U=Version Information
D=Database Menu
C=Clear diagnostic data
B=Backplane Menu
0=Protocol Serial MCM 1
1=Protocol Serial MCM 2
S=Transfer Configuration from Unit to PC
R=Transfer Configuration from PC to Unit
W=Warm Boot Module
Esc=Exit Program
```

**Caution:** Some of the commands available to you from this menu are designed for advanced debugging and system testing only, and can cause the module to stop communicating with the processor or with other devices, resulting in potential data loss or other failures. Only use these commands if you are specifically directed to do so by ProSoft Technology Technical Support staff. Some of these command keys are not listed on the menu, but are active nevertheless. Please be careful when pressing keys so that you do not accidentally execute an unwanted command.

#### Redisplaying the Menu

Press **[?]** to display the current menu. Use this command when you are looking at a screen of data, and want to view the menu choices available to you.

#### Viewing Version Information

Press **[V]** to view version information for the module.

Use this command to view the current version of the software for the module, as well as other important values. You may be asked to provide this information when calling for technical support on the product.

Values at the bottom of the display are important in determining module operation. The *Program Scan Counter* value is incremented each time a module's program cycle is complete.

**Tip:** Repeat this command at one-second intervals to determine the frequency of program execution.



#### Opening the Database View Menu

Press **[D]** to open the *Database View* menu.

Use this menu command to view the current contents of the module's database. For more information about this submenu, see Database View Menu (page 90).

#### Clearing Diagnostic Data

Press **[C]** to clear diagnostic data from the module's memory.

#### Opening the Backplane Menu

Press **[B]** from the Main Menu to view the Backplane Data Exchange List. Use this command to display the configuration and statistics of the backplane data transfer operations.

**Tip:** Repeat this command at one-second intervals to determine the number of blocks transferred each second.

#### Opening the Protocol Serial Menu

Press **[0]** or **[1]** to view the Protocol Serial Menu for ports 1 and 2, respectively.

#### Sending the Configuration File

Press **[S]** to upload (send) a configuration file from the module to your PC.

#### Receiving the Configuration File

Press **[R]** to download (receive) the current configuration file from the module.

#### Warm Booting the Module

Press **[W]** from the *Main* menu to warm boot (restart) the module.

This command will cause the program to exit and reload, refreshing configuration parameters that must be set on program initialization. Only use this command if you must force the module to reboot.

#### Exiting the Program

Press **[ESC]** to restart the module and force all drivers to be loaded. The module will use the configuration stored in the module's Flash memory to configure the module.

### 4.2.4 Database View Menu

Press **[D]** from the *Main* menu to open the *Database View* menu. Use this menu command to view the current contents of the module database. Press **[?]** to view a list of commands available on this menu.

```
DB Menu selected

DATABASE VIEW MENU
?=Display Menu
0-9=Display 0-9000
S=Show Again
-=Back 5 Pages
P=Previous Page
+=Skip 5 Pages
N=Next Page
D=Decimal Display
H=Hexadecimal Display
F=Float Display
A=ASCII Display
M=Main Menu
```

#### Viewing Register Pages

To view sets of register pages, use the keys described below:

Command	Description
<b>[0]</b>	Display registers 0 to 99
<b>[1]</b>	Display registers 1000 to 1099
<b>[2]</b>	Display registers 2000 to 2099

And so on. The total number of register pages available to view depends on your module's configuration.

#### Displaying the Current Page of Registers Again

Press **[S]** from the *Database View* menu to show the current page of registers again.

DATABASE DISPLAY 0 TO 99 <DECIMAL>									
100	101	102	4	5	6	7	8	9	10
11	12	13	14	15	16	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

This screen displays the current page of 100 registers in the database.

*Moving Back Through 5 Pages of Registers*

Press **[-]** from the *Database View* menu to skip five pages back in the database to see the 100 registers of data starting 500 registers before the currently displayed page.

*Moving Forward (Skipping) Through 5 Pages of Registers*

Press **[+]** from the *Database View* menu to skip five pages ahead in the database to see the 100 registers of data starting 500 registers after the currently displayed page.

*Viewing the Previous Page of Registers*

Press **[P]** from the *Database View* menu to display the previous page of data.

*Viewing the Next Page of Registers*

Press **[N]** from the *Database View* menu to display the next page of data.

*Viewing Data in Decimal Format*

Press **[D]** from the *Database View* menu to display the data on the current page in decimal format.

*Viewing Data in Hexadecimal Format*

Press **[H]** from the *Database View* menu to display the data on the current page in hexadecimal format.

*Viewing Data in Floating-Point Format*

Press **[F]** from the *Database View* menu to display the data on the current page in floating-point format. The program assumes that the values are aligned on even register boundaries. If floating-point values are not aligned as such, they are not displayed properly.

*Viewing Data in ASCII (Text) Format*

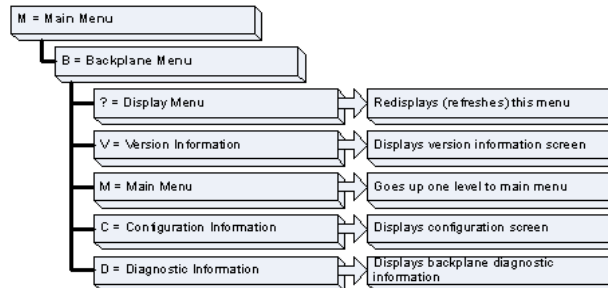
Press **[A]** from the *Database View* menu to display the data on the current page in ASCII format. This is useful for regions of the database that contain ASCII data.

*Returning to the Main Menu*

Press **[M]** to return to the *Main* menu.

### 4.2.5 Backplane Menu

Press **[B]** from the Main Menu to view the Backplane Data Exchange List. Use this command to display the configuration and statistics of the backplane data transfer operations. Press **[?]** to view a list of commands available on this menu.



#### Redisplaying the Menu

Press **[?]** to display the current menu. Use this command when you are looking at a screen of data, and want to view the menu choices available to you.

#### Viewing Version Information

Press **[V]** to view version information for the module.

Use this command to view the current version of the software for the module, as well as other important values. You may be asked to provide this information when calling for technical support on the product.

Values at the bottom of the display are important in determining module operation. The *Program Scan Counter* value is incremented each time a module's program cycle is complete.

**Tip:** Repeat this command at one-second intervals to determine the frequency of program execution.

#### Returning to the Main Menu

Press **[M]** to return to the *Main* menu.

#### Viewing Configuration Information

Press **[C]** to view configuration information for the selected port, protocol, driver or device.

Viewing Backplane Diagnostic Information

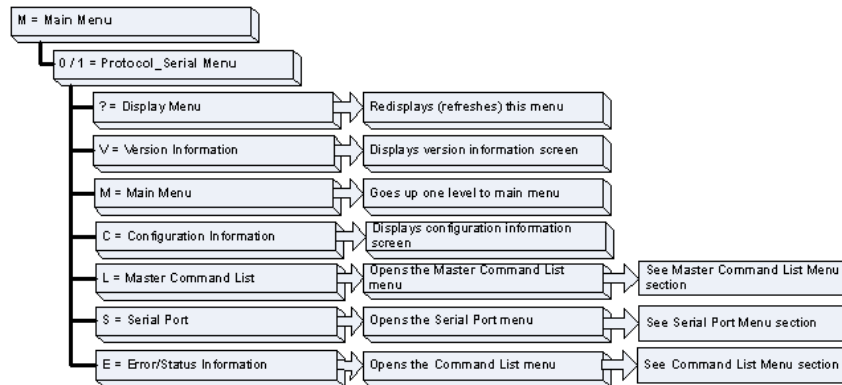
Press **[D]** to view Backplane Diagnostic information.

Use this command to display the configuration and statistics of the backplane data transfer operations between the module and the processor. The information on this screen can help determine if there are communication problems between the processor and the module.

**Tip:** Repeat this command at one-second intervals to determine the number of blocks transferred each second

**4.2.6 Protocol Serial MCM Menu**

Press **[0]** or **[1]** to view protocol serial information for ports 1 and 2, respectively. Use this command to view a variety of error and status screens for the port. Press **[?]** to view a list of commands available on this menu.



Redisplaying the Menu

Press **[?]** to display the current menu. Use this command when you are looking at a screen of data, and want to view the menu choices available to you.

Viewing Version Information

Press **[V]** to view version information for the module.

Use this command to view the current version of the software for the module, as well as other important values. You may be asked to provide this information when calling for technical support on the product.

Values at the bottom of the display are important in determining module operation. The *Program Scan Counter* value is incremented each time a module’s program cycle is complete.

**Tip:** Repeat this command at one-second intervals to determine the frequency of program execution.

Returning to the Main Menu

Press **[M]** to return to the *Main* menu.

Viewing Configuration Information

Press **[C]** to view configuration information for the selected port, protocol, driver or device.

Opening the Command List Menu

Press **[L]** to open the Command List menu. Use this command to view the configured command list for the module.

Opening the Serial Port Menu

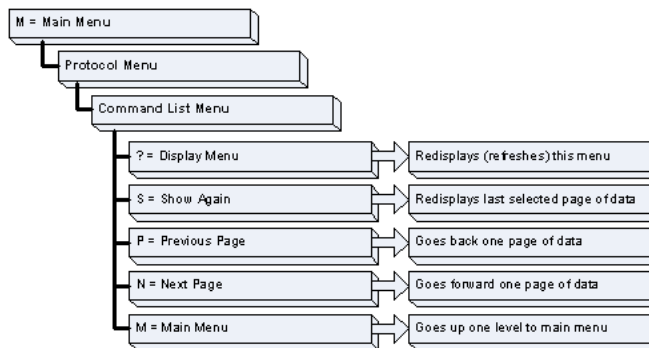
Press **[S]** to open the Serial Port menu. Use this command to view and change additional serial port driver settings.

Viewing Error and Status Data

Press **[E]** to display the error/status data for the module.

**4.2.7 Master Command Error List Menu**

Use this menu to view the command error list for the module. Press **[?]** to view a list of commands available on this menu.



Redisplaying the Current Page

Press **[S]** to display the current page of data.

Moving Back Through 5 Pages of Commands

Press **[-]** to display data for last 5 page commands.

Viewing the Previous Page of Commands

Press **[P]** to display the previous page of commands.

*Moving Forward (Skipping) Through 5 Pages of Commands*

Press **[+]** to display data for the next page of commands.

*Viewing the Next Page of Commands*

Press **[N]** to display the next page of commands.

*Returning to the Main Menu*

Press **[M]** to return to the *Main* menu.

#### **4.2.8 Serial Port Menu**

Press **[S]** to open the Serial Port menu. Use this command to view and change additional serial port driver settings. Press **[?]** to view a list of commands available on this menu.

```
SerialPort_MVI 1 MENU
?=Display Menu
V=Version Information
M=Main Menu
D=DataAnalyzer
```

*Redisplaying the Menu*

Press **[?]** to display the current menu. Use this command when you are looking at a screen of data, and want to view the menu choices available to you.

*Viewing Version Information*

Press **[V]** to view version information for the module.

Use this command to view the current version of the software for the module, as well as other important values. You may be asked to provide this information when calling for technical support on the product.

Values at the bottom of the display are important in determining module operation. The *Program Scan Counter* value is incremented each time a module's program cycle is complete.

**Tip:** Repeat this command at one-second intervals to determine the frequency of program execution.

*Returning to the Main Menu*

Press **[M]** to return to the *Main* menu.

### Opening the Data Analyzer Menu

Press **[A]** to open the Data Analyzer Menu. Use this command to view all bytes of data transferred on each port. Both the transmitted and received data bytes are displayed. Refer to Data Analyzer (page 96) for more information about this menu.

**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 press **[S]** to stop the data analyzer, and then press **[M]** to return to the main menu. This action will allow the module to resume its normal high speed operating mode.

## 4.2.9 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.

```
DATA ANALYZER VIEW MENU
?=Display Menu
5=1 mSec Ticks
6=5 mSec Ticks
7=10 mSec Ticks
8=50 mSec Ticks
9=100 mSec Ticks
0=No mSec Ticks
H=Hex Format
A=ASCII Format
B=Start
S=Stop
M=Main Menu

Port = 1, Format=HEX, Tick=10
```

**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 press **[S]** to stop the data analyzer, and then press **[M]** to return to the main menu. This action will allow the module to resume its normal high speed operating mode.

### Redisplaying the Menu

Press **[?]** to display the current menu. Use this command when you are looking at a screen of data, and want to view the menu choices available to you.



### Displaying Timing Marks in the Data Analyzer

You can display timing marks for a variety of intervals in the data analyzer screen. These timing marks can help you determine communication-timing characteristics.

<b>Key</b>	<b>Interval</b>
<b>[5]</b>	1 milliseconds ticks
<b>[6]</b>	5 milliseconds ticks
<b>[7]</b>	10 milliseconds ticks
<b>[8]</b>	50 milliseconds ticks
<b>[9]</b>	100 milliseconds ticks
<b>[0]</b>	Turn off timing marks

### Removing Timing Marks in the Data Analyzer

Press **[0]** to turn off timing marks in the Data Analyzer screen.

### Viewing Data in Hexadecimal Format

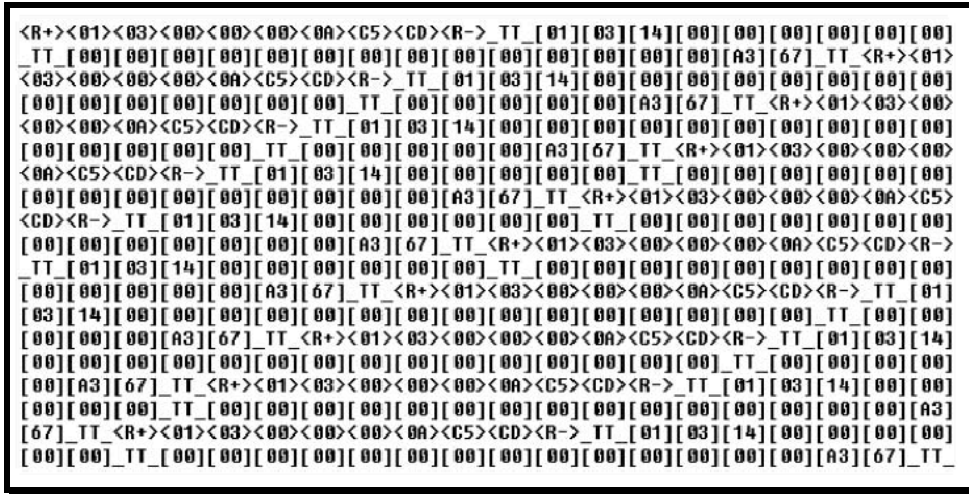
Press **[H]** from the *Database View* menu to display the data on the current page in hexadecimal format.

### Viewing Data in ASCII (Text) Format

Press **[A]** from the *Database View* menu to display the data on the current page in ASCII format. This is useful for regions of the database that contain ASCII data.

Starting the Data Analyzer

Press **[B]** to start the data analyzer. After the key is pressed, all data transmitted and received on the currently selected port will be displayed. The following illustration shows an example.



The Data Analyzer displays 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 timing mark interval has been reached. This parameter is user defined.

Stopping the Data Analyzer

Press **[S]** to stop the data analyzer. Use this option to freeze the display so the data can be analyzed. To restart the analyzer, press **[B]**.

**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 press **[S]** to stop the data analyzer, and then press **[M]** to return to the main menu. This action will allow the module to resume its normal high speed operating mode.

Returning to the Main Menu

Press **[M]** to return to the *Main* menu.

### **4.3 Reading Status Data from the Module**

The MVI69-MCM module returns a 29-word Status Data block that can be used to determine the module's operating status. This data is located in the module's database at registers 6670 to 6698 and at the location specified in the configuration. This data is transferred to the CompactLogix or MicroLogix processor continuously.



## 5 Reference

### *In This Chapter*

❖ Product Specifications .....	102
❖ Functional Overview .....	106
❖ Data Flow between MVI69-MCM Module and CompactLogix or MicroLogix Processor .....	110
❖ Normal Data Transfer .....	115
❖ Special Control and Status Blocks .....	121
❖ Modbus Protocol Specification .....	134
❖ Cable Connections .....	146
❖ MCM Database Definition.....	151
❖ Status Data Definition.....	152

## 5.1 Product Specifications

The MVI69 Modbus Master/Slave Communication Module allows Rockwell Automation® CompactLogix or MicroLogix® processors to interface easily with other Modbus protocol compatible devices.

The module acts as an input/output module between the Modbus network and the CompactLogix or MicroLogix backplane. Compatible devices include not only Modicon® PLCs (almost all support the Modbus protocol) but also a wide range of process and control devices from a variety of manufacturers. Many SCADA packages also support the Modbus protocol.

### 5.1.1 General Specifications

- Single-slot, 1769 backplane-compatible
- The module is recognized as an Input/Output module and has access to processor memory for data transfer between processor and module.
- Ladder Logic is used for data transfer between module and processor. Sample ladder file included.
- Configuration data obtained from configuration text file downloaded to module. Sample configuration file included.
- Supports CompactLogix processors with 1769 I/O bus capability and at least 800 mA of 5 Vdc backplane current available.
- Also supports MicroLogix 1500 LRP

### 5.1.2 Hardware Specifications

Specification	Description
Dimensions	Standard 1769 Single-slot module
Current Load	800 mA max @ 5 VDC Power supply distance rating of 2 (L43 and L45 installations on first 2 slots of 1769 bus)
Operating Temp.	32° F to 140° F (0° C to 60° C)
Storage Temp.	-40° F to 185° F (-40° C to 85° C)
Relative Humidity	5% to 95% (with no condensation)
LED Indicators	Battery and Module Status Application Status Serial Port Activity CFG Port Activity
CFG Port (CFG)	RJ45 (DB-9F with supplied cable) RS-232 only No hardware handshaking
App Ports (P1,P2) (Serial modules)	RS-232, RS-485 or RS-422 (jumper selectable) RJ45 (DB-9F with supplied cable) RS-232 handshaking configurable 500V Optical isolation from backplane
Shipped with Unit	RJ45 to DB-9M cables for each port 6-foot RS-232 configuration Cable

### 5.1.3 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)



### 5.1.4 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 CompactLogix or MicroLogix processor.

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

## 5.2 Functional Overview

### 5.2.1 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 MVI69-MCM module can work as a Master and as a Slave.

The MVI69-MCM 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.

### 5.2.2 Module Power Up

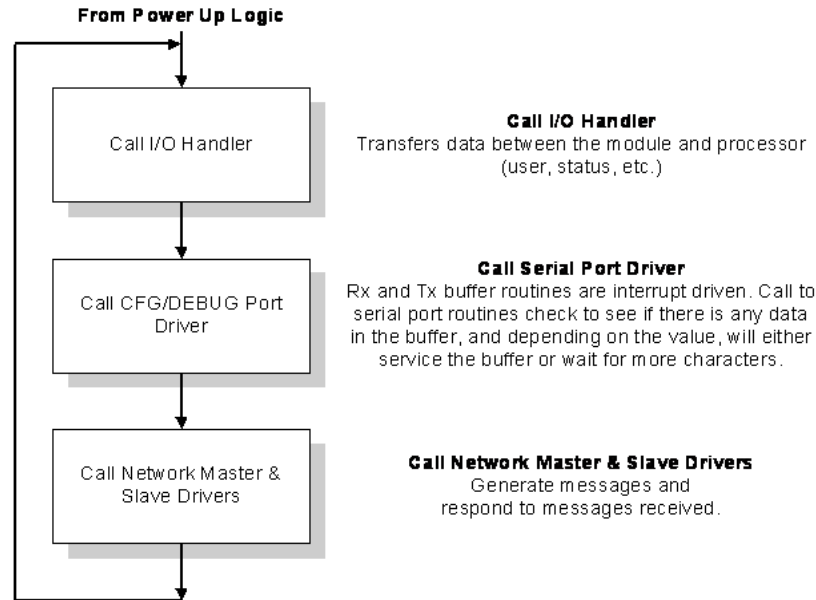
On power up the module begins performing the following logical functions:

- 1 Initialize hardware components
  - Initialize CompactLogix or MicroLogix backplane driver
  - Test and Clear all RAM
  - Initialize the serial communication ports
- 2 Module configuration
- 3 Initialize Module Register space
- 4 Enable Slave Driver on selected ports
- 5 Enable Master Driver on selected ports

After this initialization procedure is complete, the module will begin communicating with other nodes on the network, depending on the configuration.

### 5.2.3 Main Logic Loop

Upon completing the power up configuration process, the module enters an infinite loop that performs the following functions:



### 5.2.4 Backplane Data Transfer

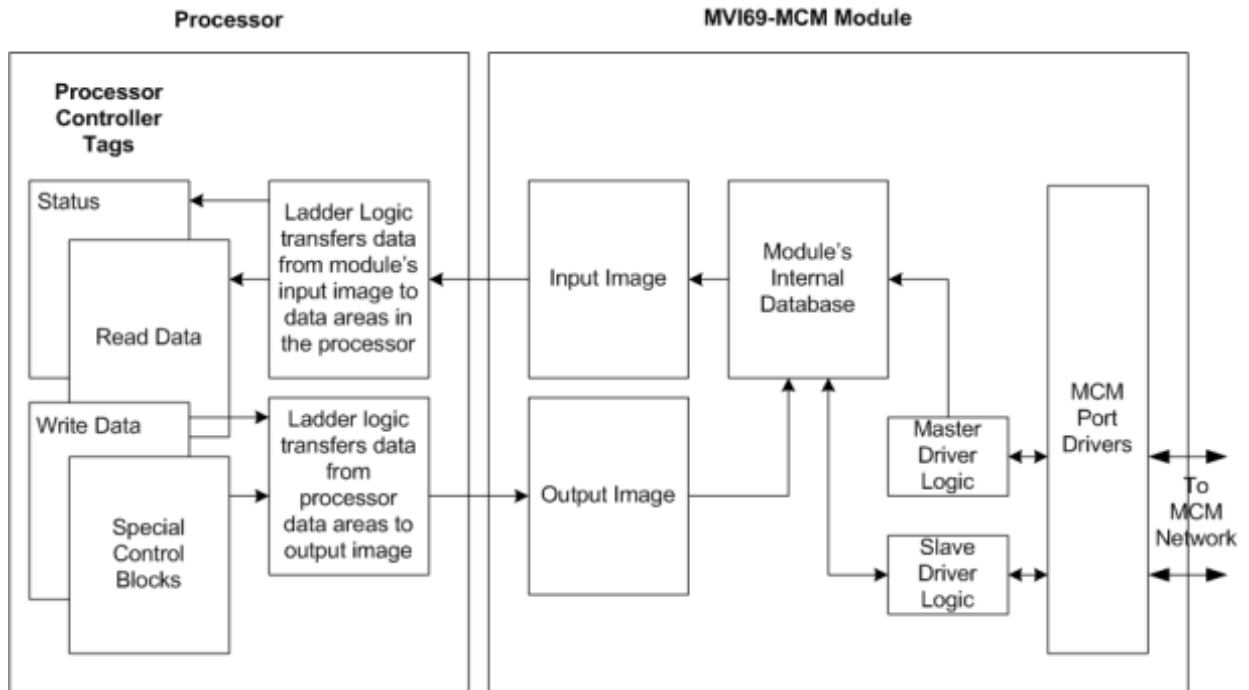
The MVI69-MCM module communicates directly over the CompactLogix or MicroLogix backplane. Data is paged between the module and the CompactLogix or MicroLogix processor across the backplane using the module's input and output images. The update frequency of the images is determined by the scheduled scan rate defined by the user for the module and the communication load on the module. Typical updates are in the range of 2 to 10 milliseconds.

The data is paged between the processor and the module using input and output image blocks. You can configure the size of the blocks using the Block Transfer Size parameter in the configuration file. You can configure blocks of 60, 120, or 240 words of data depending on the number of words allowed for your own application.

This bi-directional transference of data is accomplished by the module filling in data in the module's input image to send to the processor. Data in the input image is placed in the Controller Tags in the processor by the ladder logic. The input image for the module may be set to 62, 122, or 242 words depending on the block transfer size parameter set in the configuration file.

The processor inserts data to the module's output image to transfer to the module. The module's program extracts the data and places it in the module's internal database. The output image for the module may be set to 61, 121, or 241 words depending on the block transfer size parameter set in the configuration file.

The following illustration shows the data transfer method used to move data between the CompactLogix or MicroLogix processor, the MVI69-MCM module and the MODBUS network.

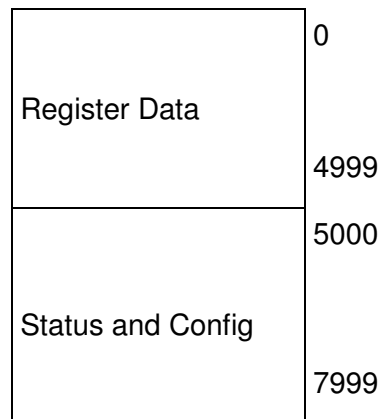


All data transferred between the module and the processor over the backplane is through the input and output images. Ladder logic must be written in the CompactLogix or MicroLogix processor to interface the input and output image data with data defined in the Controller Tags. All data used by the module is stored in its internal database. The following illustration shows the layout of the database:

**Module's Internal Database Structure**

5000 registers for user data

3000 words of configuration and status data



Data contained in this database is paged through the input and output images by coordination of the CompactLogix or MicroLogix ladder logic and the MVI69-MCM module's program. Up to 242 words of data can be transferred from the module to the processor at a time. Up to 241 words of data can be transferred from the processor to the module. The read and write block identification codes in each data block determine the function to be performed or the content of the data block. The block identification codes used by the module are listed below:

<b>Block Range</b>	<b>Descriptions</b>
-1	Status Block
0	Status Block
1 to 84	Read or write data
1000	Event Port 1
2000	Event Port 2
3000 to 3001	Port 1 slave polling control
3002 to 3006	Port 1 slave status
3100 to 3101	Port 2 slave polling control
3102 to 3106	Port 2 slave status
5000 to 5006	Port 1 command control
5100 to 5106	Port 2 command control
9958	Function Code 5 data formatted Pass-Thru Control Blocks
9956 and 9957	Function Code 6 and 16 (Floating-point) data formatted Pass-Thru Control Block
9959	Function Code 15 data formatted Pass-Thru Control Block
9998	Warm-boot control block
9999	Cold-boot control block

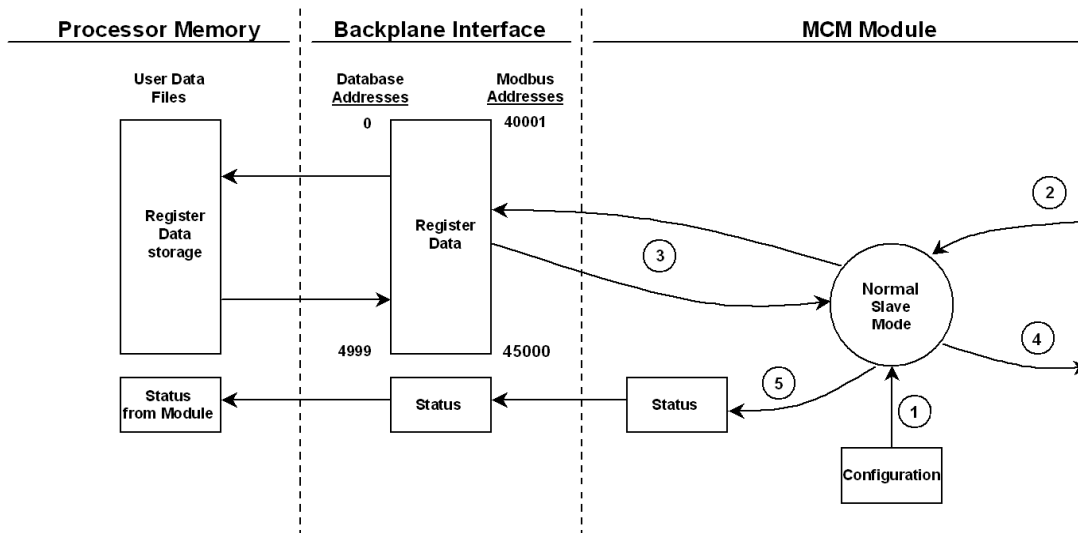
Each image has a defined structure depending on the data content and the function of the data transfer.

### 5.3 Data Flow between MVI69-MCM Module and CompactLogix or MicroLogix Processor

The following topics describe the flow of data between the two pieces of hardware (MVI69-MCM processor and MVI69-MCM module) and other nodes on the MODBUS network under the module's different operating modes. Each port on the module is configured to emulate a MODBUS Master device or a MODBUS slave device. The operation of each port depends on this configuration. The sections below discuss the operation of each mode.

#### 5.3.1 Slave Driver

The Slave Driver Mode allows the MVI69-MCM module to respond to data read and write commands issued by a Master on the MODBUS network. The following diagram shows the data flow for Normal Slave Mode.

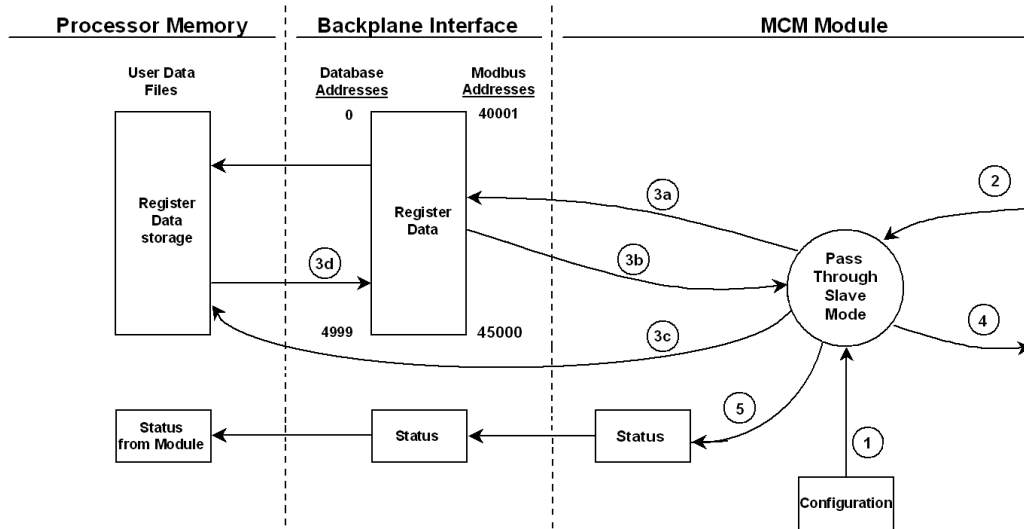


Step	Description
1	Anytime the module restarts (boots or reboots), the MODBUS slave port driver receives configuration information from a user defined .CFG file stored on the MVI69-MCM. This information configures the serial ports and defines slave node characteristics. The configuration information may also contain instructions to offset data stored in the database to addresses different from addresses requested in the received messages.
2	A Host device, such as a Modicon PLC or an HMI application, issues a read or write command to the module's node address. The port driver qualifies the message before accepting it into the module. Rejected commands will cause an Exception Response.
3	After the module accepts the command, the data is immediately transferred to or from the module's internal database. If the command is a read command, the data is read from 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 Steps 2 and 3 has been completed, either a normal response message or an Exception Response message will be sent to the Master.
5	Counters are available in the Status Block that permit the ladder logic program to determine the level of activity of the Slave Driver.

In Slave Pass-Through mode, write commands from the Master are handled differently than they are in Normal mode. In Pass-Through mode, all write requests will be passed directly to the processor and data will not be written directly into the database.

This mode is especially useful whenever both a Modbus Master and the module's processor logic need to be able to read and write values to the same internal database addresses.

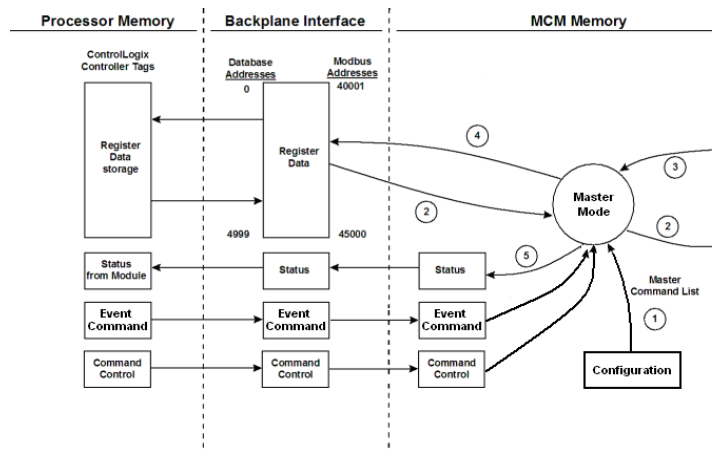
The following diagram shows the data flow for a slave port with pass-through enabled:



Step	Description
1	Same as normal mode.
2	Same as normal mode.
3	<p>a. In Pass-Through mode, if the slave driver receives a read request, it looks for the data in module's internal database, just as it would in Normal mode.</p> <p>b. The data needed to respond to the read command is retrieved directly from the internal database and returned to the Slave driver so it can build a response message.</p> <p>c. In Pass-Through mode if the slave driver receives a write request, it does not send the data directly to the module's internal database. It puts the data to be written into a special Input Image with special Block ID code to identify it as a Pass-Through Write Block and substitutes this special block in place of the next regular Read Data Block. The special block is processed by the ladder logic and the data to be written is placed into the controller tag WriteData array at an address that corresponds to the Modbus Address received in the write command.</p> <p>d. During normal backplane communications, the data from the Write Data array, including the data updated by the Pass-Through Write Block, is sent to the module's internal database. This gives the ladder logic the opportunity to also change the values stored in these addresses, if need be, before they are written to the database.</p>
4	Same as normal mode.
5	Same as normal mode.

### 5.3.2 Master Driver Mode

In the Master mode, the MVI69-MCM module issues read or write commands to slave devices on the MODBUS network. These commands are user configured in the module via the Master Command List is received from the user defined configuration file that is stored on the MVI69-MCM module or can be issued directly from the CompactLogix or MicroLogix processor (Special Command Blocks). Command status is returned to the processor for each individual command in the command list. The location of this command status list 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.



Step	Description
1	The Master driver obtains configuration data from the user defined .CFG file that is stored locally on the MVI69-MCM module itself. The configuration data obtained includes port configuration and the Master Command List. Special Ccommands can be issued directly from the CompactLogix or MicroLogix processor (using Event Commands and Command Control). These configuration and command values are used by the Master driver to determine the types and order of commands to send to slaves on the network.
2	After configuration, the Master driver begins transmitting read and/or write commands to slave nodes on the network. If writing data to slave, the data for the write command is obtained from the module's internal database to build the command.
3	Once the specified slave has successfully processed the command, it will return a response message to the Master driver for processing.
4	Data received from a slave in response to a read command is passed to the module's and stored in its internal database.
5	Status is returned to the CompactLogix or MicroLogix processor for each command in the Master Command List.

**Important:** You must take care when constructing each command in the list to ensure predictable operation of the module. If two commands write to the same internal database address of the module, the results will be invalid. 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.

**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 parameter to the database location desired.

To transfer this table to the processor, make sure that the Command Error table is in the database area covered by the Read Data.

***Standard MODBUS Protocol Exception Code Errors***

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

***Module Communication Error Codes***

<b>Code</b>	<b>Description</b>
-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

*Command List Entry Errors*

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

## 5.4 Normal Data Transfer

Normal data transfer includes the paging of the user data found in the module's internal database in registers 0 to 4999 and the status data. These data are transferred through read (input image) and write (output image) blocks. The following topics describe the structure and function of each block:

### 5.4.1 Block Request from the Processor to the Module

These blocks of data transfer information from the processor to the module. The structure of the output image used to transfer this data is shown below:

Offset	Description	Length (words)
0	Write Block ID	1
1 to (n)	Write Data	(n)

*(n) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

The Write Block ID is an index value used to determine the location in the module's database where the data will be placed.

### 5.4.2 Block Response from the Module to the Processor

These blocks of data transfer information from the module to the processor. The structure of the input image used to transfer this data is shown below:

Offset	Description	Length (words)
0	Read Block ID	1
1	Write Block ID	1
2 to (n+1)	Read Data	(n)

*(n) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

### 5.4.3 Read Block and Write Block Transfer Sequences

The Read Block ID is an index value used to determine the location of where the data will be placed in the processor controller tag array of module read data. The number of data words per transfer depends on the configured Block Transfer Size parameter in the configuration file (possible values are 60, 120, or 240).

The Write Block ID associated with the block requests data from the processor. Under normal program operation, the module sequentially sends read blocks and requests write blocks. For example, if the application uses three read and two write blocks, the sequence will be as follows:

R1W1→R2W2→R3W1→R1W2→R2W1→R3W2→R1W1→

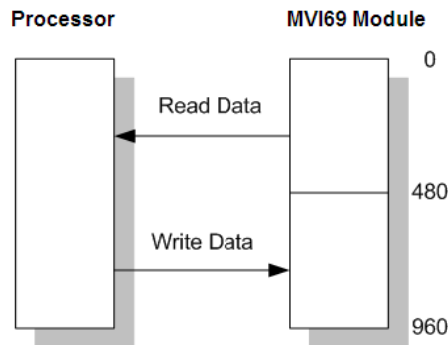
This sequence will continue until interrupted by other write block numbers sent by the controller or by a command request from a node on the MODBUS network or operator control through the module's Configuration/Debug port.

The following example shows a typical backplane communication application.

If the backplane parameters are configured as follows:

```
Read Register Start:      0
Read Register Count:     480
Write Register Start:    480
Write Register Count:    480
```

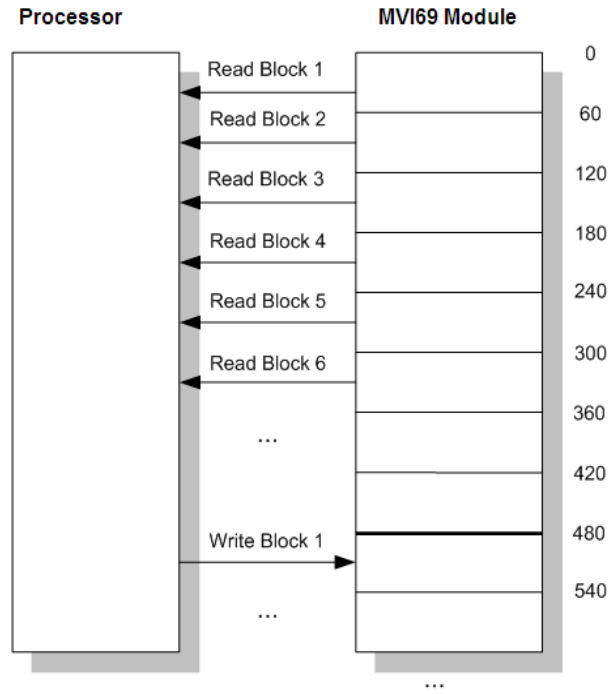
The backplane communication would be configured as follows:



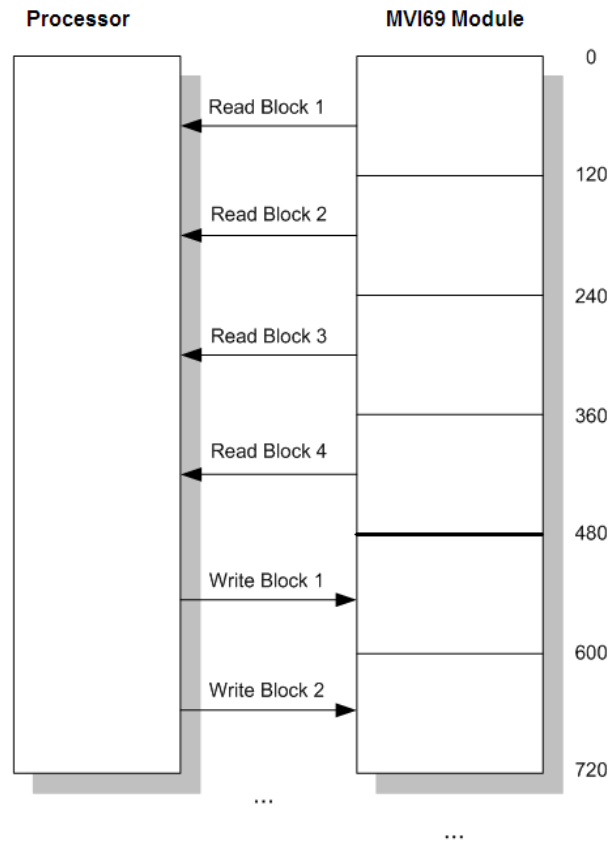
Database address 0 to 479 will be continuously transferred from the module to the processor. Database address 480 to 959 will continuously be transferred from the processor to the module.

The Block Transfer Size parameter basically configures how the Read Data and Write Data areas are broken down into data blocks (60, 120, or 240).

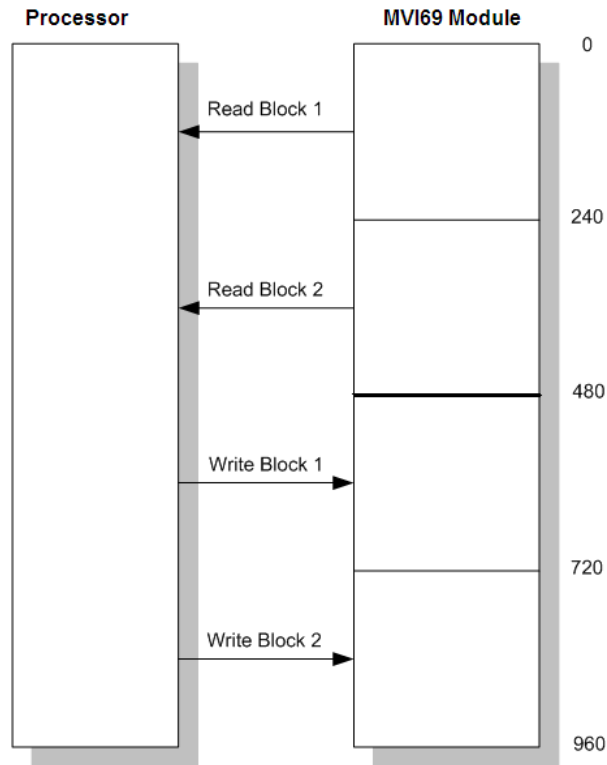
### 5.4.4 If Block Transfer Size = 60



### 5.4.5 If Block Transfer Size = 120



### 5.4.6 If Block Transfer Size = 240



### 5.4.7 Status Data Block (Read Block ID = 0)

After the last Read Block is sent, the module builds an output image (ID = 0) to transfer the module's status information to the processor. This information can be used by the PLC program to determine the current status of the module. Ladder logic should be constructed to transfer the information in this block to a user data file. The structure of this block is shown in the following table.

Offset	Content	Description
0	Read Block ID	Block identification code -1 to indicate a status block.
1	Write Block ID	Block requested from the processor by the module.
2	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
3 to 4	Product Code	These two registers contain the product code of "MCM"
5 to 6	Product Version	These two registers contain the product version for the currently running software.
7 to 8	Operating System	These two registers contain the month and year values for the program operating system.
9 to 10	Run Number	These two registers contain the Run Number value for the currently running software.
11	Port 1 Command List Requests	This field contains the number of requests made from this port to slave devices on the network.
12	Port 1 Command List Response	This field contains the number of slave response messages received on the port.

---

<b>Offset</b>	<b>Content</b>	<b>Description</b>
13	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.
14	Port 1 Requests	This field contains the total number of messages sent out of the port.
15	Port 1 Responses	This field contains the total number of messages received on the port.
16	Port 1 Errors Sent	This field contains the total number of message errors sent out of the port.
17	Port 1 Errors Received	This field contains the total number of messages errors received on the port.
18	Port 2 Command List Requests	This field contains the number of requests made from this port to slave devices on the network.
19	Port 2 Command List Response	This field contains the number of slave response messages received on the port.
20	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.
21	Port 2 Requests	This field contains the total number of messages sent out the port.
22	Port 2 Responses	This field contains the total number of messages received on the port.
23	Port 2 Errors Sent	This field contains the total number of message errors sent out of the port.
24	Port 2 Errors Received	This field contains the total number of message errors received on the port
25	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
26	Write Block Count	This field contains the total number of write blocks transferred from the processor to the module.
27	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
28	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
29	Command Block Count	This field contains the total number of command blocks received from the processor.
30	Error Block Count	This field contains the total number of block errors recognized by the module.
31	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.
32	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 an error.
33	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.
34	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.

---



## 5.5 Special Control and Status Blocks

Control and Status blocks are special blocks used to control the module or request special data from the module. The current version of the software supports eight types of special blocks.

- Slave Disable and Enable blocks (page 121) (only for Master port or ports)
- Slave Status blocks (page 124) (only for Master port or ports)
- Event Command blocks (page 125) (only for Master port or ports)
- Command Control blocks (page 127) (only for Master port or ports)
- Pass-Through blocks (page 129) (only for Slave port or ports)
- Initialize Output Data blocks (page 133) (all configurations)
- Warm Boot block (page 133, page 89) (all configurations)
- Cold Boot block (page 133) (all configurations)

Slave Disable/Enable, Slave Status, Event Command, and Command Control blocks function only when one or both ports are configured to be a Modbus Master. Pass-Through blocks function only when one or both ports are configured to be a Modbus Slave. Initialize Output Data, Warm Boot, and Cold Boot blocks function regardless of port Master/Slave configuration.

### 5.5.1 Slave Disable and Enable Control Blocks

For a variety of reasons, it may be desirable to disable and enable polling of certain network slaves using logic control from the processor. For these instances, you can use the Slave Disable and Slave Enable Special Control Block codes, along with a list of slave addresses to disable or enable.

When a port is configured as a Modbus Master, the module maintains a 256-word memory table that reserves one word for each possible Modbus slave address (see Slave Status (page 124)). This memory table is used by the module's operating firmware as a 'scratch pad' to hold the polling state of each Modbus address, to see whether or not that address is currently being polled or whether or not that address can be polled.

For example, IF:

- 1 A slave fails to respond to a command within the specified **RESPONSE TIMEOUT** (page 58);
- 2 And, the specified number of **RETRIES** (page 58) also fail;
- 3 And, the **ERROR DELAY COUNTER** (page 58) is greater than 0;

THEN:

- 1 The module will set the Slave Status register for that address to a value of 3 to prevent further polling of that slave.
- 2 The value in the **ERROR DELAY COUNTER** parameter will be added to an internal countdown register.
- 3 Each time a command addressed to the disabled slave is to be executed from the Command List, the command will be skipped instead of sent.
- 4 The internal countdown register will be decremented by one (1) each time a command is skipped.

- 5 When the internal countdown register decrements to zero (0), the Slave Status register for that slave will be set to a value of one (1), which will allow a command to be sent in an attempt to re-establish communication with the slave.

The Slave Disable and Slave Enable Special Control Blocks can be used to access the Slave Status table and control slave polling through ladder logic by forcing values into the Slave Status table in a manner similar to how the module's internal firmware operates automatically.

Using the Disable control will force a value of three (3) into the internal module register that holds the Slave Status information for that port and slave address. In this situation, when a Slave Status register is set to three (3), the slave at the corresponding address will not be polled, even if there are commands in the Command List that are enabled and addressed to that slave. Since the **ERROR DELAY COUNTER** is not involved in this operation, the status register will remain set to 3, disabling the slave, until it is re-enabled with the Enable control.

Using the Enable control will force a value of one (1) into the table. When a Slave Status register is set to one (1), the slave at the corresponding address will be polled as specified in the Command List.

Here are the Special Control Block codes for performing these functions.

Block ID	Description
3000	Disable list of slaves on Port 1
3001	Enable list of slaves on Port1
3100	Disable list of slaves on Port 2
3101	Enable list of slaves on Port 2

Port 1 slaves can be disabled using block code 3000 and can be enabled using block code 3001. Port 2 slaves can be disabled using block code 3100 and can be enabled using block code 3101. Each output (Write) block can contain a list of up to 59 slave addresses to disable or enable.

Write Block - Disable Slaves

Here is structure of the Disable Slaves output image block.

Offset	Description	Length (words)
0	3000 or 3100	1
1	Number of slaves to disable with this block	1
2 to 60	List of addresses of the slaves to disable	59
61 to ( n )	Spare (present only if Block Transfer Size > 60)	( n - 60 )

( n ) = 120 or 240 (if configured)

The module will respond with an input image block with the same identification code and indicate the number of slaves disabled by this function.

Read Block - Disable Slaves

Here is the structure of the Disable Slaves input image block.

Offset	Description	Length (words)
0	3000 or 3100	1
1	Write Block ID	1
2	Number of slaves disabled	1
3 to ( $n + 1$ )	Spare	( $n - 1$ )

(  $n$  ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

Write Block - Enable Slaves

Here is the structure of the Enable Slaves output image block:

Offset	Description	Length (words)
0	3001 or 3101	1
1	Number of slaves to enable with this block	1
2 to 60	List of addresses of slaves to enable	59
61 to ( $n$ )	Spare (present only if Block Transfer Size > 60)	( $n - 60$ )

(  $n$  ) = 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

The module will respond with an input image block with the same identification code and indicate the number of slaves enabled by this function.

Read Block - Enable Slaves

Here is the structure of the Enable Slaves input image block.

Offset	Description	Length (words)
0	3001 or 3101	1
1	Write Block ID	1
2	Number of slaves enabled	1
3 to ( $n + 1$ )	Spare	( $n - 1$ )

(  $n$  ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

**Important:** The slaves are enabled by default. Therefore, this block should only be used to re-enable slaves that have been disabled by using block 3000 or 3100 .

### 5.5.2 Slave Status Blocks

Slave status blocks send status information of each slave device on a Master port. Slaves attached to the Master port can have one of the following states:

State	Description
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
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.

Slaves are defined to the system when the module initializes the Master command list. Each slave defined will be set to a state of one in this initial step. If the Master port fails to communicate with a slave device (retry count expired on a command), the Master will set the state of the slave to a value of 2 in the status table. This suspends communication with the slave device for a user specified scan count (**ERROR DELAY COUNT** parameter in the configuration file). Each time a command in the list is scanned that has the address of a suspended slave, the delay counter value will be decremented. When the value reaches zero, the slave state will be set to one.

In order to read the slave status table, refer to the sample ladder logic. The ladder logic must send a special block to the module to request the data. Each port has a specific set of blocks to request the data as follows:

Block ID	Description
3002	Request status for slaves 0 to 59 for Port 1
3003	Request status for slaves 60 to 119 for Port 1
3004	Request status for slaves 120 to 179 for Port 1
3005	Request status for slaves 180 to 239 for Port 1
3006	Request status for slaves 240 to 255 for Port 1
3102	Request status for slaves 0 to 59 for Port 2
3103	Request status for slaves 60 to 119 for Port 2
3104	Request status for slaves 120 to 179 for Port 2
3105	Request status for slaves 180 to 239 for Port 2
3106	Request status for slaves 240 to 255 for Port 2

The following topics describe the format of these blocks:

#### Write Block - Request Slave Status

Offset	Description	Length (words)
0	3002 to 3006 or 3102 to 3106	1
1 to <i>n</i>	Spare	<i>n</i>

*n=60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

The module will recognize the request by receiving the special write block code and respond with a read block with the following format:

*Read Block - Read Slave Status*

Offset	Description	Length (words)
0	3002 to 3006 or 3102 to 3106	1
1	Write Block ID	1
2 to 61	Slave Poll Status Data	60
62 to (n + 1)	Spare (if Block Transfer Size > 60)	(n - 62)

*n = 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

**5.5.3 Event Command**

Event Command blocks send MODBUS commands directly from the ladder logic to one of the Master ports. The Event Command will be added to the high-priority queue and will interrupt normal polling so that this special command can be sent as soon as possible.

**Note:** Overuse of Event Commands may substantially slow or totally disrupt normal polling. Use Event Commands sparingly. Event Commands are meant to be used as one-shot commands triggered by special circumstances or uncommon events.

Write Block - Event Command

Here is the structure of the Event Command output image write block.

Offset	Description	Length (words)
0	Block Number: 1000 to 1255 or 2000 to 2255	1
1	Internal DB Address	1
2	Point Count	1
3	Swap Code	1
4	Function Code	1
5	Device Address	1
6 to ( n )	Spare	

( n ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

The Block Number defines the MODBUS port to be considered. The slave address is represented in the block number in the range of 0 to 255, added to 1000 for commands to be sent out Port 1 or to 2000 for commands to be sent out Port 2. The sum of these two values determines the block number to use.

Example1: To send an Event Command to Slave 34 from Master Port 1, use Block Number 1034.

Example 2; To send an Event Command to Slave 227 from Master Port 2, use Block Number 2227

Note: Slave address 0 is a broadcast address. Commands with this address will affect all slaves on the network. Not all addresses in the range 1 to 255 are valid for all slave devices. This is especially true for addresses 248 through 255.

Use the parameters passed with the block to construct the command.

- The **INTERNAL DB ADDRESS** parameter specifies the module's database location to associate with the command.
- The **POINT COUNT** parameter defines the number of registers for the command.
- The **SWAP CODE** changes the word or byte order.
- The **DEVICE ADDRESS** parameter defines the MODBUS address on the target MODBUS device to consider.
- The **FUNCTION CODE** parameter is one of those defined in the ProSoft MODBUS Command Set documentation.

The parameter fields in the block should be completed as required by the selected function code. Each command type has its own set of parameters. When the block is received, the module will process it and place the command in the command queue. The module will respond to each command block with a input image read block.

Read Block - Event Command

Here is the structure of the Event Command input image read block.

Offset	Description	Length (words)
0	1000 to 1255 or 2000 to 2255	1
1	Write Block ID	1
2	0=Fail, 1=Success	1
3 to n	Spare	

*n=60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

Word two of the block can be used by the ladder logic to determine if the command was added to the command queue of the module. The command will only fail if the command queue for the port is full (100 commands for each queue) or the command requested is invalid.

**5.5.4 Command Control**

During normal operation, the module executes commands from the Command List in the order they appear in the list and only when those commands have the Enable parameter set to a non-zero value. If the Enable parameter is set to zero (0), the command is considered disabled and not sent.

In addition to the Command List, each port also has a high-priority command queue. Transmission preference is given to commands in the queue; therefore, any command in this queue will be sent instead of sending a command from the Command List.

Commands are placed in the queue either by the Event Command blocks (page 125) or by the Special Function blocks. Event Command blocks place commands in the queue one at a time. Special Function blocks can place from one to six commands into the queue each time they are executed

**Note:** Overuse of Special Functions may substantially slow or totally disrupt normal Command List polling. Use Special Functions sparingly. Special Functions are meant to be used when a few pre-configured commands need to be sent as one-shot commands triggered by special circumstances or uncommon events.

When executed from logic, Special Function blocks place commands from anywhere in the Command List directly into the command queue, so that the command or commands will be the next one or ones to be sent. The normal way to use Special Functions is to setup commands in the Command List and set their Enable parameter set to zero (0). This will prevent the commands from being executed until added to the queue by Special Functions logic. However, any command from the command list may be added to the queue, regardless of the value in the Enable parameter. This allows normal polling order to be interrupted and to have up to six commands sent out of sequence, one after the other, before normal polling resumes.

Write Block - Command Control

Here is the structure of the Command Control output image write block.

Offset	Description	Length (words)
0	5001 to 5006 or 5101 to 5106	1
1	Command index	1
2	Command index	1
3	Command index	1
4	Command index	1
5	Command index	1
6	Command index	1
7 to ( n )	Spare	

( n ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

Blocks in the range of 5001 to 5006 are used for Port 1, and blocks in the range of 5101 to 5106 are used for Port 2. The last digit in the block code defines the number of commands to process in the block. For example, a block code of 5003 contains 3 command indexes that are for Port 1. The Command index parameters in the block have a range of 0 to 99 and correspond to Master Command List entries.

The module responds to a Command Control write block with a input image read block containing the number of commands added to the command queue for the port.

Read Block - Command Control

Here is the structure of the Command Control input image read block.

Offset	Description	Length (words)
0	5000 to 5006 or 5100 to 5106	1
1	Write Block ID	1
2	Number of commands added to command queue	1
3 to ( n + 1 )	Spare	

( n ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).



### 5.5.5 Pass-Through Control Blocks

If one or more of the slave ports on the module are configured for formatted pass-through mode, the module will send input image blocks with identification codes of 9956, 9957, 9958 or 9959 to the processor for each write command received. Any MODBUS Function 5, 6, 15 or 16 command will be passed from the port to the processor using a block identification number that identifies the Function Code received in the incoming command. Ladder logic must exist in the process to handle the receipt of all MODBUS write functions and to respond as expected to commands issued by the remote MODBUS Master device.

**Important:** MVI69-MCM modules with firmware version 1.21 and newer cannot use ladder logic written for earlier firmware versions. Please use the ladder logic or Add-On Instruction specifically labeled for your MVI69-MCM module's firmware version. Firmware version 1.21 includes the following changes to the pass-through control blocks:

- Mutual exclusion on Pass-Through Block IDs 9956, 9957, 9958, and 9959 from both ports - If both ports are configured as slave ports, when both of the slave ports receive write commands with the same Function Code, which would need to use the same block identifier from the above list, the module will process the command from the port which first received the command and will return an Exception Code error code 6 (node is busy - retry command later error) from the other port that received the command last. The Master will retry the command on the busy port after a short delay. This prevents Pass-Through blocks on both ports from overwriting each other.
- The Pass-Through Block ID is now written by the module into the first word, the (0) offset, of the processor's backplane input image. Previously this location contained a 0 (zero) value. Ladder logic for earlier firmware versions will not work with MVI69-MCM firmware version 1.21 or later.

***Function 5***

Here is the structure of the input image write block for Formatted Pass-Through Control Block ID 9958.

***For MVI69-MCM firmware versions 1.21 and newer only:***

Offset	Description	Length (words)
0	9958	1
1	9958	1
2	Number of words	1
3	Data Address	1
4 to ( $n + 1$ )	Data	( $n - 2$ )

***For MVI69-MCM firmware versions earlier than 1.21:***

Offset	Description	Length (words)
0	0	1
1	9958	1
2	Length	1
3	Data Address	1
4 to ( $n + 1$ )	Data	( $n - 2$ )

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the pass-through control block with a output image write block with the following format.

Offset	Description	Length (words)
0	9958	1
1 to ( $n$ )	Spare	( $n$ )

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

(  $n$  ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

***Function 6 and 16***

Here is the structure of the input image write block for Formatted Pass-Through Control Block IDs 9956 and 9957.

***For MVI69-MCM firmware versions 1.21 and newer only:***

Offset	Description	Length (words)
0	9956/9957	1
1	9956/9957 (floating-point)	1
2	Length	1
3	Data Address	1
4 to ( $n + 1$ )	Data	( $n - 2$ )

***For MVI69-MCM firmware versions earlier than 1.21:***

Offset	Description	Length (words)
0	0	1
1	9956/9957 (floating-point)	1
2	Number of words	1
3	Data Address	1
4 to ( $n + 1$ )	Data	( $n - 2$ )

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the pass-through control block with an output image write block with the following format.

Offset	Description	Length (words)
0	9956/9957	1
1 to ( $n$ )	Spare	( $n - 2$ )

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

(  $n$  ) = 60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).

***Function 15***

When the module receives a function code 15 when in pass-through mode, the module will write the data using block ID 9959 for multiple-bit data. First the bit mask clears the bits to be updated. This is accomplished by ANDing the inverted mask with the existing data. Next the new data ANDed with the mask is ORed with the existing data. This protects the other bits in the INT registers from being affected. This function can only be used if the Block Transfer Size parameter is set to 120 or 240 words.

***For MVI69-MCM firmware versions 1.21 and newer only:***

Offset	Description	Length (words)
0	9959	1
1	9959	1
2	Number of Words	1
3	Word Address	1
4 to 53	Data	1
54 to 104	Mask. <b>Note:</b> For block transfer size = 60, the mask bits are limited to the count of 6 words (96 bits). Therefore, the practical data size limit is also 96 bits.	50
105 to 180	Spare	50

***For MVI69-MCM firmware versions earlier than 1.21:***

Offset	Description	Length (words)
0	0	1
1	9959	1
2	Write Block ID	1
3	Number of Words	1
4	Word Address	1
5 to 55	Data	50
56 to 105	Mask	50
106 to 180	Spare	15

The ladder logic will be responsible for parsing and copying the received message and performing the proper control operation as expected by the Master device. The processor must then respond to the pass-through control block with a write block with the following format.

Offset	Description	Length (words)
0	9959	1
1 to n	Spare	

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

### 5.5.6 Initialize Output Data

When the module performs a restart operation, it will request blocks of output data from the processor to initialize the module's output data. Use the **INITIALIZE OUTPUT DATA** parameter in the configuration file to bring the module to a known state after a restart operation. The following table describes the structure of the request block.

Offset	Description	Length
0	4000 to 4083 for n = 60	1
1	4000 to 4083 for n = 60	1
2 to n	Spare	n

*n=60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

Ladder logic in the processor must recognize these blocks and place the correct information in the output image to be returned to the module. The format of the returned write block is shown in the following table.

Offset	Description	Length
0	4000 to 4083	1
1 to n	Output Data	n

### 5.5.7 Warm Boot Block (9998)

This block is sent from the CompactLogix or MicroLogix processor to the module (output image) when the module is required to perform a warm-boot (software reset) operation. The following table describes the format of the control block.

Offset	Description	Length (words)
0	9998	1
1 to n	Spare	247

*n=60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

### 5.5.8 Cold Boot Block (9999)

This block is sent from the CompactLogix or MicroLogix processor to the module (output image) when the module is required to perform the cold boot (hardware reset) operation. This block is sent to the module when a hardware problem is detected by the ladder logic that requires a hardware reset. The following table describes the format of the control block.

Offset	Description	Length (words)
0	9999	1
1 to n	Spare	247

*n=60, 120, or 240 depending on the Block Transfer Size parameter (refer to the configuration file).*

## 5.6 Modbus Protocol Specification

The following pages give additional reference information regarding the Modbus protocol commands supported by the MVI69-MCM.

### 5.6.1 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.

### 5.6.2 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.

**5.6.3 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 10218) instead of 8 inputs. The two left-most bits are provided as zeros to fill the 8-bit format.

**5.6.4 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

**5.6.5 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 from 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

**5.6.6 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".

**5.6.7 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

**5.6.8 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 MVI69-MCM module.

**00 Return Query Data**

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

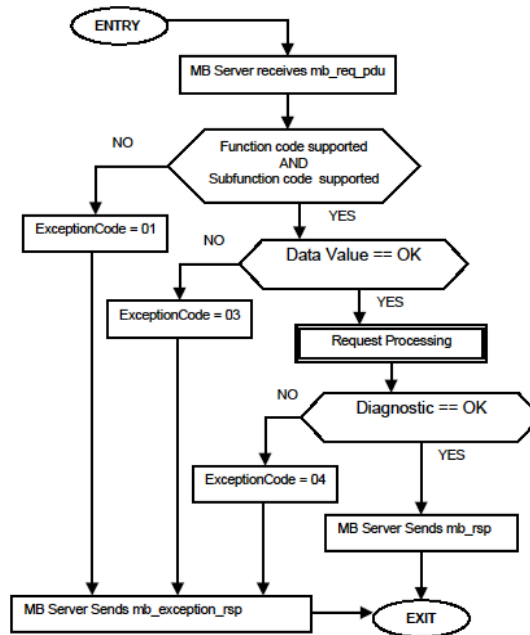
Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

**Example and State Diagram**

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

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

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



### 5.6.9 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.

### 5.6.10 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

### 5.6.11 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 slave device receives the request without a communication error and can handle the query normally, it returns a normal response.
- If the slave 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 slave 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 slave receives the request without a communication error but cannot handle it (for example, if the request is to read a non-existent address or read too many points), the slave will return an *Exception Response* informing the Master of the nature of the error by using a specific *Exception Code* in the response.

An *Exception Response* message has two fields that differentiate it from a normal response:

**Function Code Field:** In a normal response, the slave 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 slave sets the MSB of the *Function Code* byte 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 application program can recognize the *Exception Response* and can examine the data field for the *Exception Code*.

**Data Field:** In a normal response, the slave may return data or statistics in the data field (any information that was requested by the request). In an *Exception Response*, the slave returns an *Exception Code* in the data field. This tells what slave condition caused the exception.

The following table shows an example of a Master request and slave *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 read request to slave device. The Function Code (01) is for a Read Coil Status operation. It requests the status of the output coil at address 1245 (04A1 hex). Note that only that one coil is to be read, as specified by the numbers in the output fields (0001).



If the output address does not exist in the slave device, the slave will return an *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.

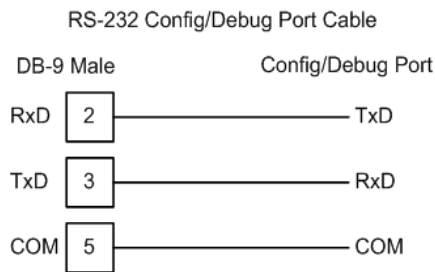
## 5.7 Cable Connections

The application ports on the MVI69-MCM 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.

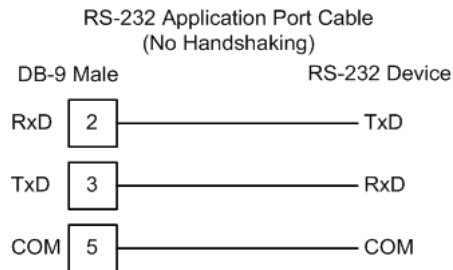
### 5.7.1 RS-232 Configuration/Debug Port

This port is physically an RJ45 connection. An RJ45 to DB-9 adapter cable is included with the module. This port permits a PC based terminal emulation program to view configuration and status data in the module and to control the module. The cable for communications on this port is shown in the following diagram:



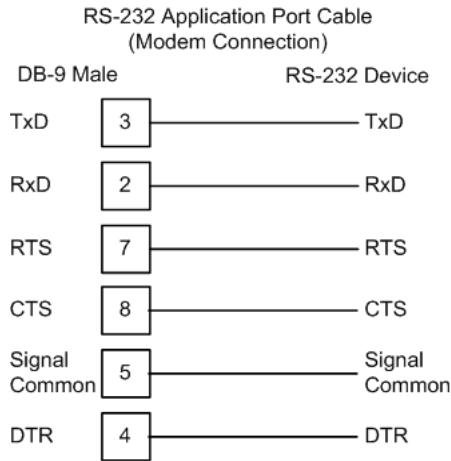
### 5.7.2 RS-232 Application Port(s)

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)**

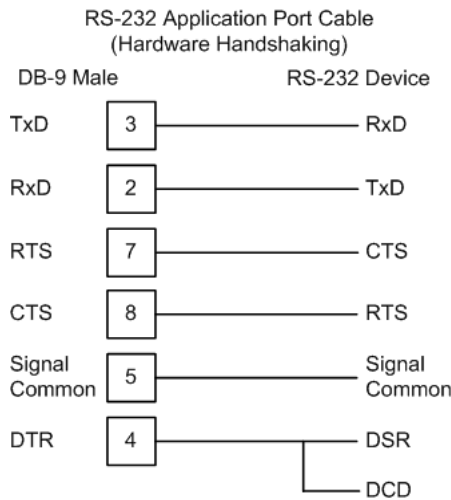
This type of connection is required between the module and a modem or other communication device.



The "Use CTS Line" parameter for the port configuration should be set to 'Y' for most modem applications.

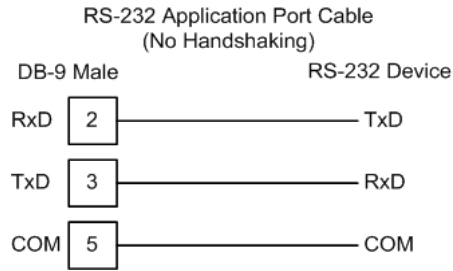
**RS-232: Null Modem Connection (Hardware Handshaking)**

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

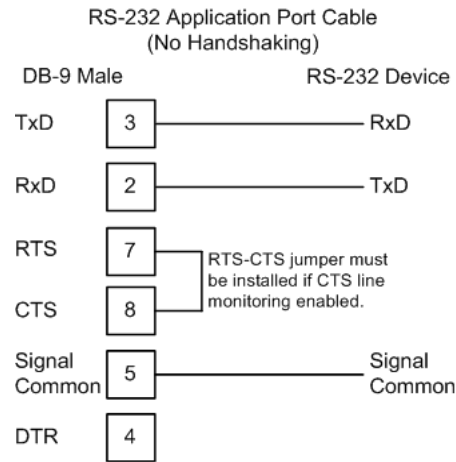


RS-232: Null Modem Connection (No Hardware Handshaking)

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

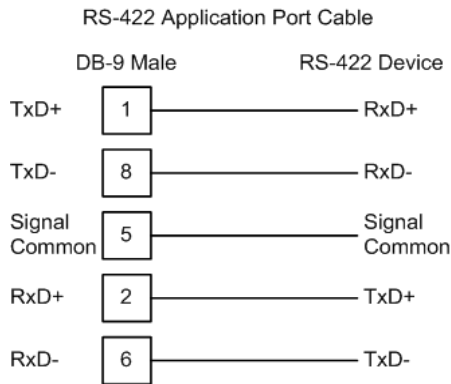


**Note:** For most null modem connections where hardware handshaking is not required, the *Use CTS Line* parameter should be set to **N** and no jumper will be required between Pins 7 (RTS) and 8 (CTS) on the connector. If the port is configured with the *Use CTS Line* set to **Y**, then a jumper is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



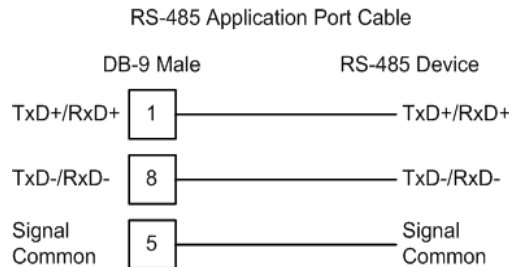
### 5.7.3 RS-422

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:



### 5.7.4 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:

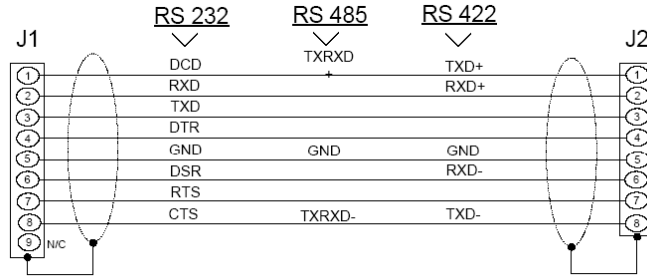
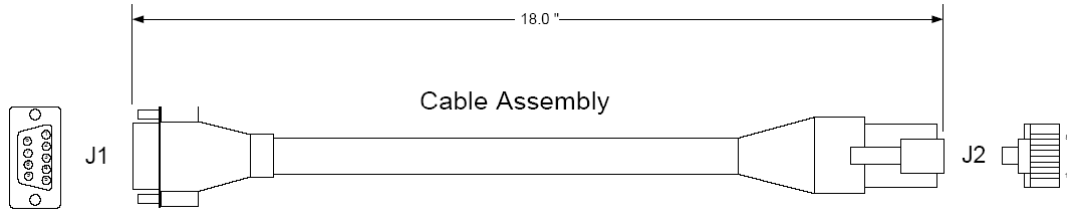


**Note:** Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In these cases, installing a 120-ohm terminating resistor between pins 1 and 8 on the module connector end of the 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.

**5.7.5 DB9 to RJ45 Adaptor (Cable 14)**



Wiring Diagram

## 5.8 MCM Database Definition

This section contains a listing of the internal database of the MVI69-MCM module. This information can be used to interface other devices to the data contained in the module.

Register Range	Content	Size
0 to 4999	User Data	5000
5000 to 5009	Backplane Configuration	10
5010 to 5039	Port 1 Setup	30
5040 to 5069	Port 2 Setup	30
5070 to 5869	Port 1 Commands	800
5870 to 6669	Port 2 Commands	800
6670 to 6702	Misc. Status Data	32
6703 to 6749	Reserved	
6750 to 6759	Port 1 Status Data	10
6760 to 6769	Port 2 Status Data	10

The User Data area holds data collected from other nodes on the network (master read commands) or data received from the processor (write blocks). Additionally, this data area is used as a data source for the processor (read blocks) or other nodes on the network (write commands).

## 5.9 Status Data Definition

This section contains a description of the members present in the **MCM1STATUS** object. This data is transferred from the module to the processor as part of each read block.

### Status Data Block Structure

Offset	Content	Description
6670	Program Scan Count	This value is incremented each time a complete program cycle occurs in the module.
6671 to 6672	Product Code	These two registers contain the product code of "MCM"
6673 to 6674	Product Version	These two registers contain the product version for the current running software.
6675 to 6676	Operating System	These two registers contain the month and year values for the program operating system.
6677 to 6678	Run Number	These two registers contain the run number value for the currently running software.
6679	Port 1 Command List Requests	This field contains the number of requests made from this port to slave devices on the network.
6680	Port 1 Command List Response	This field contains the number of slave response messages received on the port.
6681	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.
6682	Port 1 Requests	This field contains the total number of messages sent out of the port.
6683	Port 1 Responses	This field contains the total number of messages received on the port.
6684	Port 1 Errors Sent	This field contains the total number of message errors sent out of the port.
6685	Port 1 Errors Received	This field contains the total number of message errors received on the port.
6686	Port 2 Command List Requests	This field contains the number of requests made from this port to slave devices on the network.
6687	Port 2 Command List Response	This field contains the number of slave response messages received on the port.
6688	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.
6689	Port 2 Requests	This field contains the total number of messages sent out the port.
6690	Port 2 Responses	This field contains the total number of messages received on the port.
6691	Port 2 Errors Sent	This field contains the total number of message errors sent out of the port.
6692	Port 2 Errors Received	This field contains the total number of message errors received on the port.
6693	Read Block Count	This field contains the total number of read blocks transferred from the module to the processor.
6694	Write Block Count	This field contains the total number of write blocks transferred from the processor to the module.



<b>Offset</b>	<b>Content</b>	<b>Description</b>
6695	Parse Block Count	This field contains the total number of blocks successfully parsed that were received from the processor.
6696	Command Event Block Count	This field contains the total number of command event blocks received from the processor.
6697	Command Block Count	This field contains the total number of command blocks received from the processor.
6698	Error Block Count	This field contains the total number of block errors recognized by the module.
6699	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.
6700	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 an error.
6701	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.
6702	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 Support, Service & Warranty

### In This Chapter

- ❖ Contacting Technical Support ..... 155
- ❖ Return Material Authorization (RMA) Policies and Conditions..... 157
- ❖ LIMITED WARRANTY..... 159

### **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.

---

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

---

## 6.1 Return Material Authorization (RMA) Policies and Conditions

The following Return Material Authorization (RMA) Policies and Conditions (collectively, "RMA Policies") apply to any returned product. These RMA Policies are subject to change by ProSoft Technology, Inc., without notice. For warranty information, see Limited Warranty (page 159). In the event of any inconsistency between the RMA Policies and the Warranty, the Warranty shall govern.

### 6.1.1 Returning Any Product

- a) In order to return a Product for repair, exchange, or otherwise, the Customer must obtain a Return Material Authorization (RMA) number from ProSoft Technology and comply with ProSoft Technology shipping instructions.
- b) In the event that the Customer experiences a problem with the Product for any reason, Customer should contact ProSoft Technical Support at one of the telephone numbers listed above (page 155). A Technical Support Engineer will request that you perform several tests in an attempt to isolate the problem. If after completing these tests, the Product is found to be the source of the problem, we will issue an RMA.
- c) All returned Products must be shipped freight prepaid, in the original shipping container or equivalent, to the location specified by ProSoft Technology, and be accompanied by proof of purchase and receipt date. The RMA number is to be prominently marked on the outside of the shipping box. Customer agrees to insure the Product or assume the risk of loss or damage in transit. Products shipped to ProSoft Technology using a shipment method other than that specified by ProSoft Technology, or shipped without an RMA number will be returned to the Customer, freight collect. Contact ProSoft Technical Support for further information.
- d) A 10% restocking fee applies to all warranty credit returns, whereby a Customer has an application change, ordered too many, does not need, etc. Returns for credit require that all accessory parts included in the original box (i.e.; antennas, cables) be returned. Failure to return these items will result in a deduction from the total credit due for each missing item.

### **6.1.2 Returning Units Under Warranty**

A Technical Support Engineer must approve the return of Product under ProSoft Technology's Warranty:

- a) A replacement module will be shipped and invoiced. A purchase order will be required.
- b) Credit for a product under warranty will be issued upon receipt of authorized product by ProSoft Technology at designated location referenced on the Return Material Authorization
  - i. If a defect is found and is determined to be customer generated, or if the defect is otherwise not covered by ProSoft Technology's warranty, there will be no credit given. Customer will be contacted and can request module be returned at their expense;
  - ii. If defect is customer generated and is repairable, customer can authorize ProSoft Technology to repair the unit by providing a purchase order for 30% of the current list price plus freight charges, duties and taxes as applicable.

### **6.1.3 Returning Units Out of Warranty**

- a) Customer sends unit in for evaluation to location specified by ProSoft Technology, freight prepaid.
- b) If no defect is found, Customer will be charged the equivalent of \$100 USD, plus freight charges, duties and taxes as applicable. A new purchase order will be required.
- c) If unit is repaired, charge to Customer will be 30% of current list price (USD) plus freight charges, duties and taxes as applicable. A new purchase order will be required or authorization to use the purchase order submitted for evaluation fee.

**The following is a list of non-repairable units:**

- 3150 - All
- 3750
- 3600 - All
- 3700
- 3170 - All
- 3250
- 1560 - Can be repaired, only if defect is the power supply
- 1550 - Can be repaired, only if defect is the power supply
- 3350
- 3300
- 1500 - All

## 6.2 LIMITED WARRANTY

This Limited Warranty ("Warranty") governs all sales of hardware, software, and other products (collectively, "Product") manufactured and/or offered for sale by ProSoft Technology, Incorporated (ProSoft), and all related services provided by ProSoft, including maintenance, repair, warranty exchange, and service programs (collectively, "Services"). By purchasing or using the Product or Services, the individual or entity purchasing or using the Product or Services ("Customer") agrees to all of the terms and provisions (collectively, the "Terms") of this Limited Warranty. All sales of software or other intellectual property are, in addition, subject to any license agreement accompanying such software or other intellectual property.

### 6.2.1 What Is Covered By This Warranty

- a) *Warranty On New Products:* ProSoft warrants, to the original purchaser, that the Product that is the subject of the sale will (1) conform to and perform in accordance with published specifications prepared, approved and issued by ProSoft, and (2) will be free from defects in material or workmanship; provided these warranties only cover Product that is sold as new. This Warranty expires three (3) years from the date of shipment for Product purchased **on or after** January 1st, 2008, or one (1) year from the date of shipment for Product purchased **before** January 1st, 2008 (the "Warranty Period"). If the Customer discovers within the Warranty Period a failure of the Product to conform to specifications, or a defect in material or workmanship of the Product, the Customer must promptly notify ProSoft by fax, email or telephone. In no event may that notification be received by ProSoft later than 39 months from date of original shipment. Within a reasonable time after notification, ProSoft will correct any failure of the Product to conform to specifications or any defect in material or workmanship of the Product, with either new or remanufactured replacement parts. ProSoft reserves the right, and at its sole discretion, may replace unrepairable units with new or remanufactured equipment. All replacement units will be covered under warranty for the 3 year period commencing from the date of original equipment purchase, not the date of shipment of the replacement unit. Such repair, including both parts and labor, will be performed at ProSoft's expense. All warranty service will be performed at service centers designated by ProSoft.
- b) *Warranty On Services:* Materials and labor performed by ProSoft to repair a verified malfunction or defect are warranted in the terms specified above for new Product, provided said warranty will be for the period remaining on the original new equipment warranty or, if the original warranty is no longer in effect, for a period of 90 days from the date of repair.

### **6.2.2 What Is Not Covered By This Warranty**

- a) ProSoft makes no representation or warranty, expressed or implied, that the operation of software purchased from ProSoft will be uninterrupted or error free or that the functions contained in the software will meet or satisfy the purchaser's intended use or requirements; the Customer assumes complete responsibility for decisions made or actions taken based on information obtained using ProSoft software.
- b) This Warranty does not cover the failure of the Product to perform specified functions, or any other non-conformance, defects, losses or damages caused by or attributable to any of the following: (i) shipping; (ii) improper installation or other failure of Customer to adhere to ProSoft's specifications or instructions; (iii) unauthorized repair or maintenance; (iv) attachments, equipment, options, parts, software, or user-created programming (including, but not limited to, programs developed with any IEC 61131-3, "C" or any variant of "C" programming languages) not furnished by ProSoft; (v) use of the Product for purposes other than those for which it was designed; (vi) any other abuse, misapplication, neglect or misuse by the Customer; (vii) accident, improper testing or causes external to the Product such as, but not limited to, exposure to extremes of temperature or humidity, power failure or power surges; or (viii) disasters such as fire, flood, earthquake, wind and lightning.
- c) The information in this Agreement is subject to change without notice. ProSoft shall not be liable for technical or editorial errors or omissions made herein; nor for incidental or consequential damages resulting from the furnishing, performance or use of this material. The user guide included with your original product purchase from ProSoft contains information protected by copyright. No part of the guide may be duplicated or reproduced in any form without prior written consent from ProSoft.

### **6.2.3 Disclaimer Regarding High Risk Activities**

Product manufactured or supplied by ProSoft is not fault tolerant and is not designed, manufactured or intended for use in hazardous environments requiring fail-safe performance including and without limitation: the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly or indirectly to death, personal injury or severe physical or environmental damage (collectively, "high risk activities"). ProSoft specifically disclaims any express or implied warranty of fitness for high risk activities.



#### **6.2.4 Intellectual Property Indemnity**

Buyer shall indemnify and hold harmless ProSoft and its employees from and against all liabilities, losses, claims, costs and expenses (including attorney's fees and expenses) related to any claim, investigation, litigation or proceeding (whether or not ProSoft is a party) which arises or is alleged to arise from Buyer's acts or omissions under these Terms or in any way with respect to the Products. Without limiting the foregoing, Buyer (at its own expense) shall indemnify and hold harmless ProSoft and defend or settle any action brought against such Companies to the extent based on a claim that any Product made to Buyer specifications infringed intellectual property rights of another party. ProSoft makes no warranty that the product is or will be delivered free of any person's claiming of patent, trademark, or similar infringement. The Buyer assumes all risks (including the risk of suit) that the product or any use of the product will infringe existing or subsequently issued patents, trademarks, or copyrights.

- a) Any documentation included with Product purchased from ProSoft is protected by copyright and may not be duplicated or reproduced in any form without prior written consent from ProSoft.
- b) ProSoft's technical specifications and documentation that are included with the Product are subject to editing and modification without notice.
- c) Transfer of title shall not operate to convey to Customer any right to make, or have made, any Product supplied by ProSoft.
- d) Customer is granted no right or license to use any software or other intellectual property in any manner or for any purpose not expressly permitted by any license agreement accompanying such software or other intellectual property.
- e) Customer agrees that it shall not, and shall not authorize others to, copy software provided by ProSoft (except as expressly permitted in any license agreement accompanying such software); transfer software to a third party separately from the Product; modify, alter, translate, decode, decompile, disassemble, reverse-engineer or otherwise attempt to derive the source code of the software or create derivative works based on the software; export the software or underlying technology in contravention of applicable US and international export laws and regulations; or use the software other than as authorized in connection with use of Product.
- f) **Additional Restrictions Relating To Software And Other Intellectual Property**

In addition to compliance with the Terms of this Warranty, Customers purchasing software or other intellectual property shall comply with any license agreement accompanying such software or other intellectual property. Failure to do so may void this Warranty with respect to such software and/or other intellectual property.

#### **6.2.5 Disclaimer of all Other Warranties**

The Warranty set forth in What Is Covered By This Warranty (page 159) are in lieu of all other warranties, express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

### **6.2.6 Limitation of Remedies \*\***

In no event will ProSoft or its Dealer be liable for any special, incidental or consequential damages based on breach of warranty, breach of contract, negligence, strict tort or any other legal theory. Damages that ProSoft or its Dealer will not be responsible for include, but are not limited to: Loss of profits; loss of savings or revenue; loss of use of the product or any associated equipment; loss of data; cost of capital; cost of any substitute equipment, facilities, or services; downtime; the claims of third parties including, customers of the Purchaser; and, injury to property.

\*\* Some areas do not allow time limitations on an implied warranty, or allow the exclusion or limitation of incidental or consequential damages. In such areas, the above limitations may not apply. This Warranty gives you specific legal rights, and you may also have other rights which vary from place to place.

### **6.2.7 Time Limit for Bringing Suit**

Any action for breach of warranty must be commenced within 39 months following shipment of the Product.

### **6.2.8 No Other Warranties**

Unless modified in writing and signed by both parties, this Warranty is understood to be the complete and exclusive agreement between the parties, suspending all oral or written prior agreements and all other communications between the parties relating to the subject matter of this Warranty, including statements made by salesperson. No employee of ProSoft or any other party is authorized to make any warranty in addition to those made in this Warranty. The Customer is warned, therefore, to check this Warranty carefully to see that it correctly reflects those terms that are important to the Customer.

### **6.2.9 Allocation of Risks**

This Warranty allocates the risk of product failure between ProSoft and the Customer. This allocation is recognized by both parties and is reflected in the price of the goods. The Customer acknowledges that it has read this Warranty, understands it, and is bound by its Terms.

### ***6.2.10 Controlling Law and Severability***

This Warranty shall be governed by and construed in accordance with the laws of the United States and the domestic laws of the State of California, without reference to its conflicts of law provisions. If for any reason a court of competent jurisdiction finds any provisions of this Warranty, or a portion thereof, to be unenforceable, that provision shall be enforced to the maximum extent permissible and the remainder of this Warranty shall remain in full force and effect. Any cause of action with respect to the Product or Services must be instituted in a court of competent jurisdiction in the State of California.



## Index

### I

[Backplane 69] • 50  
[MCM Port x] • 53  
[Modbus Port x Commands] • 59  
[Module] • 50

### O

00 Return Query Data • 140

### A

About the MODBUS Protocol • 106  
Adding Multiple Modules (Optional) • 31  
Adding the Module to an Existing CompactLogix Project • 20, 75  
Adding the Module to an Existing MicroLogix Project • 79  
Allocation of Risks • 162

### B

Backplane Data Transfer • 107  
Backplane Fail Count • 52, 72  
Backplane Menu • 92  
Battery Life Advisory • 3  
Baud Rate • 55  
Bit Input Offset • 56  
Block Request from the Processor to the Module • 115  
Block Response from the Module to the Processor • 115  
Block Transfer Size • 52

### C

Cable Connections • 146  
Clearing a Fault Condition • 83  
Clearing Diagnostic Data • 89  
Cold Boot Block (9999) • 121, 133  
Command Control • 121, 127  
Command Count • 57  
Command Error Pointer • 57  
Command List Entry Errors • 114  
Command List Overview • 59  
Commands Supported by the Module • 134  
Configuring Module Parameters • 48  
Configuring the Floating Point Data Transfer • 61  
Configuring the MVI69-MCM Module • 19  
Configuring the RSLinx Driver for the PC COM Port • 39  
Connect your PC to the Module • 45  
Connecting Your PC to the Processor • 39  
Contacting Technical Support • 155, 157  
Controlling Law and Severability • 163  
Create a new RSLogix5000 project • 20

Create the Module • 21

### D

Data Analyzer • 96  
Data Bits • 55  
Data Flow between MVI69-MCM Module and CompactLogix or MicroLogix Processor • 110  
Database View Menu • 89, 90  
DB9 to RJ45 Adaptor (Cable 14) • 150  
Diagnostics (Function Code 08) • 140  
Diagnostics and Troubleshooting • 9, 80, 81  
Disabling the RSLinx Driver for the Com Port on the PC • 43  
Disclaimer of all Other Warranties • 161  
Disclaimer Regarding High Risk Activities • 160  
Displaying the Current Page of Registers Again • 90  
Displaying Timing Marks in the Data Analyzer • 97  
Download the Sample Program to the Processor • 39  
Downloading the Project to the Module Using a Serial COM port • 49  
Downloading to the Processor • 41

### E

Enable • 53, 65  
ENRON Floating Point Support • 61  
Error Delay Counter • 58, 121  
Error/Status Block Pointer • 52, 72  
Event Command • 121, 125, 127  
Example and State Diagram • 141  
Exiting the Program • 89

### F

Float Flag • 54  
Float Offset • 54  
Float Start • 54  
Floating Point Support • 60  
Force Multiple Coils (Function Code 15) • 142  
Force Single Coil (Function Code 05) • 138  
Function 15 • 132  
Function 5 • 130  
Function 6 and 16 • 131  
Functional Overview • 9, 106  
Functional Specifications • 105

### G

General Specifications • 102  
General Specifications - Modbus Master/Slave • 104  
Guide to the MVI69-MCM User Manual • 9

### H

Hardware Specifications • 103  
Hold Offset • 57  
How to Contact Us • 2

### I

If Block Transfer Size = 120 • 118  
If Block Transfer Size = 240 • 119

If Block Transfer Size = 60 • 117  
Import the Ladder Rung • 23  
Important Installation Instructions • 3  
Initialize Output Data • 121, 133  
Initializing Output Data • 52  
Install the Module in the Rack • 16  
Installing ProSoft Configuration Builder Software • 14  
Intellectual Property Indemnity • 161  
Internal Address • 66

## K

Keystrokes • 87

## L

Ladder Logic • 69  
Ladder Logic and Firmware Compatibility Note • 70  
LED Status Indicators • 82  
Limitation of Remedies \*\* • 162  
LIMITED WARRANTY • 157, 159

## M

Main Logic Loop • 107  
Main Menu • 88  
Markings • 4  
Master Command Error List Menu • 94  
Master Command List • 113  
Master Driver Mode • 112  
MB Address in Device • 68  
MCM Database Definition • 151  
Minimum Command Delay • 57  
Minimum Response Delay • 56  
Modbus Command Configuration • 59  
Modbus Exception Codes • 145  
MODBUS Exception Responses • 144  
Modbus Func • 67  
MODBUS Message Data • 74  
Modbus Protocol Specification • 134  
Module Communication Error Codes • 113  
Module Configuration • 50  
Module Data Object (MCM1ModuleDef) • 71  
Module Name • 50  
Module Power Up • 106  
Moving Back Through 5 Pages of Commands • 94  
Moving Back Through 5 Pages of Registers • 91  
Moving Forward (Skipping) Through 5 Pages of Commands • 95  
Moving Forward (Skipping) Through 5 Pages of Registers • 91  
MVI (Multi Vendor Interface) Modules • 3  
MVI69-MCM Sample Add-On Instruction Import Procedure • 20, 75

## N

Navigation • 87  
No Other Warranties • 162  
Node Address • 67  
Normal Data Transfer • 115

## O

Opening the Backplane Menu • 89  
Opening the Command List Menu • 94  
Opening the Data Analyzer Menu • 96  
Opening the Database View Menu • 89  
Opening the Protocol Serial Menu • 89  
Opening the Serial Port Menu • 94  
Output Offset • 57

## P

Package Contents • 13  
Parity • 55  
Pass-Through Control Blocks • 53, 74, 121, 129  
Pinouts • 3, 146, 150  
Poll Interval • 66  
Preset Multiple Registers (Function Code 16) • 143  
Preset Single Register (Function Code 06) • 139  
Printing a Configuration File • 48  
Product Specifications • 9, 102  
ProSoft Technology® Product Documentation • 2  
Protocol • 54  
Protocol Serial MCM Menu • 93

## R

Read Block - Command Control • 128  
Read Block - Disable Slaves • 123  
Read Block - Enable Slaves • 123  
Read Block - Event Command • 127  
Read Block - Read Slave Status • 125  
Read Block and Write Block Transfer Sequences • 116  
Read Coil Status (Function Code 01) • 134  
Read Holding Registers (Function Code 03) • 136  
Read Input Registers (Function Code 04) • 137  
Read Input Status (Function Code 02) • 135  
Read Register Count • 51  
Read Register Start • 51  
Reading Status Data from the Module • 99  
Receiving the Configuration File • 89  
Redisplaying the Current Page • 94  
Redisplaying the Menu • 88, 92, 93, 95, 96  
Reference • 9, 101  
Reg Count • 66  
Removing Timing Marks in the Data Analyzer • 97  
Renaming PCB Objects • 48  
Response Timeout • 58, 121  
Retry Count • 58, 121  
Return Material Authorization (RMA) Policies and Conditions • 157  
Returning Any Product • 157  
Returning to the Main Menu • 91, 92, 94, 95, 98  
Returning Units Out of Warranty • 158  
Returning Units Under Warranty • 158  
RS-232  
Modem Connection (Hardware Handshaking Required) • 147  
Null Modem Connection (Hardware Handshaking) • 147

Null Modem Connection (No Hardware Handshaking) • 148  
RS-232 Application Port(s) • 146  
RS-232 Configuration/Debug Port • 146  
RS-422 • 149  
RS-485 and RS-422 Tip • 149  
RS-485 Application Port(s) • 149  
RTS Off • 56  
RTS On • 55

## S

Sending the Configuration File • 89  
Serial Port Menu • 95  
Set the Block Transfer Parameter Size • 29  
Set the Connection Input Size Values • 30  
Set the Read/Write Data Lengths • 27  
Setting Jumpers • 15  
Setting Up the Project • 46  
Slave Address • 56  
Slave Disable and Enable Control Blocks • 121  
Slave Driver • 110  
Slave Polling Control and Status • 73  
Slave Status Blocks • 121, 124  
Special Control and Status Blocks • 121  
Standard MODBUS Protocol Exception Code Errors • 113  
Start Here • 9, 11  
Starting the Data Analyzer • 98  
Status Data Block (Read Block ID = 0) • 119  
Status Data Definition • 152  
Status Object (MCM1Status) • 72  
Stop Bits • 55  
Stopping the Data Analyzer • 98  
Sub-function Codes Supported • 140  
Support, Service & Warranty • 9, 155  
Swap Code • 67  
System Requirements • 12

## T

Time Limit for Bringing Suit • 162  
Transferring the Command Error List to the Processor • 113  
Troubleshooting • 84  
Type • 53

## U

Use CTS Line • 56  
User Data Objects • 73  
Using ProSoft Configuration Builder • 46  
Using ProSoft Configuration Builder (PCB) for Diagnostics • 85  
Using the Diagnostic Window in ProSoft Configuration Builder • 85

## V

Viewing Backplane Diagnostic Information • 93  
Viewing Configuration Information • 92, 94  
Viewing Data in ASCII (Text) Format • 91, 97

Viewing Data in Decimal Format • 9, 91  
Viewing Data in Floating-Point Format • 91  
Viewing Data in Hexadecimal Format • 91, 97  
Viewing Error and Status Data • 94  
Viewing Register Pages • 90  
Viewing the Next Page of Commands • 95  
Viewing the Next Page of Registers • 91  
Viewing the Previous Page of Commands • 94  
Viewing the Previous Page of Registers • 91  
Viewing Version Information • 88, 92, 93, 95

## W

Warm Boot Block (9998) • 121, 133  
Warm Booting the Module • 89, 121  
Warnings • 3  
What Is Covered By This Warranty • 159, 161  
What Is Not Covered By This Warranty • 160  
Word Input Offset • 56  
Write Block - Command Control • 128  
Write Block - Disable Slaves • 122  
Write Block - Enable Slaves • 123  
Write Block - Event Command • 126  
Write Block - Request Slave Status • 124  
Write Register Count • 52  
Write Register Start • 51

## Y

Your Feedback Please • 2