ProSoft TECHNOLOGY

Where Automation Connects.

inRAx®

# MVI56-MCM

**ControlLogix Platform**

Modbus Communication Module

June 23, 2009

USER MANUAL

## 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, and
**C** WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NONHAZARDOUS.
**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'EQUIPMENT, COUPER LE COURANT OU S'ASSURER QUE L'EMPLACEMENT EST DÉSIGNÉ NON DANGEREUX.

CL I Div 2 GPs A, B, C, D

Temp Code T5

II 3 G

Ex nA IIC T5 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.

## 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 nonhazardous.
**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.

### Electrical Ratings

- Backplane Current Load: 800 mA @ 5 V DC; 3mA @ 24V DC
- Operating Temperature: 0 to 60°C (32 to 140°F)
- Storage Temperature: -40 to 85°C (-40 to 185°F)
- Shock: 30g Operational; 50g non-operational; Vibration: 5 g from 10 to 150 Hz
- Relative Humidity 5% to 95% (non-condensing)
- All phase conductor sizes must be at least 1.3 mm(squared) and all earth ground conductors must be at least 4mm(squared).

### Markings:

| | |
|---|---|
| ANSI / ISA | ISA 12.12.01 Class I Division 2, GPs A, B, C, D |
| CSA/cUL | C22.2 No. 213-1987 |
| CSA CB Certified | IEC61010 |
| ATEX | EN60079-0 Category 3, Zone 2<br>EN60079-15 |

243333

## Battery Life Advisory

All modules in the MVI series use a rechargeable Lithium Vanadium Pentoxide battery to backup the 512K SRAM memory, 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 it becomes fully charged. After it is fully charged, the battery provides backup power for the CMOS setup and configuration data, the real-time clock, and the 512K SRAM memory for approximately 21 days.

Before you remove a module from its power source, ensure that the battery within the module is fully charged. A fully charged battery will hold the BIOS settings (after being removed from its power source) for a limited number of days. When the battery is fully discharged, the module will revert to the default BIOS settings.

Note: The battery is not user replaceable.

## 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 the product, documentation or support, please write or call us.

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

Copyright © ProSoft Technology, Inc. 2009. All Rights Reserved.

MVI56-MCM User Manual
June 23, 2009

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

## ProSoft Technology® Product Documentation

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

Printed documentation is available for purchase. Contact ProSoft Technology for pricing and availability.

Asia Pacific: +603.7724.2080

Europe, Middle East, Africa: +33 (0) 5.3436.87.20

Latin America: +1.281.298.9109

North America: +1.661.716.5100

# Contents

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

**Guide to the MVI56-MCM User Manual 7**

**1 Start Here 9**

1.1 System Requirements ..... 9
1.2 Deployment Checklist ..... 10
1.3 Package Contents ..... 12
1.4 Setting Jumpers ..... 13
1.5 Install the Module in the Rack ..... 14
1.6 Connect your PC to the ControlLogix Processor ..... 15

**2 Using the RSLogix 5000 v16 Add-On Instruction 17**

2.1 Create a new RSLogix 5000 project ..... 18
2.2 Download the Sample Program to the Processor ..... 29

**3 Configuration as a Modbus Master 33**

3.1 Overview ..... 33
3.2 ModDef Settings ..... 34
3.3 Master Command Examples ..... 41
3.4 Floating-Point Data Handling ..... 46
3.5 Command Control and Event Command ..... 52

**4 Configuration as a Modbus Slave 55**

4.1 Overview ..... 55
4.2 ModDef Settings ..... 56
4.3 Read and Write Same Modbus Address (Pass Through) ..... 60
4.4 Slave Configuration ..... 62
4.5 Floating-Point Data Handling ..... 63

**5 Verify Communication 67**

5.1 MVI56-MCM Status Data Definition as a Master ..... 67
5.2 Verify Master Communications ..... 68
5.3 Verify Slave Communications ..... 72

**6 Diagnostics and Troubleshooting 73**

6.1 Reading Status Data from the Module ..... 73
6.2 LED Status Indicators ..... 85

**7 Reference 89**

7.1 Product Specifications ..... 89
7.2 Functional Overview ..... 91
7.3 Cable Connections ..... 110
7.4 MVI56-MCM Database Definition ..... 114
7.5 MVI56-MCM Configuration Data ..... 115
7.6 MVI56-MCM Status Data Definition ..... 123
7.7 MVI56-MCM Command Control ..... 124
7.8 MVI56-MCM User Defined Data Types ..... 125
7.9 Modbus Protocol Specification ..... 130
7.10 Using the Sample Program - RSLogix Version 15 and earlier ..... 141

**8 Support, Service & Warranty 155**

8.1 How to Contact Us: Technical Support ..... 155
8.2 Return Material Authorization (RMA) Policies and Conditions ..... 156
8.3 LIMITED WARRANTY ..... 157

**Index 163**

# Guide to the MVI56-MCM User Manual

| Function | | Section to Read | Details |
|---|---|---|---|
| Introduction<br>(Must Do) | → | Start Here (page 9) | This Section introduces the customer to the module. Included are: package contents, system requirements, hardware installation, and basic configuration. |
| Verify Communication, Diagnostic and Troubleshooting | → | Verifying Communication (page 67)<br>Diagnostics and Troubleshooting (page 73) | This section describes how to verify communications with the network. Diagnostic and Troubleshooting procedures. |
| Reference<br>Product Specifications<br>Functional Overview | → | Reference (page 89)<br>Functional Overview (page 91)<br>Product Specifications (page 89) | These sections contain general references associated with this product, Specifications, and the Functional Overview. |
| Support, Service, and Warranty<br>Index | → | Support, Service and Warranty (page 155) | This section contains Support, Service and Warranty information.<br>Index of chapters. |

# 1 Start Here

*In This Chapter*


- System Requirements ..... 9
- Deployment Checklist ..... 10
- Package Contents ..... 12
- Setting Jumpers ..... 13
- Install the Module in the Rack ..... 14
- Connect your PC to the ControlLogix Processor ..... 15


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 Master/Slave and ControlLogix devices to a power source and to the MVI56-MCM module's application ports.

Caution: You must be able to complete the application without exposing personnel or equipment to unsafe or inappropriate working conditions.

## 1.1 System Requirements

The MVI56-MCM module requires the following minimum hardware and software components:

- Rockwell Automation ControlLogix™ processor, with compatible power supply and one free slot in the rack, for the MVI56-MCM module. The module requires 800mA of available power.
- Rockwell Automation RSLogix 5000 programming software version 2.51 or higher.
- 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
- ProSoft Configuration Builder, HyperTerminal or other terminal emulator program.

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

## 1.2 Deployment Checklist

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

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

Most applications can use the Sample Ladder Logic without any edits to the Sample Program.

**2** _____ Which slot number in the chassis will the MVI56-MCM module occupy?

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

**3** _____ Are RSLogix 5000 and RSLinx installed?

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

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

The MVI56-MCM module can transfer a maximum of 5000 (16-bit) registers to/from the ControlLogix processor. The Sample Ladder transfers 600 words to the ControlLogix processor (into the Read Data array) and obtains 600 words from the ControlLogix processor (from the Write Data array).

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

Modbus is a master/slave network. Only one master is allowed on the Com line (max 32 devices/RS485). The Master is responsible for polling data from the slaves on the network.

**6** _____ For a Modbus Master, what devices (node ID) and Modbus addresses do you need to exchange data with on the Modbus network?

As a Modbus master, you must know the node ID # of the slave devices you wish to obtain data from, as well as the Modbus address (coil 0001, register 4001 and so on) of the data that must be read from or written to that slave device.

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

The MVI56-MCM module can send data to a Modbus master as 0x coil data, 1x input coil data, 3x input registers and 4x holding registers. The sample program transfers 600 (16-bit) words or 9600 bits to the ControlLogix processor, and 600w or 18 bits from the ControlLogix processor.

**8** Serial Communication Parameters for the Modbus network:

_____ Baud rate?
_____ Data bits?
_____ Parity?
_____ Stop bits?
Required for master and slave configurations.

**9** _____ Wiring type to be used (RS232, 422 or 485). Set by jumper settings (page 13).

Required for proper implementation of the module in master and slave configurations.

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

- For version 16 or newer of RSLogix 5000, refer to Using the RSLogix 5000 v16 Add-On Instruction (page 17).
- For NEW system installations, refer to Sample Ladder Logic in New Application.
- For EXISTING system installations, refer to Integrating the Sample Ladder Logic into an Existing Project (page 147).

Note: Most applications can use the Sample Ladder Logic *without any edits to the sample program.*

## 1.3 Package Contents

The following components are included with your MVI56-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.

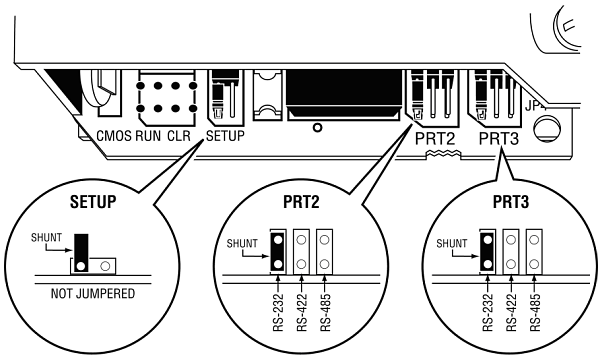
| Qty. | Part Name | Part Number | Part Description |
|---|---|---|---|
| 1 | MVI56-MCM Module | MVI56-MCM | Modbus Communication Module |
| 1 | Cable | Cable #15, RS232 Null Modem | For RS232 Connection to the CFG Port |
| 3 | Cable | Cable #14, RJ45 to DB9 Male Adapter cable | For DB9 Connection to 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 MVI56-MCM module. |

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

## 1.4 Setting Jumpers

If you use an interface other than RS-232 (default), you must change the jumper configuration to match the interface. There are three jumpers located at the bottom of the module.

The following illustration shows the MVI56-MCM jumper configuration:



1. Set the PRT 2 (for application port 1) and PRT 3 (for application port 2) jumpers for RS232, RS422, or RS485 to match the wiring needed for your application. The default jumper setting for both application ports is RS-232.
2. 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

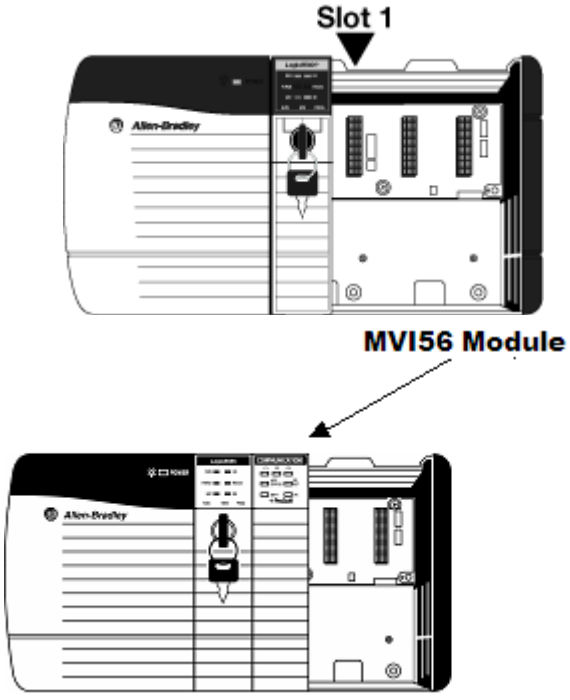
If you have not already installed and configured your ControlLogix processor and power supply, please do so before installing the MVI56-MCM module. Refer to your Rockwell Automation product documentation for installation instructions.

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

After you have checked the placement of the jumpers, insert MVI56-MCM into the ControlLogix chassis. Use the same technique recommended by Rockwell Automation to remove and install ControlLogix modules.

Warning: When you insert or remove the module while backplane power is on, an electrical arc can occur. This could cause an explosion in hazardous location installations. Verify that power is removed or the area is non-hazardous before proceeding. Repeated electrical arcing causes excessive wear to contacts on both the module and its mating connector. Worn contacts may create electrical resistance that can affect module operation.

1. Turn power OFF.
2. Align the module with the top and bottom guides, and slide it into the rack until the module is firmly against the backplane connector.




3. With a firm but steady push, snap the module into place.
4. Check that the holding clips on the top and bottom of the module are securely in the locking holes of the rack.

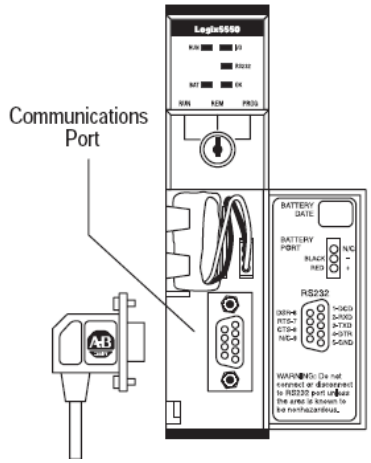
5 Make a note of the slot location. You must identify the slot in which the module is installed in order for the sample program to work correctly. Slot numbers are identified on the green circuit board (backplane) of the ControlLogix rack.
6 Turn power ON.

Note: If you insert the module improperly, the system may stop working, or may behave unpredictably.
Note: If you are installing MVI56-MCM with other modules connected to the PCI bus, the peripheral modules will not have holding clips. Make sure all of the modules are aligned with their respective slots before you snap them into place.

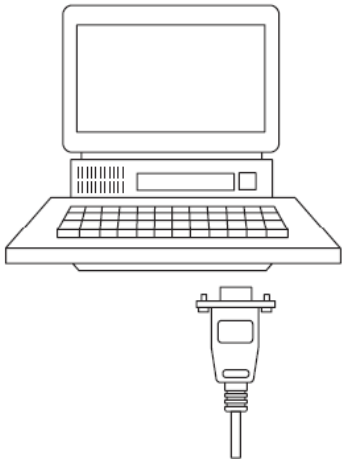
## 1.6 Connect your PC to the ControlLogix Processor

There are several ways to establish communication between your PC and the ControlLogix processor. The following steps show how to establish communication through the serial interface. Refer to your Rockwell Automation documentation for information on other connection methods.

1 Connect the right-angle connector end of the cable to your controller at the communications port.



**2** Connect the straight connector end of the cable to the serial port on your computer.



# 2 Using the RSLogix 5000 v16 Add-On Instruction

*In This Chapter*


- Create a new RSLogix 5000 project ..... 18
- Download the Sample Program to the Processor ..... 29


Important: If you are using an older version of RSLogix 5000 (version 15 or older), please refer to Sample Ladder Logic in New Application or Integrating the Sample Ladder Logic into an Existing Project (page 147).

If you have RSLogix 5000 version 16 or newer, you can use an Add-On Instruction to simplify the task of configuring the module, either as a new application, or within an existing application.

The ProSoft Solutions CD-ROM included in the package with the module contains ladder logic, product manuals, and utility programs for all ProSoft Technology products.

*Copy the manuals and sample program from the CD-ROM*

1. Insert the ProSoft Solutions CD-ROM into the CD drive of your PC. Wait for the startup screen to appear.
2. On the startup screen, click *Product Documentation*. This action opens an Explorer window. Files are arranged by type:
   - The Ladder Logic folder contains sample programs for each module, arranged by processor type, and then by product name. The sample programs for your module are in the ControlLogix/MVI56/MVI56-MCM folder.
   - The Manuals folder contains product manuals and datasheets in Adobe Acrobat Reader format (PDF) for each module, arranged in the same way as the Ladder Logic folder.
   - The Utilities folder contains additional programs and tools required for some ProSoft modules. Refer to your user manual to determine if you need to use or install any of these additional tools.
3. In the Explorer window, navigate to the files you need, and then copy them to a location on your hard drive.

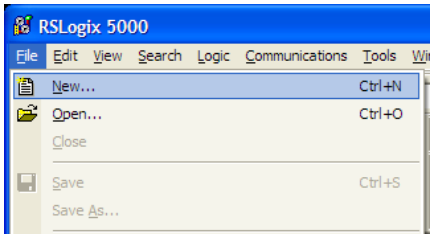
*Download the manuals and sample program from the ProSoft Technology web site*

You can always download the latest version of the sample ladder logic and user manuals for the MVI56-MCM module from the ProSoft Technology web site, at www.prosoft-technology.com/support/downloads

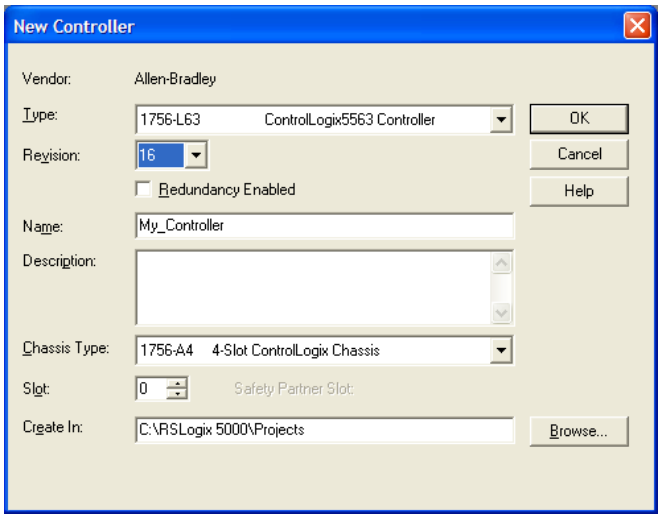
From that link, navigate to the download page for your module and choose the sample ladder program to download for your version of RSLogix 5000 and your processor.

## 2.1 Create a new RSLogix 5000 project

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



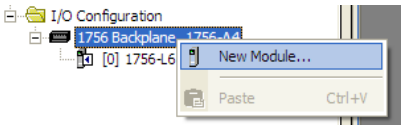

**2** Select **REVISION 16**.



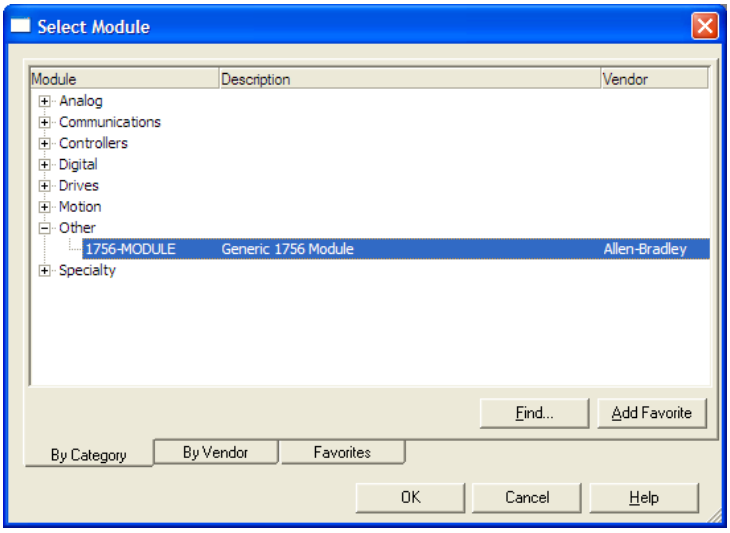
### 2.1.1 Create the Module

**1** Add the MVI56-MCM module to the project.

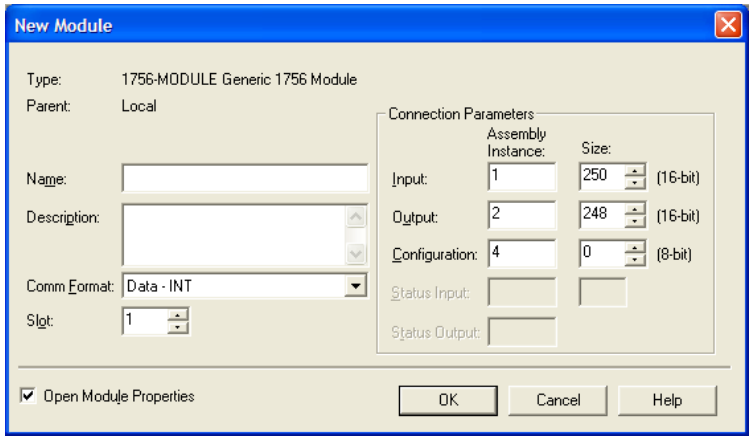
In the **CONTROLLER ORGANIZATION** window, select **I/O CONFIGURATION** and click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW MODULE...**




This action opens the **SELECT MODULE** dialog box.




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



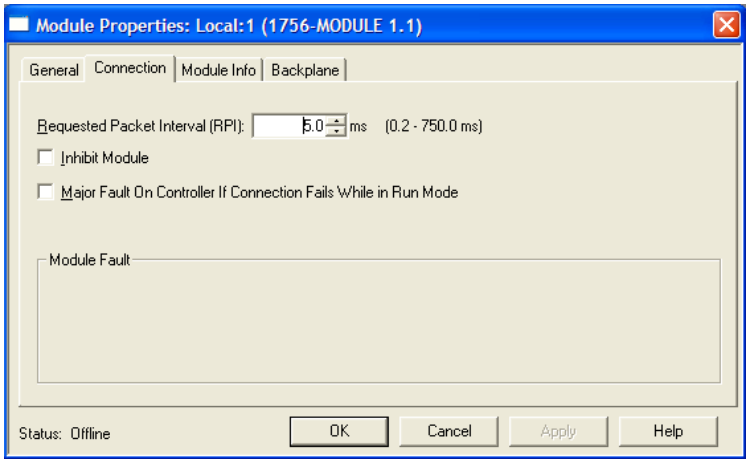
**3** In the **NEW MODULE** dialog box, enter the following values.

| Parameter | Value |
|---|---|
| **NAME** | Enter a module identification string. Example: MCM. |
| **DESCRIPTION** | Enter a description for the module. Example: Modbus Communication Module |
| **COMM FORMAT** | Select **DATA-INT.** |
| **SLOT** | Enter the slot number in the rack where the MVI56-MCM module is located. |
| **INPUT ASSEMBLY INSTANCE** | 1 |
| **INPUT SIZE** | 250 |
| **OUTPUT ASSEMBLY INSTANCE** | 2 |
| **OUTPUT SIZE** | 248 |
| **CONFIGURATION ASSEMBLY INSTANCE** | 4 |
| **CONFIGURATION SIZE** | 0 |

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

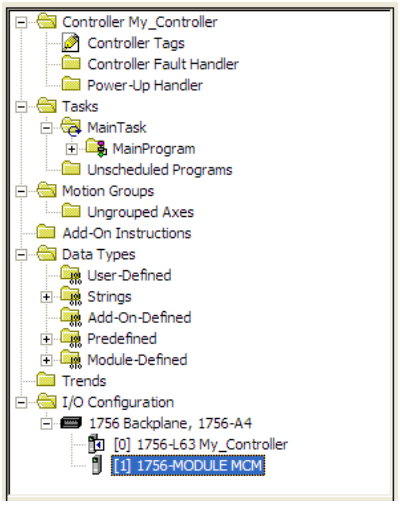
**4** Click **OK** to continue.

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



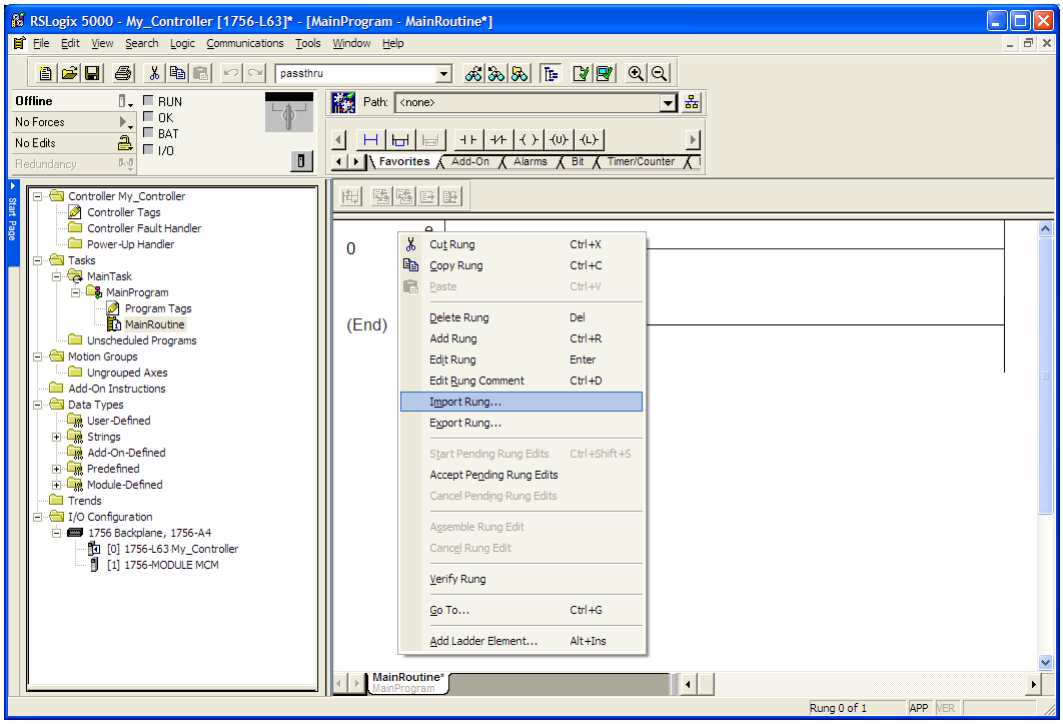
**6 Save the module.**

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

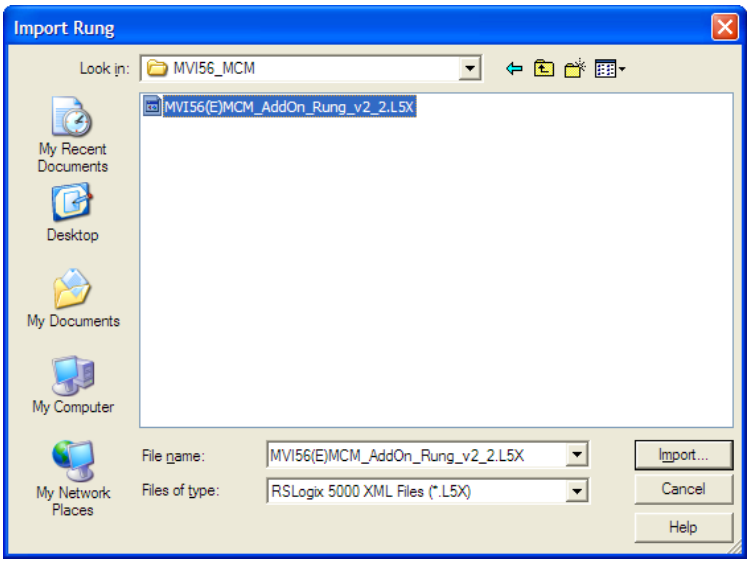



### *2.1.2 Import Add-On Instruction*

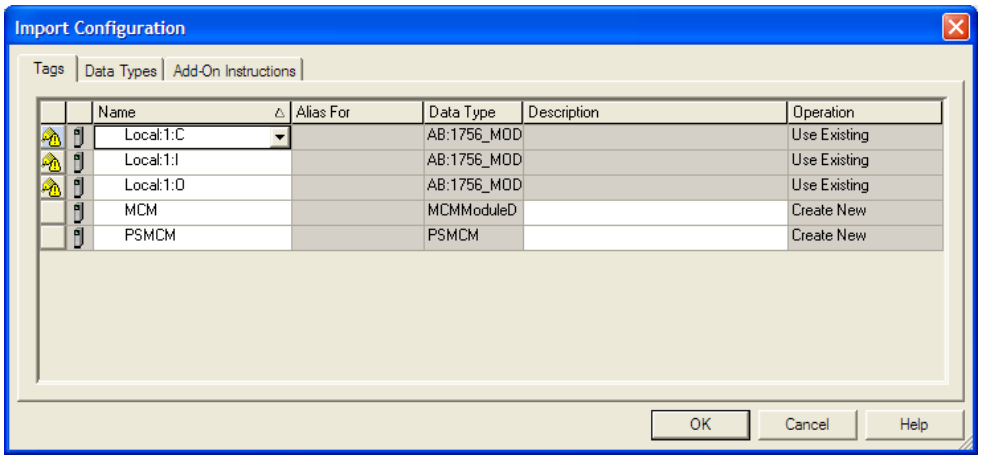
1. In the **CONTROLLER ORGANIZATION** window, expand the **TASKS** folder and subfolder until you reach the **MAINPROGRAM** folder.
2. In the **MAINPROGRAM** folder, double-click to open the **MAINROUTINE** ladder.
3. Select an empty rung in the new routine, and then click the right mouse button to open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



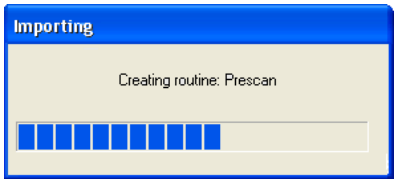
**4** Navigate to the location on your PC where you saved the Add-On Instruction (for example, "My Documents" or "Desktop"). Select the **MVI56(E)MCM_ADDON_RUNG_V2.2.L5X** file



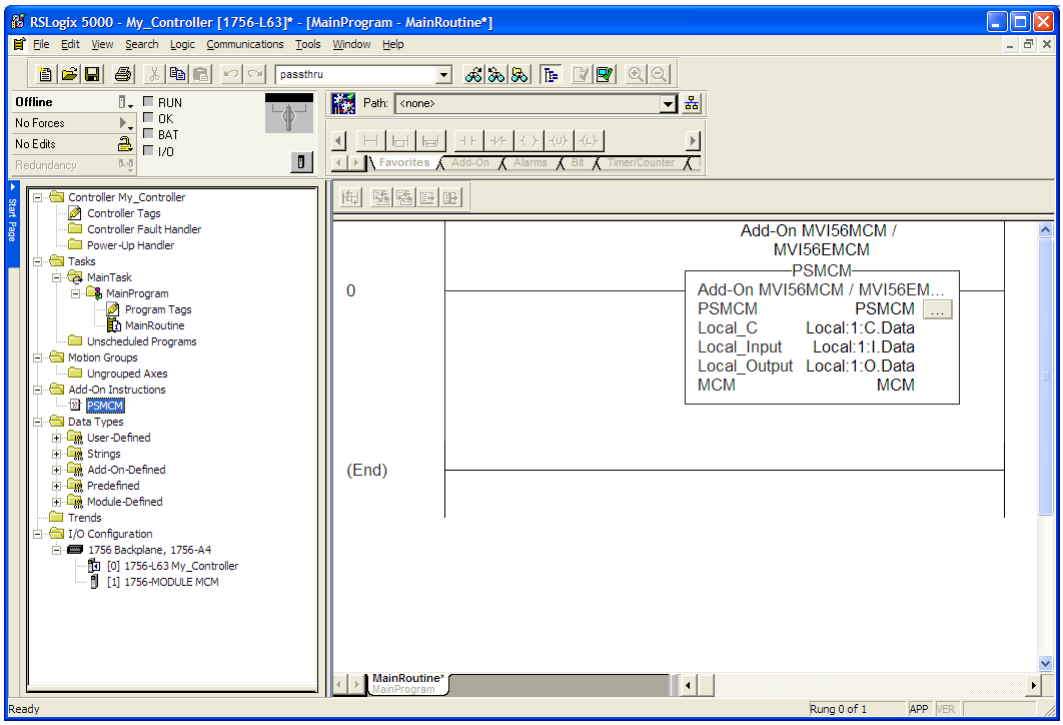

This action opens the **IMPORT CONFIGURATION** dialog box, showing the controller tags that will be created.



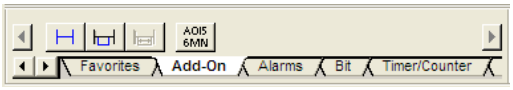

**5** Click **OK** to confirm the import. RSLogix will indicate that the import is in progress:



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



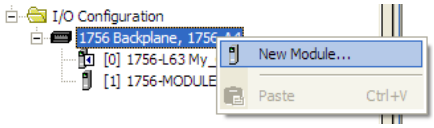

The procedure has also imported new User Defined Data Types, data objects and the Add-On instruction for your project.



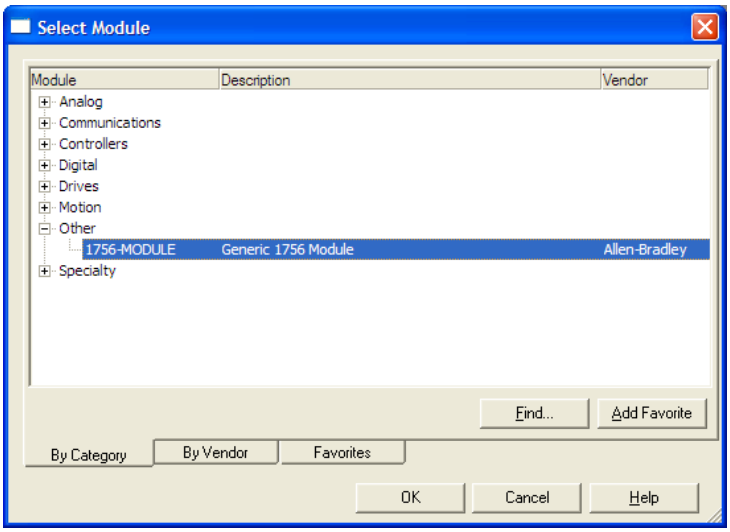

## *Adding Multiple Modules (Optional)*

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

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



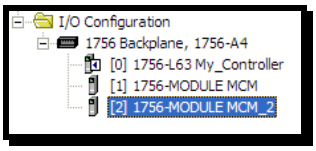
**2** Select **1756-MODULE**




**3** Fill the module properties as follows:

| Parameter | Value |
|---|---|
| **NAME** | Enter a module identification string. Example: MCM_2. |
| **DESCRIPTION** | Enter a description for the module. Example: ProSoft Modbus Communication Module. |
| **COMM FORMAT** | Select **DATA-INT.** |
| **SLOT** | Enter the slot number in the rack where the MVI56-MCM module is located. |
| **INPUT ASSEMBLY INSTANCE** | 1 |
| **INPUT SIZE** | 250 |
| **OUTPUT ASSEMBLY INSTANCE** | 2 |
| **OUTPUT SIZE** | 248 |
| **CONFIGURATION ASSEMBLY INSTANCE** | 4 |
| **CONFIGURATION SIZE** | 0 |

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

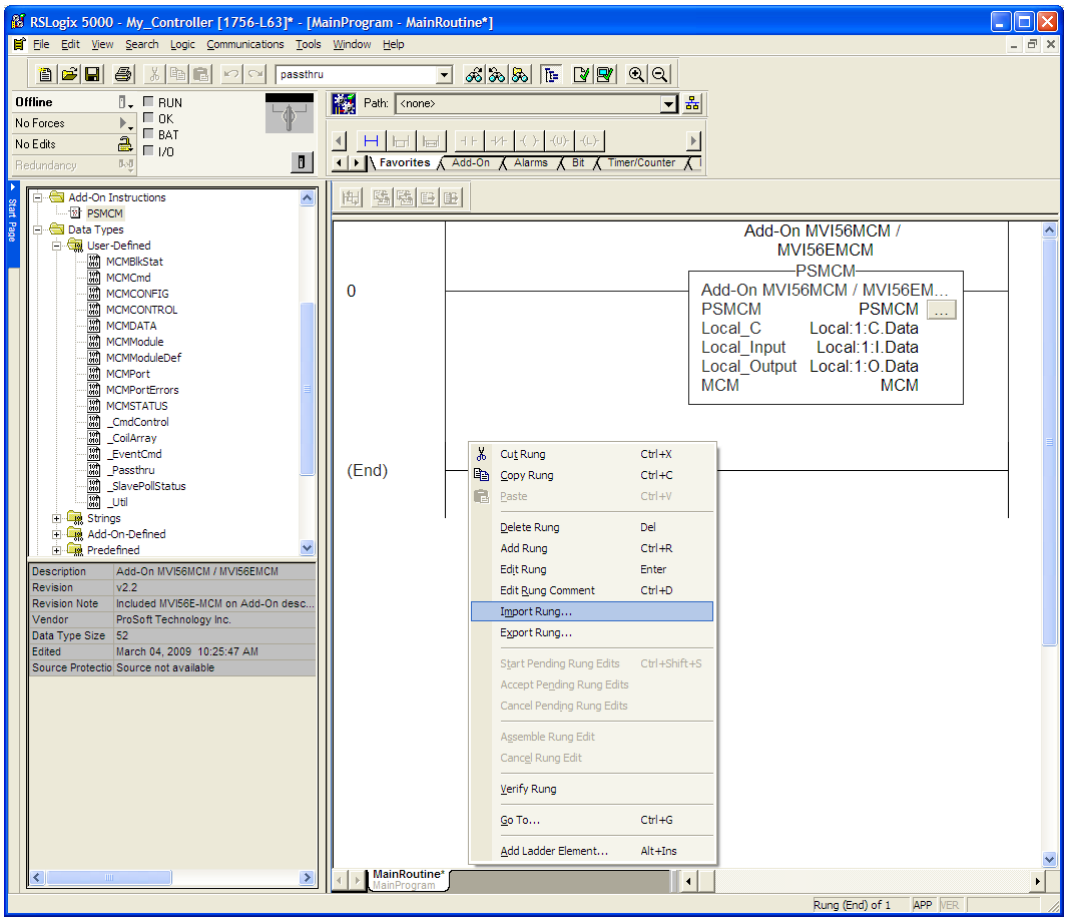



**5** Expand the **TASKS** folder, and then expand the **MAINTASK** folder.

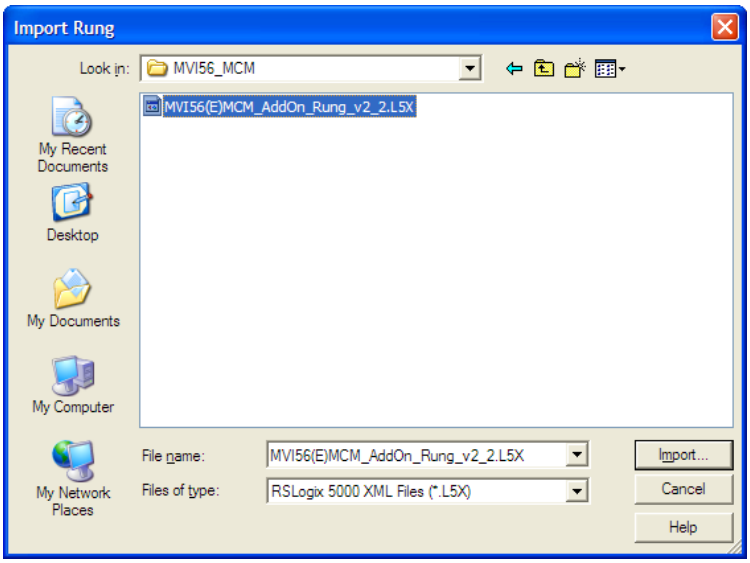
**6** On the **MAINPROGRAM** folder, click the right mouse button to open a shortcut menu. On the shortcut menu, choose **NEW ROUTINE**. As an alternative to creating a separate New Routine, you could skip to Step 8 and import the AOI for the second module into the same routine you created for the first module.

7 In the **NEW ROUTINE** dialog box, enter the name and description of your routine, and then click **OK.**

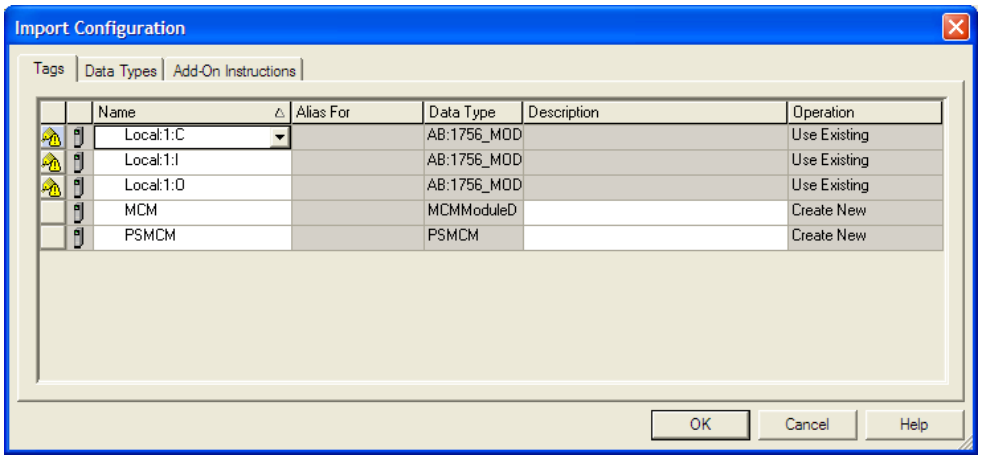
8 Select an empty rung in the new routine or an existing routine, and then click the right mouse button to open a shortcut menu. On the shortcut menu, choose **IMPORT RUNG…**



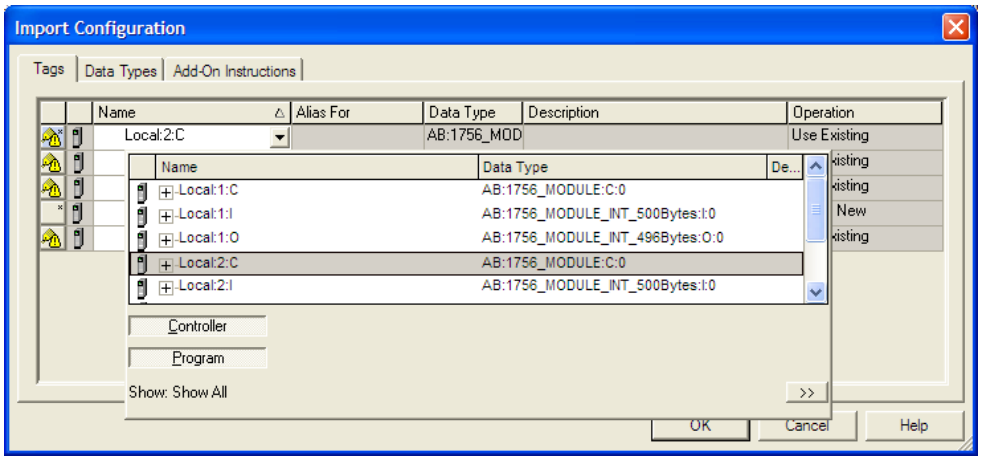
**9** Select the **MVI56(E)MCM_ADDON_RUNG_V2.2.L5X** file, and then click **IMPORT.**

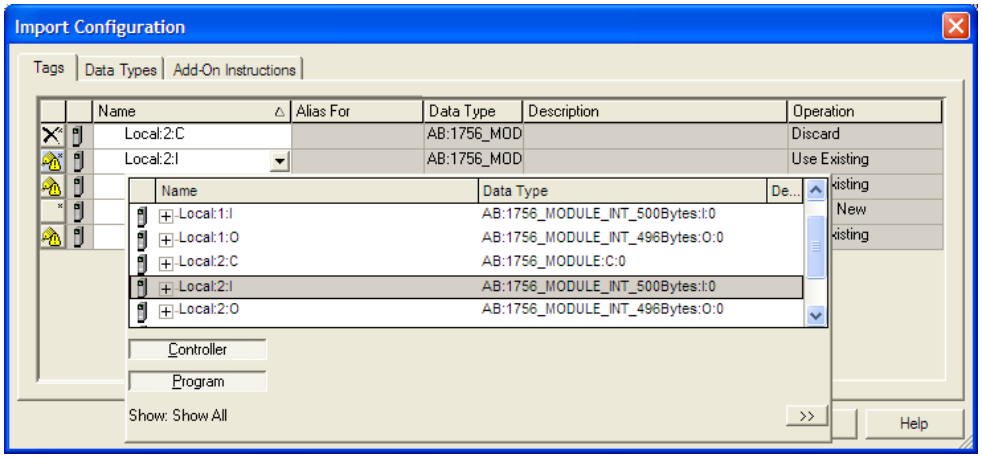


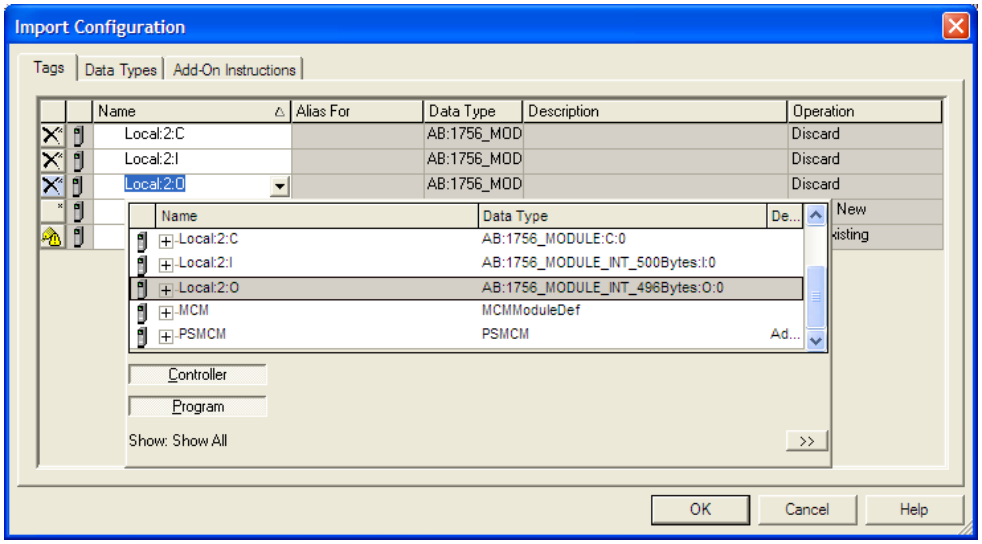

**10** This action opens the **IMPORT CONFIGURATION** window, which shows the tags that will be imported.



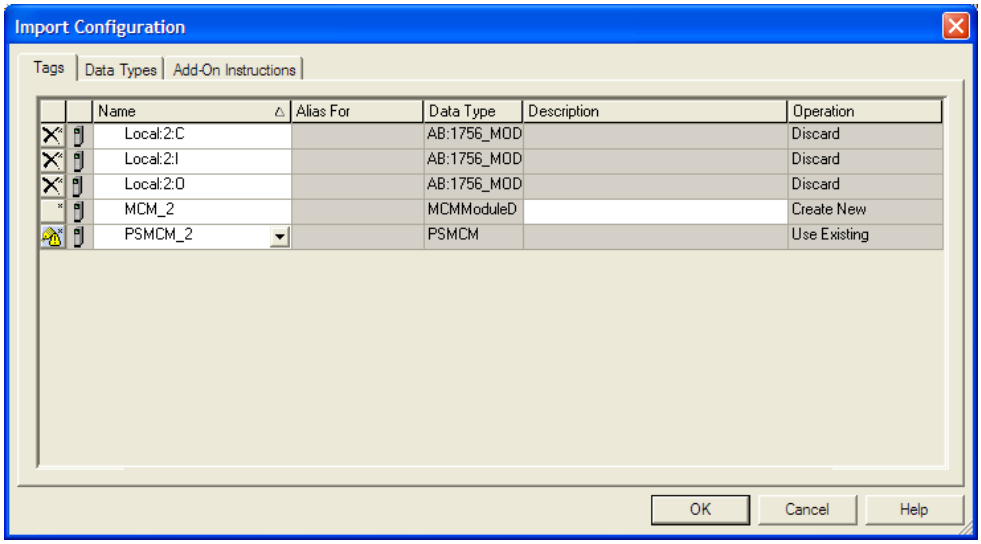
**11** Associate the I/O connection variables to the correct module. The default values are Local:1:I and Local:1:O so these require change.



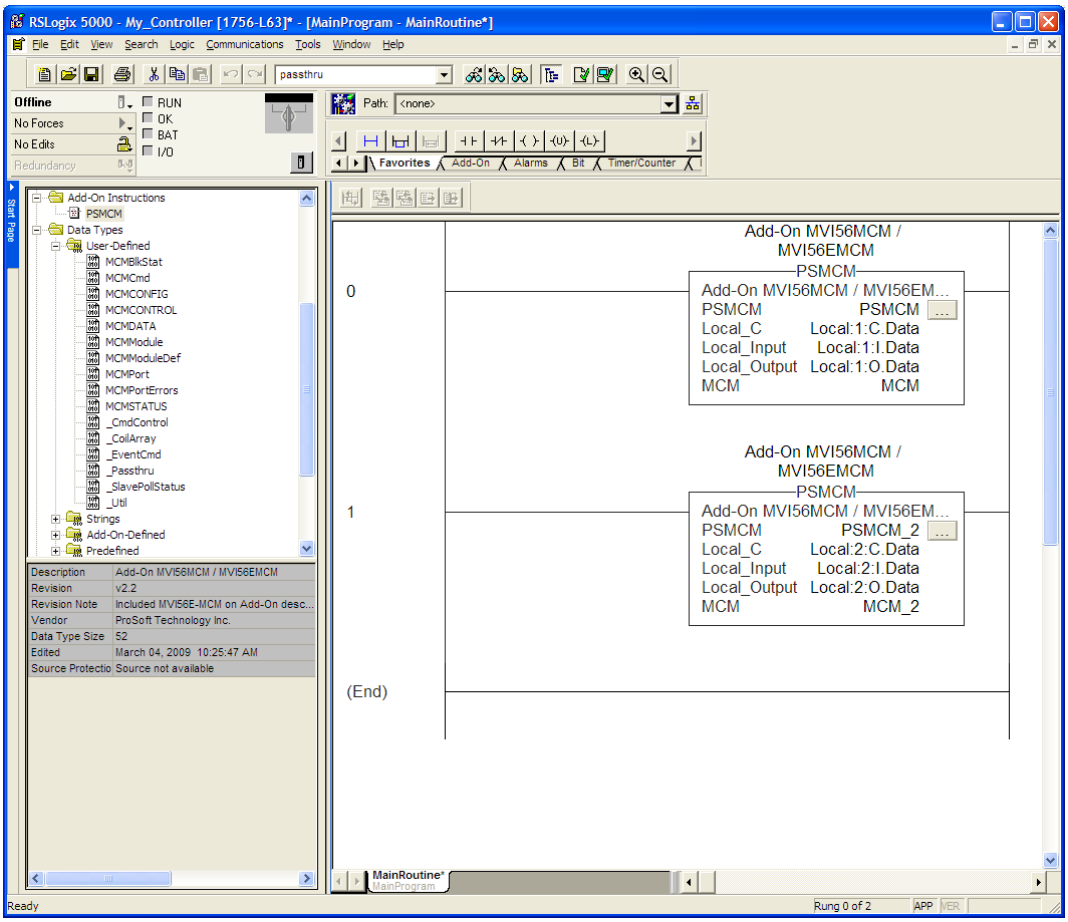



**12** Change the default tags MCM and PSMCM to avoid conflict with existing tags. This procedure will append the string "_2" as follows:




**13** Click OK to confirm.

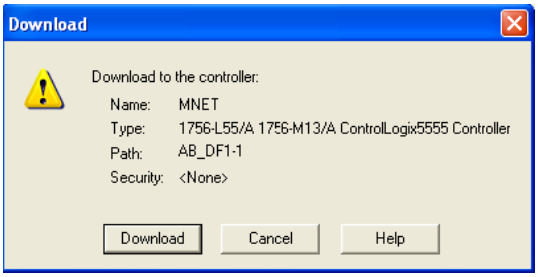


The setup procedure is now complete. Save the project and download the application to your ControlLogix processor.

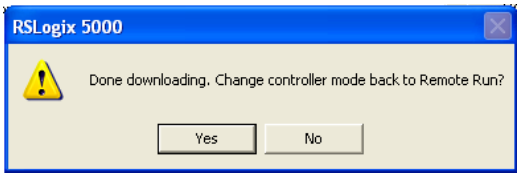
## 2.2 Download the Sample Program to the Processor

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

1. If you are not already online with the processor, open the **COMMUNICATIONS** menu, and then choose **DOWNLOAD.** RSLogix 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. If the keyswitch is in the REM position, click **OK** to switch the processor from Program mode to Run mode.

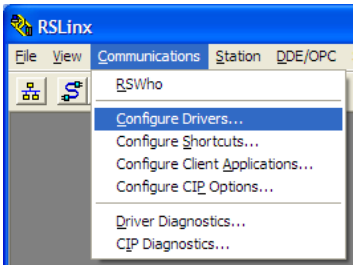



Note: If you receive an error message during these steps, refer to your RSLogix documentation to interpret and correct the error.

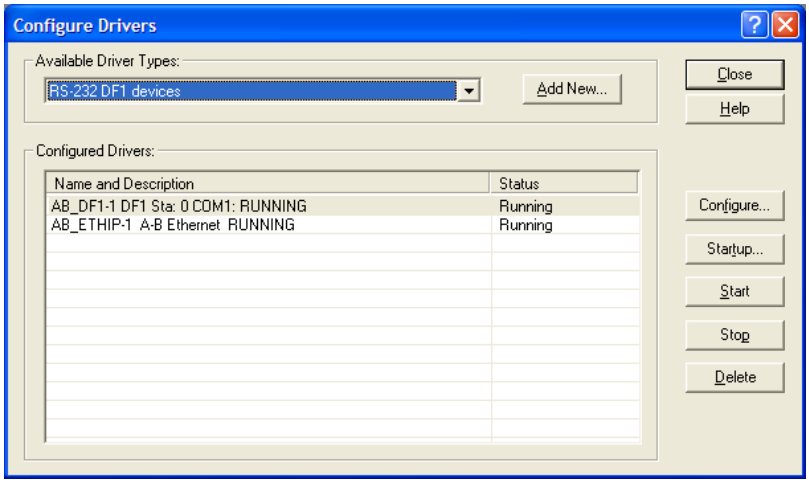
### 2.2.1 Configuring the RSLinx Driver for the PC COM Port

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

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

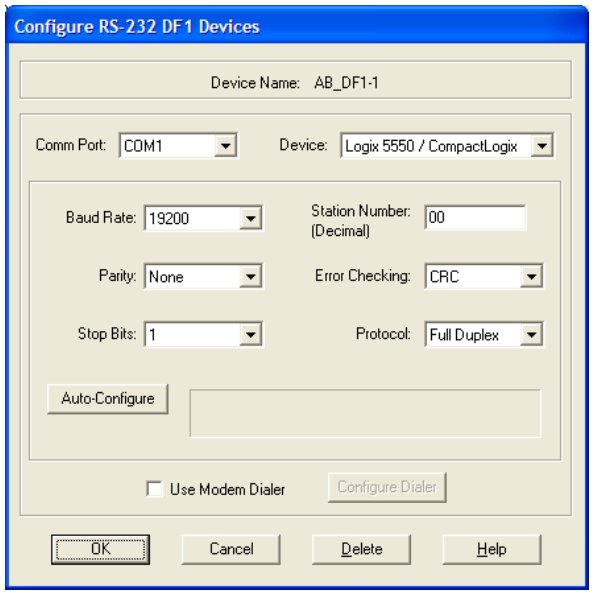



This action opens the **CONFIGURE DRIVERS** dialog box.




Note: If the list of configured drivers is blank, you must first choose and configure a driver from the Available Driver Types list. The recommended driver type to choose for serial communication with the processor is RS-232 DF1 DEVICES.

3 Click to select the driver, and then click **CONFIGURE.** This action opens the **CONFIGURE ALLEN-BRADLEY DF1 COMMUNICATIONS DEVICE** dialog box.




4 Click the **AUTO-CONFIGURE** button. RSLinx will attempt to configure your serial port to work with the selected driver.

5 When you see the message **AUTO CONFIGURATION SUCCESSFUL**, click the **OK** button to dismiss the dialog box.

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

# 3 Configuration as a Modbus Master

*In This Chapter*

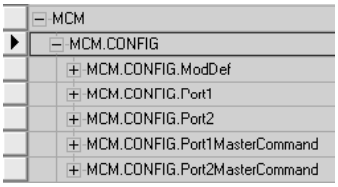

- Overview ..... 33
- ModDef Settings ..... 34
- Master Command Examples ..... 41
- Floating-Point Data Handling ..... 46
- Command Control and Event Command ..... 52


## 3.1 Overview

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

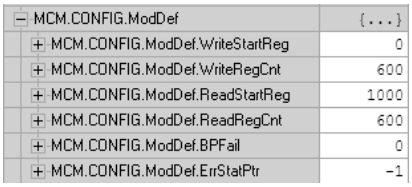
The following three RSLogix controller tags contain the Modbus Master configuration. You must configure all three sets of controller tags.

1. The **MODDEF** controller tags set up the backplane communication between the MVI56-MCM module and the ControlLogix processor. These settings include register addresses for ReadData and WriteData. You can configure up to 5000 data registers in the module to exchange data with the ControlLogix processor.
2. The **PORT1** and **PORT 2** controller tags configure the Modbus application serial port. This set of controller tags configures parameters such as baud rate, data bits, and stop bits.
3. The **PORT1MASTERCOMMAND** and **PORT2MASTERCOMMAND** controller tags define a polling table (command list) for the Modbus Master. This set of tags contains the addresses for devices on the network, the types of data (Modbus Function Codes) to read and write with those devices, and the location to store the data within the module's 5000 data registers.



## 3.2 ModDef Settings

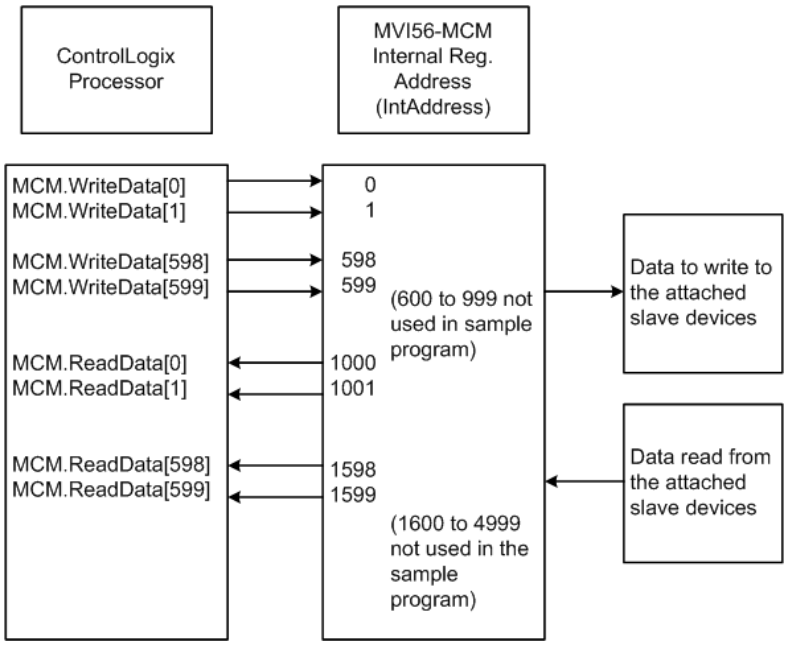
The **MCM.CONFIG.MODDEF** tag defines the 5000 data registers to use for read and write data within the MVI56-MCM module. You will use these data read and write locations in the IntAddress tag within each Master Command (page 38). The following illustration shows the values from the sample program.




The **WRITESTARTREG** determines the starting register location for **WRITEDATA[0** to **599]**. The **WRITEREGCNT** determines how many of the 5000 registers to use send data to the module. The sample ladder file uses 600 registers for write data, labeled **MCM.WRITEDATA[0** to **599].**

| Label | Description |
|---|---|
| **WRITESTARTREG** | Specifies where in the 5000 register module memory to place data sent from the WriteData tags in the ControlLogix processor. |
| **WRITEREGCNT** | Specifies how many registers of data the MVI56-MCM module will request from the ControlLogix processor. Because the module pages data in blocks of 200 words, this number must be evenly divisible by 200. |
| **READSTARTREG** | Specifies which registers in the module's read data area to send to the ReadData tags in the ControlLogix processor. |
| **READREGCNT** | Sets how many registers of data the MVI56-MCM module will send to the ControlLogix processor. This value should also be a multiple of 200. |
| **BPFAIL** | Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network. Typically used when the module is configured as a Slave. |
| **ERRSTATPTR** | Also used mainly when the module is configured as a Slave. This parameter places the STATUS data into the database of the module. |

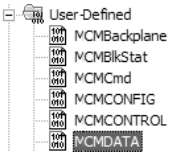
The sample configuration values configure the module database to store **WRITEDATA[0** to **599]** in registers 0 to 599, and **READDATA[0 TO 599]** in registers 1000 to 1599, as shown in the following illustration.



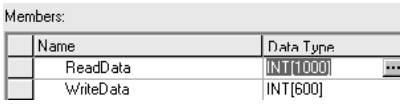

The MVI56-MCM sample program is configured for 600 registers of **READDATA** and 600 registers of **WRITEDATA,** which is sufficient for most application. This topic describes how to configure user data for applications requiring more than 600 registers of ReadData and WriteData.

Important: Because the module pages data in blocks of 200 registers at a time, you should configure your user data in multiples of 200 registers.
Caution: When you change the array size, RSLogix may reset the MCM tag values to zero. To avoid data loss, be sure to save your settings before continuing.

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

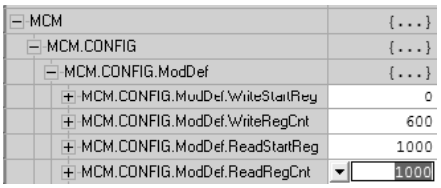


2 In the edit window, change the value of the **READDATA** array from **INT[600]** to **INT[1000]** as shown, and then click **APPLY.**



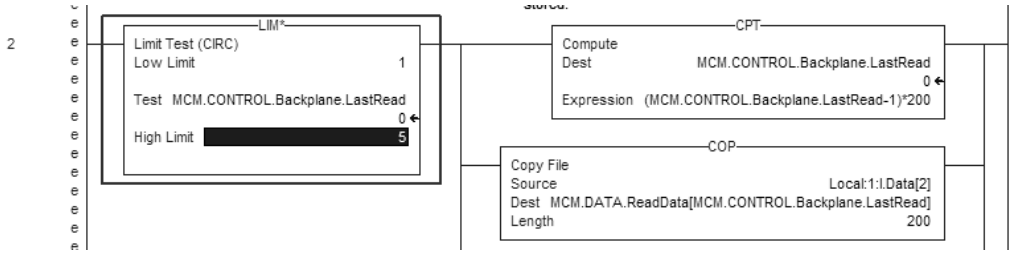

Note: If RSLogix resets your data values, refer to the backup copy of your program to re-enter your configuration parameters.

3 Next, navigate to **CONTROLLER TAGS** and double click to open an edit window. Click the **MONITOR TAGS** tab at the bottom of the edit window.
4 Click **[+]** to expand the **MCM.CONFIG.MODDEF** section, and then change the **READREGCNT** parameter from 600 to 1000.




5 Click **ReadData** to open ladder file and go to rung #2 of this file.
6 Change the High Limit on the LIM statement to allow for 5 blocks of data, as shown in the following illustration.

(1000 registers / 200 registers per block = 5 blocks of data)



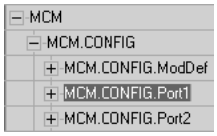

7 Verify the change to this rung. Toggle the object within RSLogix 5000.
8 Save and download ladder to the processor.
9 When Online with the ControlLogix processor, toggle the MCM.CONTROL.WarmBoot bit to download the changes made in Step 5 to the processor.

Note: Any changes made to the MCM.CONFIG or WriteData arrays must be downloaded to the MVI56-MCM module. The use of the MCM.CONTROL.WarmBoot or MCM.CONTROL.ColdBoot bit will force the MVI56-MCM module to re- read the configuration from the ControlLogix processor.

To modify the WriteData array, change the High Limit value of the LIM statement in rung #3 of the **WriteData** ladder. Also, make sure that the ReadData and WriteData arrays do not overlap in the module memory. If you have an application that requires 2000 words of WriteData, starting at register 0, then your MCM.CONFIG.ModDef.ReadStartReg should be set to a value of 2000 or greater.

### *3.2.1 Port Configuration*

The **MCM.CONFIG.PORTX** controller tags are used when the module is configured as a Modbus Master device. Port 1 and Port 2 each have their own set of parameters to configure.




Note: Any changes made within the MCM.CONFIG array must be downloaded to the MVI56-MCM module by setting the WARMBOOT or COLDBOOT bit, or cycling power to the module.

Any parameters not mentioned in this section are not used when the module is configured as a Modbus Master.

Verify you are in **MONITOR TAGS** mode. Then use the scroll bar at bottom to view description of each parameter. The following table uses that information.

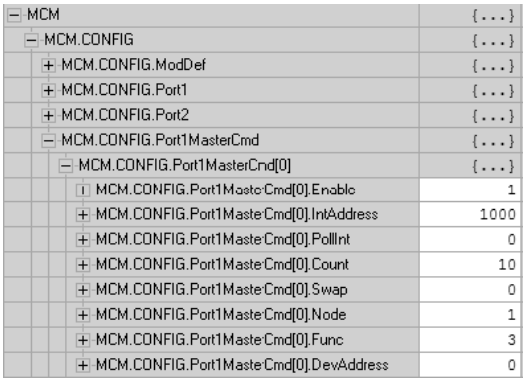
| Label | Description |
|---|---|
| Enabled | 1 = enable port, 0 = disable port |
| Type | 0=Master, 1=Slave, 2=Slave: pass-through, 3=Slave: formatted pass-through/data swapped, 4=Slave: form. pass-through |
| Protocol | 0 = Modbus RTU mode, 1 = Modbus ASCII mode |
| Baudrate | Sets the baud rate for the port. Valid values for this field are 110, 150, 300, 600, 1200, 4800, 9600, 19200, 384 or 3840 (for 38,400 baud), 576 or 5760 (for 57,600 baud) and 115,1152, or 11520 (for 115,200 baud) |
| Parity | 0 = None, 1 = Odd, 2 = Even |
| DataBits | Modbus RTU mode = 8 Modbus ASCII mode = 8 or 7 |
| StopBits | Valid values are 1 or 2. |
| CmdCount | Command list count. Sets how many registers to use for error codes as set in the CmdErrPtr value.<br>**Note:** Command count list 100 = causes the module to use the commands configured at **MCM.CONFIG.PORTX. MASTERCOMMAND[0] TO [99]**.<br>10 = process commands [0] to [9]. |
| CmdErrPtr | Internal DB location to place command error list<br>**Note:** Set Master Command Errors location. Each command will reserve one for the command error code. See Verifying Communications (page 67). **CMDERRPTR** value should be within the range of the **READDATA** array, per **MCM.CONFIG.MODDEF** of this manual. |

| Label | Description |
|---|---|
| RespTO | 0 to 65535 milliseconds response timeout for command<br>**Note:** 1000 = 1000 milliseconds (1 second) before it will either reissue the command (as set in the **RETRYCOUNT**) or if the RetryCount has already been met, then it will move on to the next command in the list. |
| RetryCount | Retry count for failed request |

### *3.2.2 Master Command Configuration*

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

Verify you are in **MONITOR TAGS** mode. Then use the scroll bar at bottom to view a description of each parameter. The following table uses that information.




| Label | Description |
|---|---|
| Enable | 0 = Disabled<br>Command will not be executed, but can be enabled using command control option in ladder logic.<br>1 = Enabled<br>Command is enabled and will be sent out to the target device.<br>2 = Conditional Write<br>Only for Func 5, 15, 6, or 16 data will be sent to the target device only when the data to be written has changed. |
| IntAddress | Determines where in the module's 5000-register database the data will be stored to or written from. For a Read command, this will determine, after information has been read from a Slave, where it will be placed in the module database. For read commands, you should configure this for a location that is configured for **READDATA.** The internal database location for the **READDATA** and **WRITEDATA** tags is determined by the configuration in the MCM.ModDef tag location.<br>For write data, the **INTADDRESS** determines where to obtain the information write to the Slave device. This must be a location that is configured as **WRITEDATA.**<br>**Note:** When using a bit level command, you must define this field at the bit level. For example, when using a function code 1, 2 for a Read command, you must have a value of 16000 to place the data in MCM.ReadData[0] (register 1000 * 16 bits per register = 16000). |

| Label | Description |
|---|---|
| PollInt | The Poll Interval (PollInt) is the number of seconds that a Master device will wait before issuing this command. |
| Count | Sets how many continuous words (Function Codes 3, 4, and 16) or bits (Function Codes 1, 2, and 15) to request from the Slave device.<br>Valid values are 1 to 125 words for function codes 3, 4, and 16, while you can specify a range of 1 to 2000 for function codes 1, 2, and 15.<br>**Note:** These values are the maximum allowed in the Modbus protocol. Some devices may support fewer words or bits than the maximum allowed. |
| Swap | Typically used when reading floating-point data, swaps the data read from the Slave device before it is placed into the module memory. For example, you receive 4 bytes of data from the Slave (ABCD).<br>0 = No swapping (ABCD)<br>1 = Word pairs switched (CDAB)<br>2 = Bytes and words switched (DCBA)<br>3 = Bytes swapped (BADC) |
| Node | Node address of the device on the network to read data from, or write data to. Valid addresses are 1 to 247. Address 0 is reserved for broadcast write commands (will broadcast a Write command to all devices on the network). |
| Func | Determines the Modbus function code that to issue in the command to the Slave device. Valid values for this field are as follows:<br>**1 = Read Coil Status**<br>This will read Modbus addresses 0001 to 9999. These bit values indicate coil status. Use Function Code 5 or 15 to write to these registers<br>**2 = Read Input Coils**<br>This will read Modbus addresses 10001 to 29999. Like Function Code 1, these are also bit values, but Function Code 2 values are Read Only data values. Use Function Code 5 or 15 to write to these registers<br>**3 = Read Holding Registers**<br>This will read Modbus addresses 40001 to 47999. This is a 16-bit word value. Use Function Codes of 6 and 16 to write to these registers.<br>**4 = Read Input Registers**<br>This will read Modbus addresses 30001 to 39999. These are also 16-bit word values, but are Read Only data. The Modbus Master cannot write to these registers.<br>**5 = Write Single Coil Status**<br>This will write to Modbus addresses 0001 to 9999. This command will write to only one coil. Use Function Code 15 to write to multiple coils.<br>**6 = Write Single Register**<br>This will write to Modbus addresses 40001 to 47999. This command writes a single register value out to a Slave device. Use Function Code 16 to write to multiple registers.<br>**15 = Multiple Coil Write**<br>This will write multiple coil values to the Slave addresses 0001 to 9999.<br>**16 = Multiple Register Write**<br>This will write multiple register values to the Slave device at addresses 40001 to 49999. |

| Label | Description |
|---|---|
| DevAddress | Specifies the Modbus Slave address for the registers associated with that command. This is the offset address for the Modbus Slave device. With Modbus, to read an address of 40001, what will actually be transmitted out port is Function Code 03 (one byte) with an address of 00 00 (two bytes). This means that to read an address of 40501, use Func 3 with a DevAddress of 500.<br>This applies to Modbus addresses 10001 to 47999.<br>Below is a definition that will help with your DevAddress configuration:<br>**Function Codes 1, 5, or 15**<br>▪ DevAddress = Modbus address in device - 0001<br>Example: Modbus address 0001 = DevAddress 0<br>▪ Modbus address 1378 = DevAddress 1377<br>**Function Code 2**<br>▪ DevAddress = Modbus address in device - 10001<br>Example: Modbus address 10001 = DevAddress 0<br>▪ Modbus address 10345 = DevAddress 344<br>**Function Codes 3, 6, or 16**<br>▪ DevAddress = Modbus address in device - 40001<br>Example: Modbus address 40001 = DevAddress 0<br>▪ Modbus address 40591 = DevAddress 590<br>**Function Code 4**<br>▪ DevAddress = Modbus address in device - 30001<br>Example: Modbus address 30001 = DevAddress 0<br>▪ Modbus address 34290 = DevAddress 4289 |

### *3.2.3 Other Modbus Addressing Schemes*

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

The two most common schemes are six-digit addressing (400101, 301000, etc…) and some devices show their addressing already as an offset address (the address that actually goes out on the Modbus communication line). This is an example.

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

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

If your device manufacturer gives you addressing like this, "Input Registers" then you will use Function Code 4, and then place the address shown in the DevAddress field. Also, most manufacturers that show this type of addressing will list the address in hex, as is the case with the device shown above. So for this example device, use Func = 4 (Input Registers) with a DevAddress of 512 decimal (200h) to read the "Switch Input Status" value.

*Why does my Slave show addressing such as 400,001 or 301,345?*

For the 6 digit addressing, use the same function codes and configuration as configured above, just the starting address has changed.

Below is a definition that will help with your DevAddress configuration:

Function Codes 1, 5, or 15 **DevAddress** = Modbus address in device - 0001

- Example: Modbus address 0001 = DevAddress 0
- Modbus address 1378 = DevAddress 1377

Function Code 2 **DevAddress =** Modbus address in device - 100001

- Example: Modbus address 100001 = DevAddress 0
- Modbus address 100345 = DevAddress 344

Function Codes 3, 6, or 16 **DevAddress** = Modbus address in device - 400001

- Example: Modbus address 400001 = DevAddress 0
- Modbus address 400591 = DevAddress 590

Function Code 4 **DevAddress** = Modbus address in device - 300001

- Example: Modbus address 300001 = DevAddress 0
- Modbus address 304290 = DevAddress 4289

For example, our device listed above could show their addressing as follows.

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

To read the same parameter "Switch_Input_Status", you would still issue a Function Code 4, and use a DevAddress of 512 decimal.

## 3.3 Master Command Examples

### *3.3.1 Read Holding Registers 4xxxx (Modbus Function Code 3)*

The 4x holding registers are used for Analog Values such as Pressure, Temperature, Current, and so on. These are 16-bit register values, but they can also store Floating-Point data (page 46). You can also write to these Modbus addresses using Modbus Function Codes 6 or 16.

Below is a sample command to read Modbus addresses 40001 to 40010 of node 1 on the Modbus network.

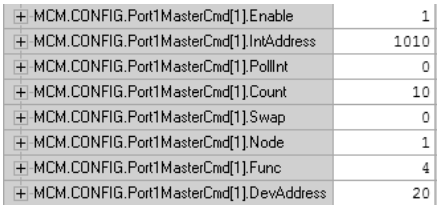
| MCM.CONFIG.Port1MasterCmd[0] | {...} |
|---|---|
| MCM.CONFIG.Port1MasterCmd[0].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[0].IntAddress | 1000 |
| MCM.CONFIG.Port1MasterCmd[0].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Count | 10 |
| MCM.CONFIG.Port1MasterCmd[0].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[0].Func | 3 |
| MCM.CONFIG.Port1MasterCmd[0].DevAddress | 0 |

| Label | Description |
|---|---|
| Enable = 1 | The module will send the command every time it goes through the command list. |
| IntAddress = 1000 | Places the data read from the Slave device into the module at address 1000. IntAddress 1000 of the module memory will be copied into the tag **MCM.DATA.READDATA[0]**. |
| Count = 10 | Reads 10 consecutive registers from the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 3 | Issues Modbus Function Code 3 to Read Holding Registers. |
| DevAddress = 0 | Function Code 3, DevAddress of 0 will read address 40001<br>Along with a count of 10, this command reads 40001 to 40010. |

### *3.3.2 Read Input Registers 3xxxx (Modbus Function Code 4)*

Like the 4x holding registers, 3x input registers are used for reading analog values that are 16-bit register values. You can also use these registers to store floating-point data (page 46). Unlike the 4x registers, 3x registers are Read Only.

Below is a sample command to read Modbus addresses 30021 to 30030 of node 1 on the Modbus network.



| | |
|---|---|
| + MCM.CONFIG.Port1MasterCmd[1].Enable | 1 |
| + MCM.CONFIG.Port1MasterCmd[1].IntAddress | 1010 |
| + MCM.CONFIG.Port1MasterCmd[1].PollInt | 0 |
| + MCM.CONFIG.Port1MasterCmd[1].Count | 10 |
| + MCM.CONFIG.Port1MasterCmd[1].Swap | 0 |
| + MCM.CONFIG.Port1MasterCmd[1].Node | 1 |
| + MCM.CONFIG.Port1MasterCmd[1].Func | 4 |
| + MCM.CONFIG.Port1MasterCmd[1].DevAddress | 20 |

| Label | Description |
|---|---|
| Enable = 1 | The module will send the command every time it goes through the command list. |
| IntAddress = 1010 | Places the data read from the Slave device into the module at address 1010. IntAddress 1010 of the module memory will be copied into the tag **MCM.DATA.READDATA[10].** |
| Count = 10 | Reads 10 consecutive registers from the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 4 | Issues Modbus Function Code 4 to Read Input Registers. |
| DevAddress =20 | Function Code 4 DevAddress of 20 will read address 30021<br>Along with a count of 10, this command reads 30021 to 30030. |

### *3.3.3 Read Coils 0xxxx (Modbus Function Code 1)*

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

Below is a sample command to read Modbus addresses 0321 to 0480 of node 1 on the Modbus network.

| | |
|---|---|
| MCM.CONFIG.Port1MasterCmd[2].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[2].IntAddress | 16320 |
| MCM.CONFIG.Port1MasterCmd[2].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[2].Count | 160 |
| MCM.CONFIG.Port1MasterCmd[2].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[2].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[2].Func | 1 |
| MCM.CONFIG.Port1MasterCmd[2].DevAddress | 320 |

| Label | Description |
|---|---|
| Enable = 1 | The module will send the command every time it goes through the command list. |
| IntAddress = 16320 | Places the data read from the Slave device into the module at address 16320. IntAddress 16320 of the module memory will be copied into the tag **MCM.DATA.READDATA[20]** because 16320 represents a bit address within the memory of the MVI56-MCM module (16320 / 16 = register 1020). |
| Count = 160 | Reads 160 consecutive bits from the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 1 | Issues Modbus Function Code 1 to Read Coils. |
| DevAddress = 320 | Function Code 1, DevAddress of 320 will read address 0321<br>Along with a count of 160, this command reads 0321 to 0480. |

### *3.3.4 Read Input Coils 1xxxx (Modbus Function Code 2)*

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

Below is a sample command to read Modbus addresses 10081 to 10090 of node 1 on the Modbus network.

| | |
|---|---|
| MCM.CONFIG.Port1MasterCmd[3].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[3].IntAddress | 16480 |
| MCM.CONFIG.Port1MasterCmd[3].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[3].Count | 16 |
| MCM.CONFIG.Port1MasterCmd[3].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[3].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[3].Func | 2 |
| MCM.CONFIG.Port1MasterCmd[3].DevAddress | 80 |

| Label | Description |
|---|---|
| Enable = 1 | The module will send the command every time it goes through the command list. |
| IntAddress = 16480 | Places the data read from the Slave device into the module at address 16480. IntAddress 16480 of the module memory will be copied into the tag **MCM.DATA.READDATA[30]** (bit16480 / 16 = register 1030). |
| Count = 16 | Reads 16 consecutive registers from the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |

| Label | Description |
|---|---|
| Func = 2 | Issues Modbus Function Code 2 to Read Input Coils. |
| DevAddress = 80 | Function Code 2, DevAddress of 80 will read address 10081<br>Along with a count of 16, this command reads 10081 to 10096. |

### 3.3.5 Write Single Coil 0xxxx (Modbus Function Code 5)

Used to write a Coil of a Slave device, these are single bit addresses within a Modbus Slave device. The IntAddress field of this command is defined down to the bit level within the module memory, and should come from an area of memory that has been defined within the **MCM.DATA.WRITEDATA** area (this is configured within **MCM.CONFIG.MODDEF.**

Below is a sample command to write Modbus addresses 0513 of node 1 on the Modbus network, only when the data associated with the IntAddress has changed.

| | |
|---|---|
| MCM.CONFIG.Port1MasterCmd[4].Enable | 2 |
| MCM.CONFIG.Port1MasterCmd[4].IntAddress | 160 |
| MCM.CONFIG.Port1MasterCmd[4].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[4].Count | 1 |
| MCM.CONFIG.Port1MasterCmd[4].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[4].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[4].Func | 5 |
| MCM.CONFIG.Port1MasterCmd[4].DevAddress | 512 |

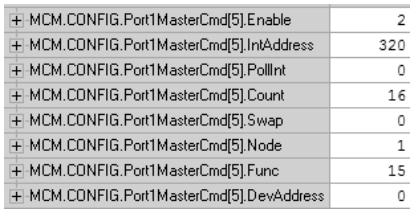
| Label | Description |
|---|---|
| Enable = 2 | The module will send the command only when the data within the IntAddress field of the module has changed. |
| IntAddress = 160 | Will write the data to the Slave device when the value at WriteData[10].0 has changed. Because this is a bit level command, the IntAddress field must be defined down to the bit level. |
| Count = 1 | Will write a single bit to the device (Function Code 5 will 1 support a count of 1). |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 5 | Issues Modbus Function Code 5 to write a single coil. |
| DevAddress = 512 | Function Code 5, DevAddress of 512 will read address 0513 |

### 3.3.6 Write Multiple Coils 0xxx (Modbus Function Code 15)

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

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

Below is a sample command to write Modbus addresses 0001 to 0016 of node 1 on the Modbus network.



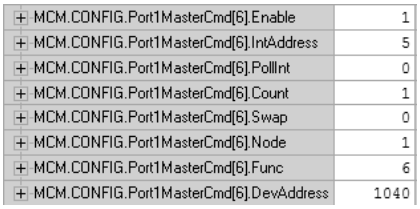
| | |
|---|---|
| MCM.CONFIG.Port1MasterCmd[5].Enable | 2 |
| MCM.CONFIG.Port1MasterCmd[5].IntAddress | 320 |
| MCM.CONFIG.Port1MasterCmd[5].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[5].Count | 16 |
| MCM.CONFIG.Port1MasterCmd[5].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[5].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[5].Func | 15 |
| MCM.CONFIG.Port1MasterCmd[5].DevAddress | 0 |

| Label | Description |
|---|---|
| Enable = 2 | The module will send the command to the Slave device only when the data associated within the IntAddress of the MVI56-MCM module memory has changed. |
| IntAddress = 320 | Writes the data in bit 320 of the module memory to the Slave device. Based on the **MCM.CONFIG.MODDEF** setting, this would be the data in **MCM.DATA.WRITEDATA[20].0** to **[20].15** in the ladder logic. |
| Count = 16 | Writes 16 consecutive bits to the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 15 | Issues Modbus Function Code 15 to write multiple coils. |
| DevAddress = 0 | Function Code 15, DevAddress of 0 will read address 0001<br>Along with a count of 16, this command writes to 0001 to 0016. |

### *3.3.7 Write Holding Register 4xxxx (Modbus Function Code 6)*

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

Below is a sample command to write Modbus addresses 41041 of node 1 on the Modbus network.



| | |
|---|---|
| MCM.CONFIG.Port1MasterCmd[6].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[6].IntAddress | 5 |
| MCM.CONFIG.Port1MasterCmd[6].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[6].Count | 1 |
| MCM.CONFIG.Port1MasterCmd[6].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[6].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[6].Func | 6 |
| MCM.CONFIG.Port1MasterCmd[6].DevAddress | 1040 |

| Label | Description |
|---|---|
| Enable = 1 | The module will send the command every time it goes through the command list. |
| IntAddress = 5 | Writes the data from address 5 of the module memory to the Slave device. Based on the **MCM.CONFIG.MODDEF** configuration, this will take the data from **MCM.DATA.WRITEDATA[5]** and write that information out to the Slave device. |
| Count = 1 | Writes 1 register (16-bit) to the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 2 | Issues Modbus Function Code 6 to write a single register. |
| DevAddress = 1040 | Function Code 6, DevAddress of 1040 will write to address 41041 of the Modbus Slave device. |

### *3.3.8 Write Multiple Registers 4xxxx (Modbus Function Code 16)*

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

Below is a sample command to write Modbus addresses 41051 to 41060 of node 1 on the Modbus network.

| | |
|---|---|
| MCM.CONFIG.Port1MasterCmd[7].Enable | 2 |
| MCM.CONFIG.Port1MasterCmd[7].IntAddress | 30 |
| MCM.CONFIG.Port1MasterCmd[7].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[7].Count | 10 |
| MCM.CONFIG.Port1MasterCmd[7].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[7].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[7].Func | 16 |
| MCM.CONFIG.Port1MasterCmd[7].DevAddress | 1050 |

| Label | Description |
|---|---|
| Enable = 2 | The module will send the command only when the data associated with the IntAddress of the module has changed. |
| IntAddress =30 | Writes the data from Internal Address 30 of the module memory to the Slave device. Based on the **MCM.CONFIG.MODDEF** configuration, this will write the data from **MCM.DATA.WRITEDATA[30] TO [39]** to the Slave device. |
| Count = 10 | Writes 10 consecutive registers to the Slave device. |
| Node = 1 | Issues the Modbus command to node 1 on the network. |
| Func = 16 | Issues Modbus Function Code 16 to write Holding Registers. |
| DevAddress = 1050 | Function Code 16, DevAddress of 1050 will write address 41051.<br>Along with a count of 10, this command writes 41051 to 41060 of the Slave device. |

## 3.4 Floating-Point Data Handling

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

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

### 3.4.1 Read Floating-Point Data

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

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

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

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

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

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

| | | | |
|---|---|---|---|
| [+] | Energy_Consumption | REAL[1] | Float |

Copy data from the **MCM.DATA.READDATA[0]** and **[1]** into the tag **ENERGY_CONSUMPTION** that has a data type of REAL. Use a **COP** statement within the ladder logic. Here is an example.

Because the tag **MCM.DATA.READDATA[0]** should only be used within the above command, an unconditional COP statement can be used.

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

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

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

### *3.4.2 Read Multiple Floating-Point Registers*

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

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

Configure the command to read these 7 floats as follows.

| MCM.CONFIG.Port1MasterCmd[0] | {...} |
|---|---|
| MCM.CONFIG.Port1MasterCmd[0].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[0].IntAddress | 1000 |
| MCM.CONFIG.Port1MasterCmd[0].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Count | 14 |
| MCM.CONFIG.Port1MasterCmd[0].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[0].Func | 3 |
| MCM.CONFIG.Port1MasterCmd[0].DevAddress | 260 |

Configure an array of 7 floats within the ControlLogix processor as shown in the following illustration.

| MCM_Float_Data | | | REAL[7] | Float |
|---|---|---|---|---|

The following **COP** statement will copy the data from **MCM.DATA.READDATA[0] TO [13]** into the array **MCM_FLOAT_DATA[0] TO [6].**

The "Length" parameter is set to the number of Floating-Point values that must be copied from the **MCM.DATA.READDATA** array.

### *3.4.3 Write Floats to Slave Device*

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

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

You must use a **COP** statement to copy the data from floating-point data tags within the ControlLogix processor, into the **MCM.DATA.WRITEDATA** array used by the MVI56-MCM module. Below is an example.

The length of this COP statement must now be 14. This will COP as many of the **MCM_FLOAT_DATA** values required to occupy the **MCM.DATA.WRITEDATA** array for a length of 14. This will take 7 registers, **MCM_FLOAT_DATA[0] TO [6],** and place that data into **MCM.DATA.WRITEDATA[0] TO [13].**

You must configure the command to write all 7 floats (14 Modbus addresses) as follows.

| MCM.CONFIG.Port1MasterCmd[0] | {...} |
|---|---|
| MCM.CONFIG.Port1MasterCmd[0].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[0].IntAddress | 0 |
| MCM.CONFIG.Port1MasterCmd[0].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Count | 14 |
| MCM.CONFIG.Port1MasterCmd[0].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[0].Func | 16 |
| MCM.CONFIG.Port1MasterCmd[0].DevAddress | 260 |

The above command will take the data from **MCM.DATA.WRITEDATA[0] TO [13]** and write this information to the Slave device node #1 addresses 40261 to 40274.

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

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

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

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

This type of device uses one Modbus address per floating-point register. To read these values from the Slave device, configure the following command within the module.

| MCM.CONFIG.Port1MasterCmd[0] | {...} |
|---|---|
| MCM.CONFIG.Port1MasterCmd[0].Enable | 1 |
| MCM.CONFIG.Port1MasterCmd[0].IntAddress | 1000 |
| MCM.CONFIG.Port1MasterCmd[0].PollInt | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Count | 7 |
| MCM.CONFIG.Port1MasterCmd[0].Swap | 0 |
| MCM.CONFIG.Port1MasterCmd[0].Node | 1 |
| MCM.CONFIG.Port1MasterCmd[0].Func | 3 |
| MCM.CONFIG.Port1MasterCmd[0].DevAddress | 7000 |

Notice that the count is now set to a value of 7. Because the Slave device utilizes only 7 Modbus addresses, a count of 7 will cause the Slave to respond with 14 registers (28 bytes) of information.

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

The **COP** statement for this type of data is the same as shown in Read Multiple Floating-Point Registers (page 48).

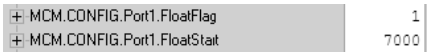
### 3.4.5 Write to Enron/Daniel Floats

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

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

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

Configure the Float Start and Float Flag parameters as shown.



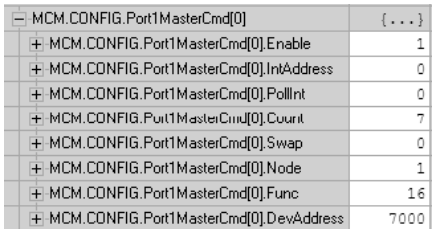

The Float Flag causes the module to use the FloatStart parameter to determine which DevAddress requires a write command to issue double the number of bytes.

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

**1** First, copy the floating-point data from the ControlLogix processor into the **MCM.DATA.WRITEDATA** array used by the MVI56-MCM module. Below is an example.

**2** The length of this COP statement must now be 14. This will COP as many of the **MCM_FLOAT_DATA** values required to occupy the **MCM.DATA.WRITEDATA** array for a length of 14. This will take 7 registers, **MCM_FLOAT_DATA[0] TO [6],** and place that data into **MCM.DATA.WRITEDATA[0] TO [13].**

The following illustration shows the command required to write these 7 Floating-Point values.



Based on the IntAddress and the configuration within the **MCM.CONFIG.MODDEF** section for WriteStartReg and WriteRegCount, the data from the tag **MCM.DATA.WRITEDATA[0] TO [6]** will be written to Modbus addresses 47001 to 47007 of the Slave device node #1.

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

## 3.5 Command Control and Event Command

You can use Command Control and Event Commands in Modbus Master mode to change the command execution based on some conditions in ladder. The module goes through the command list sequentially. For example:

- The module executes **MCM.CONFIG.PORT1MASTERCMD[0]**
- After completing that command, it will then execute **MCM.CONFIG.PORT1MASTERCMD[1],** then **MCM.CONFIG.PORT1MASTERCMD[2],** and so on.

You can use Command Control and Event Command to issue a command at the top of the command queue, interrupting the regular command list execution.

You would typically use Command Control and Event Command to

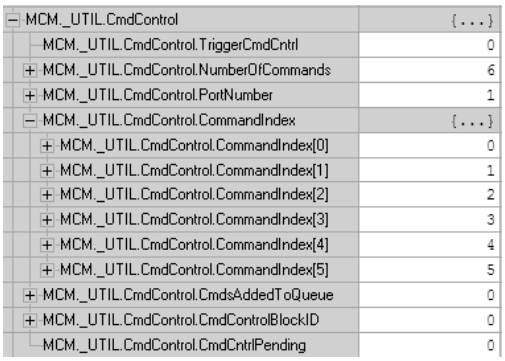
- issue a reset to a device on a once a day basis
- poll for end of hour data
- issue special commands on the startup of a process or the changing of a batch

Important: Because these special command blocks will interrupt the normal polling list, you should use them sparingly, to avoid interrupting your normal data transfer. Make sure that the data to be written to the device contains the latest value from the WriteData tag that corresponds to the Command Control or Event Command.

### *3.5.1 Command Control*

Command Control allows you to issue a command already defined in the Master command list (but disabled) and enable that command for a single pass. Command Control has a distinct advantage over Event Command in that it will still return an error code for that command as configured in **MCM.CONFIG.PORTX.CMDERRPTR.** Up to 6 commands may be enabled at the same time.

The following illustration shows how to configure Command Control using the **MCM._UTIL.CMDCONTROL** object in the ladder logic.




The following configuration will place 6 commands into the command queue.

**MCM.CONFIG.PORT1MASTERCMD[0]** to **MCM.CONFIG.PORT1MASTERCMD[5]** will be enabled with this configuration. Error codes for each command are placed in the Error Status table.

| Tag | Value | Description |
|---|---|---|
| TriggerCmdCntrl | 1 | 1 will execute the Command Control |
| NumberOfCommands | 6 | Number of commands per block |
| PortNumber | 1 | MVI56-MCM Port number (Master) |
| CommandIndex[0] to [5] | 0 to 5 | Stores the command index for Command Control block |
| CmdsAddedToQueue | | Number of commands added to queue. This is the confirmation that the Command Control block has completed successfully |
| CmdControlBlockID | | Temporary variable to calculate control block ID number |
| CmdCntrolPending | | Aux. control command - prevents a second request before acknowledgement is received |

Note: For RSLogix version 15 and lower, the ladder logic necessary for the successful execution of this block is contained in the _WriteControl ladder file, rung 4, and in the _ReadControl ladder file, rung 2.

### *3.5.2 Event Command*

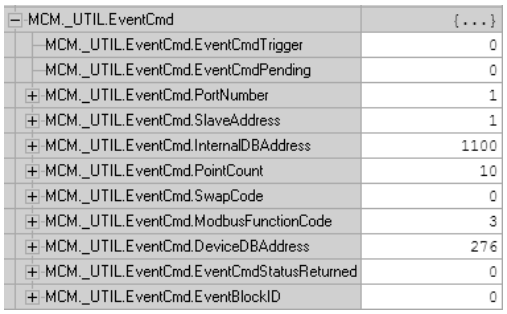
Event Command allows you to add commands directly to the command queue, interrupting the normal polling sequence of the module. Unlike Command Control, Event Commands do not return an error code into the location defined by the **MCM.CONFIG.PORTX.CMDERRPTR** value.

You do not need to define Event Commands in the regular command list. Event Command adds a command to the top of the MVI56-MCM module's command queue that is not defined within the command list.

Within an Event Command block, you define a Modbus command to add to the queue.

**Important:** Because these special command blocks will interrupt the normal polling list, you should use them sparingly, to avoid interrupting your normal data transfer. Make sure that the data to be written to the Slave contains the latest value from the WriteData tag that corresponds to the Event Command.

The following illustration describes the structure of the EventCommand block.




| Parameter | Value | Description |
|---|---|---|
| EventCmdTrigger | 1 | 1 = trigger the Event Command |
| EventCmdPending | | Used = EventCommand is executed once |
| PortNumber | 1 | Module Port # to send command out to |
| SlaveAddress | 1 | Modbus Slave ID command to be issued to |
| InternalDBAddress | 1100 | 1100 will place the data read into MCM.DATA.ReadData[100] |
| PointCount | 10 | Consecutive register/bits to read or write with the command |
| SwapCode | 0 | Swap code used with command |
| ModbusFunctionCode | 3 | Function Code 3 is read 4xxxx holding registers |
| DeviceDBAddress | 276 | Address in the Slave device to read. With Function Code 3, DeviceDBAddress of 276, the module will read starting at address 40277 in the Slave device |
| EventCmdStatusReturned | | Return value of 0 = Fail, 1 = Success |
| EventBlockID | | Block ID number for the module to recognize the Event Command, Slave address, and Port number to send the command out |

**Note:** For RSLogix version 15 and lower, the ladder logic used for the Event Command blocks is contained in _WriteControl rung 5 and _ReadControl rung 4 within the sample ladder file.
**Note:** Event Command blocks can only send 1 command to the command queue per block.
**Note:** Event Commands (like Command Control) take priority over commands in the normal command list.

# 4 Configuration as a Modbus Slave

*In This Chapter*


- Overview ..... 55
- ModDef Settings ..... 56
- Read and Write Same Modbus Address (Pass Through) ..... 60
- Slave Configuration ..... 62
- Floating-Point Data Handling ..... 63


## 4.1 Overview

When configuring the module as a Slave, you will be providing whoever is programming the Master side of the communications with a Modbus Memory Map.

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

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

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

The sample ladder file is configured to transfer 600 16-bit registers in each direction. If more than that is required, please see Applications Requiring More Than 600 Registers of ReadData or WriteData (page 35).

Find out if the Master can read from one Modbus address and write to another Modbus address, or, if the Master must use the same address to read and write data points.

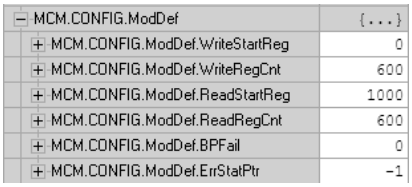
If the Master must read and write to the same Modbus address, you must use Pass-Through mode (page 60). This allows the **MCM.DATA.WRITEDATA** array to be used for all data transfer to the Master. Because the data transfer of the MVI56-MCM module cannot be bidirectional, when the Master issues a Modbus Write command in Pass-Through mode, the MVI56-MCM module builds a special block of information. This block is then parsed by the ladder logic, and the value written from the Modbus Master is then updated in the **MCM.DATA.WRITEDATA** array.

Note: You should only use Pass-Through mode when there is no other option, as there is a drawback to this mode that is not present in the standard mode.
Because the module must wait for the ladder logic to confirm receiving the new data from the Master, if the Master issues consecutive write commands, the module cannot process the second write command until it has finished with the first command. This will cause the module to respond with an error code of 6 (module busy) on the Modbus network.

## 4.2 ModDef Settings

To configure Modbus Slave mode, use the **MCM.CONFIG.MODDEF** settings.

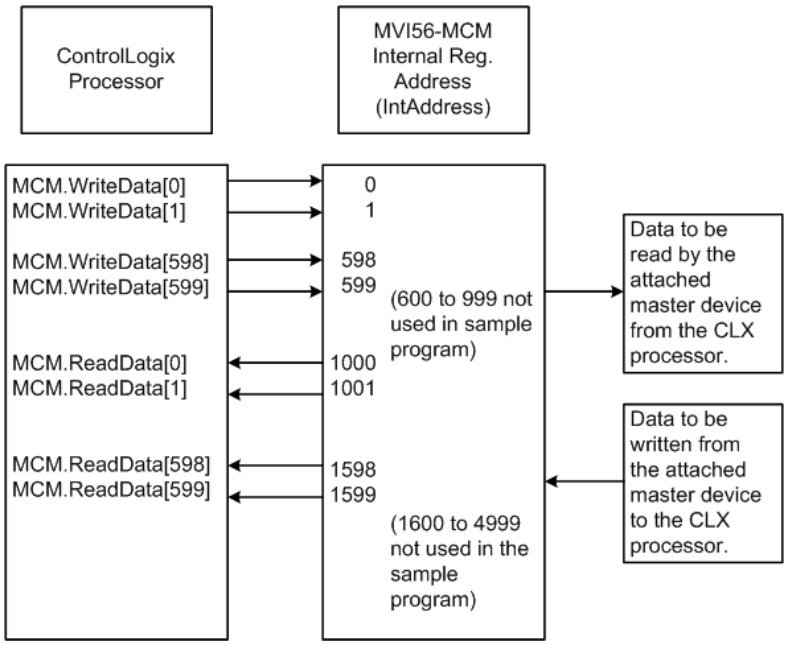
This section specifies which of the MVI56-MCM module's 5000 registers of memory to send from the ControlLogix processor to the MVI56-MCM module (WriteData) and which registers to send from the MVI56-MCM module to the ControlLogix processor (ReadData).




The **WRITESTARTREG** determines the starting register location for **WRITEDATA [0 TO 599]** and the **WRITEREGCNT** determines how many of the 5000 registers to use for information to be written out to the module. The sample ladder file will configure 600 registers for Write Data, labeled **MCM.WRITEDATA[0 TO 599].**

| Value | Description |
|---|---|
| WriteStartReg | Determines where in the 5000 register module memory to place the data obtained from the ControlLogix processor from the WriteData tags. |
| WriteRegCnt | Sets how many registers of data the MVI56-MCM module will request from the ControlLogix processor. Because the module pages data in blocks of 200 words, this number must be evenly divisible by 200. |
| ReadStartReg | Determines where in the 5000 register module memory to begin obtaining data to present to the ControlLogix processor in the ReadData tags. |
| ReadRegCnt | Sets how many registers of data the MVI56-MCM module will send to the ControlLogix processor. This value should also be a multiple of 200. |
| BPFail | Sets the consecutive number of backplane failures that will cause the module to stop communications on the Modbus network. |
| ErrStatPtr | This parameter places the STATUS data into the database of the module. This information can be read be the Modbus Master to know the status of the module. |

With the sample configuration, the following is the layout of the tags and addressing.




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

### *4.2.1 Modbus Memory Map*

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

| MVI Address | 0xxx | 1xxxx | 3xxxx | 4xxxx | Tag Address |
|---|---|---|---|---|---|
| 0 | 0001 to 0016 | 10001 to 10016 | 30001 | 40001 | WriteData[0] |
| 1 | 0017 to 0032 | 10017 to 10032 | 30002 | 40002 | WriteData[1] |
| 2 | 0033 to 0048 | 10033 to 10048 | 30003 | 40003 | WriteData[2] |
| 3 | 0049 to 0064 | 10049 to 10064 | 30004 | 40004 | WriteData[3] |
| 4 | 0065 to 0080 | 10065 to 10080 | 30005 | 40005 | WriteData[4] |
| 5 | 0081 to 0096 | 10081 to 10096 | 30006 | 40006 | WriteData[5] |
| 6 | 0097 to 0112 | 10097 to 10112 | 30007 | 40007 | WriteData[6] |
| 7 | 0113 to 0128 | 10113 to 10128 | 30008 | 40008 | WriteData[7] |
| 8 | 0129 to 0144 | 10129 to 10144 | 30009 | 40009 | WriteData[8] |
| 9 | 0145 to 0160 | 10145 to 10160 | 30010 | 40010 | WriteData[9] |
| 10 | 0161 to 0176 | 10161 to 10176 | 30011 | 40011 | WriteData[10] |
| 50 | 0801 to 0816 | 10801 to 10816 | 30051 | 40051 | WriteData[50] |
| 100 | 1601 to 1616 | 11601 to 11616 | 30101 | 40101 | WriteData[100] |

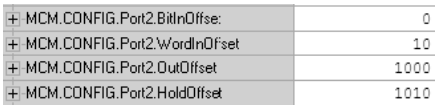
| MVI Address | 0xxx | 1xxxx | 3xxxx | 4xxxx | Tag Address |
|---|---|---|---|---|---|
| 200 | 3201 to 3216 | 13201 to 13216 | 30201 | 40201 | WriteData[200] |
| 500 | 8001 to 8016 | 18001 to 18016 | 30501 | 40501 | WriteData[500] |
| 598 | 9569 to 9584 | 19569 to 19584 | 30599 | 40599 | WriteData[598] |
| 599 | 9585 to 9600 | 19585 to 19600 | 30600 | 40600 | WriteData[599] |
| 600 to 999 | N/A | N/A | N/A | N/A | Reserved |
| 1000 | | | 31001* | 41001 | ReadData[0] |
| 1001 | | | 31002* | 41002 | ReadData[1] |
| 1002 | | | 31003* | 41003 | ReadData[2] |
| 1003 | | | 31004* | 41004 | ReadData[3] |
| 1004 | | | 31005* | 41005 | ReadData[4] |
| 1005 | | | 31006* | 41006 | ReadData[5] |
| 1006 | | | 31007* | 41007 | ReadData[6] |
| 1007 | | | 31008* | 41008 | ReadData[7] |
| 1008 | | | 31009* | 41009 | ReadData[8] |
| 1009 | | | 31010* | 41010 | ReadData[9] |
| 1010 | | | 31011* | 41011 | ReadData[10] |
| 1050 | | | 31051* | 41051 | ReadData[50] |
| 1100 | | | 31101* | 41101 | ReadData[100] |
| 1200 | | | 31201* | 41201 | ReadData[200] |
| 1500 | | | 31501* | 41501 | ReadData[500] |
| 1598 | | | 31599* | 41599 | ReadData[598] |
| 1599 | | | 31600* | 41600 | ReadData[599] |

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

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

### *4.2.2 Customizing the Memory Map*

In some cases, the above memory map will not work for the application. Sometimes a Master must read bits starting at address 0001, and also read a register starting at 40001. With the memory map in this example (page 57), this is not possible, as **WRITEDATA[0]** is seen as both 0001 to 0016, and 40001. To accommodate this, you can customize the starting location within the module for each device using the parameters shown below.



| Parameter | Value | Description |
|---|---|---|
| BitInOffset | 0 | Defines the starting address within the module for 1xxxx Modbus addressing. A value of 0 sets 10001 to 10016 as address 0 in the MVI56-MCM module. |
| WordInOffset | 10 | Defines the starting address within the module memory for 3xxxx registers. |
| OutOffset | 1000 | Defines the starting address within the module for 0xxx coils. |
| HoldOffset | 1010 | Defines the starting address within the module for 4xxxx addressing. |

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

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

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

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

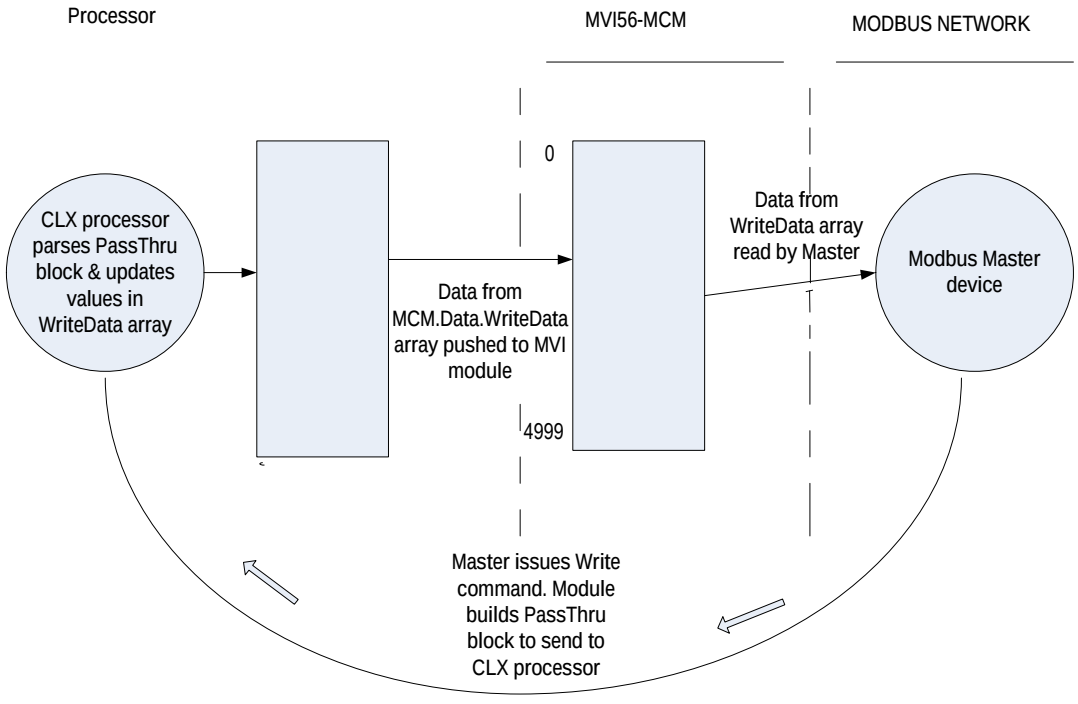
Note: The offset parameter only set the starting location for the data. As shown above, if the Master issues a Write command to address 40001, the data will go into the ControlLogix processor at address MCM.DATA.READDATA[10].

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

## 4.3 Read and Write Same Modbus Address (Pass Through)

In some applications, the Modbus Master must be able to read and write to the same Modbus address within the module. This is not possible for normal Slave communication, as data can either be read from the WriteData array, or written to the ReadData array, but not both.

Pass Through mode allows the Modbus Master to read and write the same Modbus address, using only the WriteData array. The basic theory of pass through is that the ladder logic will constantly be updating values in the MVI56-MCM module memory using the WriteData array. When the Master issues a Write command, the module will build a special block of data. This block of data is then presented to the ladder logic and then copied back into the WriteData array. The following illustration shows Pass Through operation of the module.




Ladder logic for the pass through operation is located in the subroutine _PassThru.

*Pass Through should only be used when required*. If a Master issues a Write command to the module, the module must build a special block of information. Then, it waits for confirmation from the ladder logic that the block has been processed.

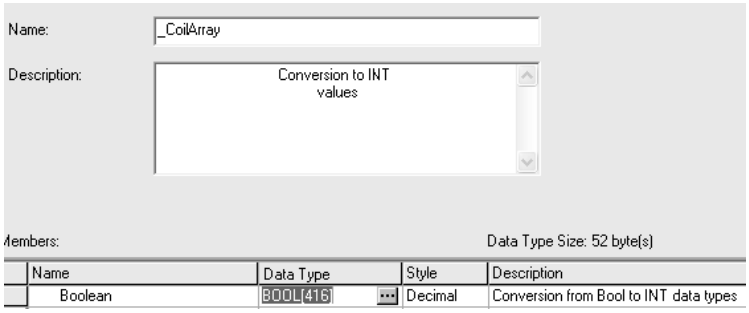
Note: If the module is waiting for the block to be processed by the ladder, and the Master device issues another Write command, the module will return an Error Code of 6 (module busy). This error causes the ladder not to process data written by the Master.

### 4.3.1 Pass Through Coil Adjustment

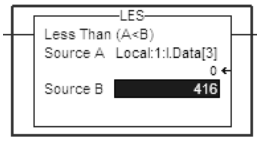
Important: The MVI56(E)MCM_AddOn_Rung_v2.2.L5X Add-On Instruction only supports the default value configured in the _CoilArray data type. To implement the Pass Through Coil Adjustment described in this topic, you must use the Sample Ladder Logic for an earlier version of RSLogix 5000 (version 15 or lower).

The Sample Ladder Logic will only allow for the first 416 coils (26 registers) to be processed. This is due to the size of the Coil Array within the **_UTIL.PASSTHRU.MBCOIL** configuration, and the Sample Ladder Logic. To increase this array the following steps must be performed.

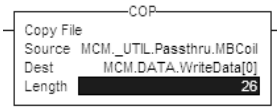
1 Increase the Coil array under the User Defined data type of **_COILARRAY** (should be a value divisible by 32).




2 Change the ladder within **_PASSTHRU** rung #2 to reflect this new change. This must be changed in 2 places. First, the LES statement should be equal to the new array size.



**3** The COP statement that copies the new status of the bits must have the length changed to Equal to the number of bits defined in the array 416 bits in array / 16 = 26.




These changes are necessary for the proper operation of the logic when the ladder logic is processing the new write data from the Master.

## 4.4 Slave Configuration

Any parameters not mentioned in this section are not used when the module is configured as a Modbus Master. Parameters in **BOLD** are required for all applications as a slave.

| Value | Description |
|---|---|
| **Enabled** | 1= enable port, 0 = disable port |
| **Type** | 1= Modbus slave port<br>2= Modbus slave port with pass thru (not recommended, see note in Overview (page 55)).<br>**Note:** This allows for a write message to this slave to be passed through the module database, and go directly into the ladder logic. The module will set the MCM.CONTROL.BPLastRead value to 9996 and the Modbus write command will be handled by rung 0 in the _PassThru ladder file. This allows for an unparsed Modbus message to be moved into the tag location MBMsg[0 to 499]. Here you will need to parse out the data value and move it into the appropriate registers using the ladder logic (not recommended, available for backwards compatibility with older versions of firmware only).<br>3 = Modbus slave port with formatted pass thru and data values swapped<br>**Note:** This mode will allow for the same register to be read and written by a Modbus master device, and will also swap the bytes within the data value (most devices will need to use a value of 4).<br>4 = Modbus slave port with formatted pass thru (no swapping).<br>**Note:** This mode will also allow for the same register location to be read and written by the master device. Rungs 1, 2, and 3 in the _PassThru ladder file will handle this information. |
| **FloatFlag** | As a Slave, emulates Enron/Daniel style floats. See Float Point Data Handling (page 63) for more information. |
| **FloatStart** | Register offset in message for floating data point. See Float Point Data Handling (page 63) for more information. |
| FloatOffset | Used to locate the floating point data into the module memory. Refer to "Floating Point Support." |

| Value | Description |
|---|---|
| **Protocol** | 0 = Modbus RTU mode, 1 = Modbus ASCII mode |
| **Baudrate** | Sets the baud rate that the port will operate at. Valid values for this field are 110, 150, 300, 600, 1200, 4800, 9600, 19200, 384 or 3840 (for 38,400 baud), 576 or 5760 (for 57,600 baud) and 115,1152, or 11520 (for 115,200 baud) |
| **Parity** | 0 = None, 1 = Odd, 2 = Even |
| **DataBits** | 8 = Modbus RTU mode, 8 or 7 = Modbus ASCII mode |
| **StopBits** | Valid values are 1 or 2 |
| RTSOn | 0 to 65536 milliseconds delay before it issues the message |
| RTSOff | 0 to 65536 milliseconds delay after it issues the message |
| MinResp | milliseconds wait before response to the master |
| UseCTS | If the parameter is set to 0, the CTS line will not be monitored. If the parameter is set to 1, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire) |
| **SlaveID** | Valid values are 1 to 247 |
| BitInOffset | Register value to offset address 10001 of the module memory |
| WordInOffset | Register value to offset address 30001 of the module memory |
| OutOffset | Register value to offset address 0001 of the module memory. |
| HoldOffset | Register value to offset address 40001 of the module memory |
| InterCharacterDelay | 0 to 65535 milliseconds time between characters to signal end of message<br>**Note:** Within Modbus RTU a character gap or quiet time on the line signals the end of the message. This is typically 3.5 character widths, as specified by the Modbus protocol. In some Radio or Modem applications, there may be more of a delay between characters. |

## 4.5 Floating-Point Data Handling

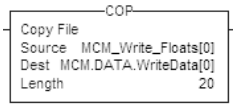
In most applications, the use of floating-point data requires no special handling.

**1** Copy the data to and from the MVI56-MCM module with a tag configured as a data type REAL in the ControlLogix processor.

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

**2** First, configure a tag within the ControlLogix processor.

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

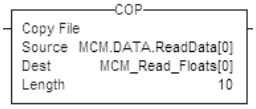



The length of the copy statement is determined by the Dest file size. To copy 10 floats from the MCM_Write_Floats array to the **MCM.DATA.WRITEDATA** array, the length of the COP statement must be set to a value of 20.

*To copy data from the MVI56-MCM module to a floating-point tag within the ControlLogix processor*

**1** Configure a tag within the ControlLogix processor as shown.

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




Once again, the COP statement will take as many of the Source elements required to fill the Dest tag for the length specified. Therefore, the COP statement will take **MCM.DATA.READDATA[0] TO [19]** to fill the **MCM_READ_FLOATS[0] TO [9].**

### *4.5.1 Enron/Daniel Float Configuration*

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

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

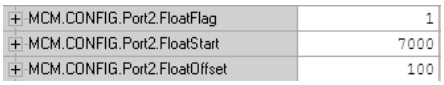
For example, an Enron Float device is required to access address 47001 for floating-point data, and each Modbus register would emulate a single float value (does not require 2 Modbus addresses for 1 float value).

A Master device requiring this type of addressing, would require that for every count of 1, the MVI56-MCM module responds to the request message with 4 bytes (one 32-bit REAL) value.

To emulate this addressing, the module has the parameters **MCM.CONFIG.PORTX.FLOATFLAG, FLOATSTART,** and **FLOATOFFSET.**

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

Here is a sample configuration for the module.




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

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

# 5 Verify Communication

*In This Chapter*


- MVI56-MCM Status Data Definition as a Master ..... 67
- Verify Master Communications ..... 68
- Verify Slave Communications ..... 72


There are several ways to verify that the MVI56-MCM module is communicating with the processor and with the Modbus Master/Slave network. You can ...

- View Exchanged Data
- View the Module Status in the RSLogix 5000 Controller Tags (page 123).

View the LED Status Indicators

## 5.1 MVI56-MCM Status Data Definition as a Master

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

| Offset | Content | Description |
|---|---|---|
| 202 | Program Scan Count | This value is incremented each time a complete program cycle occurs in the module. |
| 203 to 204 | Product Code | These two registers contain the product code of "MCM". |
| 205 to 206 | Product Version | These two registers contain the product version for the current running software. |
| 207 to 208 | Operating System | These two registers contain the month and year values for the program operating system. |
| 209 to 210 | Run Number | These two registers contain the run number value for the currently running software. |
| 211 | Port 1 Command List Requests | This field contains the number of requests made from this port to Slave devices on the network. |
| 212 | Port 1 Command List Response | This field contains the number of Slave response messages received on the port. |
| 213 | 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. |
| 214 | Port 1 Requests | This field contains the total number of messages sent from the port. |
| 215 | Port 1 Responses | This field contains the total number of messages received on the port. |
| 216 | Port 1 Errors Sent | This field contains the total number of message errors sent from the port. |

| Offset | Content | Description |
|---|---|---|
| 217 | Port 1 Errors Received | This field contains the total number of message errors received on the port. |
| 218 | Port 2 Command List Requests | This field contains the number of requests made from this port to Slave devices on the network. |
| 219 | Port 2 Command List Response | This field contains the number of Slave response messages received on the port. |
| 220 | 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. |
| 221 | Port 2 Requests | This field contains the total number of messages sent out the port. |
| 222 | Port 2 Responses | This field contains the total number of messages received on the port. |
| 223 | Port 2 Errors Sent | This field contains the total number of message errors sent out the port. |
| 224 | Port 2 Errors Received | This field contains the total number of message errors received on the port. |
| 225 | Read Block Count | This field contains the total number of read blocks transferred from the module to the processor. |
| 226 | Write Block Count | This field contains the total number of write blocks transferred from the module to the processor. |
| 227 | Parse Block Count | This field contains the total number of blocks successfully parsed that were received from the processor. |
| 228 | Command Event Block Count | This field contains the total number of command event blocks received from the processor. |
| 229 | Command Block Count | This field contains the total number of command blocks received from the processor. |
| 230 | Error Block Count | This field contains the total number of block errors recognized by the module. |
| 231 | Port 1 Current Error | For a Master Port, this field contains the index of the currently executing command. |
| 232 | Port 1 Last Error | For a Master Port, this field contains the index of the command with the error. |
| 233 | Port 2 Current Error | For a Master Port, this field contains the index of the currently executing command. |
| 234 | Port 2 Last Error | For a Master Port, this field contains the index of the command with an error. |

## 5.2 Verify Master Communications

The Modbus Master commands are configured, now it is time to verify that these commands are working correctly.

Within the MVI56-MCM module, there are a couple of ways of checking to see if the commands that have been configured in the previous location are working correctly.

The most common, and detailed method of checking the communications is using the **MCM.CONFIG.PORTX.CMDERRPTR** parameter. This parameter will tell you the individual status of each command that is issued by the module. Another method is by checking the **MCM.STATUS.PRTXERRS** location for total commands issued, responses received, errors, and so on.

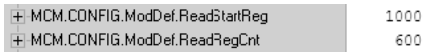
### 5.2.1 Command Error Codes

The MVI56-MCM module will return an individual error code for every command configured within the **MCM.CONFIG.PORTXMASTERCMD** section. The location of these error codes are determined by the parameter **MCM.CONFIG.PORTX.CMDERRPTR.** This parameter determines where in the module's 5000-register database the error codes for each command will be placed. The amount of error codes returned into the database is determined by the **MCM.CONFIG.PORTX.CMDCOUNT** parameter, therefore is the maximum number of commands have been selected (100), then 100 register will be placed into the module memory.

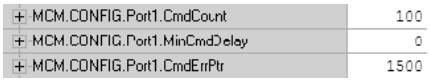
To be useful in the application, these error codes must be placed within the **MCM.DATA.READDATA** array.

Once again, the configuration in the **MCM.CONFIG.MODDEF** section for **READSTARTREG,** and **READREGCOUNT** determine which of the 5000 register will be presented to the ControlLogix processor and placed in the tag **MCM.DATA.READDATA** array.

Based on the sample configuration values for **READSTARTREG** and **READREGCNT,** this will be addresses 1000 to 1599 of the module memory. Below are the sample configuration values.




Based on these values shown above, a good place for the **MCM.CONFIG.PORTX.CMDERRPTR** is address 1500, as shown.




With the **CMDERRPTR** pointer set to address 1500 and the **CMDCOUNT** set to a value of 100, this will place your Command Error Data at addresses 1500 to 1599 of the module memory, and because of the before mentioned configuration of the **MCM.CONFIG.MODDEF READSTARTREG** and **READREGCNT** parameters, the command error data will be placed into the tags **MCM.DATA.READDATA[500] TO [599].**

Each command configured in the **MCM.CONFIG.PORTX.MASTERCMD** will occupy one register within the **READDATA** array. Based on the sample configuration values, the following table is true.

| Error Code for Command | ReadData Location |
|---|---|
| MCM.CONFIG.Port1MasterCmd[0] | MCM.DATA.ReadData[500] |
| MCM.CONFIG.Port1MasterCmd[1] | MCM.DATA.ReadData[501] |
| MCM.CONFIG.Port1MasterCmd[2] | MCM.DATA.ReadData[502] |
| MCM.CONFIG.Port1MasterCmd[3] | MCM.DATA.ReadData[503] |
| MCM.CONFIG.Port1MasterCmd[4] | MCM.DATA.ReadData[504] |
| MCM.CONFIG.Port1MasterCmd[98] | MCM.DATA.ReadData[598] |
| MCM.CONFIG.Port1MasterCmd[99] | MCM.DATA.ReadData[599] |

So know that you know where to look for the error data, you need to know what the individual error codes are.

Below is a list of the error codes for the module:

*Standard Modbus Protocol Errors*

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

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

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

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

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

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

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

*Module Communication Error Codes*

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

"Module Communication Errors" are generated by the MVI56-MCM module, and indicate communication errors with the Slave device.

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

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

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

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

*Command List Entry Errors*

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

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

For all commands that have not been configured (all parameters set to a value of 0) you will receive an error code of -44. To remove this error code, you can change your **MCM.CONFIG.PORTX.CMDCOUNT** parameter to the number of commands that are actually configured, cycle power to the module, or toggle the **MCM.CONTROL.WARMBOOT** or **COLDBOOT** bit to transfer the new values to the module.

### *5.2.2 MCM STATUS Data*

Status information can also be obtained from the MVI56-MCM module by checking the **MCM.STATUS.PRTXERRS** location. Below is a sample.

| MCM.STATUS.Prt1Errs | {...} |
|---|---|
| MCM.STATUS.Prt1Errs.CmdReq | 1768 |
| MCM.STATUS.Prt1Errs.CmdResp | 1768 |
| MCM.STATUS.Prt1Errs.CmdErr | 0 |
| MCM.STATUS.Prt1Errs.Requests | 1768 |
| MCM.STATUS.Prt1Errs.Responses | 1768 |
| MCM.STATUS.Prt1Errs.ErrSent | 0 |
| MCM.STATUS.Prt1Errs.ErrRec | 0 |

If your system is working correctly, you will see **CMDREQ, CMDRESP, REQUESTS,** and **RESPONSES** all incrementing together. If you see that **CMDERR** is incrementing, determine what command is causing the error (using the error code defined in the previous section (page 69)) and correct the issue causing the error.

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

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

| | |
|---|---|
| + MCM.STATUS.Port1LastErr | 2 |
| + MCM.STATUS.Port1PreviousErr | 1 |

This indicates the command index that last generated an error and does not indicate a command currently in error. In the above example, a value of 2 in **PORT1LASTERR** indicates that the last error was generated by **MCM.PORT1MASTERCMD[2].** This does not indicate that this command is currently in error. The value in **MCM.STATUS.PORT1PREVIOUSERR** indicates that before **MASTERCMD[2]** generated an error, **MCM.PORT1.MASTERCMD[1]** posted an error.

## 5.3 Verify Slave Communications

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

Below is an example.

| | |
|---|---|
| + MCM.STATUS.Prt2Errs.Requests | 5382 |
| + MCM.STATUS.Prt2Errs.Responses | 5382 |

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

### *5.3.1 MVI56-MCM Status Data Definition as a Slave*

Refer to MVI56-MCM Status Data Definition (page 123) for complete listing of Status information.

# 6 Diagnostics and Troubleshooting

*In This Chapter*


- Reading Status Data from the Module ..... 73
- LED Status Indicators ..... 85


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

- Status data values are transferred from the module to the processor
- Data contained in the module can be viewed through the Configuration/Debug port attached to a terminal emulator
- LED status indicators on the front of the module provide information on the module's status

## 6.1 Reading Status Data from the Module

The MVI56-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 ControlLogix processor continuously with each read block. For a complete listing of the status data object, refer to MVI56-MCM Status Data Definition (page 123).

### *6.1.1 The Configuration/Debug Menu*

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 sub-menus 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 terminal application (for example, HyperTerminal). 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.

### *6.1.2 Required Hardware*

You can connect directly from your computer's serial port to the serial port on the module to view configuration information and perform maintenance.

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

- 80486 based processor (Pentium preferred)
- 1 megabyte of memory
- At least one serial communications port available
- A null modem serial cable.

### 6.1.3 Required Software

In order to send and receive data over the serial port (COM port) on your computer to the module, you must use a communication program (terminal emulator).

A simple communication program called HyperTerminal is pre-installed with recent versions of Microsoft Windows operating systems. If you are connecting from a machine running DOS, you must obtain and install a compatible communication program. The following table lists communication programs that have been tested by ProSoft Technology.

| | |
|---|---|
| DOS | ProComm, as well as several other terminal emulation programs |
| Windows 3.1 | Terminal |
| Windows 95/98 | HyperTerminal |
| Windows NT/2000/XP | HyperTerminal |

### 6.1.4 Using the Configuration/Debug Port

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

1. Connect your computer to the module's port using a null modem cable.
2. Start the communication program on your computer and configure the communication parameters with the following settings:

| | |
|---|---|
| Baud Rate | 57,600 |
| Parity | None |
| Data Bits | 8 |
| Stop Bits | 1 |
| Software Handshaking | None |

3. Open the connection. When you are connected, press the **[?]** key on your keyboard. If the system is set up properly, you will see a menu with the module name followed by a list of letters and the commands associated with them.

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

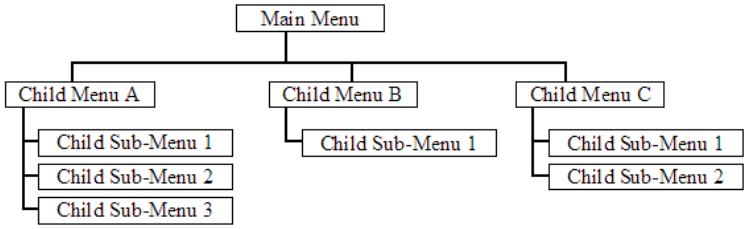
1. Verify that the null modem cable is connected properly between your computer's serial port and the module. A regular serial cable will not work.
2. Verify that RSLinx is not controlling the COM port. Refer to Disabling the RSLinx Driver for the Com Port on the PC.
3. Verify that your communication software is using the correct settings for baud rate, parity and handshaking.
4. 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, you can contact ProSoft Technology Technical Support for further assistance.

*Navigation*

All of the sub-menus for this module contain commands to redisplay the menu or return to the previous menu. You can always return from a sub-menu 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 you the menus available for this module, and briefly discusses the commands available to you.

*Keystrokes*

The keyboard commands on these menus are almost always non-case sensitive. You can enter most commands in lower case or capital 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][/].**

Also, take care to distinguish capital letter **[I]** from lower case letter **[L]** (L) and number **[1];** likewise for capital letter **[O]** and number **[0].** Although these characters look nearly the same on the screen, they perform different actions on the module.

### 6.1.5 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 on your terminal screen:

```
MODBUS MASTER/SLAVE COMMUNICATION MODULE (MVI56-MCM) 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
  V=Version Information
  W=Warm Boot Module
  Y=Transfer Module Cfg to Processor
  Communication Status : 1=Port 1   2=Port 2
  Port Configuration   : 6=Port 1   7=Port 2

  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.

*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 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.

*Viewing Block Transfer Statistics*

Press **[B]** from the Main Menu to view the Block Transfer Statistics screen.

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: To determine the number of blocks transferred each second, mark the numbers displayed at a specific time. Then some seconds later activate the command again. Subtract the previous numbers from the current numbers and divide by the quantity of seconds passed between the two readings.

*Viewing Module Configuration*

Press **[C]** to view the Module Configuration screen.

Use this command to display the current configuration and statistics for the module.

*Opening the Database Menu*

Press **[D]** to open the Database View menu. Use this menu command to view the current contents of the module's database.

*Opening the Command Error List Menu*

Press **[E]** to open the Command Error List. This list consists of multiple pages of command list error/status data. Press **[?]** to view a list of commands available on this menu.

*Viewing the Slave Status List (Port 1 and 2)*

Press **[O]** (port 1) or **[P]** (port 2) to view the 256 slave status values associated with the ports. The slave status values are defined as follows:

0 = slave is not used,
1 = slave being actively polled,
2 = slave suspended and
3 = slave disabled.

*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.

*Warm Booting the Module*

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.

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 re-boot.

*Transferring Module Configuration to the Processor*

Press **[Y]** to transfer the module's configuration data to the processor. Ladder logic is required in the processor to receive and implement the updated configuration. You will be prompted to confirm the transfer.

If the operation is not successful, an error code will be returned.

| Code | Description |
|---|---|
| 0 | Transfer successful |
| -1 | Error transferring module configuration data (block -9000) |
| -2 | Error transferring device definition data (blocks -9100 to -9103) |
| -3 | Error transferring master command list data (blocks -6000 to -6007) |

After successful data transfer, the module will perform a warm-boot operation to read in the new data.

*Communication Status (Ports 1 and 2)*

Press **[1]** or **[2]** to view the communication status and statistics of the specified Modbus port. This information can be useful for troubleshooting network problems.

*Port Configuration (Ports 1 and 2)*

Press **[6]** or **[7]** to view the configuration information for the selected Modbus port.

*Exiting the 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.

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.

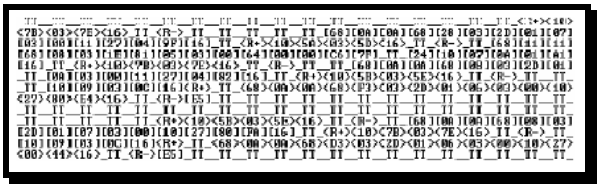
### *6.1.6 Data Analyzer*

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.

Note: The Port selection commands on the Data Analyzer menu differs very slightly in different modules, but the functionality is basically the same. Use the illustration above as a general guide only. Refer to the actual data analyzer menu on your module for the specific port commands to use.
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.

*Analyzing Data for the first application port*

Press **[1]** to display I/O data for the first application port in the Data Analyzer. The following illustration shows an example of the Data Analyzer output.



*Analyzing Data for the second application port*

Press **[2]** to display I/O data for the second application port in the Data Analyzer.

*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.

| Key | Interval |
|---|---|
| **[5]** | 1 milliseconds ticks |
| **[6]** | 5 milliseconds ticks |
| **[7]** | 10 milliseconds ticks |
| **[8]** | 50 milliseconds ticks |
| **[9]** | 100 milliseconds ticks |
| **[0]** | 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]** to display the data on the current page in hexadecimal format.

*Viewing Data in ASCII (Text) Format*

Press **[A]** 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.

```
<R+><01><03><00><00><00><0A><C5><CD><R->_TT_[01][03][14][00][00][00][00][00][00]
_TT_[00][00][00][00][00][00][00][00][00][00][00][00][00][A3][67]_TT_<R+><01>
<03><00><00><00><0A><C5><CD><R->_TT_[01][03][14][00][00][00][00][00][00][00][00]
[00][00][00][00][00][00][00]_TT_[00][00][00][00][00][A3][67]_TT_<R+><01><03><00>
<00><00><0A><C5><CD><R->_TT_[01][03][14][00][00][00][00][00][00][00][00][00][00]
[00][00][00][00][00]_TT_[00][00][00][00][00][A3][67]_TT_<R+><01><03><00><00><00>
<0A><C5><CD><R->_TT_[01][03][14][00][00][00][00][00]_TT_[00][00][00][00][00][00]
[00][00][00][00][00][00][00][00][00][A3][67]_TT_<R+><01><03><00><00><00><0A><C5>
<CD><R->_TT_[01][03][14][00][00][00][00][00][00]_TT_[00][00][00][00][00][00][00]
[00][00][00][00][00][00][00][00][A3][67]_TT_<R+><01><03><00><00><00><0A><C5><CD><R->
_TT_[01][03][14][00][00][00][00][00][00]_TT_[00][00][00][00][00][00][00][00][00]
[00][00][00][00][00][00][A3][67]_TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT_[01]
[03][14][00][00][00][00][00][00][00][00][00][00][00][00]_TT_[00][00][00][00][00]
[00][00][00][A3][67]_TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT_[01][03][14]
[00][00][00][00][00][00][00][00][00][00][00][00][00]_TT_[00][00][00][00][00][00]
[00][A3][67]_TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT_[01][03][14][00][00]
[00][00][00]_TT_[00][00][00][00][00][00][00][00][00][00][00][00][00][00][A3]
[67]_TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT_[01][03][14][00][00][00][00]
[00][00]_TT_[00][00][00][00][00][00][00][00][00][00][00][00][00][A3][67]_TT_
```

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.

### *6.1.7 Data Analyzer Tips*

From the main menu, press **[A]** for the "Data Analyzer". You should see the following text appear on the screen:

```
Data Analyzer Mode Selected
```

After the "Data Analyzer" mode has been selected, press **[?]** to view the Data Analyzer menu. You will see the following menu:

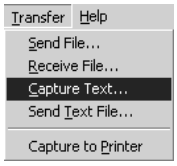
```
DATA ANALYZER VIEW MENU
 ?=Display Menu
 1=Select Port 1
 2=Select Port 2
 5=1 mSec Ticks
 6=5 mSec Ticks
 7=10 mSec Ticks
 8=50 mSec Ticks
 9=100 mSec Ticks
 0=No mSec Ticks
 H=Hex Format
 A=ASCII Format
 B=Start
 S=Stop
 M=Main Menu

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

From this menu, you can select the "Port", the "format", and the "ticks" that you can display the data in.

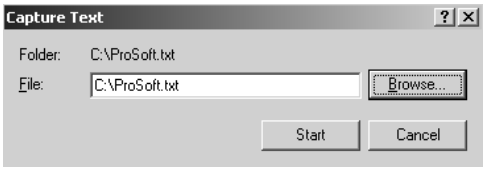
For most applications, HEX is the best format to view the data, and this does include ASCII based messages (because some characters will not display on HyperTerminal and by capturing the data in HEX, we can figure out what the corresponding ASCII characters are supposed to be).

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

After you have selected the Port, Format, and Tick, we are now ready to start a capture of this data. The easiest way to do so is to go up to the top of you HyperTerminal window, and do a **TRANSFER / CAPTURE TEXT** as shown below:




After selecting the above option, the following window will appear:




Next name the file, and select a directory to store the file in. In this example, we are creating a file ProSoft.txt and storing this file on our root C: drive. After you have done this, press the Start button.

Now you have everything that shows up on the HyperTerminal screen being logged to a file called ProSoft.txt. This is the file that you will then be able to email to ProSoft Technical Support to assist with issues on the communications network.

To begin the display of the communications data, you will then want to press **[B]** to tell the module to start printing the communications traffic out on the debug port of the module. After you have pressed **[B],** you should see something like the following:

```
[03][00][04][00][05][00][06][00][07][00][08][00][09][FB][B7]_TT__TT_<R+><01><02>
<00><00><00><0A><F8><0D><R->_TT__TT__TT_[01][02][02][00][00][B9][B8]_TT__TT_<R+>
<01><03><00><00><00><0A><C5><CD><R->_TT__TT_[01][03][14][00][00][00][01][00]_TT_
[02][00][03][00][04][00][05][00][06][00][07][00][08][00][09][CD][51]_TT__TT_<R+>
<01><01><00><00><00><A0><3C><72><R->_TT__TT_[01][01][14][00][00][01][00][02]_TT_
[00][03][00][04][00][05][00][06][00][07][00][08][00][09][00][B7][52]_TT__TT_<R+>
<01><04><00><00><00><0A><70><0D><R->_TT__TT_[01][04][14][00][00][00][01][00]_TT_
[02][00][03][00][04][00][05][00][06][00][07][00][08][00][09][FB][B7]_TT__TT_<R+>
<01><02><00><00><00><0A><F8><0D><R->_TT__TT__TT_[01][02][02][00][00][B9][B8]_TT_
_TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT__TT_[01][03][14][00][00][00][01]
[00]_TT_[02][00][03][00][04][00][05][00][06][00][07][00][08][00][09][CD][51]_TT_
_TT_<R+><01><01><00><00><00><A0><3C><72><R->_TT__TT__TT_[01][01][14][00][00][01]
[00][02]_TT_[00][03][00][04][00][05][00][06][00][07][00][08][00][09][00][B7][52]
_TT__TT_<R+><01><04><00><00><00><0A><70><0D><R->_TT__TT_[01][04][14][00][00][00]
[01][00]_TT_[02][00][03][00][04][00][05][00][06][00][07][00][08][00][09][FB][B7]
_TT__TT_<R+><01><02><00><00><00><0A><F8><0D><R->_TT__TT_[01][02][02][00][00][B9]
[B8]_TT__TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT__TT_[01][03][14][00][00]
[00][01][00]_TT_[02][00][03][00][04][00][05][00][06][00][07][00][08][00][09][CD]
[51]_TT__TT_<R+><01><01><00><00><00><A0><3C><72><R->_TT__TT__TT_[01][01][14][00]
[00][01][00][02]_TT_[00][03][00][04][00][05][00][06][00][07][00][08][00][09][00]
[B7][52]_TT__TT_<R+><01><04><00><00><00><0A><70><0D><R->_TT__TT_[01][04][14][00]
[00][00][01][00]_TT_[02][00][03][00][04][00][05][00][06][00][07][00][08][00][09]
[FB][B7]_TT__TT_<R+><01><02><00><00><00><0A><F8><0D><R->_TT__TT__TT_[01][02][02]
[00][00][B9][B8]_TT__TT_<R+><01><03><00><00><00><0A><C5><CD><R->_TT__TT_
```

The <R+> means that the module is transitioning the communications line to a transmit state.

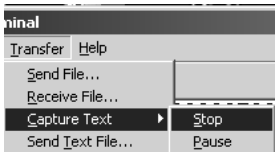
All characters shown in <> brackets are characters being sent out by the module.

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

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

After taking a minute or two of traffic capture, you will now want to stop the "Data Analyzer". To do so, press the [S] key, and you will then see the scrolling of the data stop.

When you have captured the data you want to save, open the Transfer menu and choose Capture Text. On the secondary menu, choose Stop.



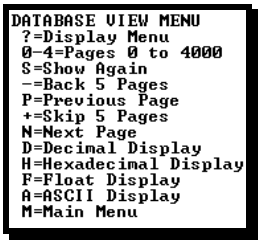

You have now captured, and saved the file to your PC. This file can now be used in analyzing the communications traffic on the line, and assist in determining communication errors.

### 6.1.8 Modbus Message Structure

Refer to Modbus Protocol Specification (page 130) for information on the structure of Modbus messages.

### 6.1.9 Modbus Database View

Press **[D]** to open the Modbus Database View menu. Use this command to view the module's internal database values. Press **[?]** to view a list of commands on this menu.




All data contained in the module's database is available for viewing using the commands. Refer to Modbus Protocol Specification (page 130) for information on the structure of Modbus messages. Each option available on the menu is discussed in the following topics.

*Viewing Register Pages*

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

| Command | Description |
|---|---|
| **[0]** | Display registers 0 to 99 |
| **[1]** | Display registers 1000 to 1099 |
| **[2]** | Display registers 2000 to 2099 |

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

*Redisplaying the Current Page*

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

*Moving Back Through 5 Pages of Registers*

Press **[-]** from the Database View menu to skip five pages back in the database to see the previous 100 registers of data.

*Viewing the Previous 100 Registers of Data*

Press **[P]** from the Database View menu to display the previous 100 registers of data.

*Moving Forward Through 5 Pages of Registers*

Press **[+]** from the Database View menu to skip five pages ahead in the database to see the next 100 registers of data.

*Viewing the Next 100 Registers of Data*

Press **[N]** from the Database View menu to select and display the next 100 registers of data.

*Viewing Data in Decimal Format*

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

*Viewing Data in Hexadecimal Format*

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

*Viewing Data in Floating Point Format*

Press **[F]** from the Database View menu. Use this command 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*

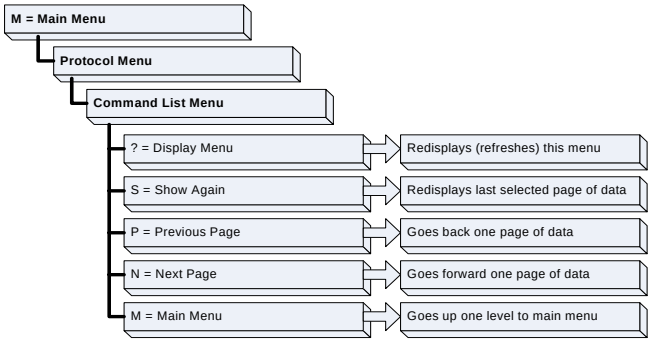
Press **[A]** to display the data on the current page in ASCII format. This is useful for regions of the database that contain ASCII data.

*Returning to the Main Menu*

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

### *6.1.10 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.

*Viewing the Previous 20 Commands*

Press **[-]** to display data for the previous 20 commands.

*Viewing the Previous Page of Commands*

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

*Viewing the Next 20 Commands*

Press **[+]** to display data for the next 20 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.

## 6.2 LED Status Indicators

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

| Module | Color | Status | Indication |
|---|---|---|---|
| CONFIG | 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 its Modbus 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 its Modbus Port 2. |
| | | Off | No data is being transferred on the port. |
| APP | Amber | On | The MVI56-MCM is working normally. |
| | | Off | The MVI56-MCM module program has recognized a communication error on one of its Modbus ports. |
| 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 will be red and the APP and BP ACT LEDs will be on. If the LEDs are latched in this mode for a long period of time, look at the configuration error words in the configuration request block. The structure of the block is shown in the following table.

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | 9000 | 1 |
| 2 | Module Configuration Errors | 1 |
| 3 | Port 1 Configuration Errors | 1 |
| 4 | Port 2 Configuration Errors | 1 |
| 5 to 248 | Spare | 244 |
| 249 | -2 or -3 | 1 |

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

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

The port configuration error words have the following definitions:

| Bit | Description | Value |
|---|---|---|
| 0 | Type code is not valid. Enter a value from 0 (master) to 1 (slave). | 0x0001 |
| 1 | The float flag parameter is not valid. | 0x0002 |
| 2 | The float start parameter is not valid. | 0x0004 |
| 3 | The float offset parameter is not valid. | 0x0008 |
| 4 | Protocol parameter is not valid. | 0x0010 |
| 5 | Baud rate parameter is not valid. | 0x0020 |
| 6 | Parity parameter is not valid. | 0x0040 |
| 7 | Data bits parameter is not valid. | 0x0080 |

| Bit | Description | Value |
|---|---|---|
| 8 | Stop bits parameter is not valid. | 0x0100 |
| 9 | Slave ID is not valid. | 0x0200 |
| 10 | Input bit or word, output word and/or holding register offset(s) are not valid. | 0x0400 |
| 11 | Command count parameter is not valid. | 0x0800 |
| 12 | Spare | 0x1000 |
| 13 | Spare | 0x2000 |
| 14 | Spare | 0x4000 |
| 15 | Spare | 0x8000 |

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

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

### 6.2.1 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 the configuration data being transferred to the module from the ControlLogix processor.

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 Support.

### 6.2.2 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.<br>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. Verify that all modules in the rack are configured in the ladder logic. |

Module Errors

| Problem Description | Steps to take |
|---|---|
| BP ACT LED remains off or blinks slowly | This indicates that backplane transfer operations are failing. Connect to the module's Configuration/Debug port to check this. |
| MVI56E modules with scrolling LED display: <*Backplane Status*> condition reads ERR | To establish backplane communications, verify the following items:<br>▪ The processor is in Run mode.<br>▪ The backplane driver is loaded in the module.<br>▪ The module is configured for read and write block data transfer.<br>▪ The ladder logic handles all read and write block situations.<br>▪ The module is configured in the processor. |
| OK LED remains red | The program has halted or a critical error has occurred. Connect to the Configuration/Debug port to see if the module is running. If the program has halted, turn off power to the rack, remove the card from the rack and re-insert the card in the rack, and then restore power to the rack. |

# 7 Reference

*In This Chapter*


- Product Specifications ..... 89
- Functional Overview ..... 91
- Cable Connections ..... 110
- MVI56-MCM Database Definition ..... 114
- MVI56-MCM Configuration Data ..... 115
- MVI56-MCM Status Data Definition ..... 123
- MVI56-MCM Command Control ..... 124
- MVI56-MCM User Defined Data Types ..... 125
- Modbus Protocol Specification ..... 130
- Using the Sample Program - RSLogix Version 15 and earlier ..... 141


## 7.1 Product Specifications

The MVI56 Modbus Master/Slave Communication Module allows Rockwell Automation® ControlLogix® processors to interface easily with other Modbus protocol compatible devices.

The module acts as an input/output module between the Modbus network and the ControlLogix 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 host SCADA packages also support the Modbus protocol.

### *7.1.1 General Specifications*

- Single Slot - 1756 backplane compatible
- Local or remote rack
- 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.
- Configuration data obtained through user-defined ladder. Sample ladder file included

### 7.1.2 Hardware Specifications

| Specification | Description |
|---|---|
| Backplane Current Load | 800 mA @ 5 V DC<br>3mA @ 24V DC |
| Operating Temperature | 0 to 60°C (32 to 140°F) |
| Storage Temperature | -40 to 85°C (-40 to 185°F) |
| Shock | 30g Operational<br>50g non-operational<br>Vibration: 5 g from 10 to 150 Hz |
| Relative Humidity | 5% to 95% (non-condensing) |
| LED Indicators | Module Status<br>Backplane Transfer Status<br>Application Status<br>Serial Activity |
| **Debug/Configuration port (CFG)** | |
| CFG Port (CFG) | RJ45 (DB-9M with supplied cable)<br>RS-232 only |
| **Application ports (PRT1 & PRT2)**<br>Full hardware handshaking control, providing radio, modem and multi-drop support | |
| Software configurable communication parameters | Baud rate: 110 to 115,200 baud, depending on protocol<br>RS-232, 485 and 422<br>Parity: none, odd or even<br>Data bits: 5, 6, 7, or 8<br>Stop bits: 1 or 2<br>RTS on/off delay: 0 to 65535 milliseconds |
| App Ports (P1, P2) (Serial modules) | RJ45 (DB-9M with supplied cable)<br>RS-232 handshaking configurable<br>500V Optical isolation from backplane |
| Shipped with Unit | RJ45 to DB-9M cables for each port<br>6-foot RS-232 configuration cable |

### 7.1.3 General Specifications - Modbus Master/Slave

| | | |
|---|---|---|
| Communication parameters | Baud Rate: 110 to 115K baud<br>Stop Bits: 1 or 2<br>Data Size: 7 or 8 bits<br>Parity: None, Even, Odd<br>RTS Timing delays: 0 to 65535 milliseconds | |
| Modbus Modes | RTU mode (binary) with CRC-16<br>ASCII mode with LRC error checking | |
| Floating Point Data | Floating point data movement supported, including configurable support for Enron implementation | |
| Modbus Function Codes | 1: Read Coil Status<br>2: Read Input Status<br>3: Read Holding Registers<br>4: Read Input Registers<br>5: Force (Write) Single Coil<br>6: Preset (Write) Single Holding Register | 8: Diagnostics<br>15: Force( Write) Multiple Coils<br>16: Preset (Write) Multiple Holding Registers<br>17: Report Slave ID<br>22: Mask Write 4x Register<br>23: Read/Write 4x Registers |

### *7.1.4 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 ControlLogix processor.

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

## 7.2 Functional Overview

This section describes how the MVI56-MCM module transfers data between itself and the processor, and how it implements the Modbus Master/Slave protocol.

### *7.2.1 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 MVI56-MCM module can work as a Master and as a Slave.

The MVI56-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.

### 7.2.2 General Concepts

The following topics describe several concepts that are important for understanding the operation of the MVI56-MCM module.

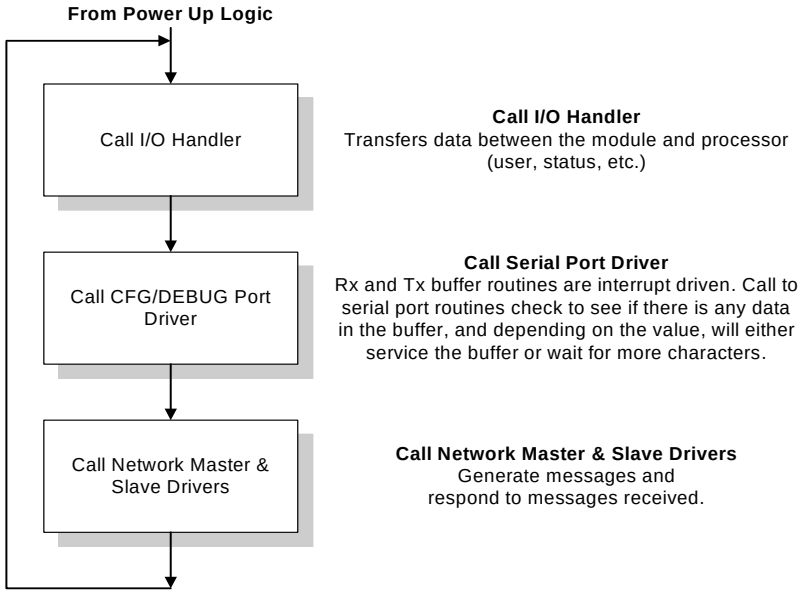
On power up the module begins performing the following logical functions:

1. Initialize hardware components
2. Initialize ControlLogix backplane driver
   - Test and Clear all RAM
   - Initialize the serial communication ports
   - Wait for Module Configuration from ControlLogix processor
3. Initialize Module Register space
4. Enable Slave Driver on selected ports
5. Enable Master Driver on selected ports

After the module has received the Module Configuration Block from the processor, the module will begin communicating with other nodes on the network, depending on the configuration.

### 7.2.3 Main Logic Loop

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



### 7.2.4 ControlLogix Processor Not in Run

Whenever the module detects that the processor is not in Run mode (for example, Fault or PGM), the Modbus Ports can be shut down as prescribed in the user configuration. When the processor is returned to a running state, the module will resume communications on the network.

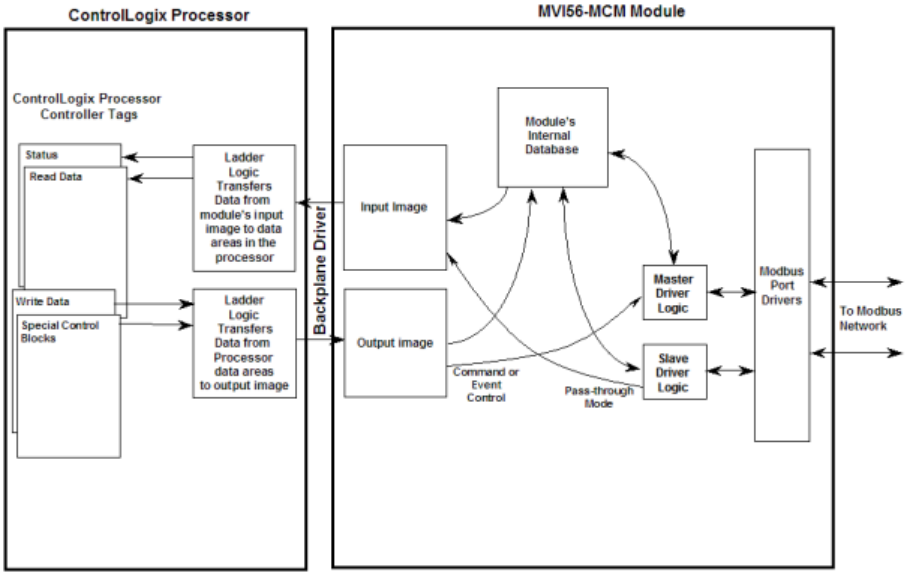
### 7.2.5 Backplane Data Transfer

The MVI56-MCM module communicates directly over the ControlLogix backplane. Data is paged between the module and the ControlLogix processor across the backplane using the module's input and output images. The update frequency of the images is determined by the scheduled scan rate defined by the user for the module and the communication load on the module. Typical updates are in the range of 2 to 10 milliseconds.

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 is set to 250 words. This large data area permits fast throughput of data between the module and the processor.

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 is set to 248 words. This large data area permits fast throughput of data from the processor to the module.

The following illustration shows the data transfer method used to move data between the ControlLogix processor, the MVI56-MCM module and the Modbus Network.

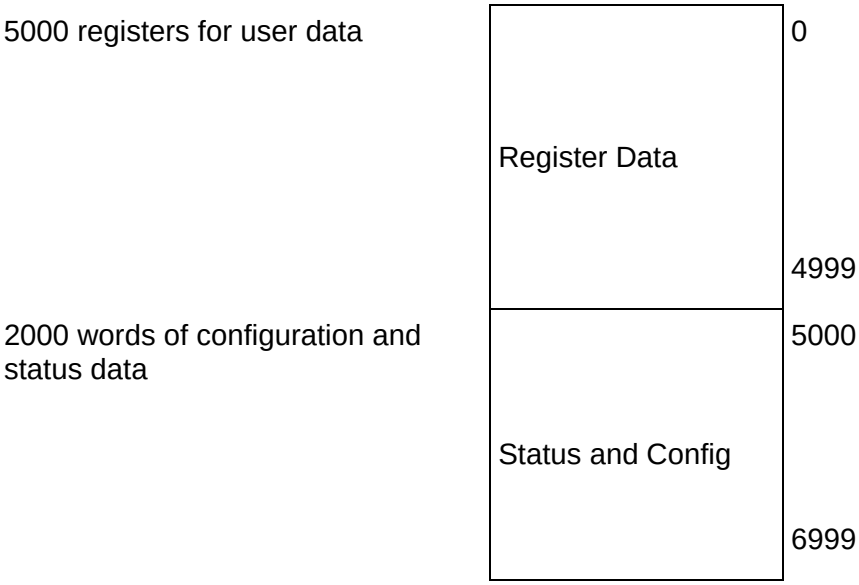


As shown in the illustration above, all data transferred between the module and the processor over the backplane is through the input and output images. Ladder logic must be written in the ControlLogix processor to interface the input and output image data with data defined in the Controller Tags. All data used by the module is stored in its internal database. This database is defined as a virtual Modbus data table with addresses from 0 to 6999.

The database is translated into a Modbus data table, according to which a Modbus command is received or sent. Refer to Modbus Memory Map (page 57) for more information.

The following illustration shows the layout of the database:

Module's Internal Database Structure




Data contained in this database is paged through the input and output images by coordination of the ControlLogix ladder logic and the MVI56-MCM module's program. Up to 248 words of data can be transferred from the module to the processor at a time. Up to 247 words of data can be transferred from the processor to the module. Each image has a defined structure depending on the data content and the function of the data transfer as defined below.

### *7.2.6 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. Refer to Integrating the Sample Ladder Logic into an Existing Project (page 147) for a description of the data objects used with the blocks and the ladder logic required. The structure and function of each block is discussed below.

*Read Block*

These blocks of data transfer information from the module to the ControlLogix processor. The following table describes the structure of the input image.

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | Write Block ID | 1 |
| 2 to 201 | Read Data | 200 |
| 202 | Program Scan Counter | 1 |
| 203 to 204 | Product Code | 2 |
| 205 to 206 | Product Version | 2 |
| 207 to 208 | Operating System | 2 |
| 209 to 210 | Run Number | 2 |
| 211 to 217 | Port 1 Error Status | 7 |
| 218 to 224 | Port 2 Error Status | 7 |
| 225 to 230 | Data Transfer Status | 6 |
| 231 | Port 1 Current Error/Index | 1 |
| 232 | Port 1 Last Error/Index | 1 |
| 233 | Port 2 Current Error/Index | 1 |
| 234 | Port 2 Last Error/Index | 1 |
| 235 to 248 | Spare | 14 |
| 249 | Read Block ID | 1 |

The Read Block ID is an index value used to determine the location of where the data will be placed in the ControlLogix processor controller tag array of module read data. Each transfer can move up to 200 words (block offsets 2 to 201) of data. In addition to moving user data, the block also contains status data for the module. This last set of data is transferred with each new block of data and is used for high-speed data movement.

The Write Block ID associated with the block requests data from the ControlLogix 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.

*Write Block*

These blocks of data transfer information from the ControlLogix processor to the module. The following table describes the structure of the output image.

| Offset | Description | Length |
|---|---|---|
| 0 | Write Block ID | 1 |
| 1 to 200 | Write Data | 200 |
| 201 to 247 | Spare | 47 |

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

### *7.2.7 Configuration Data Transfer*

When the module performs a restart operation, it will request configuration information from the ControlLogix processor. This data is transferred to the module in specially formatted write blocks (output image). The module will poll for each block by setting the required write block number in a read block (input image). Refer to Integrating the Sample Ladder Logic into an Existing Project (page 147) for a description of the data objects used with the blocks and the ladder logic required. The format of the blocks for configuration is given in the following topics.

*Module Configuration Data*

This block sends general configuration information from the processor to the module. The data is transferred in a block with an identification code of 9000. The structure of the block is displayed in the following table.

| Offset | Description | Length |
|---|---|---|
| 0 | 9000 | 1 |
| 1 to 6 | Backplane Setup | 6 |
| 7 to 31 | Port 1 Configuration | 25 |
| 32 to 56 | Port 2 Configuration | 25 |
| 57 to 59 | Port 1 Aux. Configuration | 3 |
| 60 to 62 | Port 2 Aux. Configuration | 3 |
| 63 to 247 | Spare | 185 |

The read block used to request the configuration has the following structure:

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | 9000 | 1 |
| 2 | Module Configuration Errors | 1 |
| 3 | Port 1 Configuration Errors | 1 |
| 4 | Port 2 Configuration Errors | 1 |
| 5 to 248 | Spare | 244 |
| 249 | -2 or -3 | 1 |

If there are any errors in the configuration, the bit associated with the error will be set in one of the three configuration error words. The error must be corrected before the module starts operating.

### 7.2.8 Master Command Data List

Each port on the module can be configured as a Modbus Master device containing its own list of one hundred commands. The commands are read from the processor using the following Write Block IDs: Modbus Port 1: 6000 to 6003, and Modbus Port 2: 6100 to 6103. The module will sequentially poll for each block from the processor. Ladder logic must handle all of the data transfers. The following table describes the structure of each block.

| Offset | Description | Length |
|---|---|---|
| 0 | 6000 to 6003 and 6100 to 6103 | 1 |
| 1 to 8 | Command Definition | 8 |
| 9 to 16 | Command Definition | 8 |
| 17 to 24 | Command Definition | 8 |
| 25 to 32 | Command Definition | 8 |
| 33 to 40 | Command Definition | 8 |
| 41 to 48 | Command Definition | 8 |
| 49 to 56 | Command Definition | 8 |
| 57 to 64 | Command Definition | 8 |
| 65 to 72 | Command Definition | 8 |
| 73 to 80 | Command Definition | 8 |
| 81 to 88 | Command Definition | 8 |
| 89 to 96 | Command Definition | 8 |
| 97 to 104 | Command Definition | 8 |
| 105 to 112 | Command Definition | 8 |
| 113 to 120 | Command Definition | 8 |
| 121 to 128 | Command Definition | 8 |
| 129 to 136 | Command Definition | 8 |
| 137 to 144 | Command Definition | 8 |
| 145 to 152 | Command Definition | 8 |
| 153 to 160 | Command Definition | 8 |
| 161 to 168 | Command Definition | 8 |
| 169 to 176 | Command Definition | 8 |
| 177 to 184 | Command Definition | 8 |
| 185 to 192 | Command Definition | 8 |
| 193 to 200 | Command Definition | 8 |

*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.

### 7.2.9 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:

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

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 (**ERRORDELAYCNTR** value in the **MCMPORT** object for each port). 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. This will enable polling of the Slave.

| Block ID | Description |
|---|---|
| 3002 | Request for first 128 Slave status values for Modbus Port 1 |
| 3003 | Request for last 128 Slave status values for Modbus Port 1 |
| 3102 | Request for first 128 Slave status values for Modbus Port 2 |
| 3103 | Request for last 128 Slave status values for Modbus Port 2 |

The following table describes the format of these blocks.

| Offset | Description | Length |
|---|---|---|
| 0 | 3002 to 3003 or 3102 to 3103 | 1 |
| 1 to 247 | Spare | 246 |

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

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | Write Block ID | 1 |
| 2 to 129 | Slave Poll Status Data | 128 |
| 130 to 248 | Spare | 119 |
| 249 | 3002 to 3003 or 3102 to 3103 | 1 |

Ladder logic can be written to override the value in the Slave status table. It can disable (state value of 3) by sending a special block of data from the processor to the Slave. Port 1 Slaves are disabled using block 3000, and Port 2 Slaves are disabled using block 3100. Each block contains the Slave node addresses to disable.

The following table describes the structure of the block.

| Offset | Description | Length |
|---|---|---|
| 0 | 3000 or 3100 | 1 |
| 1 | Number of Slaves in Block | 1 |
| 2 to 201 | Slave indexes | 200 |
| 202 to 247 | Spare | 46 |

The module will respond with a block with the same identification code received and indicate the number of Slaves acted on with the block. The following table describes the format of the response block.

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | Write Block ID | 1 |
| 2 | Number of Slaves processed | 1 |
| 3 to 248 | Spare | 246 |
| 249 | 3000 or 3100 | 1 |

Ladder logic can be written to override the value in the Slave status table to enable the Slave (state value of 1) by sending a special block. Port 1 Slaves are enabled using block 3001, and Port 2 Slaves are enabled using block 3101. Each block contains the Slave node addresses to enable. The following table describes the format for this block.

| Offset | Description | Length |
|---|---|---|
| 0 | 3001 or 3101 | 1 |
| 1 | Number of Slaves in Block | 1 |
| 2 to 201 | Slave indexes | 200 |
| 202 to 247 | Spare | 46 |

The module will respond with a block with the same identification code received and indicate the number of Slaves acted on with the block. The following table describes the format of this response block.

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | Write Block ID | 1 |
| 2 | Number of Slaves processed | 1 |
| 3 to 248 | Spare | 246 |
| 249 | 3001 or 3101 | 1 |

### 7.2.10 Command Control Blocks

Command control blocks are special blocks used to control the module or request special data from the module. The current version of the software supports five command control blocks: event command control, command control, write configuration, warm boot and cold boot.

*Event Command*

Event command control blocks send Modbus commands directly from the ladder logic to one of the Master Ports. The following table describes the format for these blocks.

| Offset | Description | Length |
|---|---|---|
| 0 | 1000 to 1255 or 2000 to 2255 | 1 |
| 1 | Internal DB Address | 1 |
| 2 | Point Count | 1 |
| 3 | Swap Code | 1 |
| 4 | Modbus Function Code | 1 |
| 5 | Device Database Address | 1 |
| 6 to 247 | Spare | 242 |

The block number defines the Modbus Port to be considered and the Slave node to be accessed. Blocks in the 1000 range are directed to Modbus Port 1, and blocks in the 2000 range are directed to Modbus Port 2. The Slave address is represented in the block number in the range of 0 to 255. The sum of these two values determines the block number. The other parameters passed with the block are used 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 points or registers for the command
- The **Swap Code** is used with Modbus function 3 requests to change the word or byte order
- The **Modbus Function Code** has one of the following values 1, 2, 3, 4, 5, 6, 15, or 16
- The **Device Database Address** is the Modbus register or point in the remote Slave device to be associated with the command

When the command receives the block, it will process it and place it in the command queue. The module will respond to each command block with a read block. The following table describes the format of this block.

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | Write Block ID | 1 |
| 2 | 0 = Fail, 1 = Success | 1 |
| 3 to 248 | Spare | 246 |
| 249 | 1000 to 1255 or 2000 to 2255 | 1 |

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).

*Command Control*

Command control blocks place commands in the command list into the command queue. Each port has a command queue of up to 100 commands. The module services commands in the queue before the Master command list. This gives high priority to commands in the queue. Commands placed in the queue through this mechanism must be defined in the Master command list. Under normal command list execution, the module will only execute commands with the Enable parameter set to one or two. If the value is set to zero, the command is skipped. Commands may be placed in the command list with an Enable parameter set to zero. These commands can then be executed using the command control blocks.

One to six commands can be placed in the command queue with a single request. The following table describes the format for this block.

| Offset | Description | Length |
|---|---|---|
| 0 | 5001 to 5006 or 5101 to 5106 | 1 |
| 1 | Command index (**MCM.CONFIG.PORTXMASTERCMD** [*command index value*]) | 1 |
| 2 | Command index (**MCM.CONFIG.PORTXMASTERCMD** [*command index value*]) | 1 |
| 3 | Command index (**MCM.CONFIG.PORTXMASTERCMD** [*command index value*]) | 1 |
| 4 | Command index (**MCM.CONFIG.PORTXMASTERCMD** [*command index value*]) | 1 |
| 5 | Command index (**MCM.CONFIG.PORTXMASTERCMD** [*command index value*]) | 1 |
| 6 | Command index (**MCM.CONFIG.PORTXMASTERCMD** [*command index value*]) | 1 |
| 7 to 247 | Spare | 241 |

Blocks in the range of 5001 to 5006 are used for Modbus Port 1, and blocks in the range of 5101 to 5106 are used for Modbus 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 to be used with Modbus Port 1. The Command index parameters in the block have a range of 0 to 99 and correspond to the Master command list entries.

The module responds to a command control block with a block containing the number of commands added to the command queue for the port. The following table describes the format for this block.

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | Write Block ID | 1 |
| 2 | Number of commands added to command queue | 1 |
| 3 to 248 | Spare | 246 |
| 249 | 5000 to 5006 or 5100 to 5106 | 1 |

*Write Configuration*

This block is sent from the ControlLogix processor to the module to force the module to write its current configuration back to the processor. This function is used when the module's configuration has been altered remotely using database write operations. The write block contains a value of -9000 in the first word. The module will respond with blocks containing the module configuration data. Ladder logic must handle the receipt of these blocks. The blocks transferred from the module are as follows:

Block -9000, General Configuration Data:

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | -9000 | 1 |
| 2 to 7 | Backplane Setup | 6 |
| 8 to 32 | Port 1 Configuration | 25 |
| 33 to 57 | Port 2 Configuration | 25 |
| 58 to 60 | Port 1 Configuration (continued) | 3 |
| 61 to 63 | Port 2 Configuration (continued) | 3 |
| 64 to 248 | Spare | 185 |
| 249 | -9000 | 1 |

Blocks -6000 to -6003 and -6100 to -6103, Master Command List Data for ports 1 and 2, respectively:

| Offset | Description | Length |
|---|---|---|
| 0 | Reserved | 1 |
| 1 | -6000 to -6003 and -6100 to -6103 | 1 |
| 2 to 9 | Command Definition | 8 |
| 10 to 17 | Command Definition | 8 |
| 18 to 25 | Command Definition | 8 |
| 26 to 33 | Command Definition | 8 |
| 34 to 41 | Command Definition | 8 |
| 42 to 49 | Command Definition | 8 |
| 50 to 57 | Command Definition | 8 |
| 58 to 65 | Command Definition | 8 |
| 66 to 73 | Command Definition | 8 |
| 74 to 81 | Command Definition | 8 |
| 82 to 89 | Command Definition | 8 |
| 90 to 97 | Command Definition | 8 |
| 98 to 105 | Command Definition | 8 |
| 106 to 113 | Command Definition | 8 |
| 114 to 121 | Command Definition | 8 |
| 122 to 129 | Command Definition | 8 |
| 130 to 137 | Command Definition | 8 |
| 138 to 145 | Command Definition | 8 |
| 146 to 153 | Command Definition | 8 |
| 154 to 161 | Command Definition | 8 |
| 162 to 169 | Command Definition | 8 |
| 170 to 177 | Command Definition | 8 |

| Offset | Description | Length |
|---|---|---|
| 178 to 185 | Command Definition | 8 |
| 186 to 193 | Command Definition | 8 |
| 194 to 201 | Command Definition | 8 |
| 202 to 248 | Spare | 47 |
| 249 | -6000 to -6003 and -6100 to -6103 | 1 |

Each of these blocks must be handled by the ladder logic for proper module operation.

*Warm Boot*

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

| Offset | Description | Length |
|---|---|---|
| 0 | 9998 | 1 |
| 1 to 247 | Spare | 247 |

*Cold Boot*

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

| Offset | Description | Length |
|---|---|---|
| 0 | 9999 | 1 |
| 1 to 247 | Spare | 247 |

### 7.2.11 Pass-Through Control Blocks

*Unformatted Pass-Through Control Blocks*

If one or more of the Slave Ports on the module are configured for the unformatted pass-through mode, the module will pass blocks with identification codes of 9996 to the processor for each received write command. Any Modbus function 5, 6, 15, and 16 commands will be passed from the port to the processor using this block identification number. Ladder logic must handle the receipt of all Modbus write functions to the processor and to respond as expected to commands issued by the remote Modbus Master device. The structure of the unformatted pass-through control block is shown in the following table.

| Offset | Description | Length |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 9996 | 1 |

| Offset | Description | Length |
|---|---|---|
| 2 | Number of bytes in Modbus message | 1 |
| 3 to 248 | Modbus message received | 246 |
| 249 | 9996 | 1 |

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 |
|---|---|---|
| 0 | 9996 | 1 |
| 1 to 247 | Spare | 247 |

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

*Formatted Pass-Through Control Blocks*

If one or more of the Slave Ports on the module are configured for the Formatted Pass-Through mode, the module will pass blocks with identification codes of 9996 to the processor for each received write command. Any Modbus function 5, 6, 15 or 16 commands will be passed from the port to the processor using this block identification number. Ladder logic must handle the receipt of all Modbus write functions to the processor and must respond as expected to commands issued by the remote Modbus Master device. The structure of the formatted pass-through control block is shown in the following tables:

Function 5

| Offset | Description | Length |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 9958 | 1 |
| 2 | 1 | 1 |
| 3 | Bit Address | 1 |
| 4 | Data | 1 |
| 5 to 248 | Modbus message received | 244 |
| 249 | 9958 | 1 |

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 |
|---|---|---|
| 0 | 9958 | 1 |
| 1 to 247 | Spare | 247 |

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

Function 6 and 16

| Offset | Description | Length |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 9956/9957 (Floating-point) | 1 |
| 2 | Number of data words | 1 |
| 3 | Data Address | 1 |
| 4 to 248 | Data | 244 |
| 249 | 9956/9957 | 1 |

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 |
|---|---|---|
| 0 | 9956/9957 | 1 |
| 1 to 247 | Spare | 247 |

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

Function 15

When the module receives a function code 15 while 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.

| Offset | Description | Length |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 9959 | 1 |
| 2 | Number of Words | 1 |
| 3 | Word Address | 1 |
| 4 to 53 | Data | 50 |
| 54 to 103 | Mask | 50 |
| 104 to 248 | Spare | 145 |
| 249 | 9959 | 1 |

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 |
|---|---|---|
| 0 | 9959 | 1 |
| 1 to 247 | Spare | 247 |

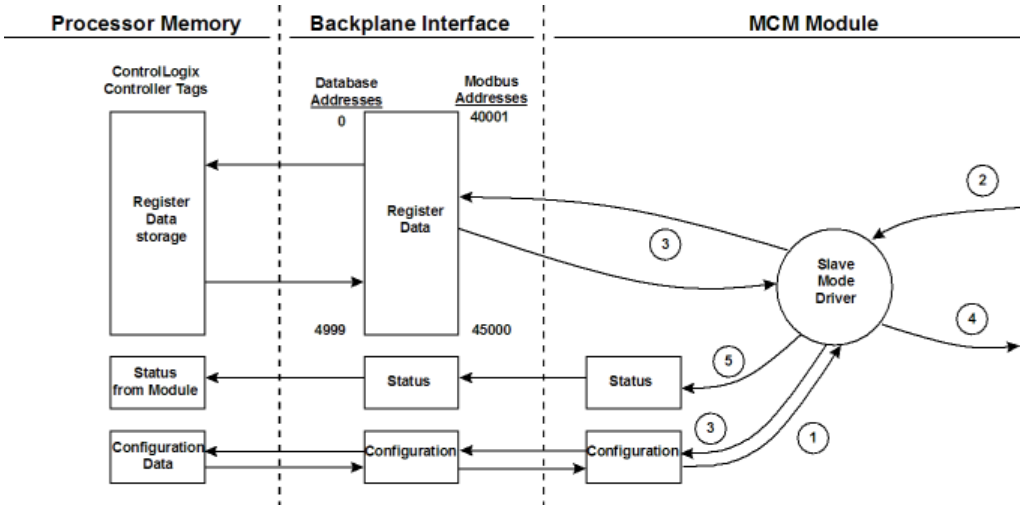
This will inform the module that the command has been processed and can be cleared from the pass-through queue.

### 7.2.12 Data Flow Between MVI56-MCM Module and ControlLogix Processor

The following topics describe the flow of data between the ControlLogix processor,MVI56-MCM module, and nodes on the Modbus network. Each port on the module can be configured to emulate a Modbus Master device or a Modbus Slave device, independently from the configuration of the other port. Only the module database is shared between ports. The sections below discuss the operation of each mode.

*Slave Driver*

The Slave Driver Mode allows the module to respond to data read and write commands issued by a Master on the Modbus network. The following flow chart and table describe the flow of data to and from the module.




**1** The Modbus Slave Port driver receives the configuration information from the ControlLogix processor. This information configures the serial port and defines the Slave node characteristics. Additionally, the configuration information contains data that can be used to offset data in the database to addresses requested in messages received from Master units.

**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.

**3** After the module accepts the command, the data is immediately transferred to or from the internal database in the module. If the command is a read command, the data is read from the database and a response message is built. If the command is a write command, the data is written directly into the database and a response message is built.

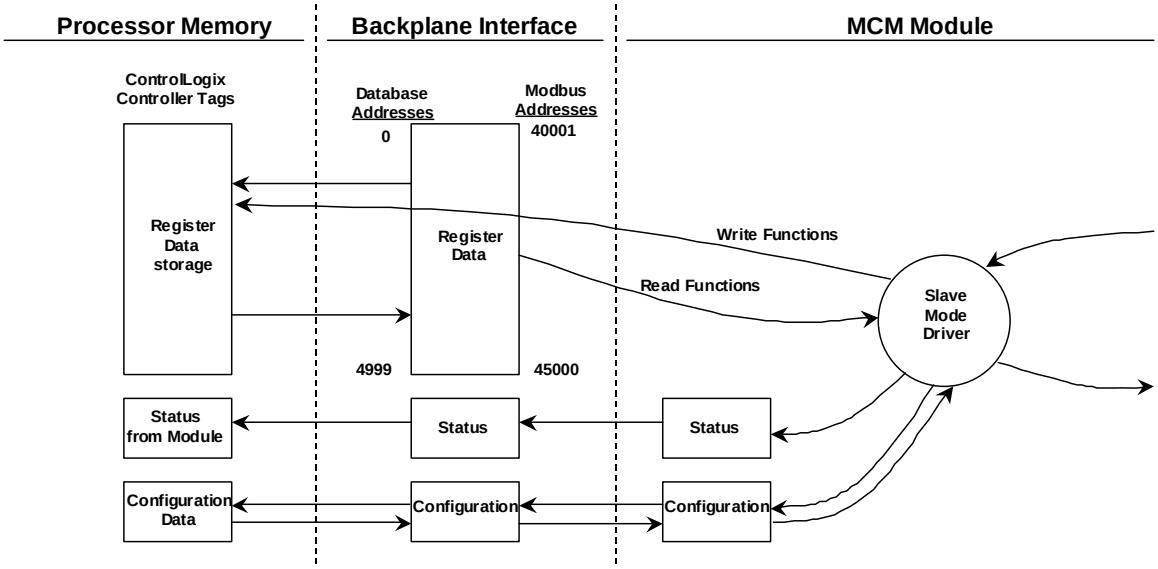
**4** After the data processing has been completed in Step 2, the response is issued to the originating Master node.

**5** Counters are available in the Status Block that permit the ladder logic program to determine the level of activity of the Slave Driver.

Refer to Integrating the Sample Ladder Logic into an Existing Project (page 147) for a complete list of the parameters that must be defined for a Slave Port.

An exception to this normal mode is when the pass-through mode is implemented. In this mode, all write requests will be passed directly to the processor and will not be placed in the database. This permits direct, remote control of the processor without the intermediate database. This mode is especially useful for Master devices that do not send both states of control. For example, a SCADA system may only send an on command to a digital control point and never send the clear state. The SCADA system expects the local logic to reset the control bit. Pass-through must be used to simulate this mode.

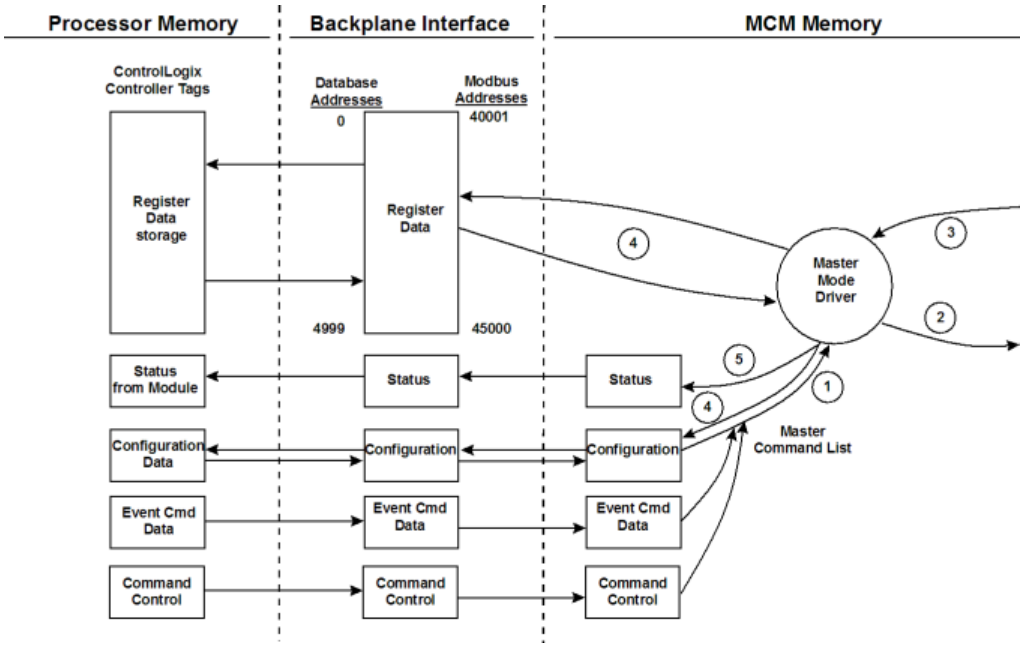
The following illustration shows the data flow for a Slave Port with pass-through enabled:




*Master Driver Mode*

In the Master mode, the MVI56-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 received from the ControlLogix processor or issued directly from the ControlLogix processor (event command control). Command status is returned to the processor for each individual command in the command list status block. The location of this status block in the module's internal database is user defined.

The following illustration describes the flow of data to and from the module.




**1** The Master driver obtains configuration data from the ControlLogix processor. The configuration data obtained includes the number of commands and the Master Command List. These values are used by the Master driver to determine the type of commands to be issued to the other nodes on the Modbus network.

**2** After configuration, the Master driver begins transmitting read and/or write commands to the other nodes on the network. If writing data to another node, the data for the write command is obtained from the module's internal database to build the command.

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

**4** Data received from the node on the network is passed into the module's internal database, assuming a read command.

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

Refer to Integrating the Sample Ladder Logic into an Existing Project (page 147) for a complete description of the parameters required to define the virtual Modbus Master Port.

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

### Master Command List

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

- Command enable mode ((0) disabled, (1) continuous or (2) conditional)
- Slave Node Address
- Command Type - Read or Write up to 125 words (2000 bits) per command
- Database Source and Destination Register Address - Determines where data will be placed and/or obtained
- Count - Select the number of words to be transferred - 1 to 125 on Function Codes 3, 4, or 16. Select the number of bits on Function Codes 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. You can view the current command error list (page 84) from the module's Config/Debug menu.

The following tables describe the error codes generated by the module.

### Standard Modbus Protocol Errors

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

### Module Communication Error Codes

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

### Command List Entry Errors

| Code | Description |
|---|---|
| -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 |

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 **(MCM.PORT1.CMDERRPTR)** parameter to the database location desired.

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

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

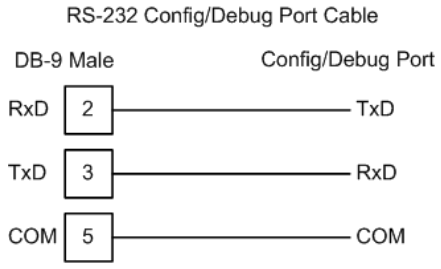
## 7.3 Cable Connections

The application ports on the MVI56-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.

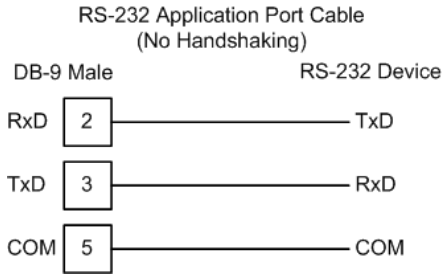
### *7.3.1 RS-232 Configuration/Debug Port*

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:



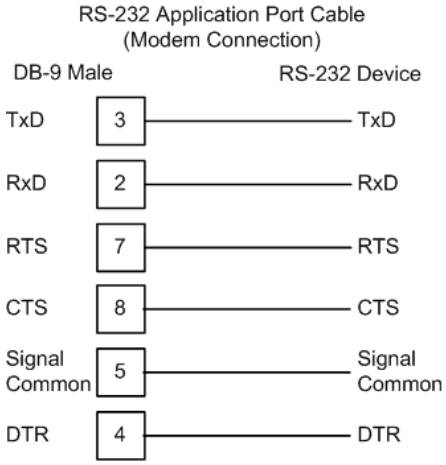
### *7.3.2 RS-232*

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, the cable to connect to the port is as shown below:




*RS-232: Modem Connection*

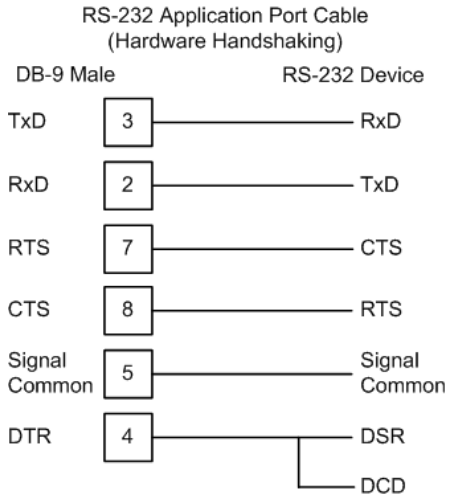
This type of connection is required between the module and a modem or other communication device.




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

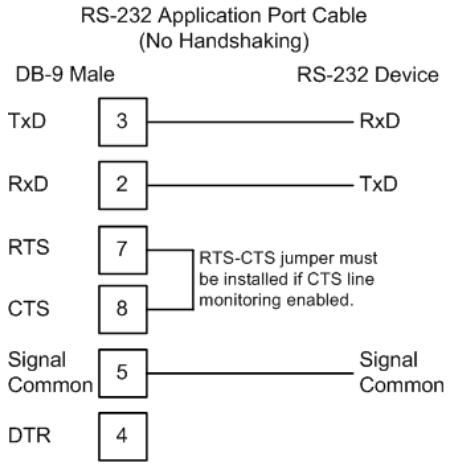
*RS-232: Null Modem Connection (Hardware Handshaking)*

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



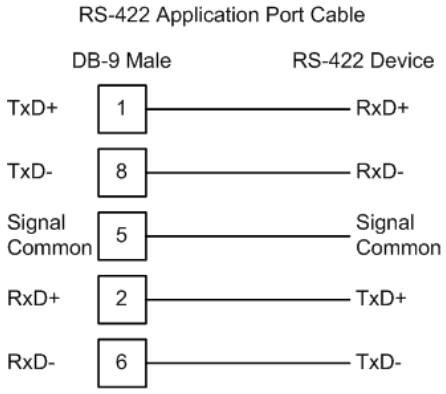

*RS-232: Null Modem Connection (No Hardware Handshaking)*

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



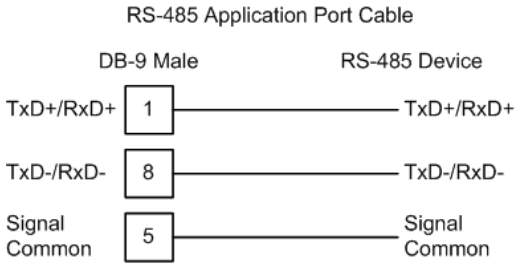

Note: If the port is configured with the "Use CTS Line" set to 'Y', then a jumper is required between the RTS and the CTS line on the module connection.

### 7.3.3 RS-422




### 7.3.4 RS-485

The RS-485 interface requires a single two or three wire cable. The Common connection is optional and dependent on the RS-485 network. The cable required for this interface is shown below:

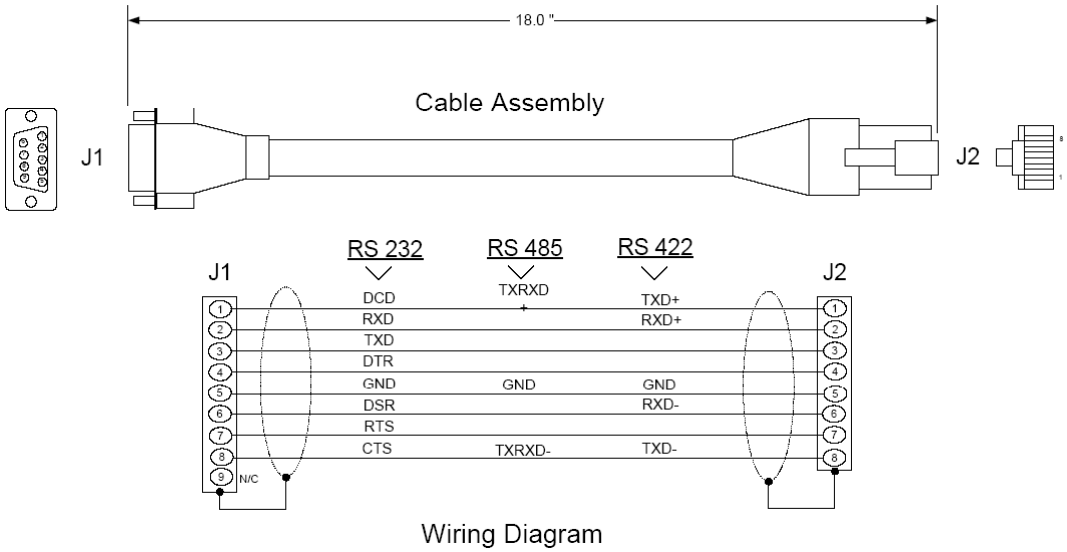



Note: Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In this case, install a 120-ohm terminating resistor on the RS-485 line.

*RS-485 and RS-422 Tip*

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

### *7.3.5 DB9 to RJ45 Adaptor (Cable 14)*




Wiring Diagram

## 7.4 MVI56-MCM Database Definition

This section contains a listing of the internal database of the MVI56-MCM module. This information can be used to interface other devices to the data contained in the module.

| Register Range | Modbus Low | Modbus High | Content | Size |
|---|---|---|---|---|
| 0 to 4999 | 40001 | 45000 | User Data | 5000 |
| 5000 to 5009 | 45001 | 45010 | Backplane Configuration | 10 |
| 5010 to 5039 | 45011 | 45040 | Port 1 Setup | 30 |
| 5040 to 5069 | 45041 | 45070 | Port 2 Setup | 30 |
| 5070 to 5869 | 45071 | 46070 | Port 1 Commands | 800 |
| 5870 to 6669 | 46071 | 47070 | Port 2 Commands | 800 |
| 6750 to 6752 | 46751 | 46753 | Port 1 Aux. Setup | 3 |
| 6760 to 6762 | 46761 | 46763 | Port 2 Aux Setup | 3 |
| 6670 to 6702 | 46671 | 46703 | Misc. Status Data | 33 |
| 6800 | 46801 | | Command Control | 1 |

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).

Detailed definition of the miscellaneous status data area can be found in MVI56-MCM Status Data Definition (page 123).

Definition of the configuration data areas can be found in the data definition section of this document and in MVI56-MCM Configuration Data Definition (page 115).

MVI56-MCM Command Control (page 124) contains a discussion of the command control section of the database.

## 7.5 MVI56-MCM Configuration Data

This section contains listings of the MVI56-MCM module's database related to the module's configuration. This data is available to any node on the network and is read from the ControlLogix processor when the module first initializes.

### *7.5.1 Backplane Setup*

| Register | Content | Description |
|---|---|---|
| 5000 | Write Start Reg | This parameter specifies the starting register in the module where the data transferred from the processor will be placed. Valid range for this parameter is 0 to 4999. |
| 5001 | Write Reg Count | This parameter specifies the number of registers to transfer from the processor to the module. Valid entry for this parameter is 0 to 5000. |
| 5002 | Read Start Reg | This parameter specifies the starting register in the module where data will be transferred from the module to the processor. Valid range for this parameter is 0 to 4999. |
| 5003 | Read Reg Count | This parameter specifies the number of registers to be transferred from the module to the processor. Valid entry for this parameter is 0 to 5000. |
| 5004 | Backplane Fail | This parameter specifies the number of successive transfer errors that must occur before the communication ports are shut down. If the parameter is set to zero, the communication ports will continue to operate under all conditions. If the value is set larger than 0 (1 to 65535), communications will cease if the specified number of failures occur. |
| 5005 | Error Status Pointer | This parameter specifies the register location in the module's database where module status data will be stored. If a value less than zero is entered, the data will not be stored in the database. If the value specified in the range of 0 to 4940, the data will be placed in the user data area. |
| 5006 to 5009 | Spare | |

### *7.5.2 Port 1 Setup*

| Register | Content | Description |
|---|---|---|
| 5010 | Enable | This parameter defines if this Modbus Port will be used. If the parameter is set to 0, the port is disabled. A value of 1 enables the port. |
| 5011 | Type | This parameter specifies if the port will emulate a Modbus Master device (0), a Modbus Slave device without pass-through (1), a Modbus Slave device with unformatted pass-through (2), a Modbus Slave device with formatted pass-through and data swapping (3), or a Modbus Slave device with formatted pass-through and no data swapping (4). |

| Register | Content | Description |
|---|---|---|
| 5012 | Float Flag | This flag specifies if the floating-point data access functionality is to be implemented. If the float flag is set to 1, Modbus functions 3, 6, and 16 will interpret floating-point values for registers as specified by the two following parameters. |
| 5013 | Float Start | 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. |
| 5014 | Float Offset | This parameter defines the start register for floating-point data in the internal database. This parameter is only used if the Float Flag is enabled. |
| 5015 | Protocol | This parameter specifies the Modbus protocol to be used on the port. Valid protocols are: 0 = Modbus RTU and 1 = Modbus ASCII. |
| 5016 | 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. Valid entries are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 384 (for 38400 bps), 576 (for 57600 bps), and 115 (for 115,200 bps). |
| 5017 | Parity | This is the parity code to be used for the port. Values are None, Odd, Even. |
| 5018 | Data Bits | This parameter sets the number of data bits for each word used by the protocol. Valid entries for this field are 5 through 8. |
| 5019 | Stop Bits | This parameter sets the number of stop bits to be used with each data value sent. Valid entries are 1 and 2. |
| 5020 | RTS On | This parameter sets the number of milliseconds to delay after RTS is asserted before the data will be transmitted. Valid values are in the range of 0 to 65535 milliseconds. |
| 5021 | RTS Off | 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. Valid values are in the range of 0 to 65535. |
| 5022 | Minimum Response Time | This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices. |
| 5023 | Use CTS Line | This parameter specifies if the CTS modem control line is to be used. If the parameter is set to 0, the CTS line will not be monitored. If the parameter is set to 1, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire). |

| Register | Content | Description |
|---|---|---|
| 5024 | Slave ID | This parameter defines the virtual Modbus Slave 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 (247 on some networks). |
| 5025 | Bit in Offset | This parameter specifies the offset address in the internal Modbus database to use with 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. |
| 5026 | Word in Offset | This parameter specifies the offset address in the internal Modbus database to use with network request 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. |
| 5027 | Out in Offset | This parameter specifies the offset address in the internal Modbus database to use with 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. |
| 5028 | Holding Reg Offset | This parameter specifies the offset address in the internal Modbus database to use with 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. |
| 5029 | Command Count | This parameter specifies the number of commands to be processed by the Modbus Master Port. |
| 5030 | Minimum Command Delay | This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries. |
| 5031 | Command Error Pointer | 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. |
| 5032 | Response Timeout | This parameter represents the message response timeout period in 1-millisecond increments. This is the time that a port configured as a Master will wait before re-transmitting a command if no response is received from the addressed Slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network. |
| 5033 | Retry Count | This parameter specifies the number of times a command will be retried if it fails. If the Master Port does not receive a response after the last retry, the Slave devices communication will be suspended on the port for Error Delay Counter scans. |

| Register | Content | Description |
|---|---|---|
| 5034 | Error Delay Counter | 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. |
| 5035 to 5039 | Spare | Reserved for future use. |

### *7.5.3 Port 2 Setup*

| Register | Content | Description |
|---|---|---|
| 5040 | Enable | This parameter defines if this Modbus Port will be used. If the parameter is set to 0, the port is disabled. A value of 1 enables the port. |
| 5041 | Type | This parameter specifies if the port will emulate a Modbus Master device (0), a Modbus Slave device without pass-through (1), a Modbus Slave device with unformatted pass-through (2), a Modbus Slave device with formatted pass-through and data swapping (3), or a Modbus Slave device with formatted pass-through and no data swapping (4). |
| 5042 | Float Flag | This flag specifies if the floating-point data access functionality is to be implemented. If the float flag is set to 1, Modbus functions 3, 6, and 16 will interpret floating-point values for registers as specified by the two following parameters. |
| 5043 | Float Start | 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. |
| 5044 | Float Offset | This parameter defines the start register for floating-point data in the internal database. This parameter is only used if the Float Flag is enabled. |
| 5045 | Protocol | This parameter specifies the Modbus protocol to be used on the port. Valid protocols are: 0 = Modbus RTU and 1 = Modbus ASCII. |
| 5046 | 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. Valid entries are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 384 (for 38400 bps), 576 (for 57600 bps), and 115 (for 115,200 bps). |
| 5047 | Parity | This is the parity code to be used for the port. Values are None, Odd, Even. |
| 5048 | Data Bits | This parameter sets the number of data bits for each word used by the protocol. Valid entries for this field are 5 through 8. |
| 5049 | Stop Bits | This parameter sets the number of stop bits to be used with each data value sent. Valid entries are 1 and 2. |

| Register | Content | Description |
|---|---|---|
| 5050 | RTS On | This parameter sets the number of milliseconds to delay after RTS is asserted before the data will be transmitted. Valid values are in the range of 0 to 65535 milliseconds. |
| 5051 | RTS Off | 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. Valid values are in the range of 0 to 65535. |
| 5052 | Minimum Response Time | This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices. |
| 5053 | Use CTS Line | This parameter specifies if the CTS modem control line is to be used. If the parameter is set to 0, the CTS line will not be monitored. If the parameter is set to 1, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire). |
| 5054 | Slave ID | This parameter defines the virtual Modbus Slave 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 (247 on some networks). |
| 5055 | Bit in Offset | This parameter specifies the offset address in the internal Modbus database to use with 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. |
| 5056 | Word in Offset | This parameter specifies the offset address in the internal Modbus database to use with network request 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. |
| 5057 | Out in Offset | This parameter specifies the offset address in the internal Modbus database to use with 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. |
| 5058 | Holding Reg Offset | This parameter specifies the offset address in the internal Modbus database to use with 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. |
| 5059 | Command Count | This parameter specifies the number of commands to be processed by the Modbus Master Port. |

| Register | Content | Description |
|---|---|---|
| 5060 | Minimum Command Delay | This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries. |
| 5061 | Command Error Pointer | 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. |
| 5062 | Response Timeout | This parameter represents the message response timeout period in 1-millisecond increments. This is the time that a port configured as a Master will wait before re-transmitting a command if no response is received from the addressed Slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network. |
| 5063 | Retry Count | This parameter specifies the number of times a command will be retried if it fails. If the Master Port does not receive a response after the last retry, the Slave devices communication will be suspended on the port for Error Delay Counter scans. |
| 5064 | Error Delay Counter | 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. |
| 5065 to 5069 | Spare | |

### 7.5.4 Port 1 Commands

| Register | Content | Description |
|---|---|---|
| 5070 to 5777 | Command #1 | This set of registers contains the parameters for the first command in the Master command list. Refer to Master Command Configuration (page 38). |
| 5078 to 5085 | Command #2 | Command #2 data set |
| - | - | - |
| 5852 to 5859 | Command #100 | Command #100 data set |

### 7.5.5 Port 2 Commands

| Register | Content | Description |
|---|---|---|
| 5870 to 5877 | Command #1 | This set of registers contains the parameters for the first command in the Master command list. Refer to Master Command Configuration (page 38). |
| 5878 to 5885 | Command #2 | Command #2 data set |
| - | - | - |
| 6662 to 6669 | Command #100 | Command #100 data set |

### 7.5.6 Misc. Status

| Register | 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 from 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 from 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 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 module to the processor. |
| 6695 | Parse Block Count | This field contains the total number of blocks successfully parsed that were received from the processor. |

| Register | Content | Description |
|---|---|---|
| 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 the 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. |
| 6703 to 6750 | Spare | |
| 6751 | Port 1 InterCharacterDelay | 0 to 65535 milliseconds time between characters to signal end of message |
| 6752 | Port 1 Fcn 99 Offset | Internal DB offset to Function 99 counter. |
| 6753 to 6760 | Spare | |
| 6760 | Spare | |
| 6761 | Port 2 InterCharacterDelay | 0 to 65535 milliseconds time between characters to signal end of message |
| 6762 | Port 2 Fcn 99 Offset | Internal DB offset to Function 99 counter. |
| 6763 to 6799 | Spare | |

### *7.5.7 Command Control*

| Register | Content | Description |
|---|---|---|
| 6800 | Command Code | Enter one of the valid control command codes in this register to control the module (9997, 9998, or 9999). |

## 7.6 MVI56-MCM Status Data Definition

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

| Offset | Content | Description |
|---|---|---|
| 202 | Program Scan Count | This value is incremented each time a complete program cycle occurs in the module. |
| 203 to 204 | Product Code | These two registers contain the product code of "MCM". |
| 205 to 206 | Product Version | These two registers contain the product version for the current running software. |
| 207 to 208 | Operating System | These two registers contain the month and year values for the program operating system. |
| 209 to 210 | Run Number | These two registers contain the run number value for the currently running software. |
| 211 | Port 1 Command List Requests | This field contains the number of requests made from this port to Slave devices on the network. |
| 212 | Port 1 Command List Response | This field contains the number of Slave response messages received on the port. |
| 213 | 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. |
| 214 | Port 1 Requests | This field contains the total number of messages sent from the port. |
| 215 | Port 1 Responses | This field contains the total number of messages received on the port. |
| 216 | Port 1 Errors Sent | This field contains the total number of message errors sent from the port. |
| 217 | Port 1 Errors Received | This field contains the total number of message errors received on the port. |
| 218 | Port 2 Command List Requests | This field contains the number of requests made from this port to Slave devices on the network. |
| 219 | Port 2 Command List Response | This field contains the number of Slave response messages received on the port. |
| 220 | 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. |
| 221 | Port 2 Requests | This field contains the total number of messages sent out the port. |
| 222 | Port 2 Responses | This field contains the total number of messages received on the port. |
| 223 | Port 2 Errors Sent | This field contains the total number of message errors sent out the port. |
| 224 | Port 2 Errors Received | This field contains the total number of message errors received on the port. |
| 225 | Read Block Count | This field contains the total number of read blocks transferred from the module to the processor. |
| 226 | Write Block Count | This field contains the total number of write blocks transferred from the module to the processor. |

| Offset | Content | Description |
|---|---|---|
| 227 | Parse Block Count | This field contains the total number of blocks successfully parsed that were received from the processor. |
| 228 | Command Event Block Count | This field contains the total number of command event blocks received from the processor. |
| 229 | Command Block Count | This field contains the total number of command blocks received from the processor. |
| 230 | Error Block Count | This field contains the total number of block errors recognized by the module. |
| 231 | 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. |
| 232 | Port 1 Last Error | For a Slave Port, this field contains the value of the last error code returned. For a Master Port, this field contains the index of the command with the error. |
| 233 | 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. |
| 234 | 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. |

## 7.7 MVI56-MCM Command Control

Command Control data is received from other nodes on the network that can control the MVI56-MCM module. Specific values are written to regions of this block to control the module. Currently, the module is programmed to handle the receipt of the following requests: write configuration to processor, warm boot and cold boot.

The remote node controls the module by writing one of the following values to register 7800 (Modbus address 47801):

| | |
|---|---|
| 9997 | Write configuration in database to the processor and warm boot the module. |
| 9998 | Warm boot the module. |
| 9999 | Cold boot the module. |

The control register is cleared (a value of 0) after the operation is executed with the exception of the 9997 command. If the module fails to successfully transfer the configuration to the processor, an error code will be returned in the control register as follows:

| | |
|---|---|
| 0 | No error, transfer successful |
| -1 | Error transferring general configuration information. |
| -2 | Error transferring Modbus Port 1 Master command list |
| -3 | Error transferring Modbus Port 2 Master command list |

Ladder logic must handle the 9997 command. No ladder logic is required when using the warm or cold boot commands.

## 7.8 MVI56-MCM User Defined Data Types

### *7.8.1 MCMModuleDef*

This object contains the data types that apply to the operation of the module.

| Tag Name | Data Type | Description |
|---|---|---|
| CONFIG | MCMCONFIG (page 125) | Module and port configuration |
| DATA | MCMDATA (page 127) | Modbus data transferred between module and processor |
| STATUS | MCMSTATUS (page 127) | Status information in each read block |
| CONTROL | MCMCONTROL (page 128) | Optional requests from the processor to the module |
| _UTIL | _Util (page 128) | Variables for internal ladder usage - should not be accessed by user application |

### *7.8.2 MCMCONFIG*

This object contains the data types that apply to the configuration of the module. Refer to MVI56-MCM Configuration Data Definition (page 115) for a complete description of each element in this object.

| Tag Name | Data Type | Description |
|---|---|---|
| ModDef | MCMModule (page 125) | Module Definition |
| Port1 | MCMPort (page 126) | Port 1 configuration settings |
| Port2 | MCMPort (page 126) | Port 2 configuration settings |
| Port1MasterCmd | MCMCmd[100] (page 126) | Master commands for Port 1 (ignore if port is configured for Slave mode) |
| Port2MasterCmd | MCMCmd[100] (page 126) | Master commands for Port 2 (ignore if port is configured for Slave mode) |

*MCMModule*

This object contains the information used to define the data movement between the module and the processor.

| Tag Name | Data Type | Description |
|---|---|---|
| WriteStartReg | INT | Start reg to transfer from PLC to module |
| WriteRegCnt | INT | Number of registers to write from PLC |
| ReadStartReg | INT | Start reg to transfer from module to PLC |
| ReadRegCnt | INT | Number of registers to transfer from module |
| BPFail | INT | Determines module operation if BP fails 0=continue,>0=number of retries before comm shutdown |
| ErrStatPtr | INT | Internal DB start register for status data (-1=Ignore) |

*MCMPort*

This object contains the serial port configuration for the MVI56-MCM module.

| Tag Name | Data Type | Description |
|---|---|---|
| Enabled | INT | 0=Port Disabled, 1=Port Enabled |
| Type | INT | 0=Master, 1=Slave, 2=Slave: pass-through, 3=Slave: formatted pass-through/data swapped, 4=Slave: form. pass-through |
| FloatFlag | INT | 0=No floating-point data, 1=Use floating-point data |
| FloatStart | INT | Register offset in message for floating-point data |
| FloatOffset | INT | Internal DB offset to start of floating-point data |
| Protocol | INT | 0=Modbus RTU, 1=Modbus ASCII |
| Baudrate | INT | Baudrate for port (110 to 115.2K) |
| Parity | INT | 0=None, 1=Odd, 2=Even, 3=Mark, 4=Space |
| DataBits | INT | 5 to 8 data bits |
| StopBits | INT | 1 or 2 stop bits |
| RTSOn | INT | 0-65535 milliseconds delay before data |
| RTSOff | INT | 0-65535 milliseconds delay after data |
| MinResp | INT | 0-65535 milliseconds minimum time before response to request |
| UseCTS | INT | 0=No, 1=Yes to use CTS modem line |
| SlaveID | INT | 1-255 Modbus Node Address (Slave) |
| BitInOffset | INT | Internal DB offset to bit input data (Slave) |
| WordInOffset | INT | Internal DB offset to word input data (Slave) |
| OutOffset | INT | Internal DB offset to bit output data (Slave) |
| HoldOffset | INT | Internal DB offset to holding register data (Slave) |
| CmdCount | INT | Command list count (Master) |
| MinCmdDelay | INT | 0-65535 milliseconds minimum time between each command (Master) |
| CmdErrPtr | INT | Internal DB location to place command error list (Master) |
| RespTO | INT | 0-65535 milliseconds response timeout for command (Master) |
| RetryCount | INT | Retry count for failed request (Master) |
| ErrorDelayCntr | INT | 0-65535 Command cycle count if error (Master) |
| Reserved | INT | Reserved (Previously was UseGuardBand parameter) |
| InterCharacterDelay | INT | 0-65535 milliseconds time between characters to signal end of message |
| Fcn99Offset | INT | Internal DB offset to function 99 counter |

*MCMCmd*

This object contains the attributes to define a Master command. An array of these objects is used for each port.

| Tag Name | Data Type | Description |
|---|---|---|
| Enable | INT | 0=Disable, 1=Continuous, 2=Event Command |
| IntAddress | INT | Module's internal address associated with the command |
| PollInt | INT | Minimum number of seconds between issuance of command (0-65535 Sec) |
| Count | INT | Number of registers associated with the command |

| Tag Name | Data Type | Description |
|---|---|---|
| Swap | INT | Swap code used with command |
| Node | INT | Node address of the target device on the network |
| Func | INT | Function code for the command |
| DevAddress | INT | Address in device associated with the command |

### 7.8.3 MCMDATA

Contains Read Data (data read from the module to the processor) and Write Data (data written from the processor to the module).

| Tag Name | Data Type | Description |
|---|---|---|
| ReadData | INT[600] | Data read from the module to the processor |
| WriteData | INT[600] | Data written from the processor to the module |

### 7.8.4 MCMSTATUS

This status data is returned on each read block and can be used to detect proper module operation.

| Tag Name | Data Type | Description |
|---|---|---|
| PassCnt | INT | Program cycle counter |
| Product | INT[2] | Product Name |
| Rev | INT[2] | Revision Level Number |
| OP | INT[2] | Operating Level Number |
| Run | INT[2] | Run Number |
| Prt1Errs | MCMPortErrors (page 127) | Port 1 error statistics |
| Prt2Errs | MCMPortErrors (page 127) | Port 2 error statistics |
| Blk | MCMBlkStat (page 128) | Block transfer statistics |
| Port1LastErr | INT | Last command index that received an error on Port 1 |
| Port1PreviousErr | INT | Previous Command index that received an error on Port 1 |
| Port2LastErr | INT | Last command index that received an error on Port 2 |
| Port2PreviousErr | INT | Previous Command index that received an error on Port 2 |

*MCMPortErrors*

This object stores the port statistics for an MVI56-MCM port.

| Tag Name | Data Type | Description |
|---|---|---|
| CmdReq | INT | Total number of command list requests sent |
| CmdResp | INT | Total number of command list responses received |
| CmdErr | INT | Total number of command list errors |
| Requests | INT | Total number of requests for port |

| Tag Name | Data Type | Description |
|---|---|---|
| Responses | INT | Total number of responses for port |
| ErrSent | INT | Total number of errors sent |
| ErrRec | INT | Total number of errors received |

*MCMBlkStat*

This object stores the block transfer statistics for the module.

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

### 7.8.5 MCMCONTROL

This object contains the attributes to define a Master command. An array of these objects is used for each port.

| Tag Name | Data Type | Description |
|---|---|---|
| WarmBoot | BOOL | Conversion from Bool to INT data types |
| ColdBoot | BOOL | Conversion from Bool to INT data types |
| BPLastRead | INT | Index of last read block |
| BPLastWrite | INT | Index of last write block |
| BlockIndex | INT | Computed block offset for data table |
| ReadDataSize | DINT | Size of Read Data Array |
| MaxReadBlock | DINT | Maximum Read Block |
| WriteDataSize | DINT | Size of Write Data Array |
| MaxWriteBlock | DINT | Maximum Write Block |

### 7.8.6 _Util

This object contains optional elements for the module.

| Tag Name | Data Type | Description |
|---|---|---|
| CmdControl | _CmdControl (page 129) | Allows for a disabled command to be sent to a device (MASTER) |
| EventCmd | _EventCmd (page 129) | Allows a command defined in ladder to be sent to a device (MASTER) |
| SlavePollStat | _SlavePollStatus (page 129) | Request Slave poll status for the port (MASTER). |
| Passthru | _Passthru (page 130) | Contains PassThru objects required when PortX.Type is set to a value between 2 and 4. |

### _CmdControl

| Tag Name | Data Type | Description |
|---|---|---|
| TriggerCmdCntrl | BOOL | Trigger command control. User application will activate this trigger |
| NumberOfCommands | INT | Number of commands per block (1 to 6) |
| PortNumber | INT | MVI56-MCM Port Number of Master Port (1 or 2) |
| CommandIndex | INT[6] | Stores the command indexes for command control |
| CmdsAddedToQueue | INT | Number of commands added to queue |
| CmdControlBlockID | INT | Temporary variable to calculate control block ID |
| CmdCntrlPending | BOOL | Auxiliary control command: prevents a second request before acknowledgment is received |

### _EventCmd

| Tag Name | Data Type | Description |
|---|---|---|
| EventCmdTrigger | BOOL | Trigger for event command. User ladder must set this bit to initiate event command |
| EventCmdPending | BOOL | Set after the ladder has sent an event cmd to the module and is waiting for the status to be returned |
| PortNumber | INT | Module Master Port number associated to this request (1 or 2) |
| SlaveAddress | INT | Slave target Modbus address |
| InternalDBAddress | INT | Internal database address |
| PointCount | INT | Number of points for this command |
| SwapCode | INT | Swap code (0= no swap, 1=swap words, 2=swap words and bytes, 3=swap bytes) |
| ModbusFunctionCode | INT | Modbus function code |
| DeviceDBAddress | INT | Modbus register address in target Modbus Slave |
| EventCmdStatusReturned | INT | (0=Fail, 1=Success) |
| EventBlockID | INT | Temporary variable to calculate event block ID |

### _SlavePollStatus

This object contains all of the Slave Polling status (used when the port is used as a Master).

| Tag Name | Data Type | Description |
|---|---|---|
| Port1Slave0Read | BOOL | |
| Port1Slave128Read | BOOL | |
| Port2Slave0Read | BOOL | |
| Port2Slave128Read | BOOL | |
| P1Slaves | INT[256] | P1 Slave Status |
| P2Slaves | INT[256] | P2 Slave Status |

*_Passthru*

| Tag Name | Data Type | Description |
|---|---|---|
| MBOffset | INT | |
| MBOffsetBit | INT | |
| MBMsgLen | INT | |
| MBMsg | SINT[500] | |
| MBControl1 | CONTROL (page 128) | |
| MBControl2 | CONTROL (page 128) | |
| MBScratch | INT[3] | |
| MBCoil | _CoilArray (page 130) | Conversion from Bool to INT data types |

*_CoilArray*

Conversion to INT values

| Tag Name | Data Type | Description |
|---|---|---|
| Boolean | BOOL[416] | Conversion from Bool to INT data types |

## 7.9 Modbus Protocol Specification

### *7.9.1 Commands Supported by the Module*

The format of each command in the list is dependent 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.

### 7.9.2 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.

### 7.9.3 Read Input Status (Function Code 02)

Query

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

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

The following table is a sample read input status request to read inputs 10197 to 10218 from Slave number 11.

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

Response

An example response to Read Input Status is as shown in Figure C4. The data is packed one bit for each input. The response includes the Slave address, function code, quantity of data characters, the data characters, and error checking. Data will be packed with one bit for each input (1=ON, 0=OFF). The lower order bit of the first character contains the addressed input, and the remainder follow. For input quantities that are not even multiples of eight, the last characters will be filled in with zeros at high order end. The quantity of data characters is always specified as a quantity of RTU characters, that is, the number is the same whether RTU or ASCII is used.

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

| Adr | Func | Byte Count | Data Discrete Input 10197 to 10204 | Data Discrete Input 10205 to 10212 | Data Discrete Input 10213 to 10218 | Error Check Field |
|---|---|---|---|---|---|---|
| 11 | 02 | 03 | AC | DB | 35 | CRC |

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

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

### 7.9.4 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 |

### 7.9.5 Read Input Registers (Function Code 04)

Query

Function code 04 obtains the contents of the controller's input registers at addresses 3xxxx. These locations receive their values from devices connected to the I/O structure and can only be referenced, not altered from within the controller, The addressing allows up to 125 registers to be obtained at each request; however, the specific Slave device may have restrictions that lower this maximum quantity. The registers are numbered for zero (30001 = zero, 30002 = one, and so on). Broadcast mode is not allowed.

The example below requests the contents of register 3009 in Slave number 11.

| Adr | Func | Data Start Reg Hi | Data Start Reg Lo | Data #of Regs Hi | Data #of Regs Lo | Error Check Field |
|---|---|---|---|---|---|---|
| 11 | 04 | 00 | 08 | 00 | 01 | CRC |

Response

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

Because the Slave interface is normally serviced at the end of the controller's scan, the data will reflect the register content at the end of the scan. Each PC will limit the quantity of register contents provided each scan; thus for large register quantities, multiple PC scans will be required, and the data provided will be form sequential scans.

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

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

### *7.9.6 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".

### 7.9.7 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 |

### 7.9.8 Diagnostics (Function Code 08)

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

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

In general, issuing a diagnostic function to a remote device does not affect the running of the user program in the remote device. User logic, like discrete and registers, is not accessed by the diagnostics. Certain functions can optionally reset error counters in the remote device.

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.

The following diagnostic functions are dedicated to serial line devices.

The normal response to the Return Query Data request is to loopback the same data. The function code and sub-function codes are also echoed.

Request

| | | |
|---|---|---|
| Function code | 1 Byte | 0x08 |
| Sub-function | 2 Bytes | |
| Data | N x 2 Bytes | |

Response

| | | |
|---|---|---|
| Function code | 1 Byte | 0x08 |
| Sub-function | 2 Bytes | |
| Data | N x 2 Bytes | |

Error

| | | |
|---|---|---|
| Error code | 1 Byte | 0x88 |
| Exception code | 1 Byte | 01 or 03 or 04 |

*Sub-function codes supported by the serial line devices*

Only Sub-functions 00 is supported by the MVI56-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.

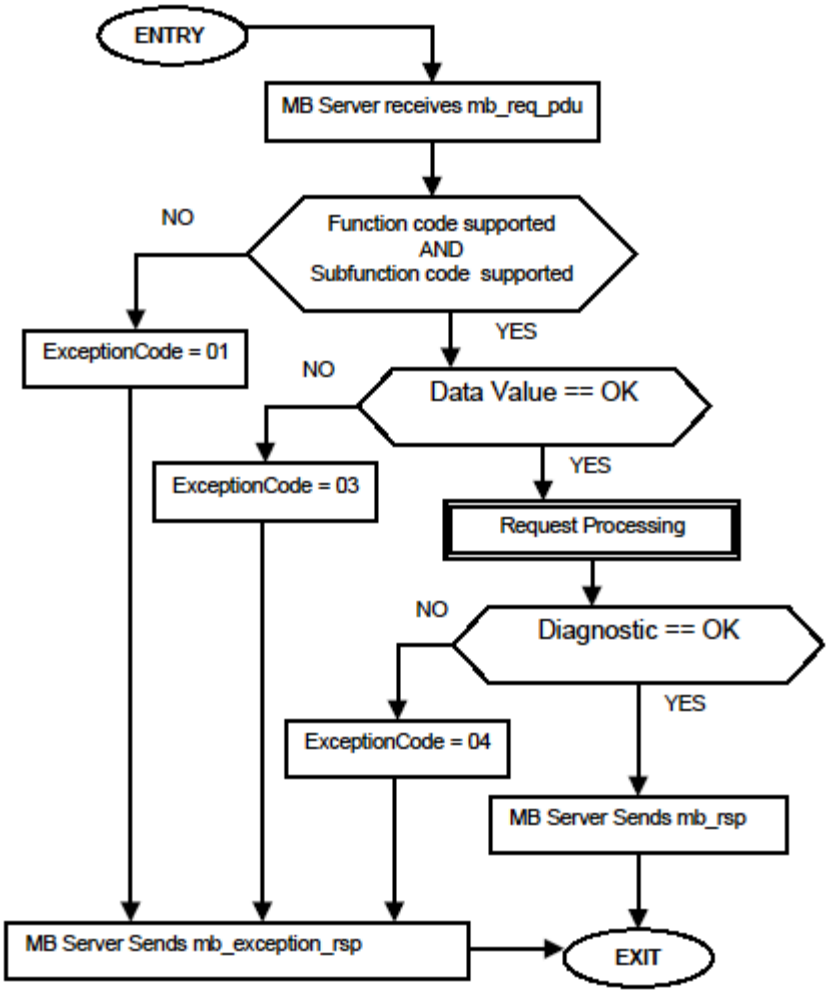
| Sub-function | Data Field (Request) | Data Field (Response) |
|---|---|---|
| 00 00 | Any | Echo Request Data |

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.




### *7.9.9 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.

### *7.9.10 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. When specified registers with contents specified.

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 |

### 7.9.11 MODBUS Exception Responses

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

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

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

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

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

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

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

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

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

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

*MODBUS Exception Codes*

| Code | Name | Meaning |
|---|---|---|
| 01 | Illegal Function | The function code received in the query is not an allowable action for the server (or 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 server (or 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 server (or 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 server (or 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 server (or Slave) was attempting to perform the requested action. |
| 05 | Acknowledge | Specialized use in conjunction with programming commands. The server (or 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 client (or Master). The client (or Master) can next issue a poll program complete message to determine if processing is completed. |
| 06 | Slave Device Busy | Specialized use in conjunction with programming commands. The server (or Slave) is engaged in processing a long-duration program command. The client (or Master) should retransmit the message later when the server (or 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 server (or Slave) attempted to read record file, but detected a parity error in the memory. The client (or Master) can retry the request, but service may be required on the server (or 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. |

## 7.10 Using the Sample Program - RSLogix Version 15 and earlier

The sample program included with your MVI56-MCM Module contains predefined controller tags, configuration information, data types, and ladder logic that allow the module to communicate between the ControlLogix processor and a network of Modbus devices. For most applications, the sample program will work without modification.

### *7.10.1 Using the Sample Program in a New Application*

*Open the Sample Program in RSLogix*

The sample program for your MVI56-MCM module includes custom tags, data types and ladder logic for data I/O, status and command control. For most applications, you can run the sample program without modification, or, for advanced applications, you can incorporate the sample program into your existing application.

The ProSoft Solutions CD provides one or more versions of the sample ladder logic. The version number appended to the file name corresponds with the firmware version number of your ControlLogix processor. The firmware version and sample program version must match.

The ProSoft Solutions CD-ROM included in the package with the module contains ladder logic, product manuals and utility programs for all ProSoft Technology products.

*Copy the manuals and sample program from the CD-ROM*

1. Insert the ProSoft Solutions CD-ROM into the CD drive of your PC. Wait for the startup screen to appear.
2. On the startup screen, click *Product Documentation*. This action opens an Explorer window. Files are arranged by type:
   - The Ladder Logic folder contains sample programs for each module, arranged by processor type, and then by product name. The sample programs for your module are in the ControlLogix/MVI56/MVI56-MCM folder.
   - The Manuals folder contains product manuals and datasheets in Adobe Acrobat Reader format (PDF) for each module, arranged in the same way as the Ladder Logic folder.
   - The Utilities folder contains additional programs and tools required for some ProSoft modules. Refer to your user manual to determine if you need to use or install any of these additional tools.
3. In the Explorer window, navigate to the files you need, and then copy them to a location on your hard drive.

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

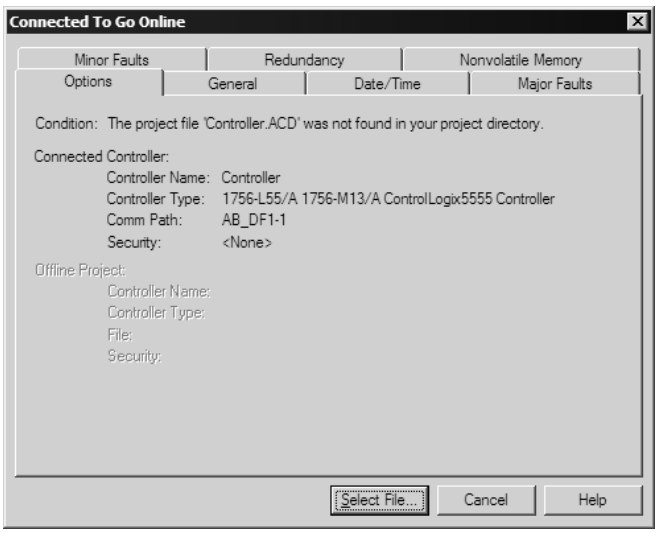
You can always download the latest version of the sample ladder logic and user manuals for the MVI56-MCM module from the ProSoft Technology web site, at http://www.prosoft-technology.com/support/downloads (http://www.prosoft-technology.com/support/downloads)

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

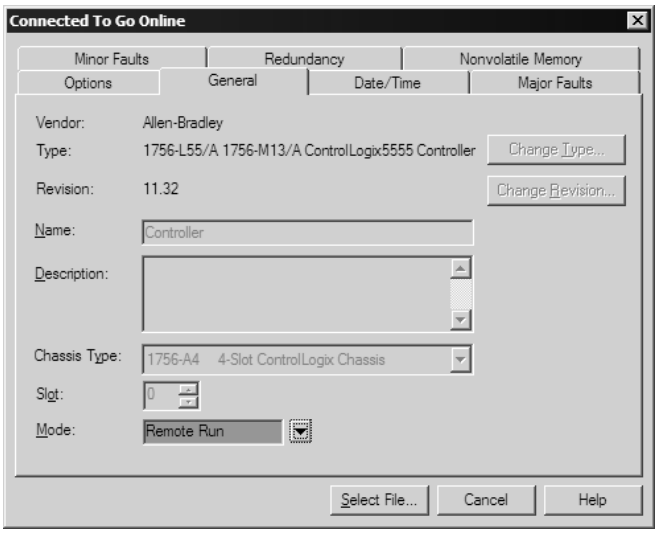
*To determine the firmware version of your processor*

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

1. Connect an RS-232 serial cable from the COM (serial) port on your PC to the communication port on the front of the processor.
2. Start RSLogix 5000 and close any existing project that may be loaded.
3. Open the Communications menu and choose **Go Online**. RSLogix will establish communication with the processor. This may take a few moments.
4. When RSLogix has established communication with the processor, the Connected To Go Online dialog box will open.



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




**6** Select the sample ladder logic file for your firmware version:

*To open the sample program*

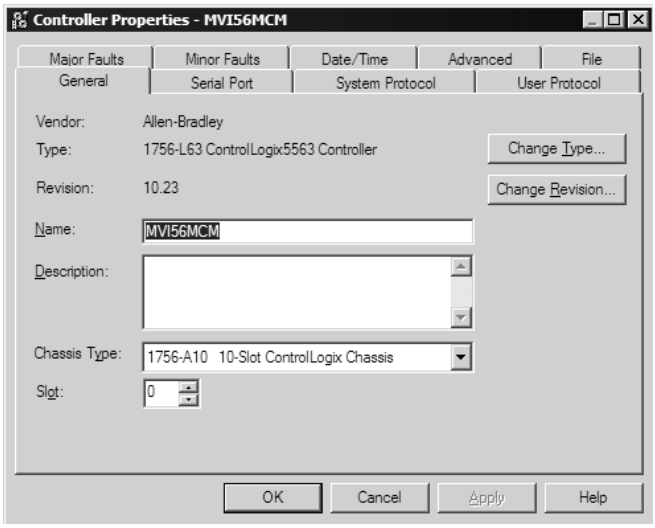
**1** On the Connected to Go Online dialog box, click the Select File button.
**2** Choose the sample program file that matches your firmware version, and then click the Select button.
**3** RSLogix will load the sample program.

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

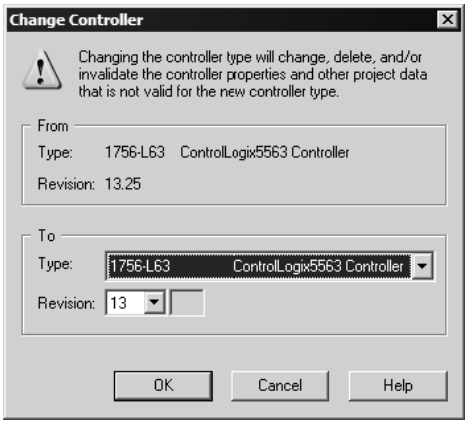
*Choose the Controller Type*

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

1. In the Controller Organization list, select the folder **CONTROLLER MVI56MCM**, and then click the right mouse button to open a shortcut menu.
2. On the shortcut menu, choose **PROPERTIES.** This action opens the **CONTROLLER PROPERTIES** dialog box.




3. Click the **CHANGE CONTROLLER** button. This action opens the **CHANGE CONTROLLER** dialog box.



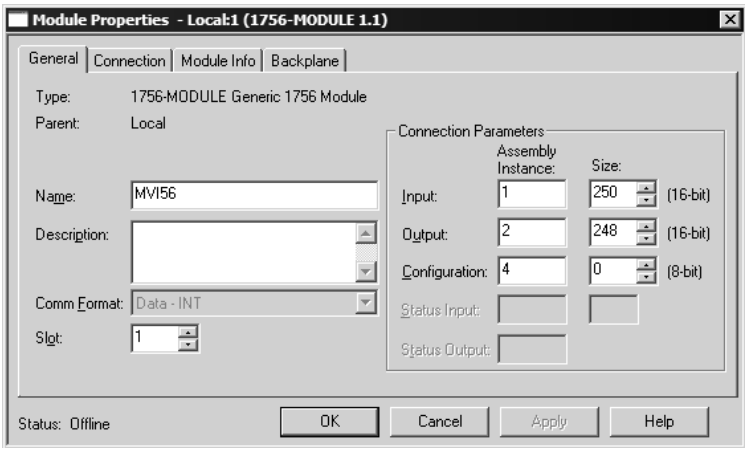

4. Open the Type: dropdown list, and then select your ControlLogix controller.
5. Select the correct firmware revision for your controller, if necessary.
6. Click **OK** to save your changes and return to the previous window.

*Select the Slot Number for the Module*

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

*To change the slot number*

1. In the **CONTROLLER ORGANIZATION** list, select the module **[1] 1756-MODULE MVI56,** and then click the right mouse button to open a shortcut menu.
2. On the shortcut menu, choose **PROPERTIES.** This action opens the **MODULE PROPERTIES** dialog box.




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

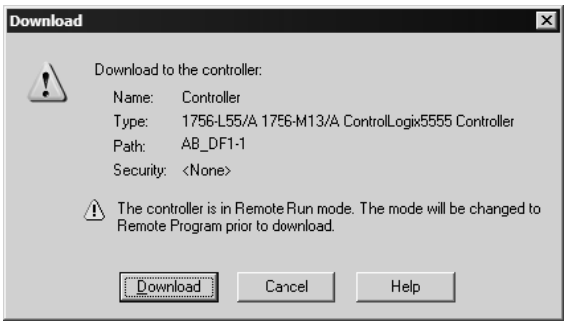
RSLogix will automatically apply the slot number change to all tags, variables and ladder logic rungs that use the MVI56-MCM slot number for computation.

*Download the Sample Program to the Processor*

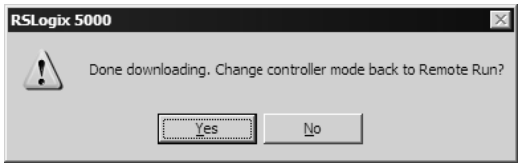
*To download the sample program from RSLogix 5000 to the ControlLogix processor:*

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

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




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




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

### *7.10.2 Using the Sample Program in an Existing Application*

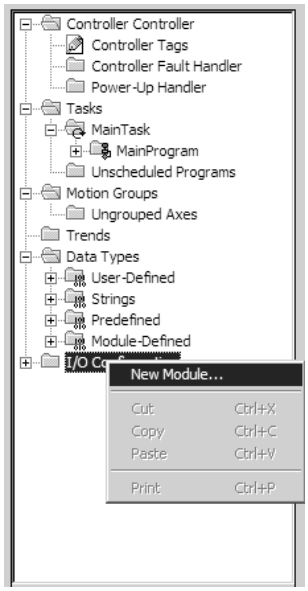
1. Open the Sample Ladder Logic in RSLogix 5000.
2. Start another instance of RSLogix 5000, and then open your existing application.

You will be adding the MVI56-MCM module definition, and then copying controller tags, ladder logic, and user defined data types from the sample application into your existing application.

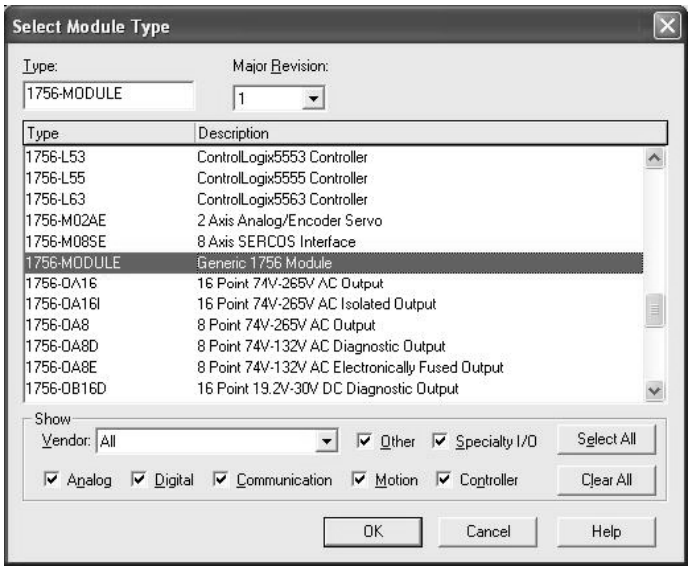
*Define Module in I/O Configuration*

Note: You cannot perform this procedure while you are online to the controller.

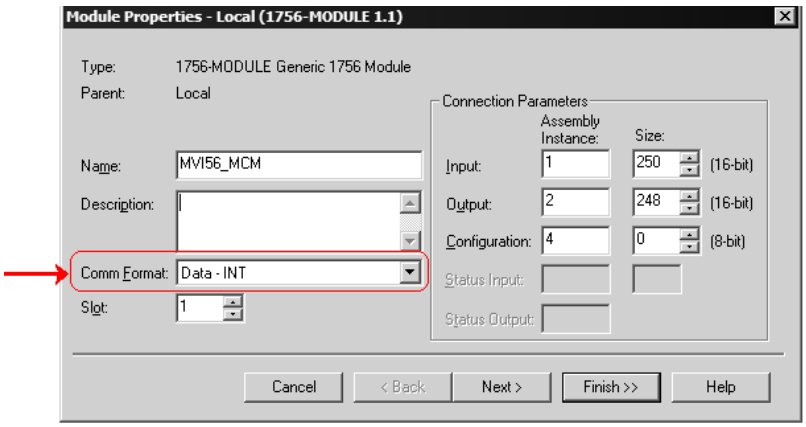
1. In the **CONTROLLER ORGANIZATION** list in RSLogix 5000, click the right mouse button on the **I/O CONFIGURATION** icon to open a shortcut menu. On the shortcut menu, choose **NEW MODULE…**. This action opens the **SELECT MODULE TYPE** dialog box.



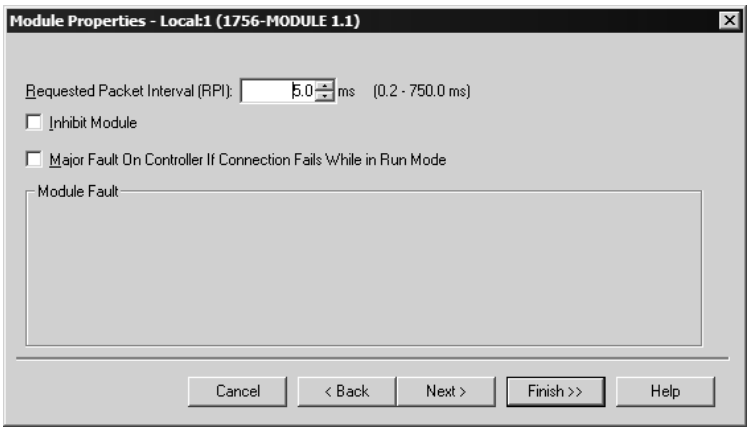
**2** In the **SELECT MODULE TYPE** dialog box, select **1756-MODULE (GENERIC 1756 MODULE)** from the list and, and then click **OK.** This action opens the **MODULE PROPERTIES** dialog box.




**3** In the **MODULE PROPERTIES** dialog box, enter the Name, Description and Slot options for your application, using the examples in the following illustration. You must select the Comm Format as **DATA - INT** in the dialog box. Failure to set the correct parameters will result in backplane communication problems between the module and the processor.



**4** Click the **NEXT** button and set the Request Packet Interval to 5.0ms as shown in the following illustration.




**5** Click **FINISH** to save the module into your existing application.

*Copy the User Defined Data Types*

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

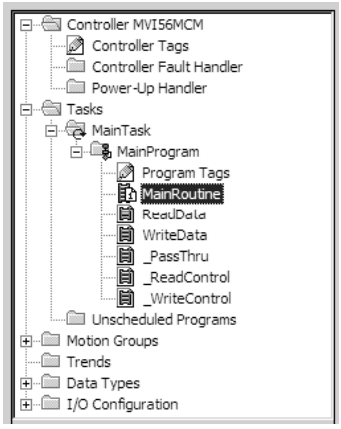
**1** Arrange the two RSLogix 5000 windows on your desktop so that they are side-by-side.
**2** In the **CONTROLLER ORGANIZATION** pane in the Sample Program, expand the **DATA TYPES** folder until the list of User-Defined data types is visible.
**3** In the Sample Program window, select one data type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

Note: Data types prefixed with an underscore [_] are used in optional routines, and need not be copied unless your application requires them. Refer to MVI56-MCM User Defined Data Types for a description of the usage for each data type.

*Copy the Sample Ladder Logic*

Next, copy the Sample Ladder Logic from the sample program to your existing program.

1 In the **CONTROLLER ORGANIZATION** pane in the Sample Program, expand the **TASKS** folder until the list of program routines is visible.




2 In the Sample Program window, select one routine at a time, and then drag the routine to the MainProgram folder in your existing program.
3 Save your program.

The sample program contains the following tasks.

MainRoutine

The **MAINROUTINE** checks for the presence of new read data from the module for the processor. The module cycles through its list of read blocks to transfer data from the module to the processor. Whenever new data is available, the module will set the value for the block in the module's input image **(LOCAL:1:I:DATA[249]).** The ladder logic must constantly scan this input word for a new value. The ladder logic should only perform the **READDATA** and **WRITEDATA** tasks, in that order, when a new value is present in **LOCAL:1:I:DATA[249],** otherwise data may be lost or scrambled.

If the new data is available, the **LASTREAD** and word (249) will not be equal. This will force the program to call the **READDATA** subroutine to process the new data received. After the new data is placed in the Modbus Data Table, the program will send new data to the module using the **WRITEDATA** subroutine.

ReadData

The **READDATA** task handles all new data and status information received from the module and placing it in the proper location in the processor. Data is transferred from the module to the processor using the module's input image **(LOCAL:1:I:DATA[ ]).** This task should set the last read block number (MCM1.BP.LastRead) to the current block number sent from the module **(LOCAL:1:I:DATA[249])** and stores the newly received read block number **(DATA[249])** into the **LASTREAD** variable.

Note: The _READCONTROL routine handles the command control responses received from the module. If command control, event command, or Slave status blocks are not going to be used in the application, then the _READCONTROL rung (rung 4 in the sample READDATA task) and the _READCONTROL and _WRITECONTROL ladder files may be removed.

If the module is configured for zero blocks, it will send blocks with identification codes of zero and -1. These blocks will only contain status data, and no user data will be included in these blocks.

The ladder obtains status information when the module is configured for either 1 or 0 blocks of read data. If the module is configured with 0 for the ReadRegCnt, then blocks -1 and 0 will be given by the module on the input image. If the ReadRegCnt is 200 or less, then you will receive block 0 and block 1.

The ladder logic also determines if the new data received in the input image is user data. If user data is present, the ladder logic will place the data in the correct location in the processor's read data area **(MCM.READDATA[ ])**. Up to 200 data words can be transferred in each block transfer. In addition to the user data, the block also contains important status data. This data should be copied to the correct data area in the module **(MCM.STATUS).** This status data can be used to determine the "health" of the MVI56-MCM module. This rung computes offset into the Modbus Data Table for the received data block and to store the data into the Modbus Data Table.

If the requested block is within the valid range of data blocks for the Modbus Data Table, the offset into the table is computed as (Block ID number - 1) * 200. This is the starting offset in the Modbus Data Table where the 200 bits of data will be stored.

When the processor receives a pass-through block the received data will be handled at the _Pass-Thru routine. If the module is being used as a Modbus Master (PortX.Type=0) or a standard Modbus Slave (Port X.Type = 1) then this rung of logic and the _PassThru routine are not required. If the module is being used as PortX.Type = 2 to 4, then this rung and ladder routine is required.

WriteData

The WriteData task sends data from the processor to the MVI56-MCM module. Data is transferred from the processor to the module using the module's output image **(LOCAL:1:O:DATA[ ]).** This task should store the currently requested data set in the module's **MCM.BP.LASTWRITE** data object. This object is used in all subsequent ladder logic in case the input word **(LOCAL:1:I:DATA[1])** changes during processing.

Note: The _WRITECONTROL routine handles the command control blocks sent to the module. If command control, event command, or Slave status blocks are not going to be used in the application, then the _WRITECONTROL rung (rung 7 in the sample WRITEDATA task) and the _READCONTROL and _WRITECONTROL ladder files may be removed.

_PassThru

This optional task is used to send pass-through data between the processor and the Modbus devices connected to the MVI56-MCM module. Pass-Through functionality allows the Modbus Master to read and write the same Modbus address on a Modbus Slave. If pass-through mode is not chosen, then the attached Modbus Master device must read from one set of Modbus register/bits and write to another set of Modbus register/bits. Pass-Through mode takes a Modbus write command (Function Codes 5, 6, 15, and 16) and passes that to the ControlLogix processor. The pass-through ladder logic then parses that information and updates the **MCM.DATA.WRITEDATA** array with the new value that has been written by the Modbus Master.

_ReadControl

This optional task is used to get status and event data from the Modbus devices connected to the MVI56-MCM module. Special command blocks requested from the module in the _WriteControl routine are processed and handled in this routine. If command control, event command, or Slave status blocks are not going to be used in the application, then this rung and the _ReadControl and _WriteControl ladder files may be removed.

_WriteControl

This optional task is used to send commands to the Modbus devices connected to the MVI56-MCM module. Command Control, Event Command, and Slave status blocks are sent to the module in this task.

*Copy the Controller Tags*

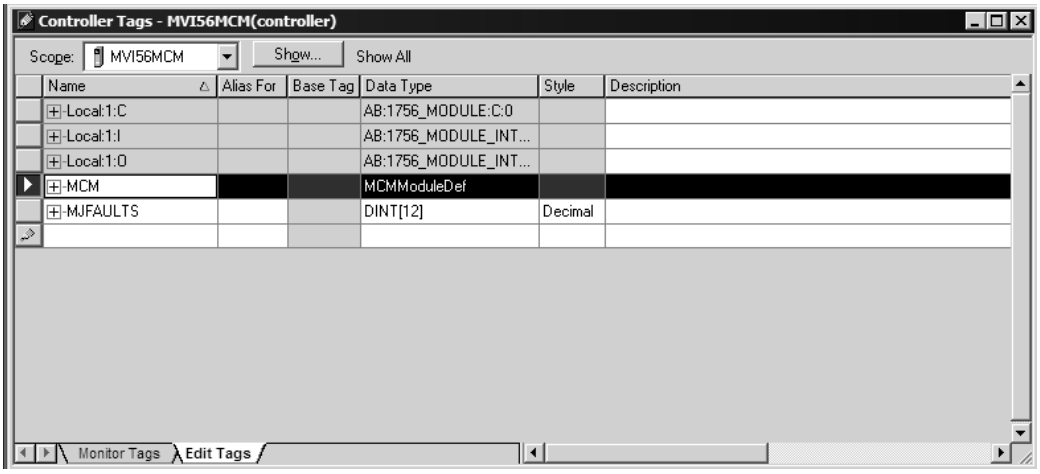
Next, copy the Controller Tags from the sample program to your existing program. The sample program includes the following tags in the MCM structure.

- **MCM.MODDEF** configures the database in the module. The module uses this database to store input and output data transferred between the processor and the Modbus devices connected to the MVI56-MCM module.
- **MCM.PORT1** and **MCM.PORT2** configure the module's serial ports for Modbus communications. The sample program configures Port 1 as a Modbus Master, and Port 2 as a Modbus Slave.
- **MCM.P1.CMD** and **MCM.P2.CMD** configure the Modbus Master commands for the module. These commands are active only if a port is configured as a Modbus Master.
- **MCM.READDATA** contains data read by the ControlLogix processor from the MVI56-MCM module.
- **MCM.WRITEDATA** contains data read from the ControlLogix processor to the module's internal database.

The remaining controller tags contain error and status information, and special commands to execute.

1. In the **CONTROLLER ORGANIZATION** pane in each instance of RSLogix 5000, expand the **CONTROLLER** folder.
2. Double-click the **CONTROLLER TAGS** icon in each instance of RSLogix 5000. This action opens the **CONTROLLER TAGS** dialog box.

3 In the **CONTROLLER TAGS** dialog box in each instance of RSLogix 5000, click the **EDIT TAGS** tab, located at the bottom of the dialog box.
4 In the Sample Program, select the line containing the **MCM** tag structure.

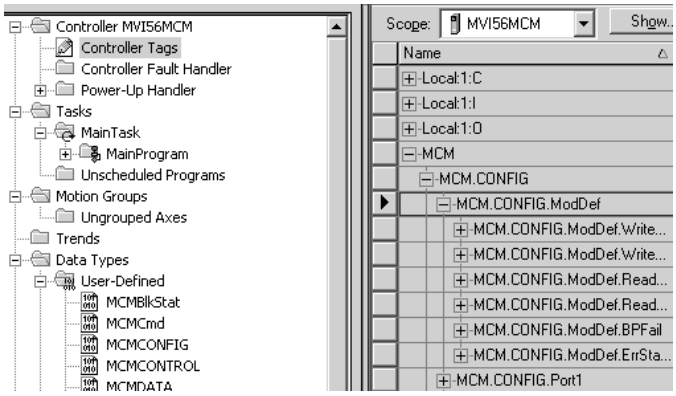



5 Drag the **MCM** tag structure to the blank line at the bottom of the list in the **EDIT TAGS** tab in your existing program.

### Editing the Controller Tags

The MVI56-MCM module configuration is stored in the **MCM.CONFIG** structure in the **CONTROLLER TAGS** dialog box. The sample program configures the module as a Modbus Master on Port 1, and a Modbus Slave on Port 2.

To edit the module configuration, in the Controller Organization pane, expand the **CONTROLLER** folder, and then double-click **CONTROLLER TAGS**. This action opens **CONTROLLER TAGS - MVI56MCM**, as shown in the following illustration.




- To configure a Modbus Master, refer to Configuring the MVI56-MCM as a Modbus Master.
- To configure a Modbus Slave, refer to Configuring the MVI56-MCM as a Modbus Slave.

Note: In order for any of the new values entered into these fields to be used by the module, you must restart the module (WarmBoot, ColdBoot, or cycle power).

# 8 Support, Service & Warranty

*In This Chapter*


- How to Contact Us: Technical Support ..... 155
- Return Material Authorization (RMA) Policies and Conditions ..... 156
- LIMITED WARRANTY ..... 157


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 contents of file
   - Module Operation
   - Configuration/Debug status information
   - LED patterns

**2** Information about the processor and user data files as viewed through and LED patterns on the processor.
**3** Details about the serial devices interfaced, if any.

## 8.1 How to Contact Us: Technical Support

| | |
|---|---|
| **Internet** | Web Site: www.prosoft-technology.com/support |
| | E-mail address: support@prosoft-technology.com |

**Asia Pacific**

+603.7724.2080, support.asia@prosoft-technology.com
Languages spoken include: Chinese, English

**Europe (location in Toulouse, France)**

+33 (0) 5.34.36.87.20, support.EMEA@prosoft-technology.com
Languages spoken include: French, English

**North America/Latin America (excluding Brasil) (location in California)**

+1.661.716.5100, support@prosoft-technology.com
Languages spoken include: English, Spanish
*For technical support calls within the United States, an after-*hours *answering system allows pager access to one of our qualified technical and/or application support engineers at any time to answer your questions.*

**Brasil (location in Sao Paulo)**

+55-11-5084-5178, eduardo@prosoft-technology.com
Languages spoken include: Portuguese, English

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

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

### *8.2.1 All Product Returns:*

a) In order to return a Product for repair, exchange or otherwise, the Customer must obtain a Returned Material Authorization (RMA) number from ProSoft and comply with ProSoft 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, 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 using a shipment method other than that specified by ProSoft 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, and so on.

### *8.2.2 Procedures for Return of Units Under Warranty:*

A Technical Support Engineer must approve the return of Product under ProSoft'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 at designated location referenced on the Return Material Authorization.

### *8.2.3 Procedures for Return of Units Out of Warranty:*

a) Customer sends unit in for evaluation
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

### *8.2.4 Purchasing Warranty Extension:*

a) ProSoft's standard warranty period is three (3) years from the date of shipment as detailed in "Limited Warranty (page 157)". The Warranty Period may be extended at the time of equipment purchase for an additional charge, as follows:

- Additional 1 year = 10% of list price
- Additional 2 years = 20% of list price
- Additional 3 years = 30% of list price

## 8.3 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, 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.

### 8.3.1 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 years from the date of shipment (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. 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 used replacement parts. 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 warranteed 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.

### 8.3.2 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.

### *8.3.3 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 of 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.

### *8.3.4 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.

### *8.3.5 Disclaimer of all Other Warranties*

The Warranty set forth in What Is Covered By This Warranty (page 158) 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.

### *8.3.6 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 included, 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.

### *8.3.7 Time Limit for Bringing Suit*

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

### 8.3.8 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.

### 8.3.9 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.

### 8.3.10 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

_

_CmdControl • 128, 129
_CoilArray • 130
_EventCmd • 128, 129
_Passthru • 128, 130
_PassThru • 152
_ReadControl • 152
_SlavePollStatus • 128, 129
_Util • 125, 128
_WriteControl • 152

**0**

00 Return Query Data • 136

**A**

About the MODBUS Protocol • 91
Adding Multiple Modules (Optional) • 23
All Product Returns: • 156
Allocation of Risks • 161
Analyzing Data for the first application port • 78
Analyzing Data for the second application port • 79

**B**

Backplane Data Transfer • 93
Backplane Setup • 115
Battery Life Advisory • 3

**C**

Cable Connections • 110
Choose the Controller Type • 144
Clearing a Fault Condition • 87
Cold Boot • 103
Command Control • 52, 101, 122
Command Control and Event Command • 52
Command Control Blocks • 100
Command Error Codes • 69, 72
Command List Entry Errors • 71, 109
Commands Supported by the Module • 130
Communication Status (Ports 1 and 2) • 78
Configuration as a Modbus Master • 33
Configuration as a Modbus Slave • 55
Configuration Data Transfer • 96
Configuring the RSLinx Driver for the PC COM Port • 30
Connect your PC to the ControlLogix Processor • 15
Controlling Law and Severability • 161
ControlLogix Processor Not in Run • 93
Copy the Controller Tags • 152
Copy the Sample Ladder Logic • 150
Copy the User Defined Data Types • 149
Create a new RSLogix 5000 project • 18
Create the Module • 19
Customizing the Memory Map • 58

**D**

Data Analyzer • 78
Data Analyzer Tips • 80
Data Flow Between MVI56-MCM Module and ControlLogix Processor • 106
DB9 to RJ45 Adaptor (Cable 14) • 114
Define Module in I/O Configuration • 147
Deployment Checklist • 10
Diagnostics (Function Code 08) • 135
Diagnostics and Troubleshooting • 7, 73
Disclaimer of all Other Warranties • 160
Disclaimer Regarding High Risk Activities • 159
Displaying Timing Marks in the Data Analyzer • 79
Download the Sample Program to the Processor • 29, 146

**E**

Editing the Controller Tags • 153
Enron/Daniel Float Configuration • 64
Event Command • 53, 100
Example and state diagram • 136
Exiting the Program • 78

**F**

Floating-Point Data Handling • 41, 42, 46, 62, 63
Force Multiple Coils (Function Code 15) • 137
Force Single Coil (Function Code 05) • 134
Formatted Pass-Through Control Blocks • 104
Function 15 • 105
Function 5 • 104
Function 6 and 16 • 105
Functional Overview • 7, 91
Functional Specifications • 91

**G**

General Concepts • 92
General Specifications • 89
General Specifications - Modbus Master/Slave • 90
Guide to the MVI56-MCM User Manual • 7

**H**

Hardware Specifications • 90
How to Contact Us
Technical Support • 155, 156

**I**

Import Add-On Instruction • 21
Important Installation Instructions • 2
Install the Module in the Rack • 14
Intellectual Property Indemnity • 159

**K**

Keystrokes • 75

## L

LED Status Indicators • 85
Limitation of Remedies ** • 160
LIMITED WARRANTY • 157

## M

Main Logic Loop • 92
Main Menu • 75
MainRoutine • 150
Master Command Configuration • 34, 38, 120
Master Command Data List • 97
Master Command Error List Menu • 84, 109
Master Command Examples • 41
Master Command List • 109
Master Driver Mode • 107
MCM STATUS Data • 71
MCMBlkStat • 127, 128
MCMCmd • 125, 126
MCMCONFIG • 125
MCMCONTROL • 125, 128, 130
MCMDATA • 125, 127
MCMModule • 125
MCMModuleDef • 125
MCMPort • 125, 126
MCMPortErrors • 127
MCMSTATUS • 125, 127
Misc. Status • 121
Modbus Database View • 83
MODBUS Exception Codes • 140
MODBUS Exception Responses • 139
Modbus Memory Map • 57, 58, 94
Modbus Message Structure • 83
Modbus Protocol Specification • 83, 130
ModDef Settings • 34, 56
Module Communication Error Codes • 70, 109
Module Configuration Data • 96
Moving Back Through 5 Pages of Registers • 83
Moving Forward Through 5 Pages of Registers • 83
MVI (Multi Vendor Interface) Modules • 2
MVI56-MCM Command Control • 114, 124
MVI56-MCM Configuration Data • 114, 115, 125
MVI56-MCM Database Definition • 114
MVI56-MCM Status Data Definition • 67, 72, 73, 114, 123
MVI56-MCM Status Data Definition as a Master • 67
MVI56-MCM Status Data Definition as a Slave • 72
MVI56-MCM User Defined Data Types • 125

## N

Navigation • 75
No Other Warranties • 161
Normal Data Transfer • 94

## O

Open the Sample Program in RSLogix • 141
Opening the Command Error List Menu • 76
Opening the Data Analyzer Menu • 76
Opening the Database Menu • 76
Other Modbus Addressing Schemes • 40
Overview • 33, 55, 62

## P

Package Contents • 12
Pass Through Coil Adjustment • 61
Pass-Through Control Blocks • 103
Pinouts • 2, 10, 13, 70, 110, 114
Port 1 Commands • 120
Port 1 Setup • 115
Port 2 Commands • 120
Port 2 Setup • 118
Port Configuration • 37
Port Configuration (Ports 1 and 2) • 78
Preset Multiple Registers (Function Code 16) • 138
Preset Single Register (Function Code 06) • 135
Procedures for Return of Units Out of Warranty: • 156
Procedures for Return of Units Under Warranty: • 156
Product Specifications • 7, 89
ProSoft Technology® Product Documentation • 4
Purchasing Warranty Extension: • 157

## R

Read and Write Same Modbus Address (Pass Through) • 55, 60
Read Block • 95
Read Coil Status (Function Code 01) • 131
Read Coils 0xxxx (Modbus Function Code 1) • 42
Read Floating-Point Data • 47
Read Floats with Single Modbus Register Address (Enron/Daniel Float) • 50
Read Holding Registers (Function Code 03) • 133
Read Holding Registers 4xxxx (Modbus Function Code 3) • 41
Read Input Coils 1xxxx (Modbus Function Code 2) • 43
Read Input Registers (Function Code 04) • 133
Read Input Registers 3xxxx (Modbus Function Code 4) • 42
Read Input Status (Function Code 02) • 132
Read Multiple Floating-Point Registers • 48, 50
ReadData • 150
Reading Status Data from the Module • 73
Redisplaying the Current Page • 83, 84
Reference • 7, 89
Removing Timing Marks in the Data Analyzer • 79
Required Hardware • 73
Required Software • 74
Return Material Authorization (RMA) Policies and Conditions • 156
Returning to the Main Menu • 80, 84, 85
RS-232 • 111
  Modem Connection • 111
  Null Modem Connection (Hardware Handshaking) • 112
  Null Modem Connection (No Hardware Handshaking) • 112
RS-232 Configuration/Debug Port • 110
RS-422 • 113
RS-485 • 113

RS-485 and RS-422 Tip • 113

## S

Select the Slot Number for the Module • 145
Setting Jumpers • 11, 13
Slave Configuration • 62
Slave Driver • 106
Slave Status Blocks • 98
Standard Modbus Protocol Errors • 70, 109
Start Here • 7, 9
Starting the Data Analyzer • 79
Stopping the Data Analyzer • 80
Sub-function codes supported by the serial line devices • 136
Support, Service & Warranty • 7, 155
System Requirements • 9

## T

The Configuration/Debug Menu • 73
Time Limit for Bringing Suit • 160
Transferring Module Configuration to the Processor • 77
Transferring the Command Error List to the Processor • 97, 110
Troubleshooting • 88

## U

Unformatted Pass-Through Control Blocks • 103
Using the Configuration/Debug Port • 74
Using the RSLogix 5000 v16 Add-On Instruction • 11, 17
Using the Sample Program - RSLogix Version 15 and earlier • 141
Using the Sample Program in a New Application • 141
Using the Sample Program in an Existing Application • 11, 17, 94, 96, 107, 108, 147

## V

Verify Communication • 7, 37, 67
Verify Master Communications • 68
Verify Slave Communications • 72
Viewing Block Transfer Statistics • 76
Viewing Data in ASCII (Text) Format • 79, 84
Viewing Data in Decimal Format • 84
Viewing Data in Floating Point Format • 84
Viewing Data in Hexadecimal Format • 79, 84
Viewing Module Configuration • 76
Viewing Register Pages • 83
Viewing the Next 100 Registers of Data • 84
Viewing the Next 20 Commands • 85
Viewing the Next Page of Commands • 85
Viewing the Previous 100 Registers of Data • 83
Viewing the Previous 20 Commands • 84
Viewing the Previous Page of Commands • 85
Viewing the Slave Status List (Port 1 and 2) • 77
Viewing Version Information • 77

## W

Warm Boot • 103
Warm Booting the Module • 77
Warnings • 2
What Is Covered By This Warranty • 158, 160
What Is Not Covered By This Warranty • 158
Write Block • 95
Write Configuration • 102
Write Floats to Slave Device • 49
Write Holding Register 4xxxx (Modbus Function Code 6) • 45
Write Multiple Coils 0xxx (Modbus Function Code 15) • 44
Write Multiple Registers 4xxxx (Modbus Function Code 16) • 46
Write Single Coil 0xxxx (Modbus Function Code 5) • 44
Write to Enron/Daniel Floats • 51
WriteData • 151

## Y

Your Feedback Please • 3