



Where Automation Connects.



MVI69E-MBS

**Modbus Serial Enhanced
Communication Module**

September 29, 2025

USER MANUAL

Your Feedback Please

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

ProSoft Technology, Inc.

+1 (661) 716-5100

+1 (661) 716-5101 (Fax)

www.prosoft-technology.com

ps.support@belden.com

MVI69E-MBS User Manual
For Public Use.

September 29, 2025

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

Content Disclaimer

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

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

© 2025 ProSoft Technology. All Rights Reserved.



For professional users in the European Union

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



Prop 65 Warning – Cancer and Reproductive Harm – www.P65Warnings.ca.gov

Agency Approvals and Certifications

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

Open-Source Information

Open-Source Software used in the product

The product contains, among other things, Open-Source Software files, as defined below, developed by third parties and licensed under an Open-Source Software license. These Open-Source Software files are protected by copyright. Your right to use the Open-Source Software is governed by the relevant applicable Open-Source Software license conditions. Your compliance with those license conditions will entitle you to use the Open-Source Software as foreseen in the relevant license. In the event of conflicts between other ProSoft Technology, Inc. license conditions applicable to the product and the Open-Source Software license conditions, the Open-Source Software conditions shall prevail. The Open-Source Software is provided royalty-free (i.e. no fees are charged for exercising the licensed rights). Open-Source Software contained in this product and the respective Open-Source Software licenses are stated in the module webpage, in the link Open-Source.

If Open-Source Software contained in this product is licensed under GNU General Public License (GPL), GNU Lesser General Public License (LGPL), Mozilla Public License (MPL) or any other Open-Source Software license, which requires that source code is to be made available and such source code is not already delivered together with the product, you can order the corresponding source code of the Open-Source Software from ProSoft Technology, Inc. - against payment of the shipping and handling charges - for a period of at least 3 years since purchase of the product. Please send your specific request, within 3 years of the purchase date of this product, together with the name and serial number of the product found on the product label to:

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

Warranty regarding further use of the Open-Source Software

ProSoft Technology, Inc. provides no warranty for the Open-Source Software contained in this product, if such Open-Source Software is used in any manner other than intended by ProSoft Technology, Inc. The licenses listed define the warranty, if any, from the authors or licensors of the Open-Source Software. ProSoft Technology, Inc. specifically disclaims any warranty for defects caused by altering any Open-Source Software or the product's configuration. Any warranty claims against ProSoft Technology, Inc. in the event that the Open-Source Software contained in this product infringes the intellectual property rights of a third party are excluded. The following disclaimer applies to the GPL and LGPL components in relation to the rights holders:

"This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License and the GNU Lesser General Public License for more details."

For the remaining Open-Source components, the liability exclusions of the rights holders in the respective license texts apply. Technical support, if any, will only be provided for unmodified software.

Table of Contents

1	Start Here	7
1.1	System Requirements	7
1.2	Deployment Checklist.....	8
1.3	Package Contents	8
1.4	Setting Jumpers	9
1.5	Installing the Module in the Rack	10
2	Configuring the Module in RSLogix	13
2.1	Creating the Module in a Studio 5000 Project	13
2.1.1	Creating a Module in the Project Using an Add-On Profile.....	14
2.1.2	Creating a Module in the Project Using a Generic 1769 Module Profile.....	18
2.2	Installing ProSoft Configuration Builder	21
2.3	Generating the AOI (.L5X File) in ProSoft Configuration Builder	21
2.3.1	Setting Up the Project in PCB	21
2.3.2	Creating and Exporting the .L5X File	23
2.4	Importing the Add-On Instruction	25
2.5	Adding Multiple Modules in the Rack (Optional)	28
2.5.1	Adding Another Module in PCB	28
2.5.2	Adding Another Module in Studio 5000.....	30
3	Configuring the MVI69E-MBS Using PCB	35
3.1	Basic PCB Functions.....	35
3.1.1	Creating a New PCB Project and Exporting an .L5X File	35
3.1.2	Renaming PCB Objects	35
3.1.3	Editing Configuration Parameters	36
3.1.4	Printing a Configuration File	37
3.2	Module Configuration Parameters	38
3.2.1	Module Parameters	38
3.2.2	Modbus Port x Parameters.....	39
3.2.3	Modbus Port x Commands.....	43
3.2.4	Ethernet 1	46
3.3	Downloading the Configuration File to the Processor	47
3.4	Uploading the Configuration File from the Processor	50
4	Using Controller Tags	53
4.1	Controller Tags.....	53
4.1.1	MVI69E-MBS Controller Tags	54
4.2	User-Defined Data Types (UDTs)	55
4.2.1	MVI69E-MBS User-Defined Data Types.....	55
4.3	MBS Controller Tag Overview.....	57
4.3.1	MBS.CONFIG.....	57
4.3.2	MBS.DATA	57
4.3.3	MBS.CONTROL	58
4.3.4	MBS.STATUS	63
4.3.5	MBS.UTIL	64

5	MVI69E-MBS Backplane Data Exchange	65
5.1	General Concepts of the MVI69E-MBS Data Transfer	65
5.2	Backplane Data Transfer.....	65
5.3	Normal Data Transfer	67
5.3.1	Write Block: Request from the Processor to the Module	67
5.3.2	Read Block: Response from the Module to the Processor	67
5.3.3	Read and Write Block Transfer Sequences	68
5.4	Data Flow Between the Module and Processor.....	71
5.4.1	Slave Mode	71
5.4.2	Master Mode	73
6	Legacy Mode	75
6.1	Webpage Configuration.....	75
6.2	PCB Configuration.....	78
6.2.1	Comment Parameter	79
6.2.2	Backplane69 Parameter	80
6.2.3	MCM Port x Parameters.....	81
6.2.4	Modbus Port x Commands.....	83
6.2.5	Ethernet 1 Parameter	86
6.3	Downloading PCB Configuration to the MVI69E-MBS.....	87
6.4	Optional Add-On Instruction	89
6.4.1	Setting Up the Optional AOI	91
6.4.2	Synchronizing the IP Settings from the MVI69E-MBS to the Processor.....	93
6.4.3	Synchronizing the IP Settings from the Processor to the MVI69E-MBS.....	94
6.4.4	Reading the Date/Time from the MVI69E-MBS to the Processor	95
6.4.5	Writing the Date/Time from the Processor to the MVI69E-MBS	96
7	Diagnostics and Troubleshooting	97
7.1	LED Status Indicators.....	97
7.2	Ethernet LED Indicators	99
7.3	Clearing a Fault Condition.....	99
7.4	Troubleshooting.....	100
7.4.1	Processor Errors	100
7.4.2	Module Errors	100
7.5	Connecting the PC to the Module's Ethernet Port	101
7.5.1	Setting Up a Temporary IP Address	102
7.6	Using the Diagnostics Menu in PCB	104
7.6.1	Diagnostics Menu	106
7.6.2	Monitoring Network Configuration Information.....	106
7.6.3	Monitoring Backplane Information.....	107
7.6.4	Port x Module Information	108
7.6.5	Monitoring Data Values in the Module's Database	109
7.7	Communication Error Codes.....	110
7.7.1	Standard Modbus Protocol Exception Code Errors	110
7.7.2	Module Communication Error Codes	110
7.7.3	Command List Entry Errors.....	110
7.8	Connecting to the Module's Webpage	111
8	Reference	112
8.1	Product Specifications.....	112
8.1.1	Hardware Specifications.....	112

8.1.2	General Specifications - Modbus Master/Slave	113
8.2	About the Modbus Protocol	113
8.2.1	Modbus Master.....	114
8.2.2	Modbus Slave.....	114
8.2.3	Function Codes Supported by the Module.....	114
8.2.4	Read Coil Status (Function Code 01)	115
8.2.5	Read Input Status (Function Code 02).....	116
8.2.6	Read Holding Registers (Function Code 03)	117
8.2.7	Read Input Registers (Function Code 04).....	118
8.2.8	Force Single Coil (Function Code 05)	119
8.2.9	Preset Single Register (Function Code 06).....	120
8.2.10	Diagnostics (Function Code 08).....	121
8.2.11	Force Multiple Coils (Function Code 15).....	123
8.2.12	Preset Multiple Registers (Function Code 16)	124
8.3	Floating-Point Support.....	125
8.3.1	Enron Floating Point Support	126
8.3.2	Configuring the Floating-Point Data Transfer	126
8.4	Function Blocks	131
8.4.1	Event Command Blocks (1000 to 1255, 2000 to 2255)	132
8.4.2	Slave Polling Disable Blocks	133
8.4.3	Slave Polling Enable Blocks (3001, 3101)	133
8.4.4	Slave Polling Status Blocks (3002 to 3006, 3102 to 3106)	134
8.4.5	Command Control Blocks (5001 to 5006, 5101 to 5106).....	135
8.4.6	Add Event with Data Blocks (8000, 8001).....	136
8.4.7	Get Event with Data Status Block (8100).....	137
8.4.8	Get Configuration File Information Block (9000 or -9000)	138
8.4.9	Get Configuration File Block (9001 or -9001).....	139
8.4.10	Get General Module Status Data Block (9250).....	140
8.4.11	Set Port and Command Active Bits Block (9500)	141
8.4.12	Get Port and Command Active Bits Block (9501)	142
8.4.13	Pass-through Formatted Word Data Block for Functions 6 & 16 (9956)	143
8.4.14	Pass-through Formatted Float Data Block for Functions 6 & 16 (9957)	144
8.4.15	Pass-through Formatted Block for Function 5 (9958)	145
8.4.16	Pass-through Formatted Block for Function 15 (9959)	146
8.4.17	Pass-through Formatted Block for Function 23 (9961)	147
8.4.18	Pass-through Block for Function 99 (9970).....	148
8.4.19	Pass Module Time to Processor Block (9973)	149
8.4.20	Reset Status Block (9997).....	150
8.4.21	Cold-boot Control Block (9999)	150
8.5	Ethernet Port Connection	151
8.5.1	Ethernet Cable Specifications	151
8.6	Modbus Application Port Connection	153
8.6.1	RS-232 Wiring	153
8.6.2	RS-422 Wiring	156
8.6.3	RS-485 Wiring	156
8.6.4	DB9 to RJ45 Adaptor (Cable 14)	157

9	Support, Service, and Warranty	158
----------	---------------------------------------	------------

9.1	Contacting Technical Support	158
9.2	Warranty Information.....	158

1 Start Here

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

- Studio 5000 Logix Designer ®: launch the program, configure ladder logic, and transfer the ladder logic to the processor
- Microsoft Windows®: install and launch programs, execute menu commands, navigate dialog boxes, and enter data
- Hardware installation and wiring: install the module, and safely connect Modbus and CompactLogix or MicroLogix 1500-LRP devices to a power source and to the MVI69E-MBS module's application port(s).

1.1 System Requirements

The MVI69E-MBS module requires the following minimum components:

- Rockwell Automation CompactLogix or MicroLogix 1500-LRP® processor (firmware version 10 or higher), with compatible power supply.

Important: The MVI69E-MBS module has a power supply distance rating of 4 (L43 and L45 installations on first 2 slots of 1769 bus). It consumes 500 mA at 5 VDC.

Important: For 1769-L23x processors, please make note of the following limitation:

1769-L23E-QBFC1B = 450 mA at 5 VDC (No MVI69E module can be used with this processor.)

- The module requires 500 mA of available 5 VDC power
- Rockwell Automation Studio 5000 Logix Designer programming software
- Rockwell Automation RSLinx® communication software version 2.51 or higher
- ProSoft Configuration Builder (PCB) (included)
- ProSoft Discovery Service (PDS) (included in PCB)
- Pentium® II 450 MHz minimum.
- Supported operating systems:
 - Microsoft Windows® 10 Professional
 - Microsoft Windows® 7 Professional
 - Microsoft Windows® XP Professional

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

1.2 Deployment Checklist

Before you begin to configure 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.

- Are you creating a new application or integrating the module into an existing application? Most applications can use the Sample Add-On Instruction or Sample Ladder Logic without any modification.
- Which slot number in the chassis does the MVI69E-MBS module occupy? For communication to occur, you must enter the correct slot number in the sample program.
- Are the Studio 5000 and RSLinx software installed? RSLogix and RSLinx are required to communicate to the CompactLogix or MicroLogix 1500-LRP processor.
- How many words of data do you need to transfer in your application (from CompactLogix or MicroLogix 1500-LRP to Module / to CompactLogix or MicroLogix 1500-LRP from Module)?
- Is this module replacing an existing legacy MVI69-MCM module (refer to section *Legacy Mode* on page 75)?

1.3 Package Contents

The following components are included with your MVI69E-MBS module, and are all required for installation and configuration.

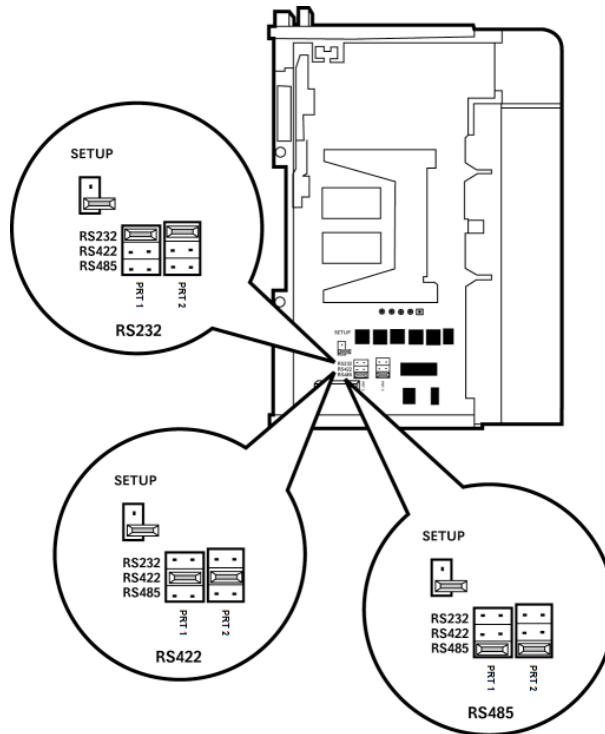
Important: Before beginning the installation, please verify that all the following items are present.

Qty.	Part Name	Part Number	Part Description
1	MVI69E-MBS Module	MVI69E-MBS	Modbus Serial Enhanced Communication Module
2	Adapter Cable	Cable #14	RJ45 to DB9 Male Adapter cable. For DB9 connection to module's serial application ports
2	Screw Terminal Adapter	1454-9F	DB9 female to 9-pin screw terminal. Used for RS422 or RS485 connections to Port 1 and 2 of the module

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

1.4 Setting Jumpers

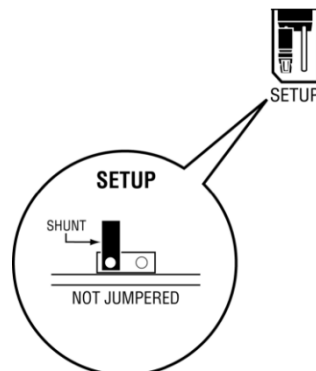
When the module is manufactured, the port selection jumpers are set to RS-232. To use RS-422 or RS-485, you must set the jumpers to the correct position. The following diagram describes the jumper settings.



Note: Jumper pin placement on the circuit board may vary.

The Setup Jumper acts as "write protection" for the module's firmware. In "write protected" mode, the Setup pins are not connected, and the module's firmware cannot be overwritten. The module is shipped with the Setup jumper OFF. If an update of the firmware is needed, apply the Setup jumper to both pins.

The following illustration shows the MVI69E-MBS jumper configuration, with the Setup Jumper OFF.



1.5 Installing the Module in the Rack

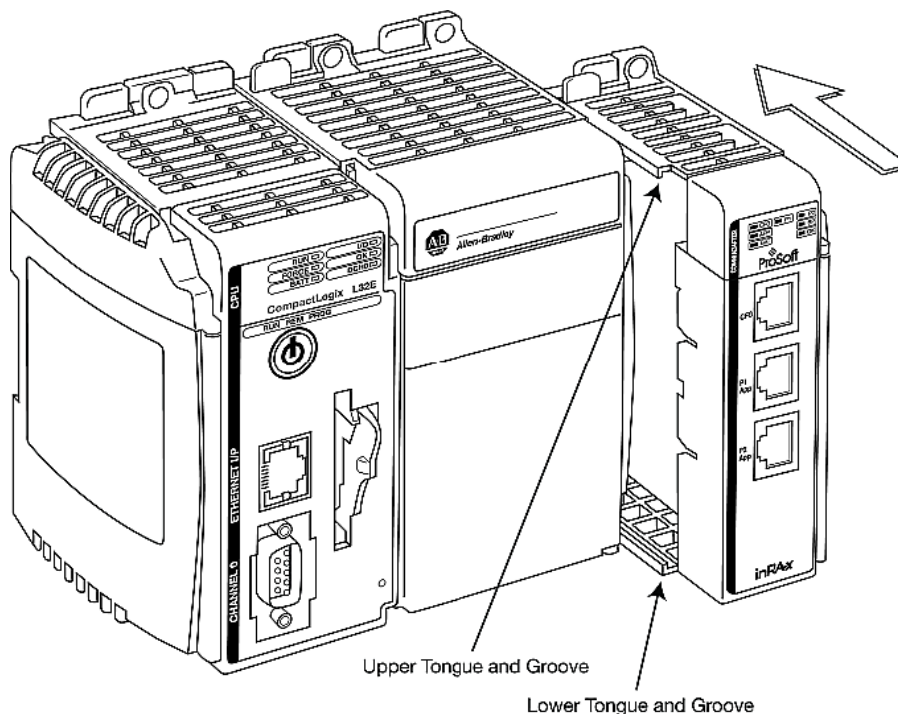
Make sure the processor and power supply are installed and configured before installing the MVI69E-MBS module. Refer to the Rockwell Automation product documentation for installation instructions.

Warning: Please 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 to be connected to verify that suitable safety procedures are in place before installing or servicing the device.

After you verify the jumper placements, insert the MVI69E-MBS into the rack. Use the same technique recommended by Rockwell Automation to remove and install CompactLogix or MicroLogix 1500-LRP modules.

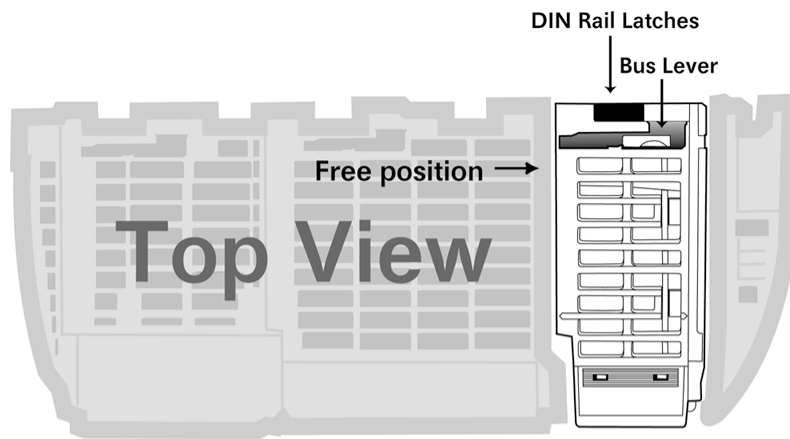
Warning: This module is not hot-swappable! Always remove power from the rack before inserting or removing this module, or damage may result in the module, the processor, or other connected devices.

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

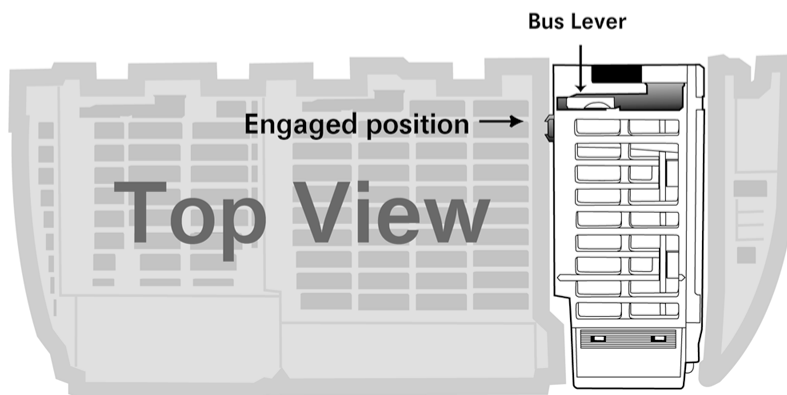


- 2 Move the module back along the tongue-and-groove slots until the bus connectors on the MVI69 module and the adjacent module line up with each other.

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

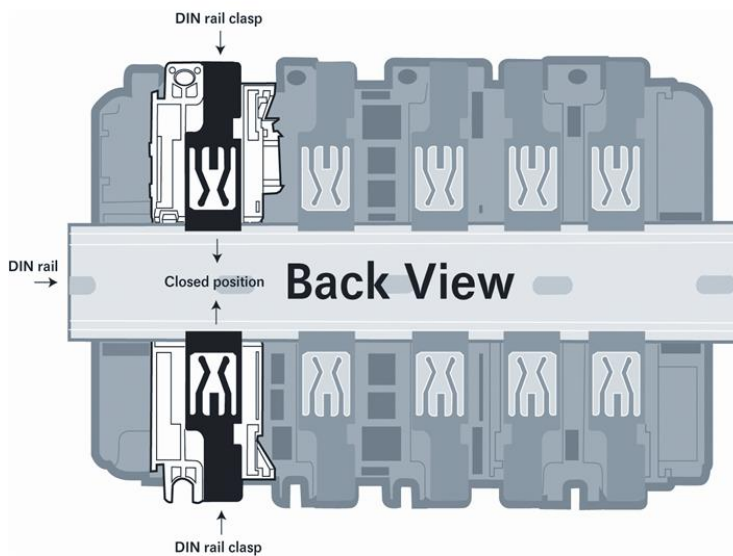
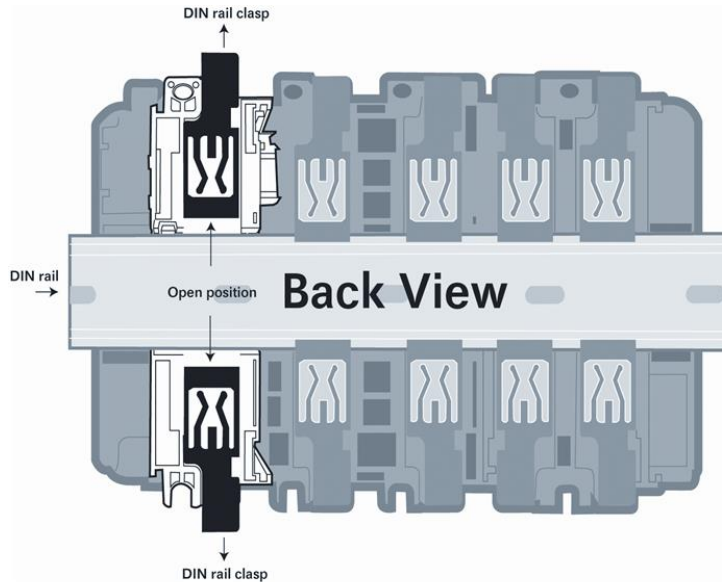


Move the Bus Lever to the left until it clicks



- 4 Close all DIN-rail latches.

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



2 Configuring the Module in RSLogix

To add the MVI69E-MBS module in Studio 5000, you must:

- 1 Create a new project in Studio 5000.
- 2 Add the module to the Studio 5000 project. There are two ways to do this:
 - You can use the Add-On Profile from ProSoft Technology. This is the preferred way, but requires RSLogix version 15 or later.
 - You can manually create the module using a generic 1769 profile, and then manually configure the module parameters. Use this method if you have RSLogix version 14 or earlier.
- 3 Create an Add-On Instruction file using ProSoft Configuration Builder (PCB) and export the Add-On Instruction to a Studio 5000 compatible file (.L5X file).
- 4 Import the Add-On Instruction (the .L5X file) into Studio 5000.

The .L5X file contains the Add-On Instruction, user-defined data types, controller tags and ladder logic required to configure the MVI69E-MBS module.

2.1 Creating the Module in a Studio 5000 Project

In a studio 5000 project, there are two ways to add the MVI69E-MBS module to the project.

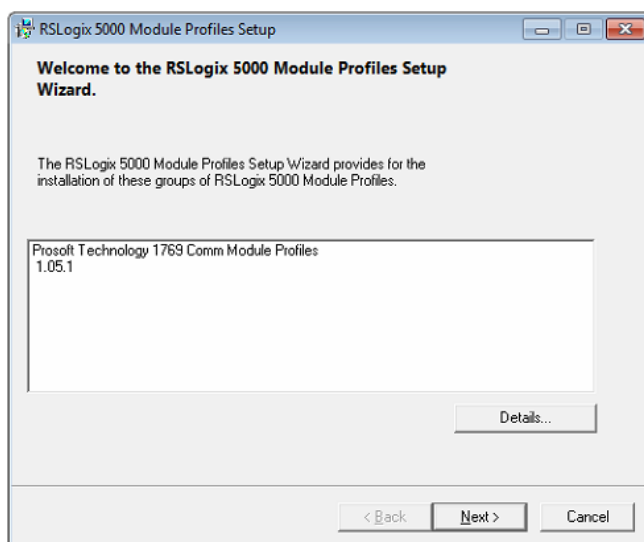
- You can use an Add-On Profile (AOP) from ProSoft Technology. The AOP contains all the configuration information needed to add the module to the project. This is the preferred way, but requires RSLogix version 15 or later. Refer to *Creating a Module in the Project Using an Add-On Profile* (page 14).
- If using an AOP is not an option, you can manually create and configure the module using a generic 1769 profile. Use this method if you have RSLogix version 14 or earlier. Refer to *Creating a Module in the Project Using a Generic 1769 Module Profile* (page 18).

2.1.1 Creating a Module in the Project Using an Add-On Profile

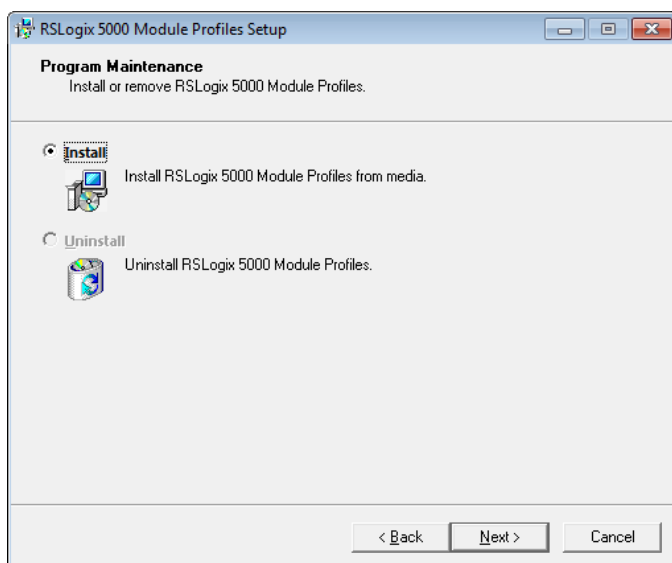
2.1.1.1 Installing an Add-On Profile

Download the AOP file (MVI69x_RevX.X_AOP.zip) from the product webpage (www.prosoft-technology.com) onto your local hard drive and then extract the files from the zip archive. Make sure you have shut down Studio 5000 and RSLinx before you install the Add-On Profile (AOP).

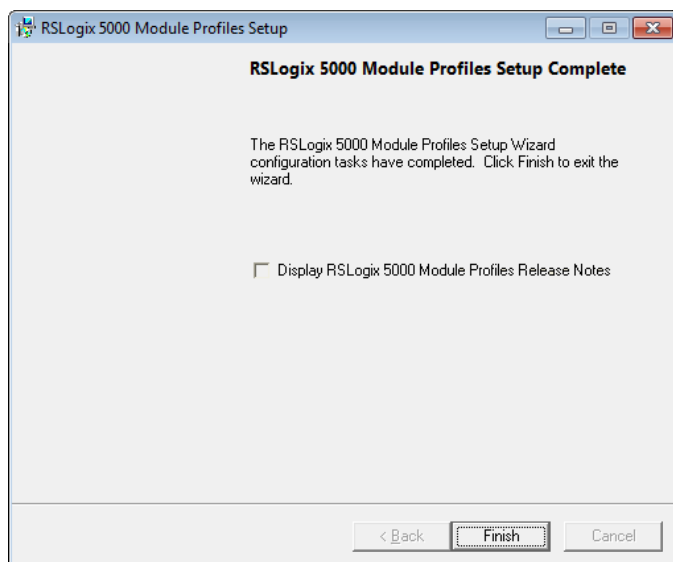
- 1 Run the **MPSetup.exe** file to start the Setup Wizard. Follow the Setup Wizard to install the AOP.



- 2 Continue to follow the steps in the wizard to complete the installation.

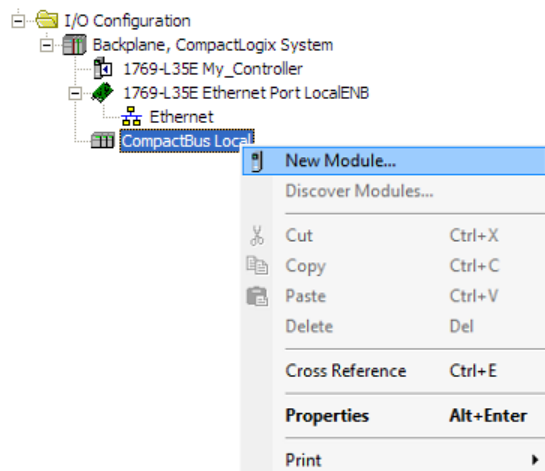


- 3 Click **FINISH** when complete. The AOP is now installed in Studio 5000. You do not need to reboot the PC.

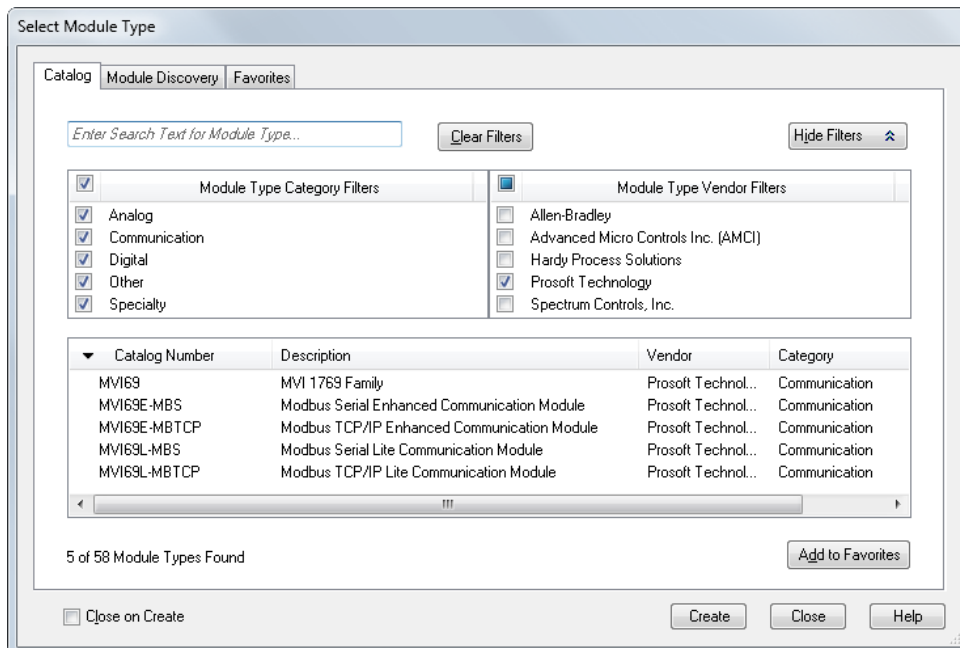


2.1.1.2 Using an Add-On Profile

- 1 In Studio 5000, expand the **I/O CONFIGURATION** folder in the Project tree. Right-click the appropriate communications bus and, and then click **NEW MODULE**.

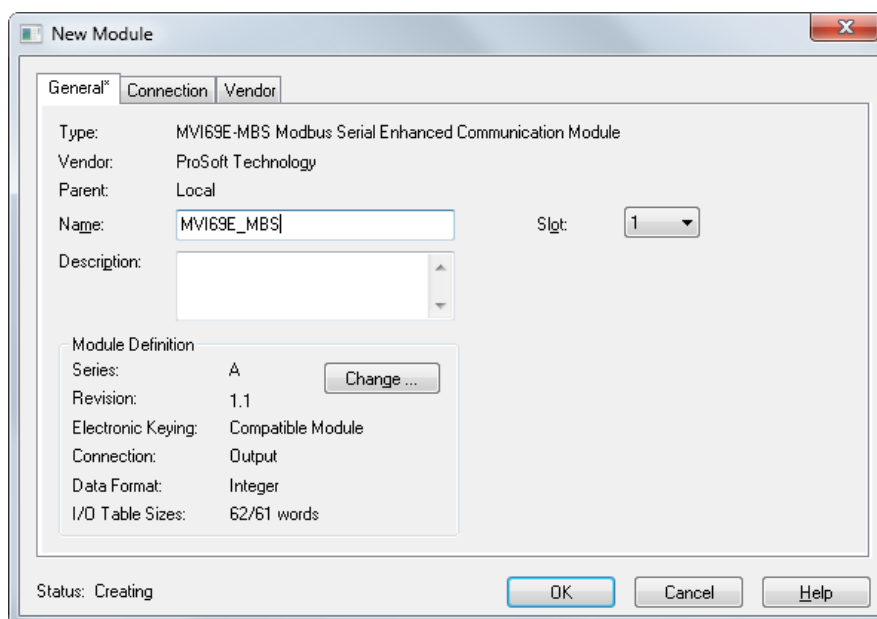


- 2 This opens the *Select Module Type* dialog box. In the *Module Type Vendor Filters* area, uncheck all boxes except the **PROSOFT TECHNOLOGY** box. A list of ProSoft Technology modules appears in the dialog box.



- 3 Select the **MVI69E-MBS** module in the list and click **CREATE**:

- 4 A *New Module* dialog box opens. Edit the **NAME** and **SLOT** for the module and click **OK**.



Note : The **I/O TABLE SIZES** above should reflect the *Block Transfer Size* parameter set in PCB (Section *Module Configuration Parameters* (page 38)).

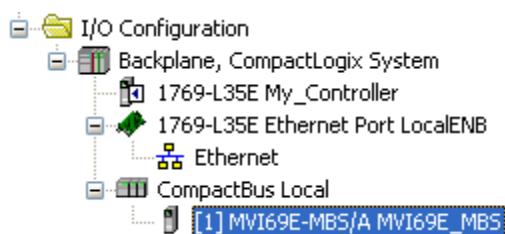
A *Block Transfer Size* of 60 uses an **I/O TABLE SIZE** of 62/61 words.

A *Block Transfer Size* of 120 uses an **I/O TABLE SIZE** of 122/121 words.

A *Block Transfer Size* of 240 uses an **I/O TABLE SIZE** of 242/241 words.

Applications that require smaller amounts of data or faster update times, such as ControlNet networks, will benefit from smaller block transfer sizes.

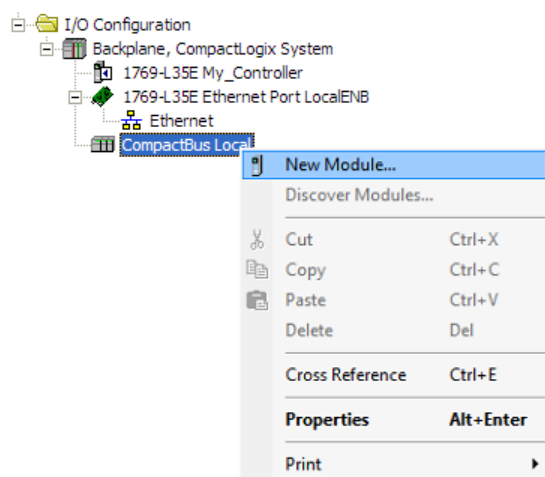
- 5 The MVI69E-MBS is now visible in the I/O Configuration tree.



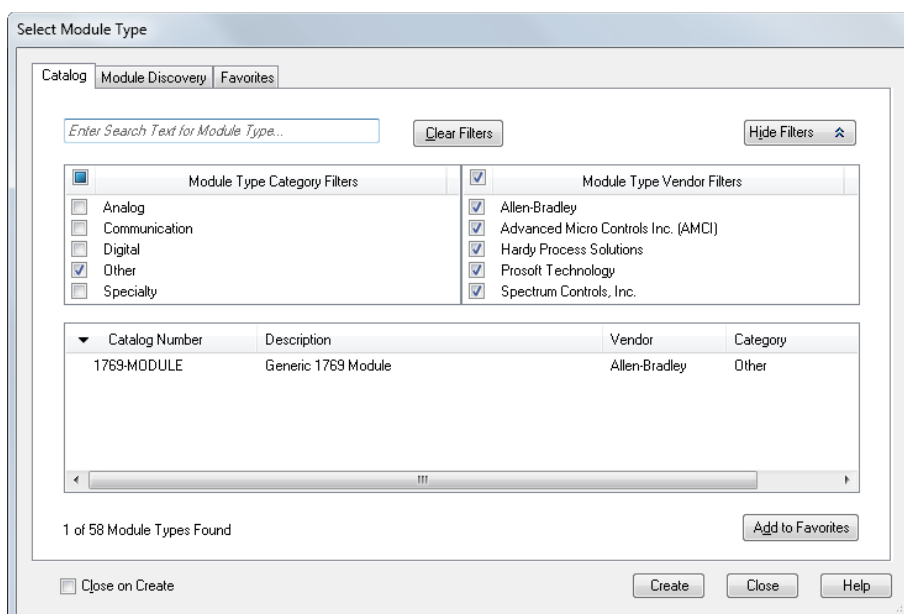
2.1.2 Creating a Module in the Project Using a Generic 1769 Module Profile

This procedure is not required if you installed the ProSoft Technology Add-On Profile for this module.

- 1 Expand the **I/O CONFIGURATION** folder in the Project tree. Right-click the appropriate communications bus and choose **NEW MODULE**.



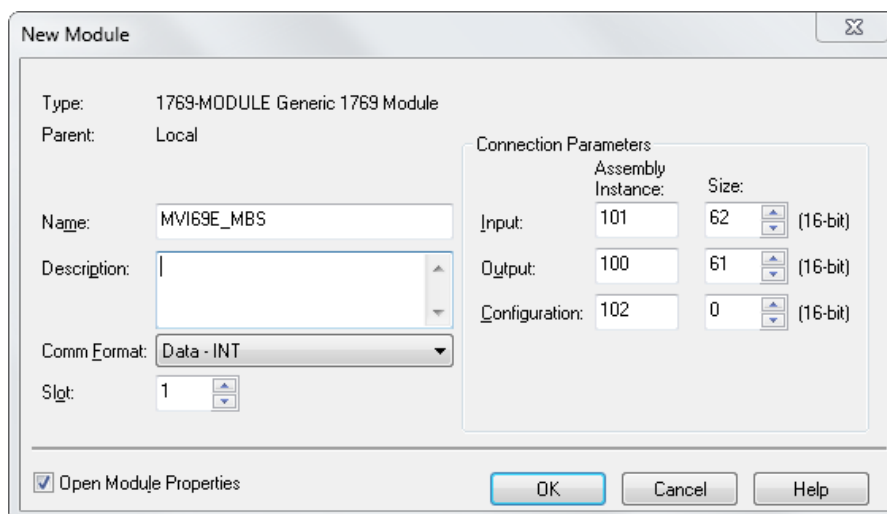
- 2 This opens the *Select Module Type* dialog box.
- 3 In the *Select Module Type* dialog, select the **1769-MODULE** and click on the **CREATE** button.



4 Set the *Module Properties* values as follows:

Parameter	Value
Name	Enter a module identification string. Example: MVI69EMBS
Description	Enter a description for the module. Example: ProSoft communication module for Serial Modbus communications.
Comm Format	Select DATA-INT
Slot	Enter the slot number in the rack where the MV69E-MBS module is installed.
Input Assembly Instance	101
Input Size	62 / 122 / 242
Output Assembly Instance	100
Output Size	61 / 121 / 241
Configuration Assembly Instance	102
Configuration Size	0

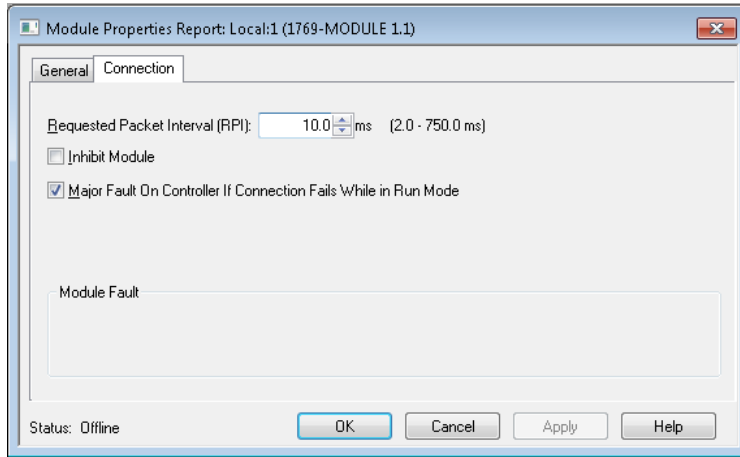
The following illustration shows an example where the module was configured for a block transfer size of 60 words (input block size = 62 words, output block size = 61 words):



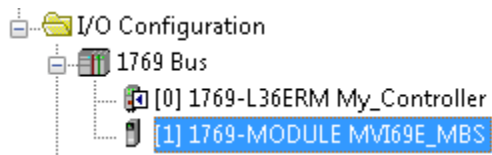
The following options are available:

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

- 5 On the *Connection* tab, set the **REQUESTED PACKET INTERVAL** value for your project and click **OK**. A value of **10.0** ms or more is recommended.



- 6 The MVI69E-MBS module is now visible in the *I/O Configuration* tree.



2.2 Installing ProSoft Configuration Builder

Use the ProSoft Configuration Builder (PCB) software to configure the module. The latest version of the ProSoft Configuration Builder (PCB) is located on our website:

www.prosoft-technology.com.

The installation filename contains the PCB version number (Example: **PCB_4.3.4.5.0238.EXE**.)

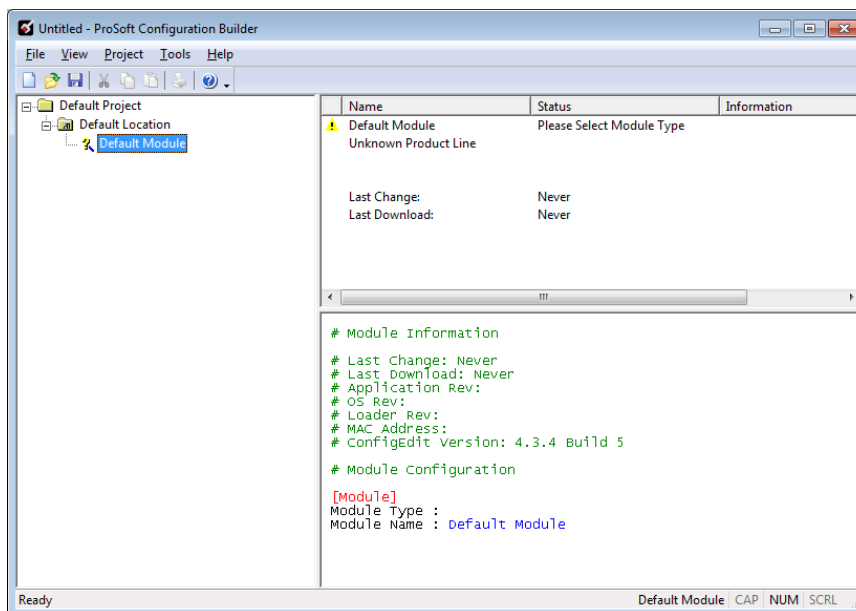
- 1 Open a browser window and navigate to www.prosoft-technology.com
- 2 Navigate to the ProSoft Configuration Builder download link, and save the file to your Windows desktop.
- 3 After the download completes, double-click the file to install. If you are using Windows 7, right-click the PCB installation file and then choose **RUN AS ADMINISTRATOR**. Follow the instructions that appear on the screen.
- 4 If you want to find additional software specific to your MVI69E-MBS, enter the model number into the ProSoft website search box and press the **Enter** key.

2.3 Generating the AOI (.L5X File) in ProSoft Configuration Builder

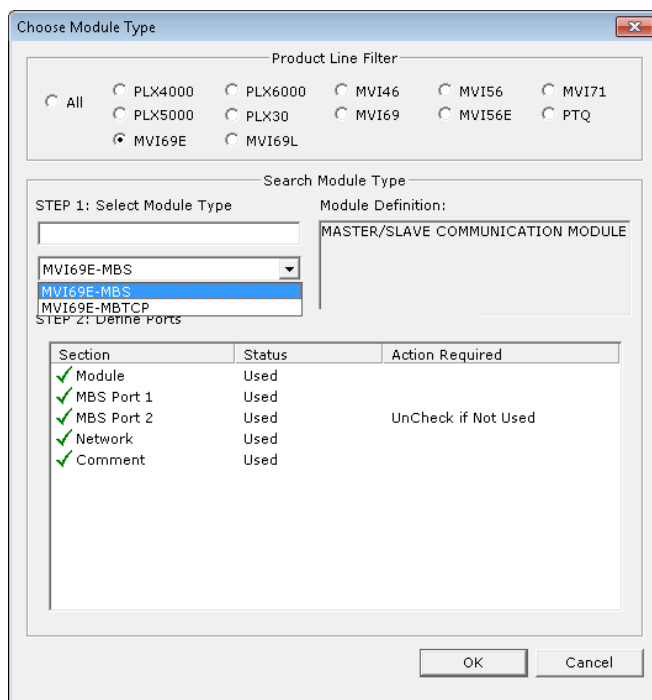
The following sections describe the steps required to set up a new configuration project in ProSoft Configuration Builder (PCB), and to export the .L5X file for the project.

2.3.1 Setting Up the Project in PCB

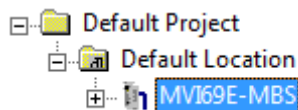
- 1 Start **PROSOFT CONFIGURATION BUILDER** (PCB).
- 2 The *PCB* window consists of a tree view on the left, and an information pane and a configuration pane on the right side of the window. The tree view consists of folders for *Default Project* and *Default Location*, with a *Default Module* in the *Default Location* folder. The following illustration shows the *PCB* window with a new project.



- 3 In the Tree view, right-click **DEFAULT MODULE**, and then click **CHOOSE MODULE TYPE**. This opens the *Choose Module Type* dialog box.





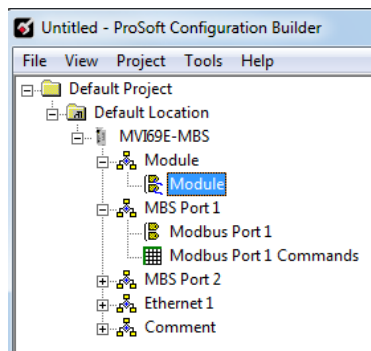
- 4 In the *Product Line Filter* area of the dialog box, click **MVI69**. In the *Select Module Type* dropdown list, click **MVI69E-MBS**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window. The MVI69E-MBS icon is now visible in the tree view.



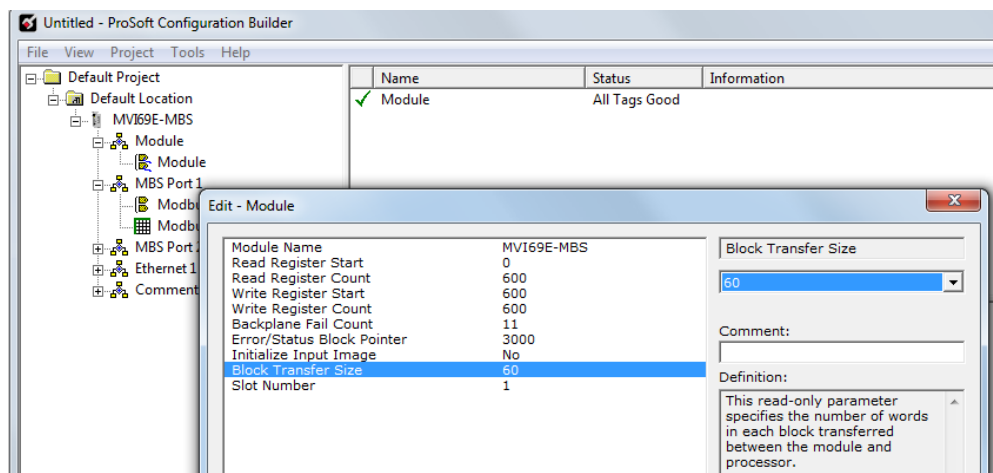
2.3.2 Creating and Exporting the .L5X File

There are two parameters in the PCB configuration that affect the format of the .L5X file that is exported. Before exporting the .L5X file to the PC/Laptop, check the *Block Transfer Size* and *Slot Number* parameters.

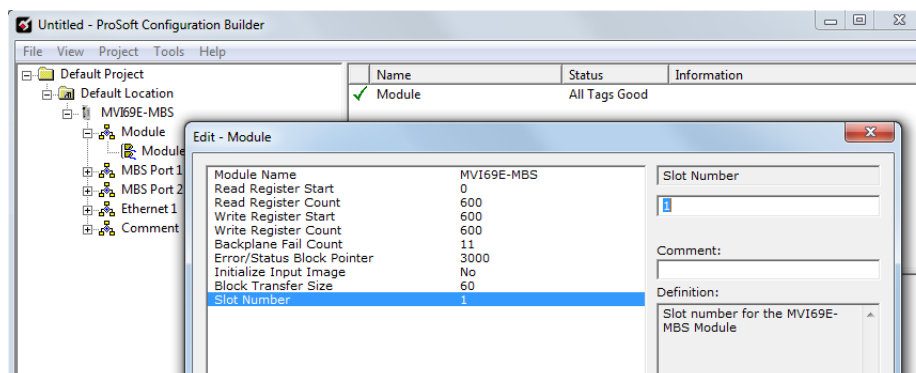
- 1 Expand the **MVI69E-MBS** icon by clicking the **[+]** symbol beside it. Similarly, expand the  icon. Double-click the  icon to open the *Edit - Module* dialog box.



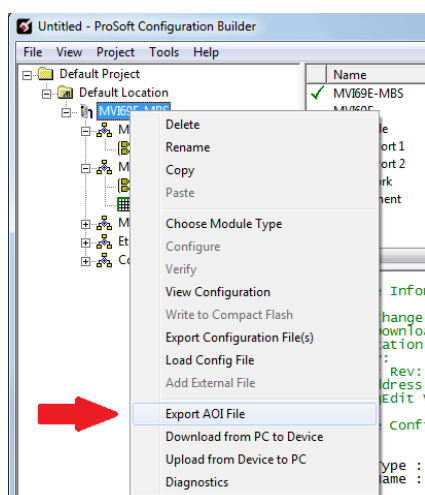
- 2 Set the *Block Transfer Size* to the desired size of the data blocks transferred between the module and processor (60, 120 or 240 words). Block transfer size information can be found starting in the section on *Normal Data Transfer* (page 67).



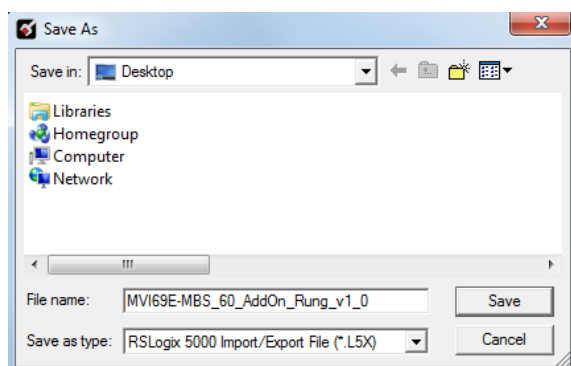
- 3 Edit the *Slot Number* indicating where the module is placed in the 1769 bus.



- 4 Click **OK** to close the *Edit – Module* dialog box. The .L5X file is now ready to be exported to the PC/Laptop.
- 5 Right-click the **MVI69E-MBS** icon in the project tree and choose **EXPORT AOI FILE**.



- 6 Save the .L5X file to the PC/Laptop in an easily found location, such as *Windows Desktop*.

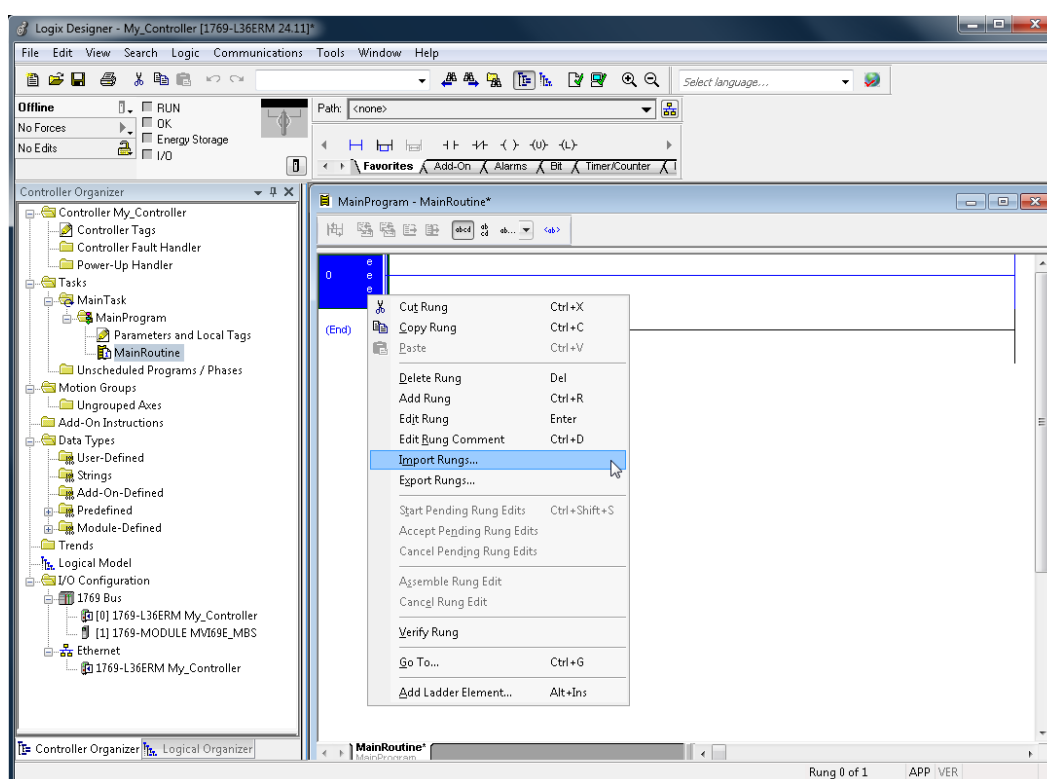


2.4 Importing the Add-On Instruction

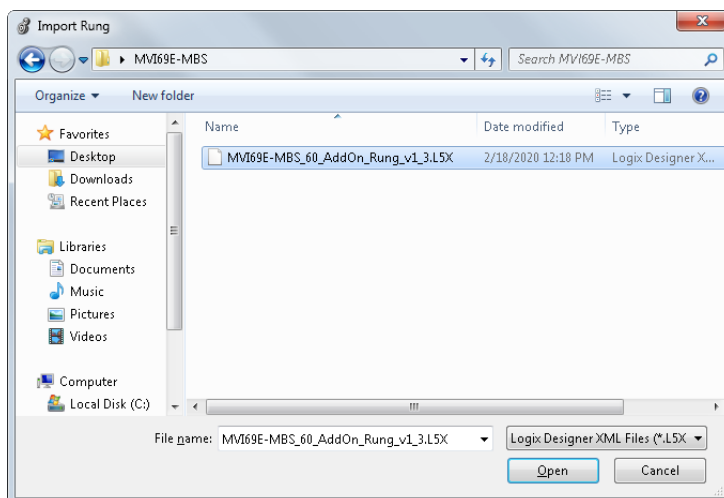
- 1 Open the application in Studio 5000.
- 2 Expand the **TASKS** folder, and expand the **MAINTASK** folder.
- 3 Expand the **MAINPROGRAM** folder. The **MAINROUTINE** contains rungs of logic. The very last rung in this routine is blank. This is where you can import the Add-On Instruction.

Note: You can place the Add-On Instruction in a different routine than the MainRoutine. Make sure to add a rung with a jump instruction (JSR) in the MainRoutine to jump to the routine containing the Add-On Instruction.

- 4 Right-click an empty rung in the routine and choose **IMPORT RUNGS**.

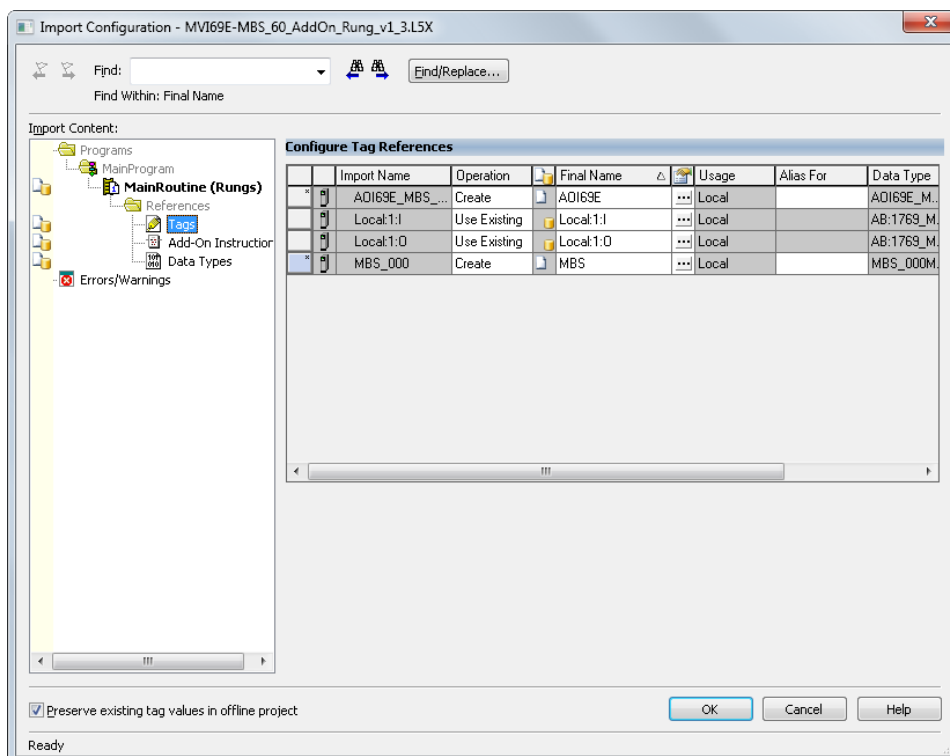


- 5 Select the **.L5X** file that you exported from PCB (*Creating and Exporting the .L5X File* (page 23)) and click **OPEN**.



- 6 This opens the *Import Configuration* dialog box. Click **TAGS** under **MAINROUTINE** to display the controller tags in the Add-On Instruction.

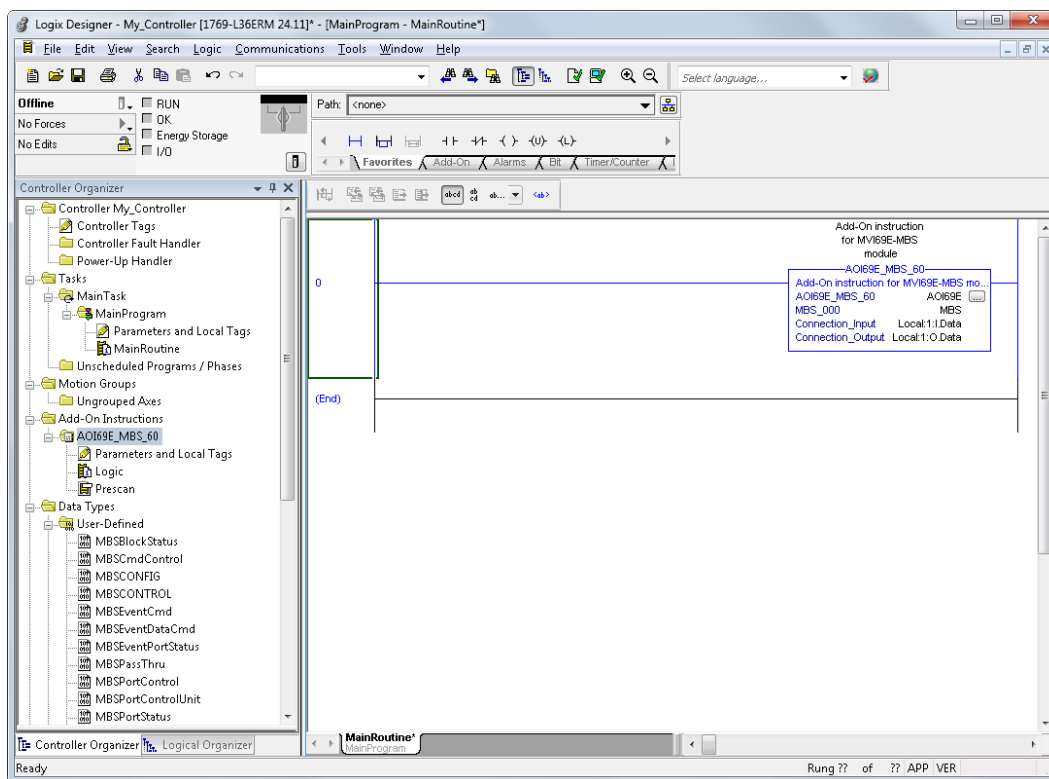
Note: If you are using RSLogix version 16 or earlier, the *Import Configuration* dialog box does not contain the *Import Content* tree.



- 7 If the module is not located in the default slot (or is in a remote rack), edit the connection input and output variables that define the path to the module in the **FINAL NAME** column (**NAME** column for RSLogix version 16 or less). For example, if your module is located in slot 3, change *Local:1:I* in the **FINAL NAME** column to *Local:3:I*. Do the same for *Local:1:O*.

Note: If your module is in Slot 1 of the local rack, this step is not required.

- 8 Click **OK** to confirm the import.
- 9 When the import is completed, the new rung with the Add-On Instruction is visible.



The procedure also imports new user-defined data types, data objects and the Add-On instruction to be used in the project with the MVI69E-MBS module.

2.5 Adding Multiple Modules in the Rack (Optional)

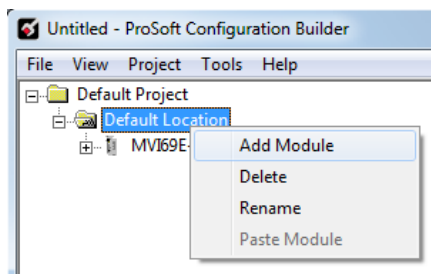
Important: This procedure is for multiple MVI69E-MBS modules running in the same CompactLogix or MicroLogix 1500-LRP rack.

You can add additional modules of the same type to the rack.

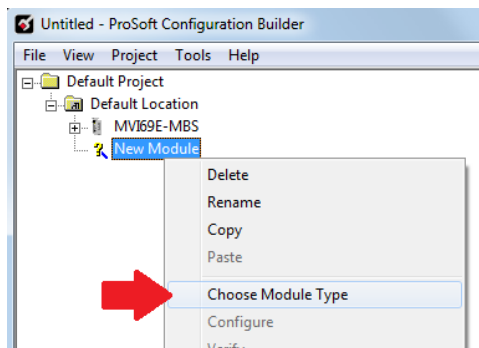
- Add a new MVI69E-MBS module to the ProSoft Configuration Builder (PCB) project.
- Export the module configuration as an L5X file.
- Add a new MVI69E-MBS to the Studio 5000 project.
- Import the .L5X file into Studio 5000 for the new module as an Add-On Instruction.

2.5.1 Adding Another Module in PCB

- 1 Start ProSoft Configuration Builder.
- 2 Right click **DEFAULT LOCATION** (which you can rename) and choose **ADD MODULE**.



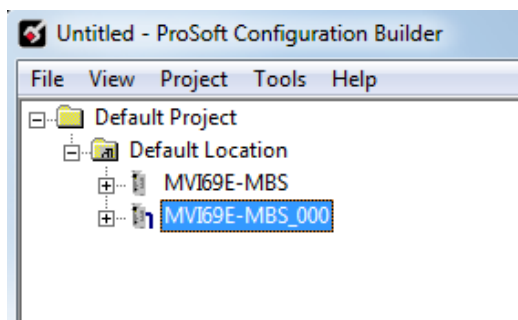
- 3 Right-click **NEW MODULE** and choose **CHOOSE MODULE TYPE**.



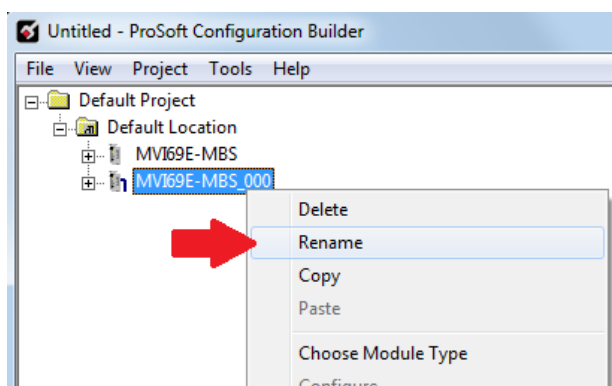
- 4 In the *Choose Module Type* dialog box, select **MVI69E** in the **PRODUCT LINE FILTER** area, and then select **MVI69E-MBS** as the **MODULE TYPE**. Click **OK**.

- 5 Select the **MVI69E-MBS** module in the tree and repeat the above steps to add a second (or more) module in the PCB project.

Note: You must give each MVI69E-MBS module a unique name. The default name on a duplicate module appends a number to the end such as **MVI69E-MBS_000**, **MVI69E-MBS_001**, etc.



- 6 You can rename the module by right clicking the module and choosing **Rename**.

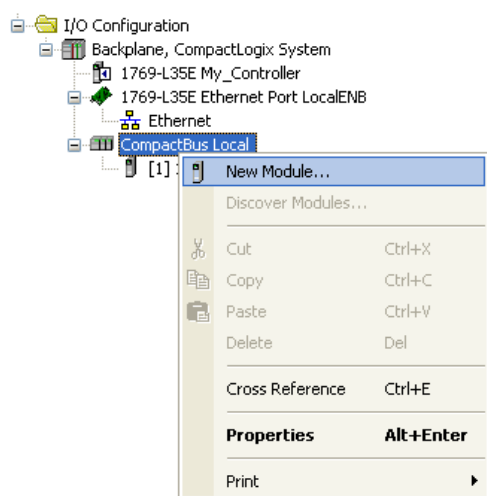


- 7 Configure the module parameters. See *Module Configuration Parameters* (page 38) and then export the AOI .L5X file for the new module (right-click the module and choose **EXPORT AOI FILE**). See *Creating and Exporting the .L5X File* (page 23).

2.5.2 Adding Another Module in Studio 5000

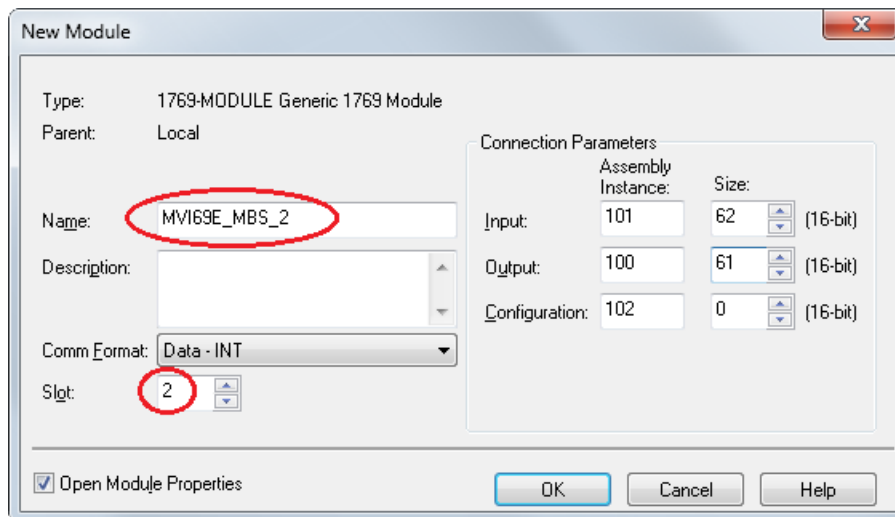
You can place multiple MVI69E-MBS modules in the same rack provided it does not exceed the power distance rating of the CompactLogix or MicroLogix 1500-LRP rack (see *System Requirements* (page 7)). Adding an additional module to the rack is similar to installing a new module; however, the name of the module must be unique.

- 1 Start Studio 5000 and open the project.
- 2 In Studio 5000, locate the **I/O CONFIGURATION** folder. Right click **COMPACTBUS LOCAL** and choose **NEW MODULE**.

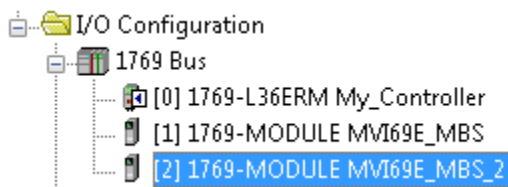


- 3 In the *Select Module Type* dialog box, select the MVI69E-MBS module.
 - If you are using an Add-On Profile (AOP), this adds the MVI69E-MBS module and configures the relevant parameters. You must be using RSLogix version 15 or later to use an AOP.
 - If using an AOP is not an option, select **GENERIC 1769 MODULE** and click **CREATE**.

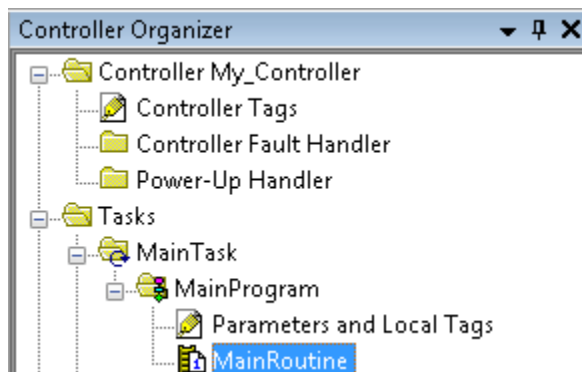
- 4 The *New Module* dialog box appears. Enter a unique name for the new module, and confirm the slot number of the new module.



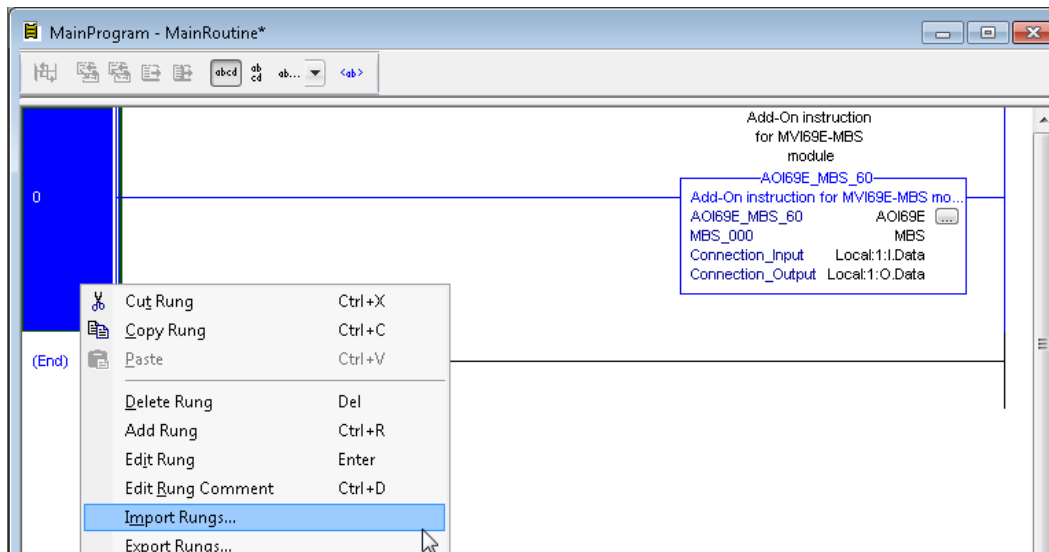
- 5 Click **OK**. The new module is now visible.



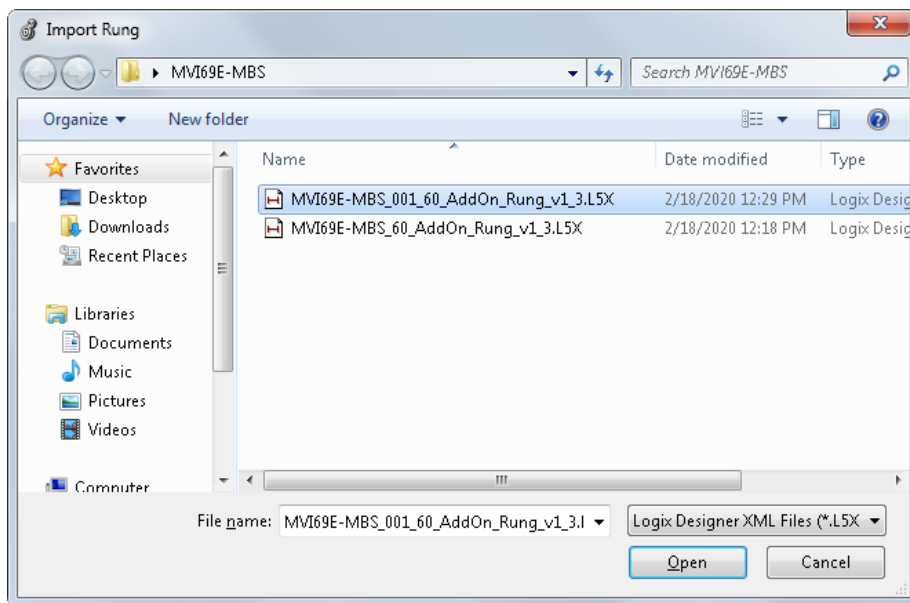
- 6 You must also import the Add-On Instruction (AOI) for the new module (see *Adding Another Module in PCB* (page 28)). In the *Controller Organizer* pane, double-click **MAINROUTINE** to open the ladder for the routine.



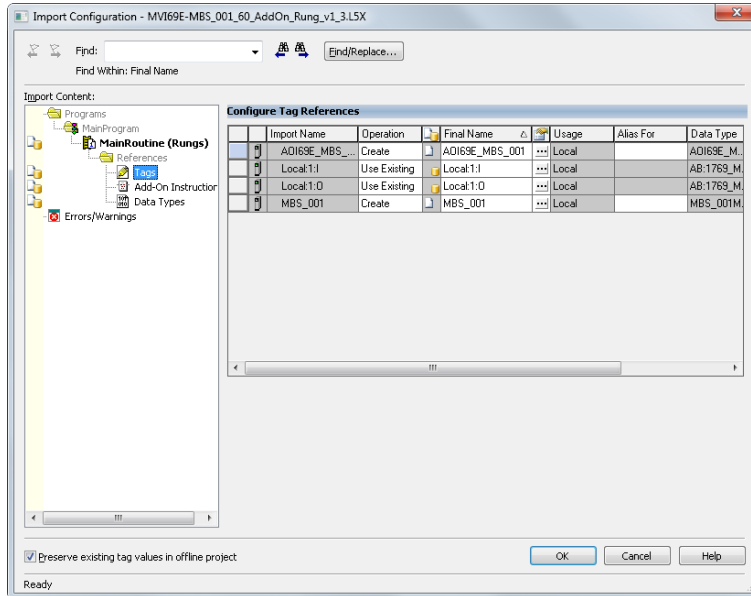
- 7 Right-click an empty rung in the routine, and then choose **IMPORT RUNGS...**



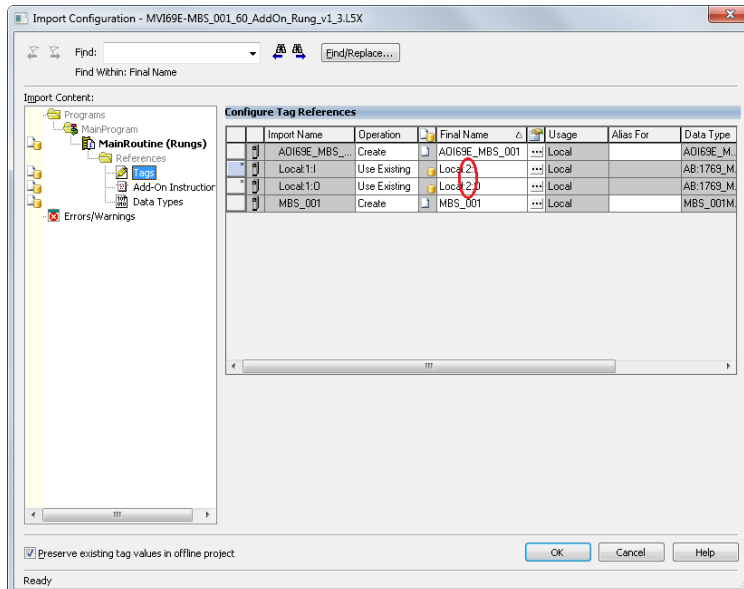
- 8 Select the .L5X file you created and exported for the new module, and click **IMPORT**. The new .L5X file has a unique filename that is specific to the new module.



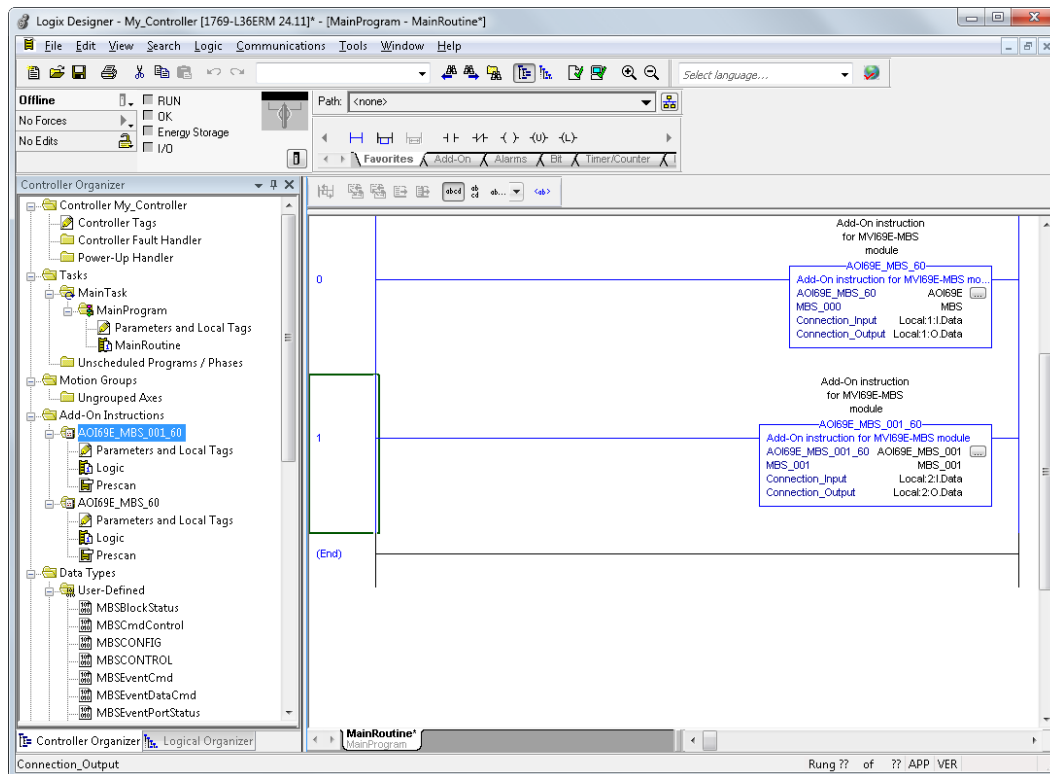
- 9 This opens the **IMPORT CONFIGURATION** dialog box. Click **TAGS** to show the controller tags in the AOI. You must edit the **FINAL NAME** column of the tags for the second module to make them unique.



- 10 Associate the I/O connection variables to the correct module in the corresponding slot number. The default values are Local:1:I and Local:1:O. You must edit these values if the card is placed in a slot location other than slot 1 (Local:1:x means the card is located in slot 1). Since the second card is placed in slot 2, change the **FINAL NAME** to Local:2:I and Local:2:O. Also, you can append a '_2' at the end of the **FINAL NAME** of 'AOI69_MBS' and 'MBS' arrays as shown below.



11 Click OK.



12 The setup procedure is now complete. Save the project. It is ready to download to the CompactLogix or MicroLogix 1500-LRP processor.

3 Configuring the MVI69E-MBS Using PCB

ProSoft Configuration Builder (PCB) provides a quick and easy way to manage module configuration files customized to meet your application needs.

Create and edit the module's configuration in ProSoft Configuration Builder. Use PCB to download the configuration file to the CompactLogix or MicroLogix 1500-LRP processor, where it is stored in the MBS.CONFIG controller tag generated by the previously exported AOI. When the MVI69E-MBS module boots up, it requests the processor to send the configuration over the backplane in special Configuration Blocks.

To create a new PCB project and export a .L5X file for the processor, see section *Generating the AOI (.L5X File) in ProSoft Configuration Builder* (page 21).

3.1 Basic PCB Functions

3.1.1 Creating a New PCB Project and Exporting an .L5X File



Please see section *Generating the AOI (.L5X File) in ProSoft Configuration Builder* (page 21).

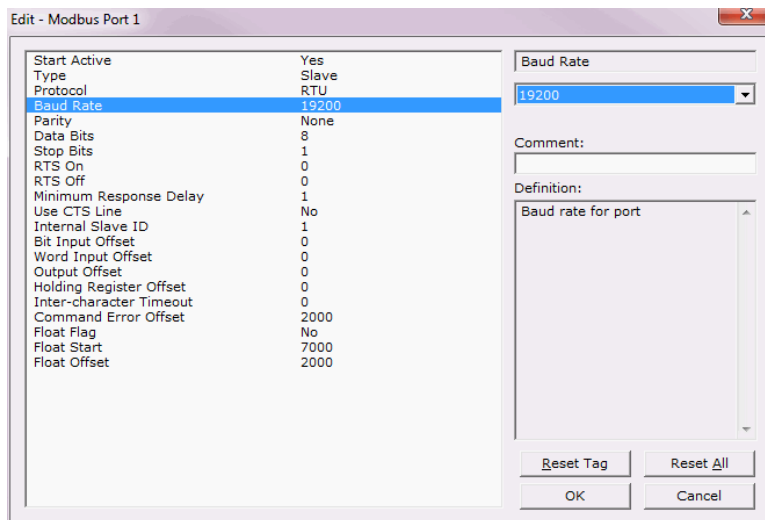
3.1.2 Renaming PCB Objects

You can rename objects such as the *Default Project* and *Default Location* folders in the tree view. You can also rename the Module icon to customize the project.


- 1 Right-click the object you want to rename and then choose **RENAME**.
- 2 Type the new name for the object and press **Enter**.

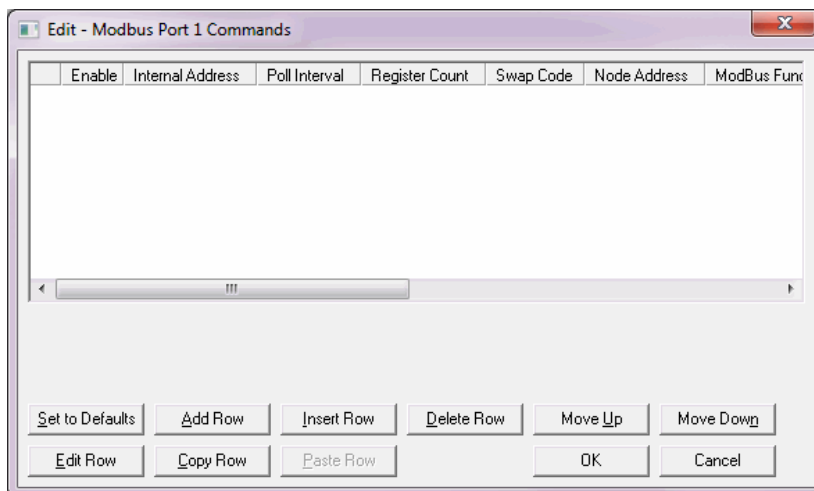
3.1.3 Editing Configuration Parameters

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

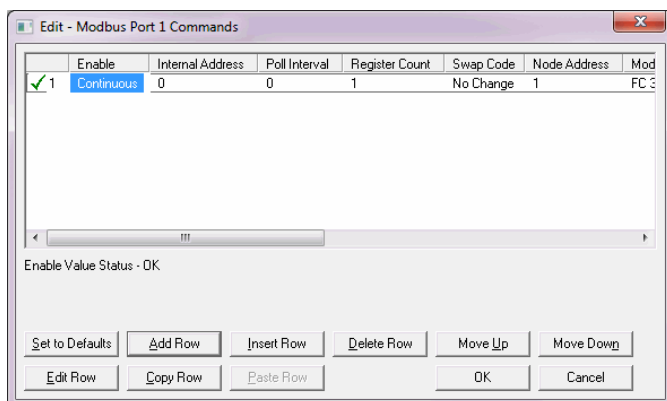


Note: Depending on the parameter, you must enter text, or a valid number, or select from a list of options.

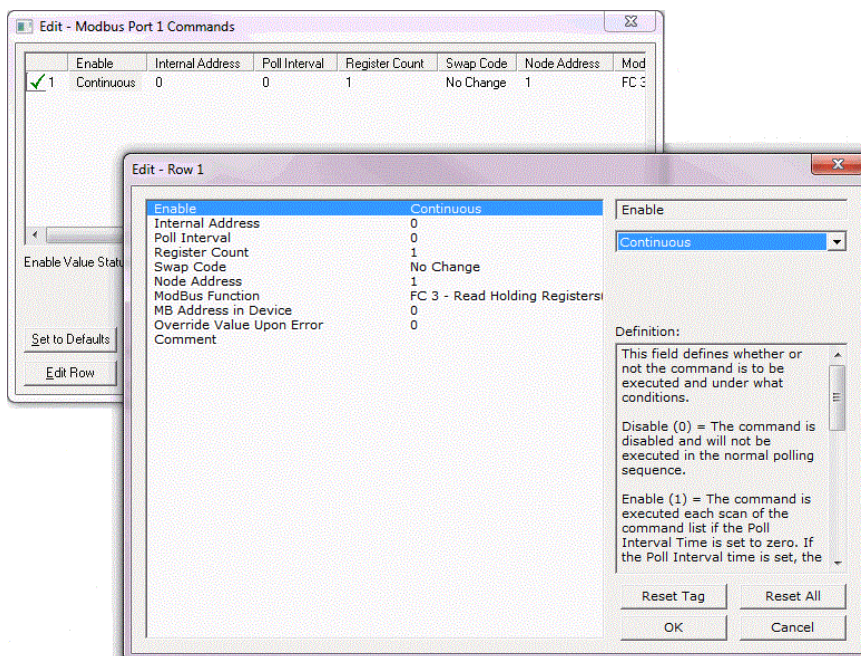
- 4 Click **OK** to save your changes.
- 5 Double-click any  icon to open an *Edit* dialog box with a table. Use this dialog box to build and edit Modbus Master commands.



- 6 To add a row to the table, click **ADD ROW**.



- 7 To edit the row, click **EDIT ROW**. This opens an *Edit* dialog box.



3.1.4 Printing a Configuration File

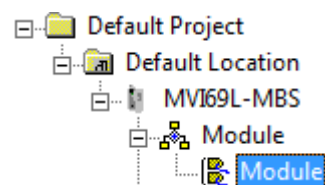
- 1 In the main PCB window, right-click the **MVI69E-MBS** icon and then choose **VIEW CONFIGURATION**.
- 2 In the *View Configuration* dialog box, click the **FILE** menu and then click **PRINT**.
- 3 In the *Print* dialog box, choose the printer to use from the drop-down list, select the printing options, and then click **OK**.

3.2 Module Configuration Parameters

3.2.1 Module Parameters

This section contains general module configuration parameters, including database allocation and backplane transfer options.

In the ProSoft Configuration Builder (PCB) tree view, double-click **MODULE**.



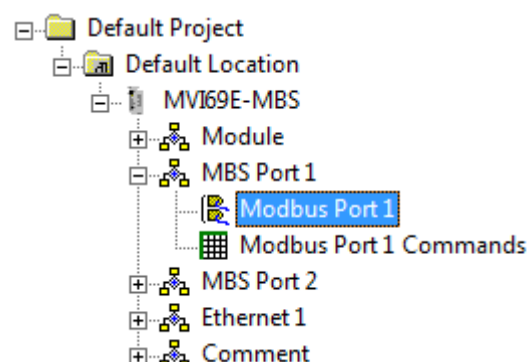
Parameter	Value	Description
Module Name	ASCII characters (max. 38)	Assigns a name to the module that can be viewed using the configuration/debug port. Use this parameter to identify the module and the configuration file.
Read Register Start	0 to 9999	Specifies the start of the Read Data area in module memory. Data in this area is transferred from the module to the processor.
Read Register Count	0 to 10,000	Specifies the size of the Read Data area.
Write Register Start	0 to 9999	Specifies the start of the Write Data area in module memory. Data in this area is transferred from the processor to the module.
Write Register Count	0 to 10,000	Specifies the size of the Write Data area.
Backplane Fail Count	0 to 65535	Specifies the number of consecutive backplane transfer failures that can occur before communications are halted.
Error/Status Block Pointer	-1 to 9939	Starting register location in the module's database for the error/status table. If a value of -1 is entered, the error/status data is not placed in the database. This data must be placed in the read data range of module memory. This data includes the module version information and all server error/status data. Refer to <i>MBS.STATUS</i> (page 63) for more information.
Initialize Input Image	Yes or No	This parameter determines if the input image data and the module's Read Register Data values are initialized with Read Register Data values from the processor. If you set the parameter to No , the Read Register Data values in the module are set to 0 upon initialization. If you set the parameter to Yes , the data is initialized with Read Register Data values from the processor. This option requires associated ladder logic to pass the data from the processor to the module.
Block Transfer Size	60, 120 or 240	Specifies the number of words in each block transferred between the module and processor.
Slot Number	1 to x	Specifies the slot in the CompactLogix or MicroLogix 1500-LRP rack for the module.

Important: The sum of the *Read Register Count* and *Write Register Count* cannot exceed 10,000 total registers. Furthermore, neither the Read Data nor the Write Data area may extend above register 9999. The Read Data and Write Data areas must not overlap.

3.2.2 Modbus Port x Parameters

This section applies to both **MBS PORT 1** and **MBS PORT 2**.

In the ProSoft Configuration Builder tree view, double-click the **MODBUS PORT X** icon.



3.2.2.1 Configuration Parameters Common to Master and Slave

Parameter	Value	Description
Start Active	Yes or No	Specifies whether the port and commands are active upon module boot-up.
Type	Master, Slave, or Slave with Pass-Through	This parameter specifies which device type the port emulates. See <i>Slave Mode</i> (page 71) for more information on Slave Pass-Through options.
Protocol	RTU or ASCII	Specifies the Modbus protocol for the port.
Baud Rate	Multiple options	Specifies the baud rate for the port.
Parity	None, Odd, Even	Specifies the type of parity error checking. All devices on this port must use the same parity setting.
Data Bits	7 or 8	Sets the number of data bits for each word used by the protocol. All devices communicating through this port must use the same number of data bits.
Stop Bits	1 or 2	Sets the number of stop bits that signal the end of a character in the data stream. For most applications, use one stop bit. For slower devices that require more time to re-synchronize, use two stop bits. All devices communicating through this port must use the same number of stop bits.
RTS On	0 to 65535 milliseconds	Sets the number of milliseconds to delay after <i>Ready To Send</i> (RTS) is asserted before data is transmitted.
RTS Off	0 to 65535 milliseconds	Sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal is set low.
Use CTS Line	Yes or No	Specifies if the <i>Clear To Send</i> (CTS) modem control line is to be used or not. If you set the parameter to No , the CTS line is not monitored. If you set the parameter to Yes , the CTS line is monitored and must be high before the module sends data. Normally, this parameter is required when half-duplex modems are used for communication (2-wire). This procedure is commonly referred to as <i>hardware handshaking</i> .
Float Flag	Yes or No	Specifies how the Slave driver responds to Function Code 3, 6, and 16 commands (read and write Holding Registers) from a remote Master when it is moving 32-bit floating-point data. Note: Most applications using floating-point data do not need this parameter enabled.

		<p>If the remote Master expects to receive or sends one complete 32-bit floating-point value for each count of one (1), then set this parameter to YES. When set to YES, the Slave driver returns values from two consecutive 16-bit internal memory registers (32 total bits) for each count in the read command, or receive 32-bits per count from the Master for write commands.</p> <p>Example: Count = 10, Slave driver sends 20 16-bit registers for 10 total 32-bit floating-point values.</p> <p>If, however, the remote Master sends a count of two (2) for each 32-bit floating-point value it expects to receive or send, or, if you do not plan to use floating-point data in your application, then set this parameter to No, which is the default setting.</p> <p>You also need to set the <i>Float Start</i> and <i>Float Offset</i> parameters to appropriate values whenever the <i>Float Flag</i> parameter is set to YES.</p>
Float Start	0 to 9998	<p>Defines the first register of floating-point data. All requests with register values greater than or equal to this value is considered floating-point data requests. This parameter is only used if the <i>Float Flag</i> is enabled. For example, if you enter a value of 7000, all requests for registers 7000 and above are considered as floating-point data.</p>
Float Offset	0 to 9998	<p>Defines the start register for floating-point data in the internal database. This parameter is used only if the <i>Float Flag</i> is enabled. For example, if you set the <i>Float Offset</i> value to 3000 and the float start parameter to 7000, data requests for register 7000 use the internal Modbus register 3000.</p>

3.2.2.2 Additional Configuration Parameters as Master

The *Type* parameter must be **MASTER** to configure these parameters. See *Configuration Parameters Common to Master and Slave* (page 39).

Parameter	Value	Description
Response Timeout	0 to 65535 milliseconds	Specifies the command response timeout period in 1 millisecond increments. This is the time that a port configured as a Master waits for a response from the addressed slave before re-transmitting the command (Retries) or skipping to the next command in the Command List. The value to specify depends on the communication network used and the expected response time (plus or minus) of the slowest device on the network.
Retry Count	0 to 10	Specifies the number of times a command is retried if it fails.
Minimum Command Delay	0 to 32767 milliseconds	Specifies the number of milliseconds to wait between receiving the end of a slave's response to the most recently transmitted command and the issuance of the next command. You can use this parameter to place a delay after each command to avoid sending commands on the network faster than the slaves can receive them. This parameter does not affect retries of a command, as retries are issued when a command failure is recognized.
Error Delay Counter	0 to 60000	Specifies the number of poll attempts to be skipped before trying to re-establish communications with a slave that has failed to respond to a command within the time limit set by the <i>Response Timeout</i> parameter. After the slave fails to respond, the master skips sending commands that should have been sent to the slave until the number of skipped commands matches the value entered in this parameter. This creates a sort of <i>slow poll</i> mode for slaves that are experiencing communication problems.
Inter-character Timeout	0 to 65535 milliseconds	Specifies a time delay in milliseconds to be added to the 3.5 character time delay used by the module to recognize the end of a message. Certain applications may require validation of Modbus messages with more than 3.5 character time between consecutive bytes (example: modem applications). A value of 0 causes the default end of message delay to be used.
Command Error Offset	-1 to 9998	Sets the address in the module's database where the command error data is placed. If the value is set to -1, the data is not transferred to the database. The valid range of values for this parameter is -1 to 4899. For example, if this parameter is configured for 1000, the command errors are copied to the database as follows: 1000: error code for command 0 1001: error code for command 1 ... An error code of 0 means that the command was successfully sent (no error).

3.2.2.3 Additional Configuration Parameters as Slave

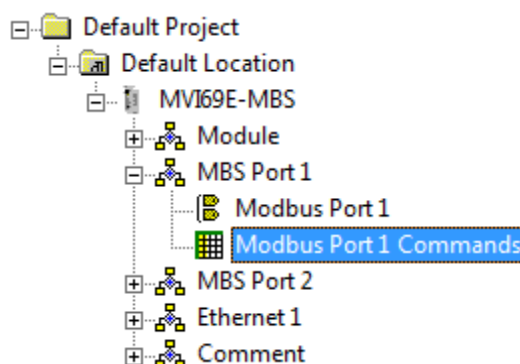
The *Type* parameter must be **SLAVE** or **PASSTHRU SLAVE** to configure these parameters. See *Configuration Parameters Common to Master and Slave* (page 39).

Parameter	Value	Description
Minimum Response Delay	0 to 65535 milliseconds	Sets the number of milliseconds to wait before responding to a command received on the port from a remote Master. This delay is sometimes required to accommodate slower Master devices.
Internal Slave ID	1 to 247	Defines the Slave Node Address for the internal database. All requests received by the port with this address are processed by the module. Verify that each device has a unique address on a network.
Bit Input Offset	0 to 9999	Specifies the offset address into the internal Modbus database for network requests for Modbus function 2 commands. For example, if you set the value to 150, an address request of 0 returns the value at register 150 in the database.
Word Input Offset	0 to 9998	Specifies the offset address into the internal Modbus database for network requests for Modbus function 4 commands. For example, if you set the value to 150, an address request of 0 returns the value at register 150 in the database.
Output Offset	0 to 9998	Specifies the offset address into the internal Modbus database for network requests for Modbus function 1, 5 or 15 commands. For example, if you set the value to 100, an address request of 0 corresponds to register 100 in the database.
Holding Register Offset	0 to 9998	Specifies the offset address in the internal Modbus database for network requests for Modbus function 3, 6, or 16 commands. For example, if you set the value to 50, a request for address 0 corresponds to the register 50 in the database.

3.2.3 Modbus Port x Commands

This section defines the master command list specifications for a Master port. The information in this section applies to both **MBS PORT 1** and **MBS PORT 2**.

In the ProSoft Configuration Builder tree view, double-click the **MODBUS PORT X COMMANDS** icon.



In order to interface the MVI69E-MBS with Modbus slave devices, you must create a command list. The commands in the list specify the slave device to be addressed, the function to be performed (read or write), the data area in the device to interface with and the registers in the internal database to be associated with the device data.

The Master command list supports up to 250 commands. The command list is processed from top (Command #0) to bottom.

Read commands are executed without condition. You can set write commands to execute only if the data in the write command changes (Conditional Enable). If the register data values in the command have not changed since the command was last issued, the command is not executed. You can use this feature to optimize network performance.

The MVI69E-MBS Master (and Slave) communication drivers support several data read and write commands. When a command is configured, the type of data (bit, 16-bit integer, 32-bit float, etc), and the level of Modbus support in the slave equipment needs to be considered.

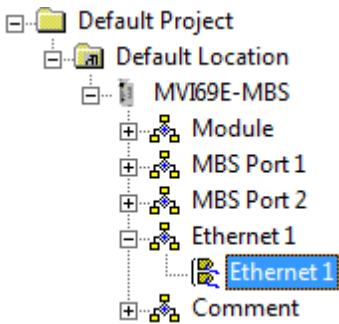
Parameter	Value	Description
Enable	0 to 4	This field defines whether the command is to be executed under certain conditions.
<p>Disabled (0) = The command is disabled and is not executed in the normal polling sequence.</p> <p>Continuous (1) = The command is executed each scan of the command list if the <i>Poll Interval</i> (see below) is set to zero. If the <i>Poll Interval</i> is set to a nonzero value, the command is executed when the interval timer expires.</p> <p>Conditional (2) = For write commands only. The command executes only if the internal data associated with the command changes.</p> <p>Bit/Word Override upon Error (3) = For read commands only. If a command error occurs, the module overrides the associated database area with the <i>Override Value Upon Error</i> parameter value.</p> <p>Float Override upon Error (4) = For read commands only. If a command error occurs, the module overrides the associated database area (2x word count) with the <i>Override Value Upon Error</i> parameter value.</p>		

Internal Address	0 to 9999 (word-level) or 0 to 159,999 (bit-level)	<p>Specifies the module's internal database register to be associated with the command.</p> <p>For Modbus functions 3, 4, 6, and 16, this parameter is interpreted as a word-level or register-level address with an allowable range of 0 to 9999.</p> <p>For Modbus functions 1, 2, 5, and 15, this parameter is interpreted as a bit-level address with an allowable range of 0 to 159,999. Note: This bit address range is available with ProSoft Configuration Builder (PCB) v4.6 or later. Previous versions have a range of 0 to 65535.</p> <p>If the command is a read function, the data read from the slave device is <i>stored</i> beginning at the module's internal database register value entered in this field. This register value must be in the Read Data area of the module's memory, defined by the <i>Read Register Start</i> and <i>Read Register Count</i> parameters in the <i>Module</i> section.</p> <p>If the command is a write function, the data to be written to the slave device is <i>sourced</i> beginning from the module's internal database register specified. This register value must come from the Write Data area of the module's memory, defined by the <i>Write Register Start</i> and <i>Write Register Count</i> parameters in the <i>Module</i> section.</p> <p>Note: When using a bit level command, you must define this field as a bit address. For example, when using function codes 1 or 2 for a Read command, you must enter a value of 160 to place the data starting at bit 0 of module memory 10. Think of it as the 160th bit of module memory (Module memory register 10 * 16 bits per register = 160). Similarly, use this formula for function codes 5 or 15 for writing bits.</p>
Poll Interval	0 to 65535 (1/10 second)	<p>Specifies the minimum interval between executions of continuous commands (<i>Enable</i> code = 1).</p> <p>Example: The parameter is entered in 1/10th of a second. Therefore, if a value of 100 is entered, the command executes no more frequently than every 10 seconds. When the command reaches the top of the command queue and 10 seconds has not elapsed, it is skipped until the poll interval has expired.</p>
Register Count	1 to 125 (words) or 1 to 2000 (coils)	<p>Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point.</p> <p>For Modbus Function Codes 1, 2 and 15, this parameter sets the number of single bit digital points (inputs or coils) to be associated with the command.</p> <p>Note: Up to 2000 coils are supported for Modbus Function Codes 1 and 2. Up to 1968 coils are supported for Modbus Function Code 15. For Modbus Function Codes 3, 4 and 16, this parameter sets the number of 16-bit registers to be associated with the command.</p>
Swap Code	0,1,2,3	<p>Defines if the data received from the Modbus slave is to be ordered differently than received from the slave device. This parameter is helpful when dealing with floating-point or other multi-register values, as there is no standard method of storage of these data types in slave devices. You can set this parameter to order the register data received in an order useful by other applications.</p> <p>No Change (0): No change is made in the byte ordering (ABCD = ABCD) Word Swap (1): The words are swapped (ABCD= CDAB)</p>

		<p>Word and Byte Swap (2): The words are swapped, then the bytes in each word are swapped (ABCD=DCBA)</p> <p>Byte Swap (3): The bytes in each word are swapped (ABCD=BADC)</p> <p>Note: Each pair of characters is a byte. Ex: AB and CD. Two pairs of characters is 16-bit register Ex: ABCD.</p>
Node Address	1 to 255 (0 = broadcast)	Specifies the Modbus slave node address on the network to be considered. Most Modbus devices only accept an address in the range of 1 to 247. If you set the value to zero, the command is a broadcast message on the network. The Modbus protocol permits broadcast commands for write operations. Do not use this node address for read operations.
Modbus Function	1,2,3,4,5,6,15,16	<p>Specifies the Modbus function to be executed by the command. These function codes are defined in the Modbus protocol.</p> <ul style="list-style-type: none"> 1 – Read Coil Status (0xxxx) 2 – Read Input Status (1xxxx) 3 – Read Holding Registers (4xxxx) 4 – Read Input Registers (3xxxx) 5 – Force (Write Single) Coil (0xxxx) 6 – Force (Write Single) Holding Register (4xxxx) 15 – Preset (Write) Multiple Coils (0xxxx) 16 – Preset (Write) Multiple Registers (4xxxx)
MB Address in Device	0 to 65535	<p>Specifies the register or digital point address offset within the Modbus slave device. The MBS Master reads or writes from/to this address within the slave. Refer to the documentation of each Modbus slave device for their register and digital point address assignments.</p> <p>Note: The value entered here does not need to include the "Modbus Prefix" addressing scheme. Also, this value is an offset of the zero-based Modbus addressing scheme.</p> <p>Example: Using a Modbus Function Code 3 to read from address 40010 in the slave, a value of '9' would be entered in this parameter. The firmware (internally) adds a '40001' offset to the value entered. This is the same for all Modbus addresses (0x, 1x, 3x, 4x).</p>
Override Value Upon Error		<p>This parameter is only applicable for <i>Enable Codes</i> 3 (Bit/Word Override) or 4 (Float Override).</p> <p>If an error occurs associated with a read command the module automatically populates the associated database area with this override value.</p>

3.2.4 Ethernet 1

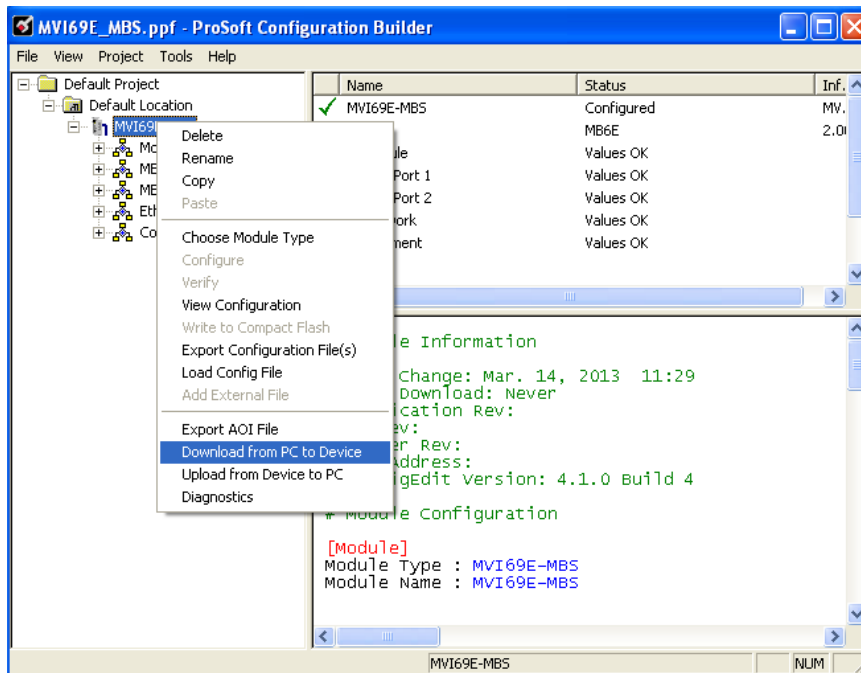
This section defines the permanent IP address, Subnet Mask, and Gateway of the module.
In the ProSoft Configuration Builder tree view, double-click the **ETHERNET 1** icon.



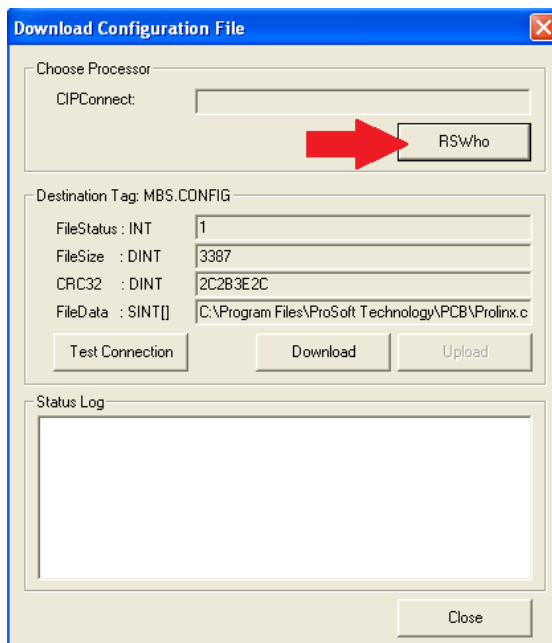
Parameter	Description
IP Address	Unique IP address assigned to the module
Netmask	Subnet mask of module
Gateway	Gateway (if used)

3.3 Downloading the Configuration File to the Processor

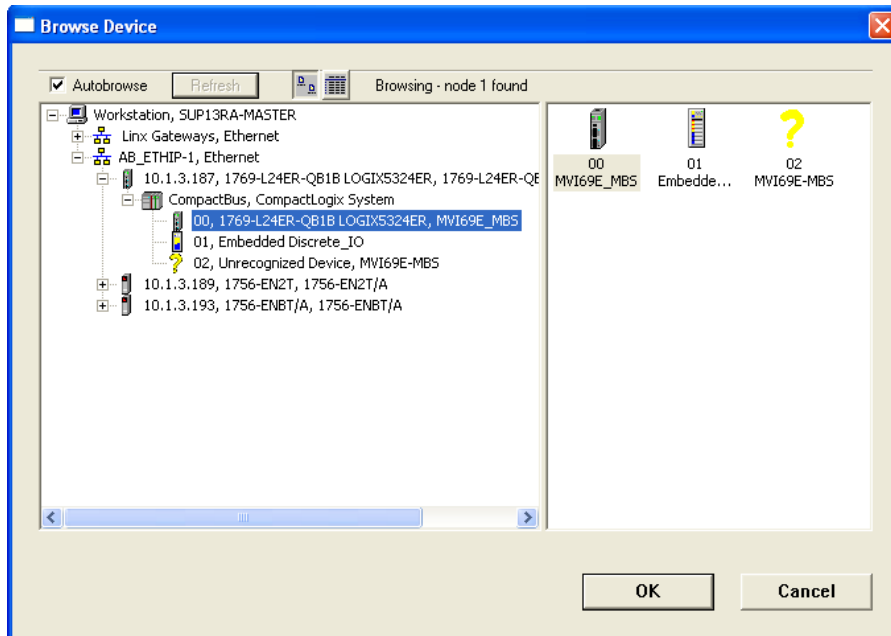
- 1 In the ProSoft Configuration Builder tree view, right-click the module icon and choose **DOWNLOAD FROM PC TO DEVICE**.



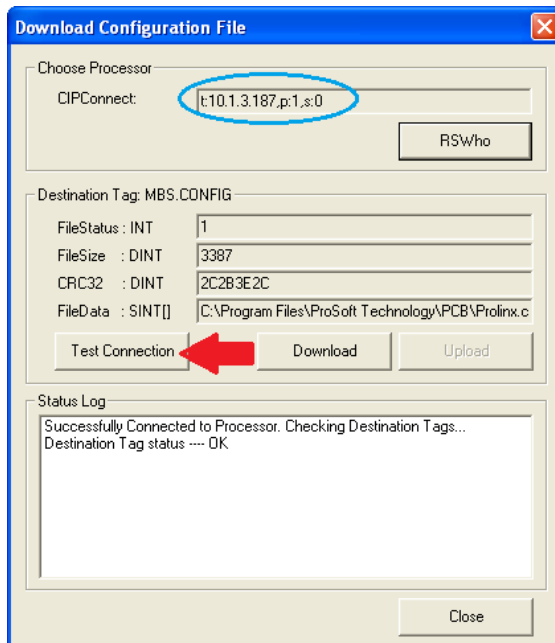
- 2 In the *Download Configuration File* dialog box, click **RSWho**.



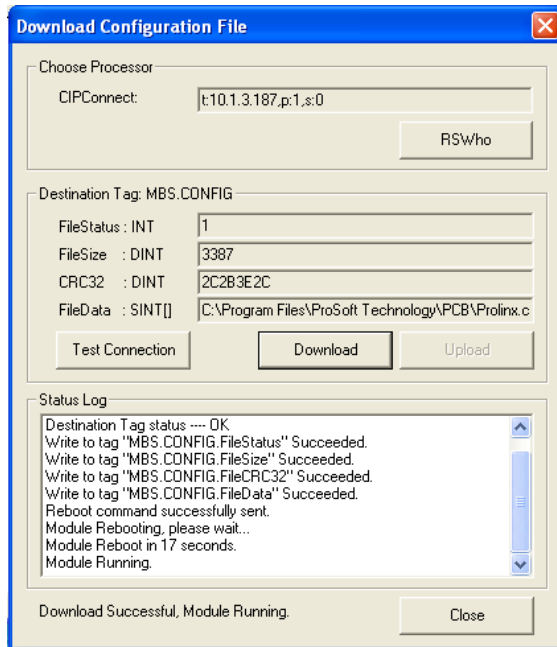
- 3 Browse to, and then highlight the CompactLogix or MicroLogix 1500-LRP processor and click **OK**.



- 4 Notice the CIPConnect path has been updated in the *Download Configuration File* dialog box. Click **TEST CONNECTION** to verify the path is active and can successfully connect to the processor.



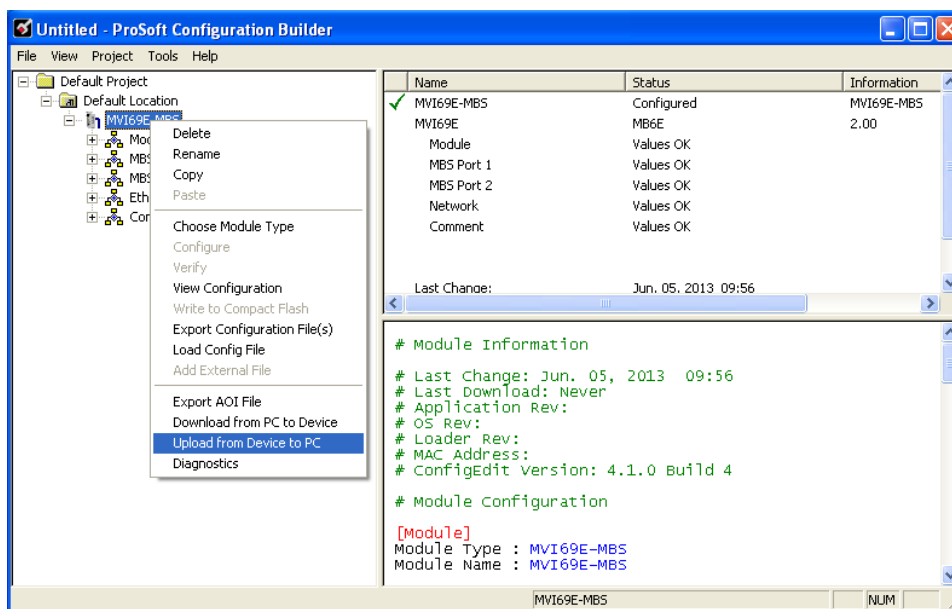
- 5 When ready, click **DOWNLOAD** to download the configuration file to the processor. Following the download process, the module is automatically rebooted.



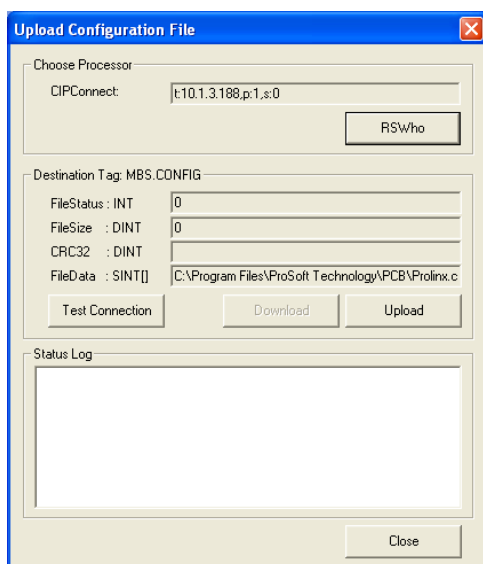
- 6 After rebooting, the ladder logic sends the configuration data from the processor to the module. When that is complete, the module starts Modbus communications.

3.4 Uploading the Configuration File from the Processor

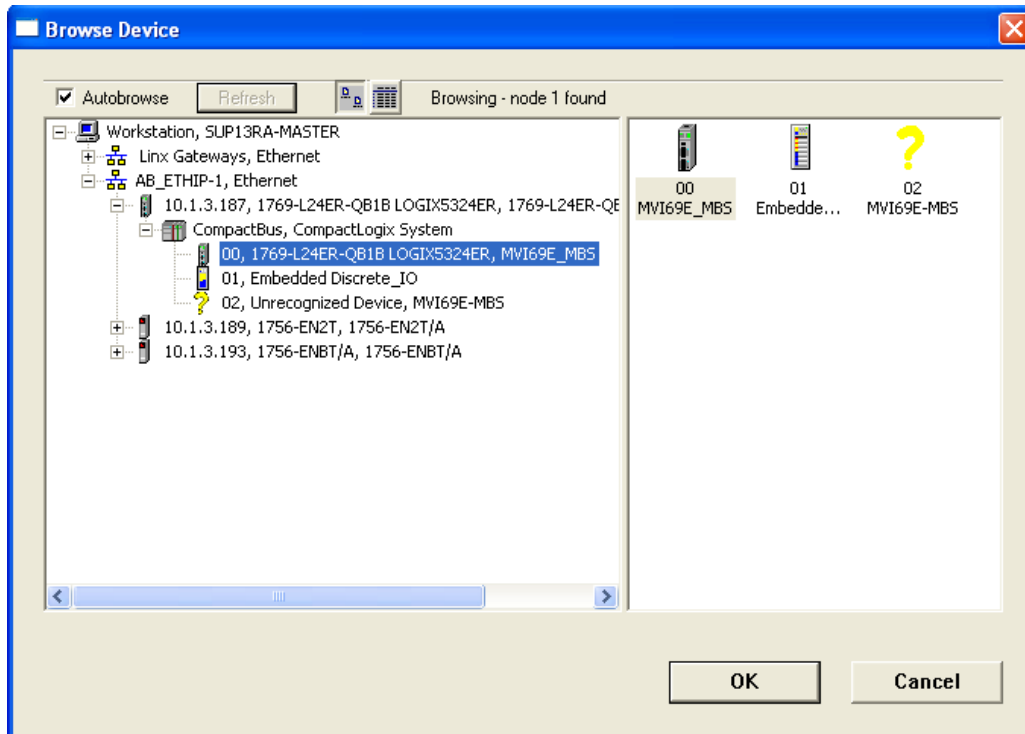
- 1 In the ProSoft Configuration Builder tree view, right-click the **MVI69E-MBS** icon and choose **UPLOAD FROM DEVICE TO PC**.



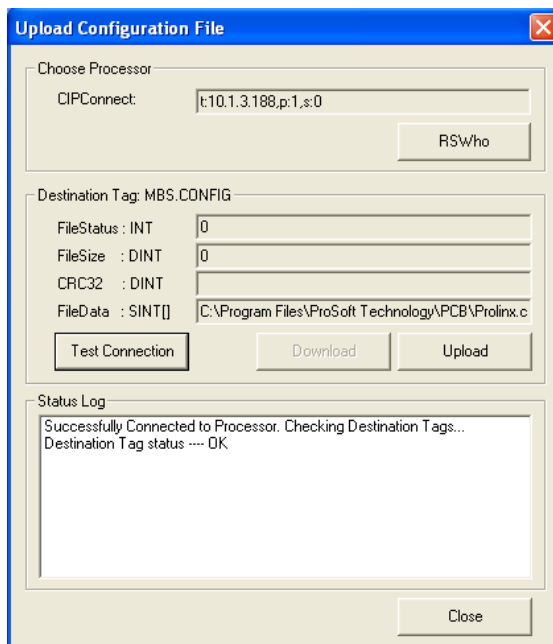
- 2 In the *Upload Configuration File* dialog box, the CIPConnect path should already be constructed if you have previously downloaded the configuration file from the same PC.



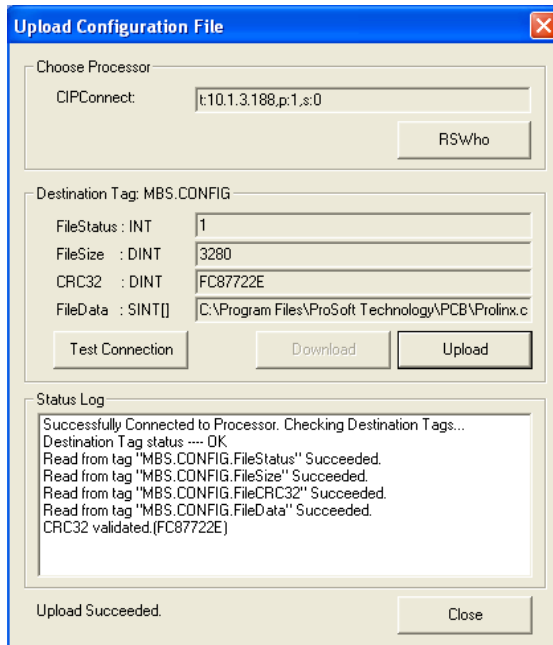
If not, click **RSWho**, browse to, then select the CompactLogix or MicroLogix 1500-LRP Processor, and click **OK**.



- 3 Click **TEST CONNECTION** to verify the path is active and can successfully connect to the processor.



- 4 When ready, click **UPLOAD**. When upload is complete, click **CLOSE**.



- 5 PCB now displays the uploaded configuration file.

4 Using Controller Tags

Controller tags are a feature of the RSLogix software and are part of the MVI69E-MBS Add-On Instruction. For information on importing the Add-On Instruction into RSLogix, see section *Importing the Add-On Instruction* (page 25).

4.1 Controller Tags

Data related to the MVI69E-MBS is stored in the ladder logic in variables called controller tags. You use controller tags to manage communication between the MVI69E-MBS module and the CompactLogix or MicroLogix 1500-LRP processor:

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

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

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

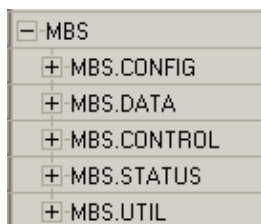
The controller tags are included in the MVI69E-MBS Add-On Instruction ladder logic. After you import the Add-On Instruction, you can find the controller tags in the *Controller Tags* subfolder, located in the *Controller* folder in the *Controller Organizer* pane of the main Studio 5000 window. This controller tag structure is arranged as a tree structure. Individual controller tags are found at the lowest level of the tree structure. Each individual controller tag is defined to hold data of a specific type, such as integer or floating-point data.

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

The MVI69E-MBS Add-On Instruction is extensively commented to provide information on the purpose and function of each user-defined data type and controller tag. For most applications, the Add-On Instruction works without needing any modification.



4.1.1 MVI69E-MBS Controller Tags

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



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

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

Scope:	 MVI69_MBS	Show:	All Tags	 Y. E...		
	Name	Value	Force	Style	Data Type	Description
	+ AOI69_MBS	{ ... }	{ ... }		AOI69_MBS_60	Add-On instruc...
	+ Local:1:C	{ ... }	{ ... }		AB:1769_MODULE:C:0	
	+ Local:1:I	{ ... }	{ ... }		AB:1769_MODULE_INT...	
	+ Local:1:O	{ ... }	{ ... }		AB:1769_MODULE_INT...	
	- MBS	{ ... }	{ ... }		MBSModuleDef	Main module d...
	+ MBS.CONFIG	{ ... }	{ ... }		MBSCONFIG	Main module d...
	- MBS.DATA	{ ... }	{ ... }		MBSDATA	Main module d...
	+ MBS.DATA.ReadData	{ ... }	{ ... }	Deci...	INT[600]	Main module d...
	+ MBS.DATA.WriteData	{ ... }	{ ... }	Deci...	INT[600]	Main module d...
	+ MBS.CONTROL	{ ... }	{ ... }		MBSCONTROL	Main module d...
	+ MBS.STATUS	{ ... }	{ ... }		MBSSTATUS	Main module d...
	+ MBS.UTIL	{ ... }	{ ... }		MBSUTIL	Main module d...

The controller tags in the Add-On Instruction are commented in the **DESCRIPTION** column.

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

4.2 User-Defined Data Types (UDTs)

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

The Add-On Instruction for the MVI69E-MBS module has pre-defined UDTs. You can find them in the *User-Defined* subfolder, located in the *Data Types* folder in the *Controller Organizer* pane of the main RSLogix window. Like the controller tags, the UDTs are organized in a multiple-level tree structure.

4.2.1 MVI69E-MBS User-Defined Data Types

Twenty different UDTs are defined for the MVI69E-MBS Add-On Instruction. The main UDT, *MBSMODULEDEF*, contains all the data types for the module and was used to create the main controller tag structure, *MBS*. There are five UDTs one level below *MBSMODULEDEF*. These lower-level UDTs were used to create the *MBS.CONFIG*, *MBS.DATA*, *MBS. CONTROL*, *MBS.STATUS*, and *MBS.UTIL* controller tag structures.

Name: MBSModuleDef

Description: Main module definition

Members: Data Type Size: 69428 byte(s)

	Name	Data Type	Style	Description	External Access
+	CONFIG	MBSCONFIG		Configuration file	Read/Write
+	DATA	MBSDATA		Database data	Read/Write
+	CONTROL	MBSCONTROL		Special tasks request	Read/Write
+	STATUS	MBSSTATUS		Status	Read/Write
+	UTIL	MBSUTIL		Tags used for internal	Read/Write

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

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

Name:

Description:

Main module definition

Members: Data Type Size: 69428 byte(s)

Name	Data Type	Style	Description	External Access
[-] CONFIG	MBSCONFIG		Configuration file	Read/Write
[-] DATA	MBSDATA		Database data	Read/Write
[-] ReadData	INT[600]	Decimal	Read from the module	Read/Write
[-] WriteData	INT[600]	Decimal	Write to the module	Read/Write
[-] CONTROL	MBSCONTROL		Special tasks request	Read/Write
[-] STATUS	MBSSTATUS		Status	Read/Write
[-] UTIL	MBSUTIL		Tags used for internal	Read/Write

top

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

The UDTs are commented in the **DESCRIPTION** column.

Tip: If more than 600 words of Read or Write Data are needed, the *MBS.DATA.ReadData* and *MBS.DATA.WriteData* controller tag arrays can be expanded. Simply edit the size of the *ReadData* or *WriteData* integer array in the **DATA TYPE** column of the MBSDATA UDT. In the example below, the *ReadData* array size has been changed to 2000. Save and download the ladder program for this change to take effect.

Name:

Description:

Module database tags

Members: Data Type Size: ?? byte(s)

Name	Data Type	Style	Description	External Access
* ReadData	INT[2000]	...	Read from the module	Read/Write
WriteData	INT[600]	Decimal	Write to the module	Read/Write

top

4.3 MBS Controller Tag Overview

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

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

4.3.1 MBS.CONFIG

When ProSoft Configuration Builder (PCB) downloads the configuration file from the PC to the processor, the processor stores the configuration file data in the *MBS.CONFIG.FileData* array. Its CRC is also included in this array.

You cannot edit this array directly. You must use PCB to edit the module configuration since PCB calculates a unique CRC to protect data integrity. Any change to the configuration parameters directly in this array will not match the calculated CRC.

Tag Name	Description
MBS.CONFIG.FileData	This parameter contains the MBS configuration data after it has been downloaded from PCB. It is displayed in ASCII format. Note: MBS configuration changes cannot be made directly in this array; the configuration must be downloaded with PCB.
MBS.CONFIG.FileSize	Configuration file size (<i>MBS.CONFIG.FileData</i> array) in bytes.
MBS.CONFIG.FileCRC32	CRC checksum of the configuration file stored in the array.
MBS.CONFIG.FileStatus	Configuration file status. 0 = No file present, 1 = File present

4.3.2 MBS.DATA

This structure contains the Read Data and Write Data arrays for processor-to-module communication.

Tag Name	Description
MBS.DATA.ReadData	Data area copied from the module to the processor. This array stores the Modbus data coming into the module from the Modbus network.
MBS.DATA.WriteData	Data area copied from the processor to the module. This array stores the outgoing data sent from the module to the Modbus network.

4.3.3 MBS.CONTROL

This array handles special tasks requested by the processor.

4.3.3.1 MBS.CONTROL.PortControl

This array allows port commands to be controlled by the processor.

Tag Name	Range	Description
MBS.CONTROL.PortControl. Set	0 or 1	Sends Port Control to module
MBS.CONTROL.PortControl. Get	0 or 1	Reads Port Control from module
MBS.CONTROL.PortControl. Portx	n/a	Definition of Port x Control
MBS.CONTROL.PortControl. Portx.Active	0 or 1	Port Control: Disable = 0, Enable = 1
MBS.CONTROL.PortControl. Portx.CmdEnableBits[x]	0 or 1	Index of command to be controlled. Example: Command 20 in port 1 command list can be controlled at <i>CmdEnableBits[1].3</i> . This is the 20 th bit offset.

4.3.3.2 MBS.CONTROL.CmdControl

This array allows the processor to dynamically enable configured commands for execution.

Tag Name	Range	Description
MBS.CONTROL.CmdControl. CmdControlTrigger	0 or 1	One-shot command control: Disable = 0, Enable = 1
MBS.CONTROL.CmdControl. NumberOfCommands	0 to 6	Total number of commands to be executed via Command Control
MBS.CONTROL.CmdControl. PortNumber	1 or 2	Port number to be associated with Command Control function
MBS.CONTROL.CmdControl. CommandIndex[x]	0 or 249	Command Index of port command [x] to be enabled. Up to 6 command indexes can be populated at a time.

4.3.3.3 MBS.CONTROL.EventCmd_DBData

This array allows the processor to dynamically build Modbus commands with data associated to the MBS database. This feature is meant for periodic execution such as resetting the clock and zeroing-out counters.

Tag Name	Range	Description
MBS.CONTROL.EventCmd_DB Data.EventCmdTrigger	0 or 1	Toggle to send Event Command. 0 = Disable, 1 = Enable
MBS.CONTROL.EventCmd_DB Data.PortNumber	1 or 2	Port number to be associated with command Control function
MBS.CONTROL.EventCmd_DB Data.SlaveID	1 to 248	Slave ID of Modbus slave
MBS.CONTROL.EventCmd_DB Data.InternalDBAddress	0 to 9999	Used only if UseModuleDBAddress=1. MVI69E database address (word address for functions 3,4,6,16 and bit address for 1,2,5,15)
MBS.CONTROL.EventCmd_DB Data.PointCount	0 to 125	Number of bit/words associated with this command.
MBS.CONTROL.EventCmd_DB Data.SwapCode	0 to 3	Swap code 0 = no swap, 1 = word swap, 2 = words & byte swap, 3 = byte swap
MBS.CONTROL.EventCmd_DB Data.ModbusFunctionCode	-	Modbus function code (1,2,3,4,5,6,15, or 16)
MBS.CONTROL.EventCmd_DB Data.DeviceDBAddress	0 to 9999	Modbus address of the target slave database
MBS.CONTROL.EventCmd_DB Data.EventCmdStatusReturned	-	Event status returned by the module

4.3.3.4 MBS.CONTROL.EventCmd_ProcessorData

This array allows the processor to dynamically build Modbus commands with processor data. This feature is meant for periodic execution such as resetting the clock and zeroing-out counters.

Tag Name	Range	Description
MBS.CONTROL.EventCmd_Processor Data.CmdTrigger	0 or 1	Toggle to send Event Command. 0 = Disable, 1 = Enable
MBS.CONTROL.EventCmd_Processor Data.GetStatusTrigger	0 or 1	Toggle to retrieve event status. 0 = Disable, 1 = Enable
MBS.CONTROL.EventCmd_Processor Data.PortNumber	1 or 2	Port number to be associated with command
MBS.CONTROL.EventCmd_Processor Data.SlaveAddress	1 to 248	Slave ID of Modbus slave
MBS.CONTROL.EventCmd_Processor Data.ModbusFunctionCode	-	Modbus function code (5,6,15, or 16)
MBS.CONTROL.EventCmd_Processor Data.DeviceDBAddress	0 to 9999	Modbus address of the target slave database
MBS.CONTROL.EventCmd_Processor Data.PointCount	0 to 125	Number of bit/words associated with this command.
MBS.CONTROL.EventCmd_Processor Data.Data[x]	0 to 49	Data values to be sent to the slave
MBS.CONTROL.EventCmd_Processor Data.EventCmdStatusReturned	-	Command status
MBS.CONTROL.EventCmd_Processor Data.PortxStatus	-	Port x Status array
MBS.CONTROL.EventCmd_Processor Data.PortxStatus.Status	-	Status code. See <i>Communication Error Codes</i> (page 110).
MBS.CONTROL.EventCmd_Processor Data.PortxStatus.LastError	-	Last error code

4.3.3.5 MBS.CONTROL.SlavePoll

This array allows the processor to enable, disable and retrieve status for slaves.

Tag Name	Range	Description
MBS.CONTROL.SlavePoll.Portx	-	Port x slave polling control.
MBS.CONTROL.SlavePoll.Portx. EnableSlaves	0 or 1	Slave Poll request. 0 = Disable, 1 = Enable Note: The AOI resets the tag to 0 after being enabled.
MBS.CONTROL.SlavePoll.Portx. EnableSlaveCount	1 to 60	Total number of slaves to be enabled in the <i>MBS.CONTROL.SlavePoll.Portx.EnableSlavesIDs[x]</i> array.
MBS.CONTROL.SlavePoll.Portx. EnableSlavesIDs[x]	-	The list of slave ID's to be enabled. Example: To enable slave ID's 1, 3, and 4 ; enter the following values: MBS.CONTROL.SlavePoll.Portx.EnableSlavesIDs[0] = 1 MBS.CONTROL.SlavePoll.Portx.EnableSlavesIDs[1] = 3 MBS.CONTROL.SlavePoll.Portx.EnableSlavesIDs[2] = 4 A value of ' 3 ' would be entered in the <i>MBS.CONTROL.SlavePoll.Portx.EnableSlaveCount</i> tag since there are 3 slave ID's to enable. Note: The AOI will not reset the tag array to 0 after being enabled.
MBS.CONTROL.SlavePoll.Portx. DisableSlaves	0 or 1	Triggers disable slaves request. 0 = Disable, 1 = Enable Note: The AOI resets the tag to 0 after being enabled.
MBS.CONTROL.SlavePoll.Portx. DisableSlaveCount	1 to 60	Number of slaves to be disabled.
MBS.CONTROL.SlavePoll.Portx. DisableSlavesIDs[x]	-	The list of the slave ID's to be disabled. Example: To disable slave ID's 1, 3, and 4 ; enter the following values: MBS.CONTROL.SlavePoll.Portx.DisableSlavesIDs[0] = 1 MBS.CONTROL.SlavePoll.Portx.DisableSlavesIDs[1] = 3 MBS.CONTROL.SlavePoll.Portx.DisableSlavesIDs[2] = 4 A value of ' 3 ' would be entered in the <i>MBS.CONTROL.SlavePoll.Portx.DisableSlaveCount</i> tag since there are 3 slave ID's to disable. Note: The AOI will not reset the tag array to 0 after being enabled.
MBS.CONTROL.SlavePoll.Portx. GetSlavesStatus	0 or 1	Triggers request to read slave status. 0 = Disable, 1 = Enabled
MBS.CONTROL.SlavePoll.Portx. SlavesStatus[x]	-	Data array with status.

4.3.3.6 MBS.CONTROL.Time

This array allows the processor to get or set module time.

Tag Name	Range	Description
SetTime	0 or 1	Sends the PLC time to the module 0 = Disable, 1 = Enable
GetTime	0 or 1	Retrieves the time from the module to PLC 0 = Disable, 1 = Enable
Year	0 to 9999	Four digit year value. Example: 2015
Month	1 to 12	Month
Day	1 to 31	Day
Hour	0 to 23	Hour
Minute	0 to 59	Minute
Second	0 to 59	Second
Milliseconds	0 to 999	Millisecond

4.3.3.7 MBS.CONTROL.GetStatus

This tag allows the processor to retrieve status from the module.

Tag Name	Range	Description
GetStatus	0 or 1	Triggers status retrieval from the module 0 = Disable, 1 = Enable

4.3.3.8 MBS.CONTROL.ResetStatus

This tag allows the processor to reset the module status counters.

Tag Name	Range	Description
ResetStatus	0 or 1	Triggers module status counter reset 0 = Disable, 1 = Enable

4.3.3.9 MBS.CONTROL.ColdBoot

This tag allows the processor to Coldboot the module (full reboot).

Tag Name	Range	Description
ColdBoot	0 or 1	Triggers a cold boot of the module 0 = Disable, 1 = Enable

4.3.3.10 MBS.CONTROL.WarmBoot

This tag allows the processor to Warmboot the module (driver reboot).

Tag Name	Range	Description
WarmBoot	0 or 1	Triggers a warm boot the module 0 = Disable, 1 = Enable

4.3.4 MBS.STATUS

This array contains status data for the module.

Tag Name	Description
MBS.STATUS.PassCnt	Program cycle counter – this value is incremented each time a complete program cycle occurs in the module
MBS.STATUS.Product	Product code
MBS.STATUS.Rev	Firmware revision level number
MBS.STATUS.OP	Operating level number
MBS.STATUS.Run	Run number
MBS.STATUS.PortxStats	Port x status
MBS.STATUS.PortxStats.CmdListReq	Total number of requests made from this port to slave devices on the network
MBS.STATUS.PortxStats.CmdListResp	Total number of slave response messages received on the port
MBS.STATUS.PortxStats.CmdListErr	Total number of command errors processed on the port. These errors could be due to a bad response or command
MBS.STATUS.PortxStats.PortReq	Total number of messages sent out of the port
MBS.STATUS.PortxStats.PortResp	Total number of messages received on the port
MBS.STATUS.PortxStats.PortErrSent	Total number of message errors sent out of the port.
MBS.STATUS.PortxStats.PortErrRec	Total number of message errors received on the port
MBS.STATUS.PortxStats.CurrErr	Not used
MBS.STATUS.PortxStats.LastErr	Not used
MBS.STATUS.Block	Backplane transfer status
MBS.STATUS.Block.Read	Total number of read blocks transferred from the module to the processor
MBS.STATUS.Block.Write	Total number of write blocks transferred from the processor to the module
MBS.STATUS.Block.Parse	Total number of blocks successfully parsed that were received from the processor
MBS.STATUS.Block.Event	Total number of event command blocks received from the processor
MBS.STATUS.Block.Cmd	Total number of command blocks received from the processor
MBS.STATUS.Block.Err	Total number of block transfer errors recognized by the module
MBS.STATUS.PortxLastErr	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.
MBS.STATUS.PortxPreviousErr	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.

4.3.5 MBS.UTIL

The array is used for internal ladder processing, and must not be modified.

Tag Name	Description
MBS.UTIL.ReadDataSizeGet	Holds Read Data array size
MBS.UTIL.WriteDataSizeGet	Holds Write Data array size
MBS.UTIL.ReadDataBlkCount	Number of Read Data blocks – this value is the <i>Read Register Count</i> divided by the <i>Block Transfer Size</i>
MBS.UTIL.WriteDataBlkCount	Number of Write Data blocks – this value is the <i>Write Register Count</i> divided by the <i>Block Transfer Size</i>
MBS.UTIL.RBTSremainder	Remainder from the <i>Read Register Count</i> divided by the <i>Block Transfer Size</i>
MBS.UTIL.WBTSremainder	Remainder from the <i>Write Register Count</i> divided by the <i>Block Transfer Size</i>
MBS.UTIL.BlockIndex	Computed block offset for data
MBS.UTIL.LastRead	Latest Read Block ID received from the module
MBS.UTIL.LastWrite	Latest Write Block ID to be sent to the module
MBS.UTIL.LastWriteInit	Latest Write Block ID used during initialization
MBS.UTIL.ConfigFile	Holds variables for configuration file transfer
MBS.UTIL.ConfigFile.WordLength	Length of configuration data to be included in block transfer
MBS.UTIL.ConfigFile.BlockCount	Not used
MBS.UTIL.ConfigFile.FileOffset	Offset in configuration file to use as a starting point for copying over configuration data
MBS.UTIL.ConnectionInputSize	Holds size of the Connection Input array
MBS.UTIL.BlockTransferSize	Size of the backplane transfer blocks
MBS.UTIL.SlotNumber	Slot number of the module in the rack
MBS.UTIL.EventBlockID	Holds Block ID for Event Command
MBS.UTIL.EventCmdPending	Keeps an Event Command message from being sent to the module before the previous Event Command is completed
MBS.UTIL.PollStatusOffset	Offset in slave status data array to use as a starting point for copying over slave status data
MBS.UTIL.CmdsAddedToQueue	Number of Command Control messages added to the command queue
MBS.UTIL.CmdControlBlockID	Holds Block ID for Command Control
MBS.UTIL.CmdCntrlPending	Keeps a Command Control message from being sent to the module before the previous Command Control is completed
MBS.UTIL.EventDataCmdPending	Keeps an Event Command with Data message from being sent to the module before the previous Event Command with Data is completed
MBS.UTIL.BootTimer	Timer used to clear both cold and warm boot requests
MBS.UTIL.PassThru[] Array	Holds variables used for processing pass-through messages

5 MVI69E-MBS Backplane Data Exchange

5.1 General Concepts of the MVI69E-MBS Data Transfer

The MVI69E-MBS module uses ladder logic to communicate with the CompactLogix or MicroLogix 1500-LRP processor across the backplane. The ladder logic handles the module data transfer, configuration data transfer, special block handling, and status data receipt.

The following topics describe several concepts that are important for understanding the operation of the MVI69E-MBS module. The following is the order of operations on power-up:

- 1) The module begins the following logical functions:
 - Initialize hardware components
 - Initialize CompactLogix or MicroLogix 1500-LRP backplane driver
 - Test and clear all RAM
- 2) Read configuration from the CompactLogix or MicroLogix 1500-LRP processor through ladder logic
- 3) Allocate and initialize Module Register space
- 4) Enable Modbus application port(s)

After the module has received the module configuration, the module begins communicating with other devices on the Modbus network, depending on the Modbus configuration of the module.

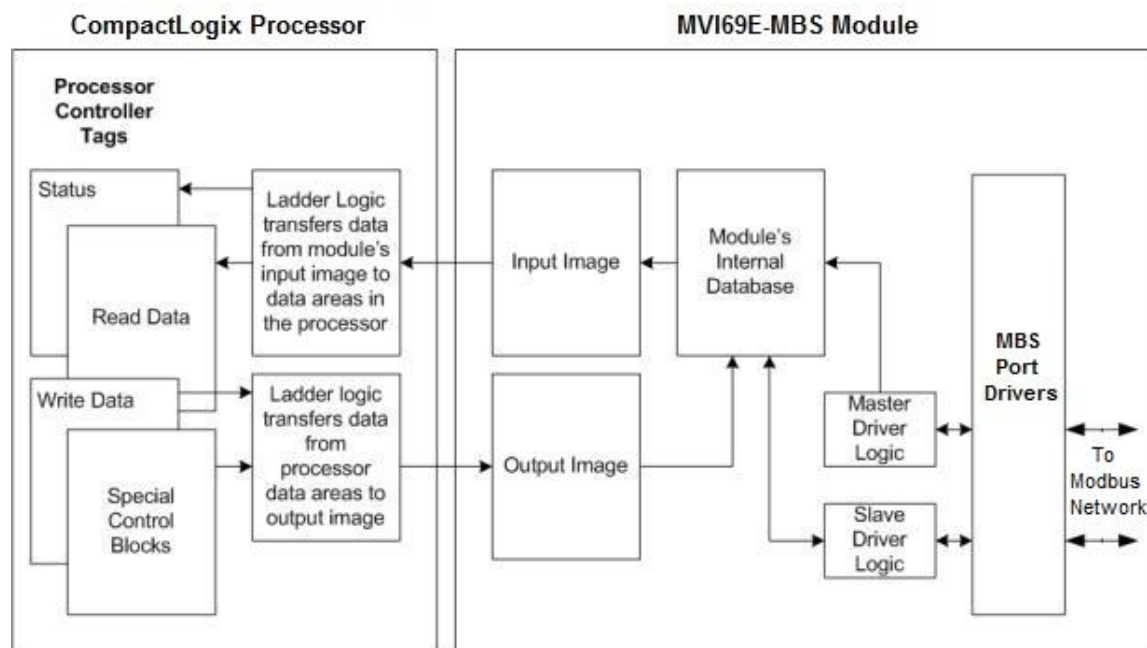
5.2 Backplane Data Transfer

The MVI69E-MBS module communicates directly over the CompactLogix or MicroLogix 1500-LRP backplane. Data is paged between the module and the CompactLogix or MicroLogix 1500-LRP 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 that you define for the module and the communication load on the module. Typical updates are in the range of 1 to 10 milliseconds per block of information.

This bi-directional data transfer is accomplished by the module filling in data in the module's input image to send to the processor. Data in the input image is placed in the Controller Tags in the processor by the ladder logic. The input image for the module may be set to 62, 122, or 242 words depending on the block transfer size parameter set in the configuration file. This data area permits fast throughput of data between the module and the processor. Applications that require smaller amounts of data or faster update times, such as ControlNet networks, will benefit from smaller block transfer sizes.

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

The following illustration shows the data transfer method used to move data between the CompactLogix or MicroLogix 1500-LRP processor, the MVI69E-MBS module and the Modbus Network.



All data transferred between the module and the processor over the backplane is through the input and output images. Ladder logic in the CompactLogix or MicroLogix 1500-LRP processor interfaces 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 virtual MBS data tables with addresses from 0 to the maximum number of points for each data type.

5.3 Normal Data Transfer

5.3.1 Write Block: Request from the Processor to the Module

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

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

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

The Write Block ID is an index value that determines the location in the module's database where the data is placed.

5.3.2 Read Block: Response from the Module to the Processor

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

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

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

5.3.3 Read and Write Block Transfer Sequences

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

The Write Block ID associated with the block requests data from the processor. Under normal program operation, the module sequentially sends read blocks and requests write blocks.

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

R1W1→R2W2→R3W1→R1W2→R2W1→R3W2→R1W1→

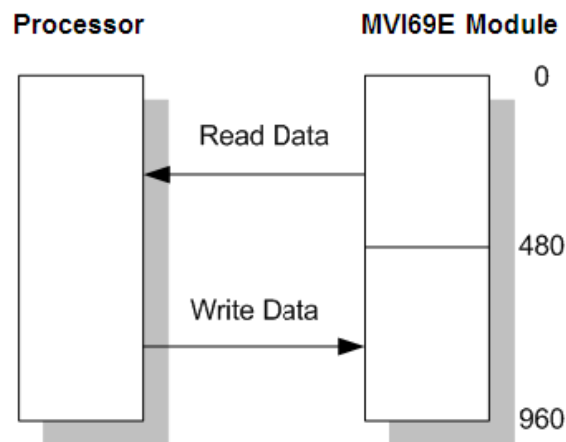
This sequence continues until interrupted by other write block numbers sent by the controller or by a command request from a node on the Modbus network or operator control through the module's Configuration/Debug port.

The following example shows a typical backplane communication application.

If the backplane parameters are configured as follows:

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

The backplane communication would be configured as follows:

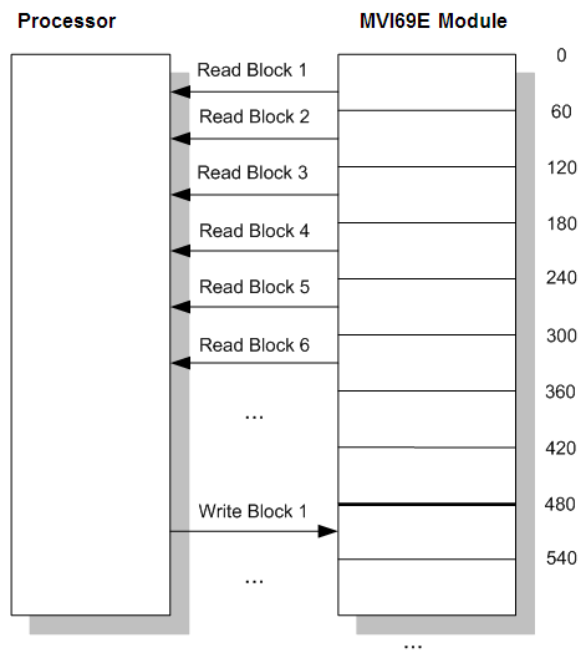


Database address 0 to 479 is continuously transferred from the module to the processor.

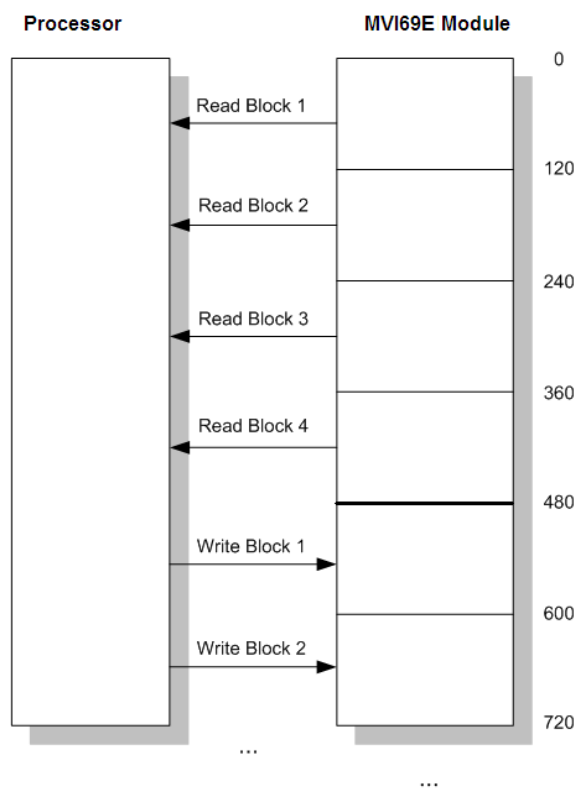
Database address 480 to 959 is continuously transferred from the processor to the module.

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

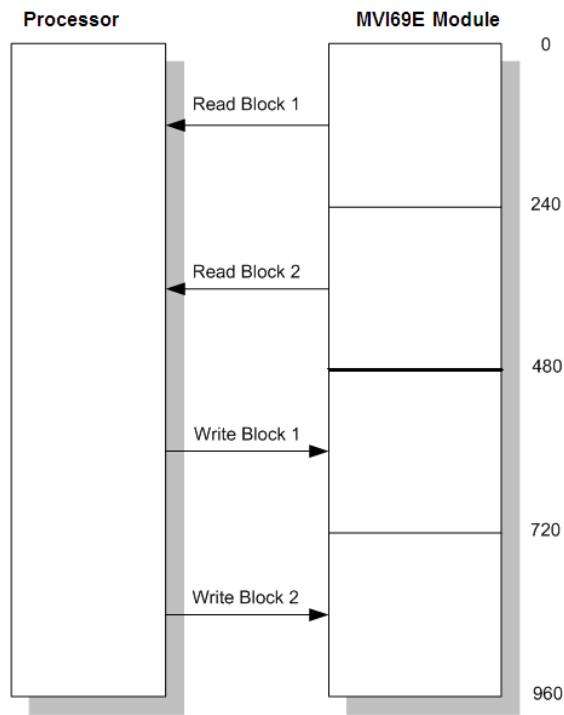
5.3.3.1 If Block Transfer Size = 60



5.3.3.2 If Block Transfer Size = 120



5.3.3.3 If Block Transfer Size = 240

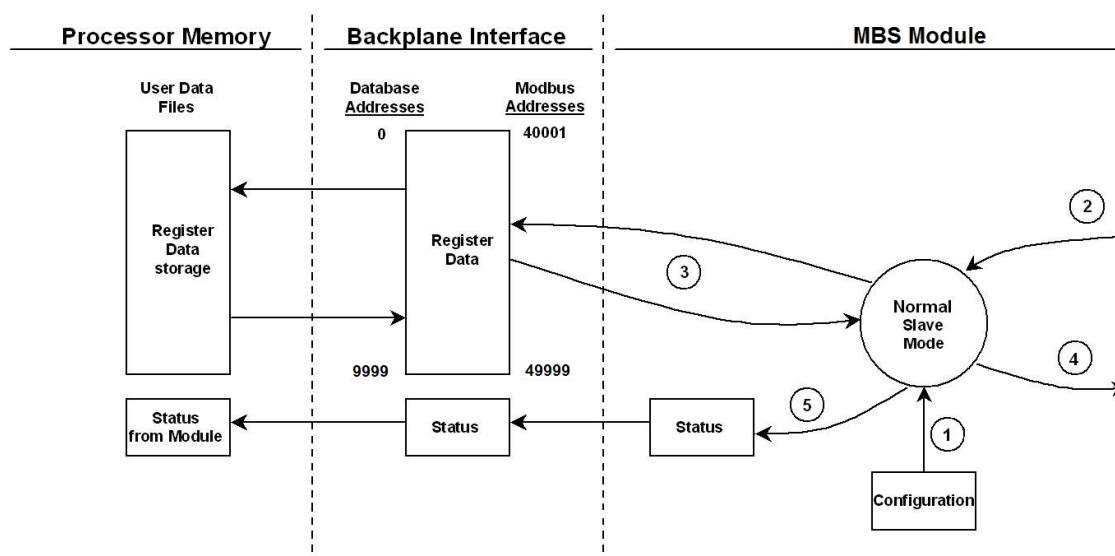


5.4 Data Flow Between the Module and Processor

The following topics describe the flow of data between the two pieces of hardware (CompactLogix or MicroLogix 1500-LRP processor and MVI69E-MBS module) and other nodes on the Modbus network. You can configure each port on the module to emulate a Modbus Master device or a Modbus Slave device.

5.4.1 Slave Mode

In Slave Driver mode, the MVI69E-MBS module responds to read and write commands issued by a master on the Modbus network. The following diagram shows the data flow for normal Slave mode.

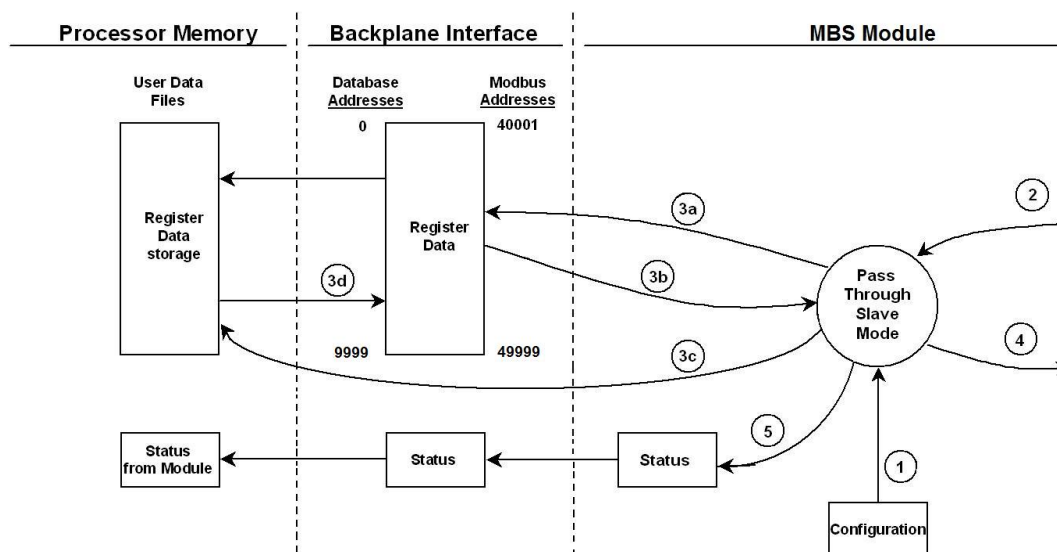


Step	Description
1	Any time the module restarts (boots or reboots), the Modbus slave port driver receives configuration information from the MBS controller tags. This information configures the serial ports and defines slave node characteristics. The configuration information may also contain instructions to offset data stored in the database to addresses different from addresses requested in the received messages.
2	A Modbus Master device, such as a Modicon PLC or an HMI application, issues a read or write command to the module's node address. The port driver qualifies the message before accepting it into the module. Rejected commands cause an Exception Response.
3	After the module accepts the command, the data is immediately transferred to or from the module's internal database. On a read command, the data is read from of the database and a response message is built. On a write command, the data is written directly into the database and a response message is built.
4	After Steps 2 and 3 have been completed, either a normal response message or an Exception Response message is sent to the Master.
5	Counters are available in the Status Block to permit the ladder logic program to determine the level of activity of the Slave driver.

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

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

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



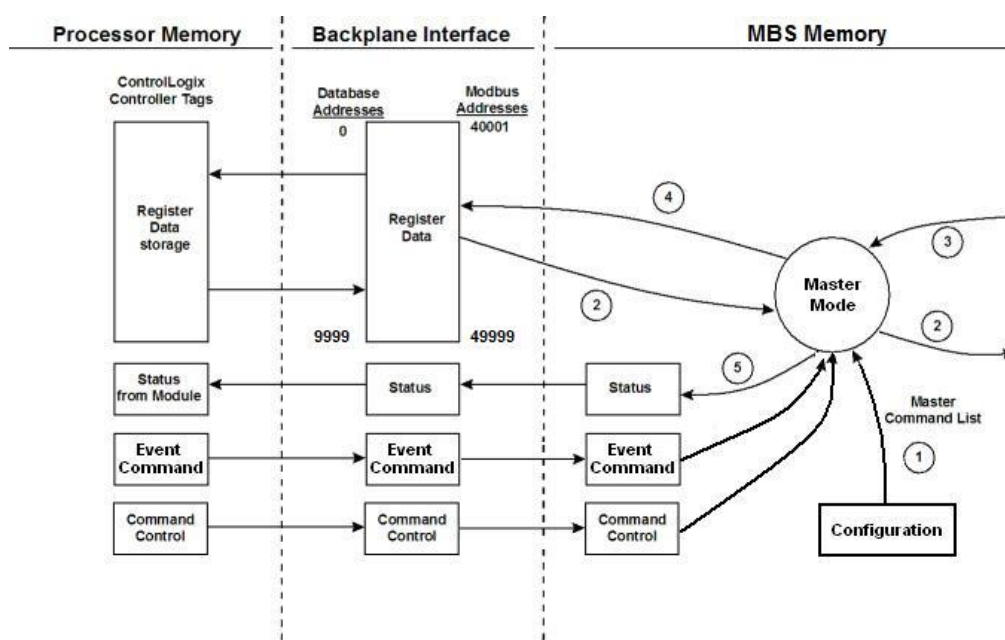
Step	Description
1	Same as normal mode.
2	Same as normal mode.
3	<p>a. In Pass-Through mode, if the Slave driver receives a read request, it looks for the data in module's internal database, just as it would in Normal mode.</p> <p>b. The data needed to respond to the read command is retrieved directly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

5.4.2 Master Mode

In Master mode, the MVI69E-MBS module issues read or write commands to slave devices on the Modbus network. These commands are user-configured in PCB; refer to *Modbus Port x Commands* (page 43). This list is transferred to the module when the module receives its configuration from the processor.

The commands can also be issued directly from the CompactLogix or MicroLogix 1500-LRP processor (Special Command Blocks).

Command status is returned to the processor for each individual command in the command list. The location of this command status list in the module's internal database is user-defined. The following flow chart and associated table describe the flow of command data into and out of the module.



Step	Description
1	Upon module boot-up, the Master driver obtains configuration data from the MBS controller tags. The configuration data retrieved includes port configuration and the Master Command List. Special Commands can be issued directly from the CompactLogix or MicroLogix 1500-LRP processor using Event Commands and Command Control. These command values are used by the Master driver to determine the types and order of commands to send to slaves on the network.
2	After configuration, the Master driver begins transmitting read and/or write commands to slave nodes on the network. If the Master driver is writing data to a slave, the data for the write command is retrieved from the module's internal database.
3	Once the specified slave has successfully processed the command, it returns a response message to the Master driver for processing.
4	Data received from a slave in response to a read command is stored in the module's internal database.
5	Status is returned to the processor for each command in the Master Command List.

Important: Take care when constructing each command in the list to ensure predictable operation of the module. If two commands are writing to the same internal database address of the module, the results are invalid. All commands containing invalid data are ignored by the module.

5.4.2.1 Master Command List

For a port to function in Master Mode, its Master Command List must be defined in Prosoft Configuration Builder; refer to *Modbus Port x Commands* (page 43). This list contains up to 310 individual entries, with each entry containing the information required to construct a valid command. A valid command includes the following items:

- Command enable mode: (0) disabled, (1) continuous or (2) conditional
- Source or destination database address: The module database address where data is written or read.
- Count: The number of words or bits to be transferred – up to 125 words for Function Codes 3, 4, or 16, and up to 2000 bits for Function Codes 1, 2, or 15.

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

- Slave node address
- Modbus Function Code: This is the type of command that is issued.
- Source or destination address in the slave device

5.4.2.2 Command Error Codes

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. The definition for these command error codes is listed in *Communication Error Codes* (page 110). View the command error codes through the Ethernet diagnostics port; refer to *Diagnostics and Troubleshooting* (page 97). They can also be transferred from the module's database to the processor.

To transfer the Command Error List to the processor, set the *Command Error Offset* parameter in the port configuration to a module database address that is in the module's Read Data area; refer to *Additional Configuration Parameters as Master* (page 41).

Note: The Command Error List must be placed in the Read Data area of the database, so it can be transferred to the processor in the input image.

6 Legacy Mode

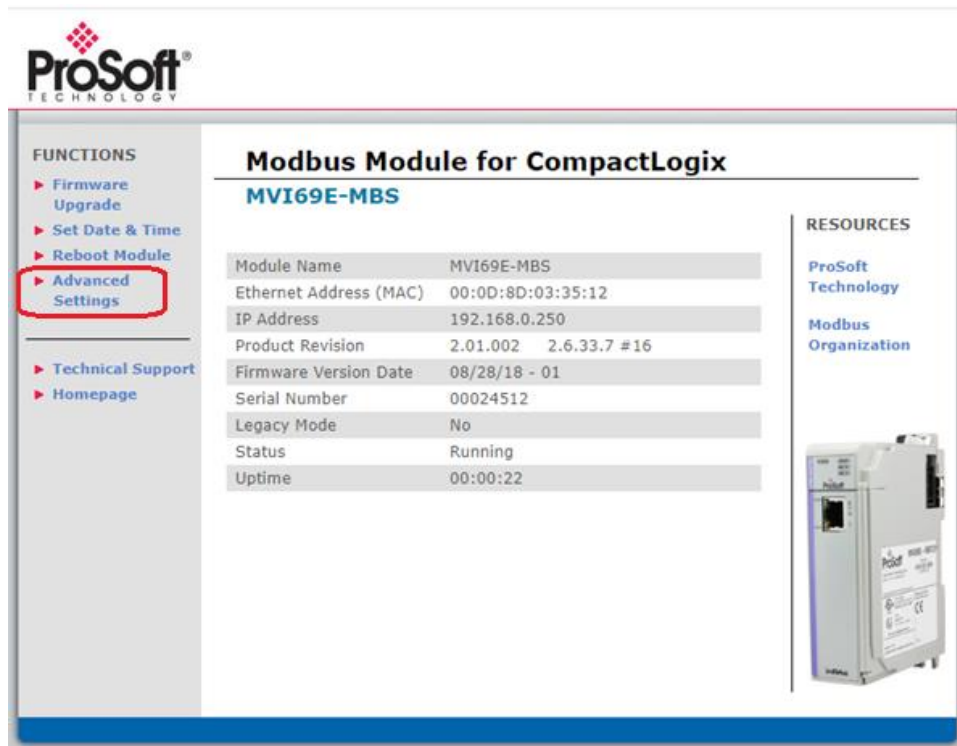
Legacy Mode allows you to replace an existing MVI69-MCM module with the MVI69E-MBS. This feature is only supported with MVI69E-MBS firmware version 1.11.001 or later.

The MVI69E-MBS Legacy Mode is backward-compatible with the MVI69-MCM. You may replace an existing MVI69-MCM with the MVI69E-MBS module (Legacy Mode) without any changes to the existing CompactLogix ladder logic application.

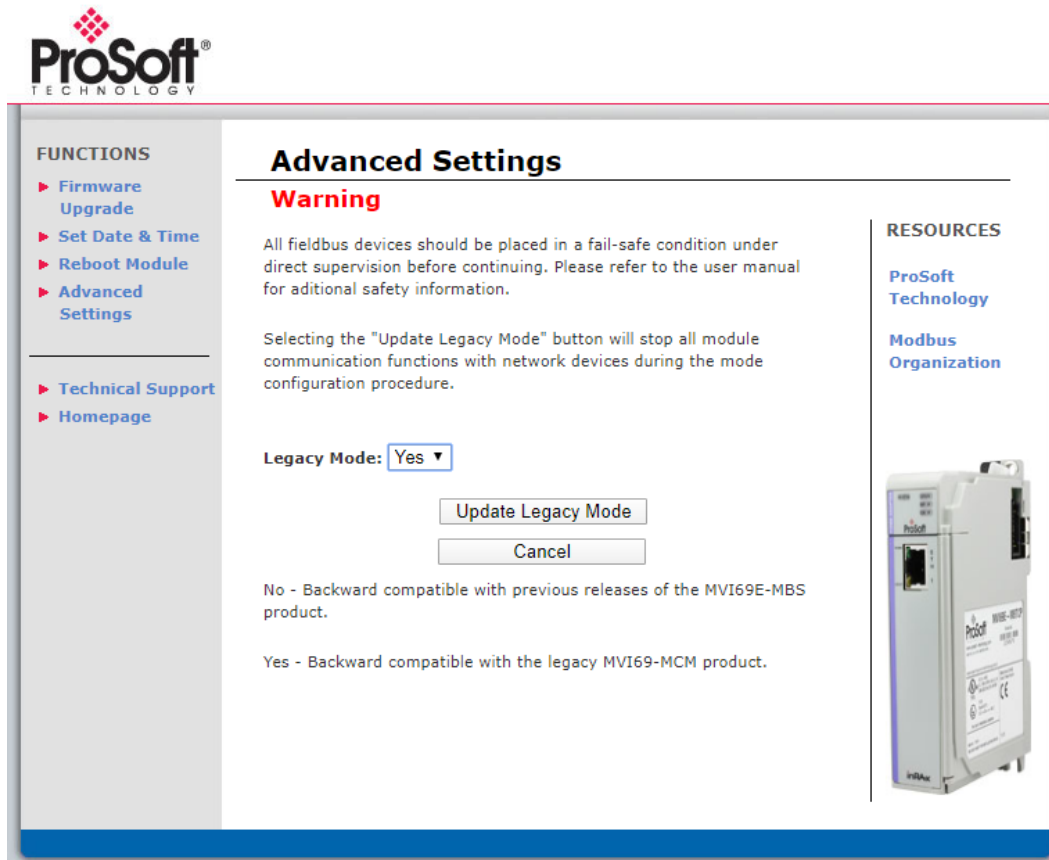
You may also convert the existing MVI69-MCM PCB configuration to the MVI69E-MBS module in Legacy Mode. This conversion procedure is supported by PCB v4.4.24.20.0302 or later.

6.1 Webpage Configuration

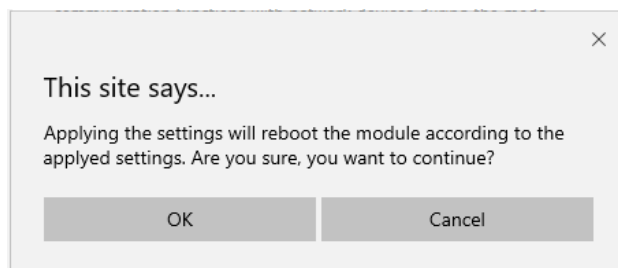
- 1 Open the MVI69E-MBS webpage. For further information, please see *Connecting to the Module's Webpage* on page 111.
- 2 Click on the *Advanced Settings* option.



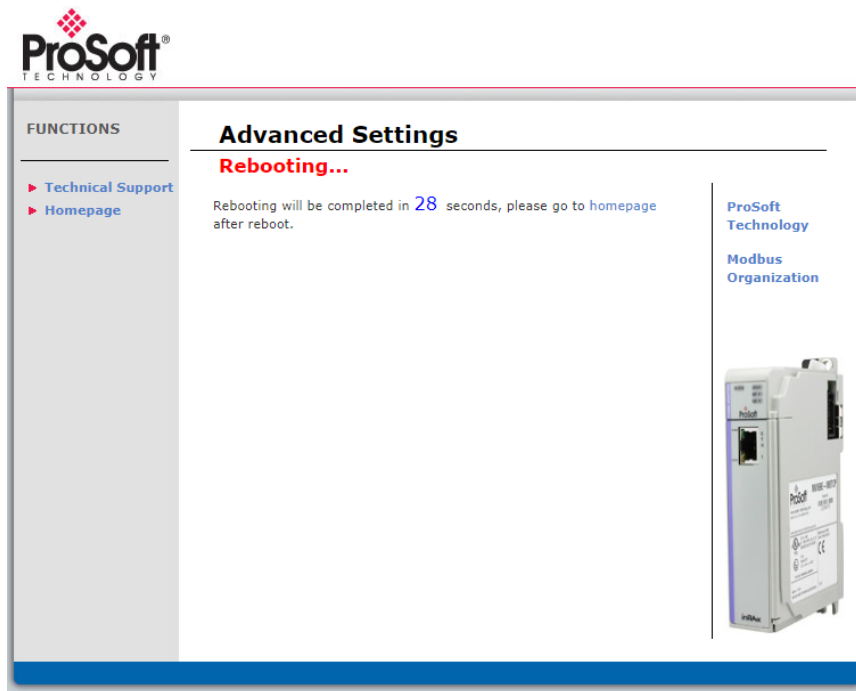
- 3 In the *Advanced Settings* page, change the **LEGACY MODE** field to 'Yes', then click on the **UPDATE LEGACY MODE** button.



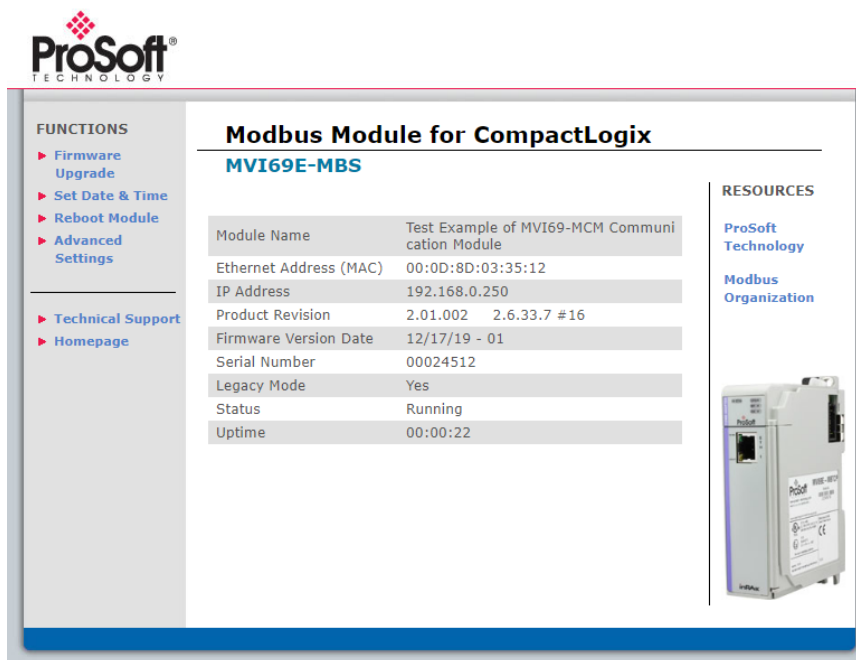
- 4 Confirm the update by clicking **OK**.



- 5 The module will reboot during the update process.



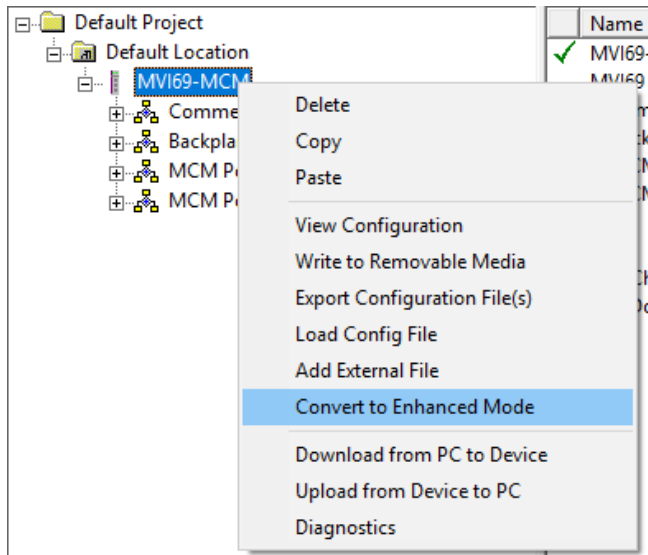
- 6 Once complete, the homepage displays **Legacy Mode – Yes**.



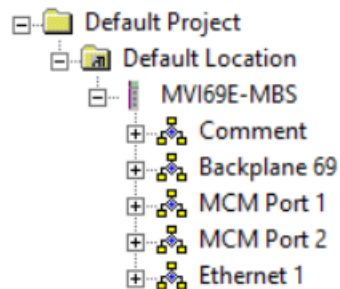
6.2 PCB Configuration

You will need to convert the existing 'MVI69-MCM' PCB project to an 'MVI69E-MBS' project.

- 1 Open the existing MVI69-MCM project in PCB.
- 2 Right-click on the *MVI69-MCM* icon and select **CONVERT TO ENHANCED MODE**.

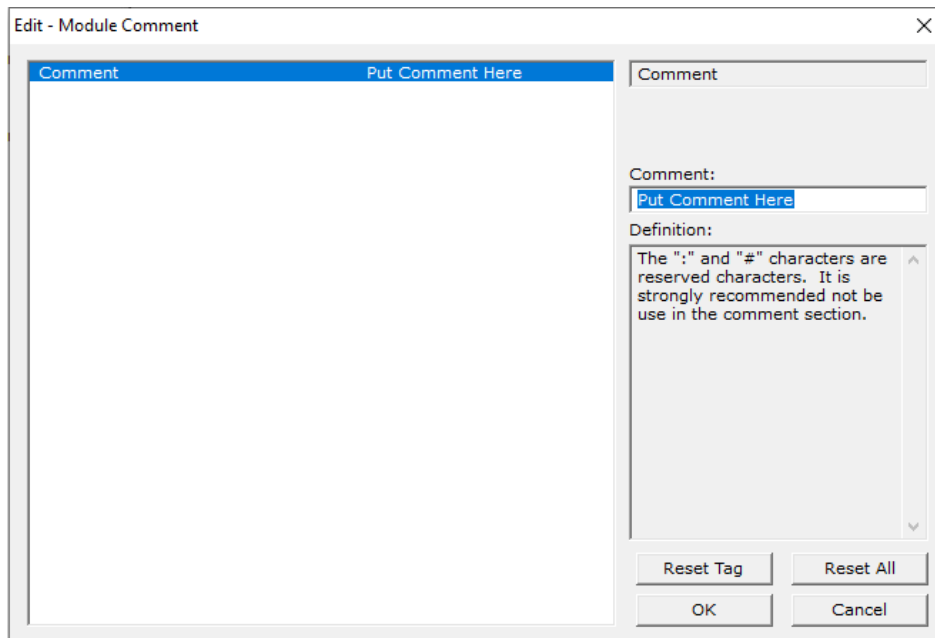


- 3 After the conversion, the PCB module parameters are updated.



6.2.1 Comment Parameter

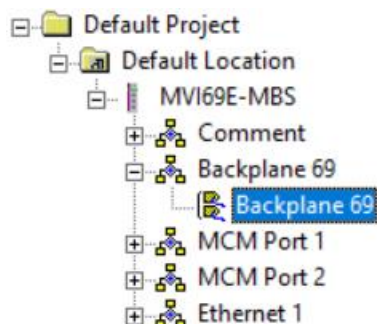
Under the **MODULE COMMENT** option in the module configuration, you can make a note that this configuration is a Legacy conversion.



6.2.2 Backplane69 Parameter

This section contains general module configuration parameters, including database allocation and backplane transfer options.

In the ProSoft Configuration Builder (PCB) tree view, double-click on the **BACKPLANE 69** icon.

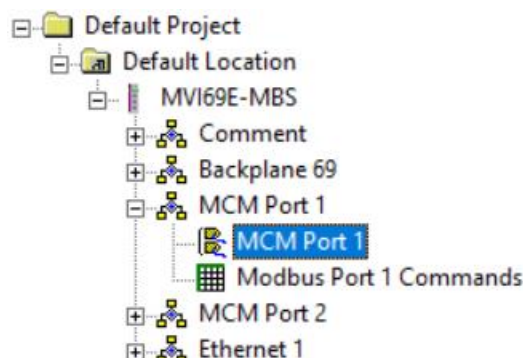


Parameter	Value	Description
Read Register Start	0 to 5000	Specifies the start of the Read Data area in module memory. Data in this area is transferred from the module to the processor.
Read Register Count	0 to 5000	Specifies the size of the Read Data area.
Write Register Start	0 to 5000	Specifies the start of the Write Data area in module memory. Data in this area is transferred from the processor to the module.
Write Register Count	0 to 5000	Specifies the size of the Write Data area.
Backplane Fail Count	0 to 65536	Specifies the number of consecutive backplane transfer failures that can occur before communications are halted.
Error/Status Block Pointer	-1 to 5000	Starting register location in the module's database for the error/status table. If a value of -1 is entered, the error/status data is not placed in the database. This data must be placed in the Read Data Range of module memory. This data includes the module version information and all server error/status data. Refer to <i>MBS.STATUS</i> (page 63) for more information.
Initialize Output Data	Yes or No	This parameter determines if the input image data and the module's Read Register Data values are initialized with Read Register Data values from the processor. If you set the parameter to No , the Read Register Data values in the module are set to 0 upon initialization. If you set the parameter to Yes , the data is initialized with Read Register Data values from the processor. This option requires associated ladder logic to pass the data from the processor to the module.
Block Transfer Size	60, 120 or 240	Specifies the number of words in each block transferred between the module and processor.

6.2.3 MCM Port x Parameters

This section applies to both MCM Port 1 and MCM Port 2.

In the ProSoft Configuration Builder tree view, double-click the MCM Port x icon.



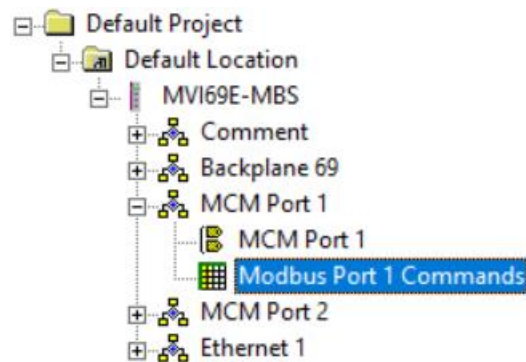
Parameter	Value	Description
Enable	Yes or No	Specifies whether the port and commands are active upon module boot-up.
Type	Master, Slave, or Slave with Pass-Through	This parameter specifies which device type the port emulates. See <i>Slave Mode</i> (page 71) for more information on Slave Pass-Through options.
Enron-Daniels	Yes or No	Use Floating point data offset.
Protocol	RTU or ASCII	Specifies the Modbus protocol for the port.
Baud Rate	Multiple options	Specifies the baud rate for the port.
Parity	None, Odd, Even	Specifies the type of parity error checking. All devices on this port must use the same parity setting.
Data Bits	7 or 8	Sets the number of data bits for each word used by the protocol. All devices communicating through this port must use the same number of data bits.
Stop Bits	1 or 2	Sets the number of stop bits that signal the end of a character in the data stream. For most applications, use one stop bit. For slower devices that require more time to re-synchronize, use two stop bits. All devices communicating through this port must use the same number of stop bits.
RTS On	0 to 65535 ms	Sets the number of milliseconds to delay after <i>Ready To Send</i> (RTS) is asserted before data is transmitted.
RTS Off	0 to 65535 ms	Sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal is set low.
Minimum Response Delay	0 to 65535 ms	(Slave mode) Number of milliseconds to delay before response to a Modbus command.
Use CTS Line	Yes or No	Specifies if the Clear To Send (CTS) modem control line is to be used or not. If you set the parameter to No , the CTS line is not monitored. If you set the parameter to Yes , the CTS line is monitored and must be high before the module sends data. Normally, this parameter is required when half-duplex modems are used for communication (2-wire). This procedure is commonly referred to as <i>hardware handshaking</i> .
Command Count	0 to 100	(Master mode) Specifies the number of commands to be processed by the Modbus master port.

Minimum Command Delay	0 to 32767 ms	(Master mode) Specifies the number of milliseconds to wait between receiving the end of a slave's response to the most recently transmitted command and the issuance of the next command. Use this parameter to place a delay after each command to avoid sending commands on the network faster than the slaves can receive them. This parameter does not affect retries of a command, as retries are issued when a command failure is recognized.
Cmd Err Pointer	0 to 4900	(Master mode) Internal DB location to place command error list
Response Timeout	0 to 65535 ms	(Master mode) Specifies the command response timeout period in 1 ms increments. This is the time that a Master waits for a response from the addressed slave before re-transmitting the command (Retries) or skipping to the next command in the Command List. The value to specify depends on the communication network used and the expected response time (plus or minus) of the slowest device on the network.
Retry Count	0 to 10	(Master mode) Specifies the number of times a command is retried if it fails.
Error Delay Count	0 to 60000	(Master mode) Specifies the number of poll attempts to be skipped before trying to re-establish communications with a slave that has failed to respond to a command within the time limit set by the <i>Response Timeout</i> parameter. After the slave fails to respond, the master skips sending commands that should have been sent to the slave until the number of skipped commands matches the value entered in this parameter. This creates a sort of <i>slow poll</i> mode for slaves that are experiencing communication problems.
Slave Address	1 to 255	(Slave mode) Modbus Slave address on network.
Bit Input Offset	0 to 3998	(Slave mode) Specifies the offset address into the internal Modbus database that is to be used with network for Modbus function 2 commands.
Word Input Offset	0 to 3998	(Slave mode) Specifies the offset address into the internal Modbus database that is to be used with network for Modbus function 4 commands.
Output Offset	0 to 3998	(Slave mode) Specifies the offset address into the internal Modbus database that is to be used with network requests for Modbus function 1, 5, or 15 commands.
Holding Register Offset	0 to 3998	(Slave mode) Specifies the offset address in the internal Modbus database that is to be used with network for Modbus function 3, 6, or 16 commands.
Inter-character Timeout	0 to 65535 ms	(Master mode) Specifies a time delay to be added to the 3.5 character time delay used by the module to recognize the end of a message.
Command Error Offset	0 to 4998	(Master mode) Internal Database offset location of command error

6.2.4 Modbus Port x Commands

This section defines the Modbus master command list specifications for **MBS PORT 1** and **MBS PORT 2**.

In the ProSoft Configuration Builder tree view, double-click the **MODBUS PORT X COMMANDS** icon.



In order to interface the MVI69E-MBS with Modbus slave devices, you must create a command list. The commands in the list specify the slave device to be addressed, the function to be performed (read or write), the data area in the device to interface with and the registers in the internal database to be associated with the device data.

The Master command list supports up to 100 commands. The command list is processed from top (Command #0) to bottom.

Read commands are executed without condition. You can set write commands to execute only if the data in the write command changes (Conditional Enable). If the register data values in the command have not changed since the command was last issued, the command is not executed. You can use this feature to optimize network performance.

The MVI69E-MBS Master supports several data read and write commands. When a command is configured, the type of data (bit, 16-bit integer, 32-bit float, etc), and the level of Modbus support in the slave equipment needs to be considered.

Parameter	Value	Description
Enable	0 to 2	<p>This field defines whether the command is to be executed under certain conditions.</p> <p>Disabled (0) = The command is disabled and is not executed in the normal polling sequence.</p> <p>Continuous (1) = The command is executed each scan of the command list if the <i>Poll Interval</i> (see below) is set to zero. If the <i>Poll Interval</i> is set to a nonzero value, the command is executed when the interval timer expires.</p> <p>Conditional (2) = For write commands only. The command executes only if the internal data associated with the command changes.</p>
Internal Address	0 to 4999 (word-level) or 0 to 65535 (bit-level)	<p>Specifies the module's internal database register to be associated with the command. Allowable range is 0 to 4999 for Modbus Function Codes 3, 4, 6, or 16, and 0 to 65535 for Modbus Function Codes 1, 2, 5, or 15.</p>

		<p>If the command is a read function, the data read from the slave device is <i>stored</i> beginning at the module's internal database register value entered in this field. This register value must be in the Read Data area of the module's memory, defined by the <i>Read Register Start</i> and <i>Read Register Count</i> parameters in the Module section.</p> <p>If the command is a write function, the data to be written to the slave device is <i>sourced</i> beginning from the module's internal database register specified. This register value must come from the Write Data area of the module's memory, defined by the <i>Write Register Start</i> and <i>Write Register Count</i> parameters in the Module section.</p> <p>Note: When using a bit level command, you must define this field at the bit level. For example, when using function codes 1 or 2 for a Read command, you must have a enter of 160 to place the data in the MBS.DATA.ReadData[10] controller tag in Studio 5000. Think of it as the 160th bit of MBS internal memory (MBS Internal register 10 * 16 bits per register = 160). Use this formula for function codes 5 or 15 for writing bits also.</p> <p>This controller tag is a 16bit signed integer. This means you can only enter values of -32768 to 32767 in the tag. If a value to be entered is above the 32767 (but below 65535) threshold, it displays as a negative value in the tag. Simply subtract 65536 from the value to get the 'acceptable' value to enter into the tag.</p> <p>Example: You need to use an Internal bit Address of 48000, but you cannot enter '48000' into the tag because it causes an error. $48000 - 65536 = -17536$ You enter -17536 in the Internal Address parameter for this command.</p>
Poll Interval	0 to 65535 (seconds)	<p>Specifies the minimum interval between executions of continuous commands (<i>Enable</i> code = 1).</p> <p>Example: If a value of 100 is entered, the command executes no more frequently than every 100 seconds. When the command reaches the top of the command queue and 100 seconds has not elapsed, it is skipped until the poll interval has expired.</p>
Register Count	1 to 125 (words) or 1 to 2000 (coils)	<p>Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point.</p> <p>For Modbus Function Codes 1, 2 and 15, this parameter sets the number of single bit digital points (inputs or coils) to be associated with the command.</p> <p>Note: Up to 2000 coils are supported for Modbus Function Codes 1 and 2. Up to 1968 coils are supported for Modbus Function Code 15.</p> <p>For Modbus Function Codes 3, 4 and 16, this parameter sets the number of 16-bit registers to be associated with the command.</p>
Swap Code	0,1,2,3	<p>Defines if the data received from the Modbus slave is to be ordered differently than received from the slave device. This parameter is helpful when dealing with floating-point or other multi-register values, as there is no standard method of storage of these data types in slave devices. You can set this parameter to order the register data received in an order useful by other applications.</p> <p>No Change (0)= No change is made in the byte ordering (ABCD = ABCD) Word Swap (1)= The words are swapped (ABCD= CDAB) Word and Byte Swap (2) = The words are swapped, then the bytes in each word are swapped (ABCD=DCBA) Byte Swap (3) = The bytes in each word are swapped (ABCD=BADC)</p>

		<p>Note: Each pair of characters is a byte. Ex: AB and CD. Two pairs of characters is 16-bit register Ex: ABCD.</p>
Node Address	1 to 255 (0 = broadcast)	<p>Specifies the Modbus slave node address on the network to be considered. Most Modbus devices only accept an address in the range of 1 to 247. If you set the value to zero, the command is a broadcast message on the network. The Modbus protocol permits broadcast commands for write operations. Do not use this node address for read operations.</p>
Modbus Function	1,2,3,4,5,6,15,16	<p>Specifies the Modbus function to be executed by the command. These function codes are defined in the Modbus protocol.</p> <ul style="list-style-type: none"> 1 – Read Coil Status (0xxxx) 2 – Read Input Status (1xxxx) 3 – Read Holding Registers (4xxxx) 4 – Read Input Registers (3xxxx) 5 – Force (Write Single) Coil (0xxxx) 6 – Force (Write Single) Holding Register (4xxxx) 15 – Preset (Write) Multiple Coils (0xxxx) 16 – Preset (Write) Multiple Registers (4xxxx)
MB Address in Device	0 to 65535	<p>Specifies the register or digital point address offset within the Modbus slave device. The MBS Master reads or writes from/to this address within the slave. Refer to the documentation of each Modbus slave device for their register and digital point address assignments.</p> <p>Note: The value entered here does not need to include the "Modbus Prefix" addressing scheme. Also, this value is an offset of the zero-based Modbus addressing scheme.</p> <p>Example: Using a Modbus Function Code 3 to read from address 40010 in the slave, a value of '9' would be entered in this parameter. The firmware (internally) adds a '40001' offset to the value entered. This is the same for all Modbus addresses (0x, 1x, 3x, 4x).</p>

6.2.5 Ethernet 1 Parameter

The **ETHERNET 1** option allows you to configure the module’s IP Address, Subnet Mask, and Gateway.

Edit - Ethernet 1

IP Address	192.168.0.250
Netmask	255.255.255.0
Gateway	192.168.0.1

IP Address

192 . 168 . 0 . 250

Comment:

Definition:

Default private class 3 address

Reset Tag

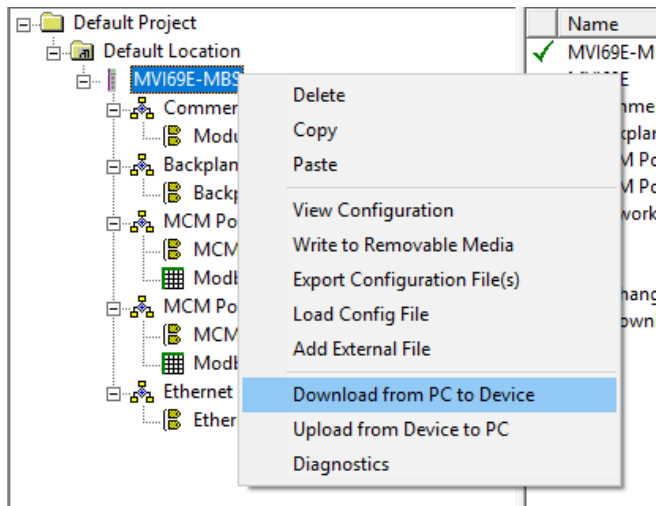
Reset All

OK

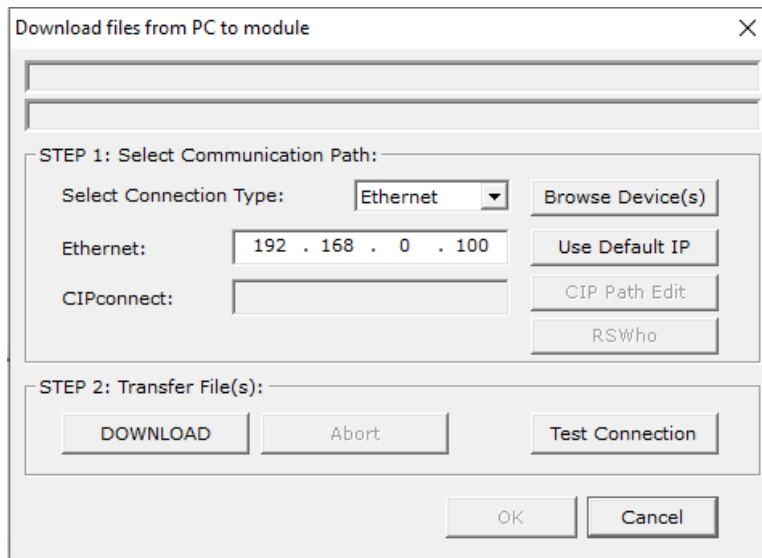
Cancel

6.3 Downloading PCB Configuration to the MVI69E-MBS

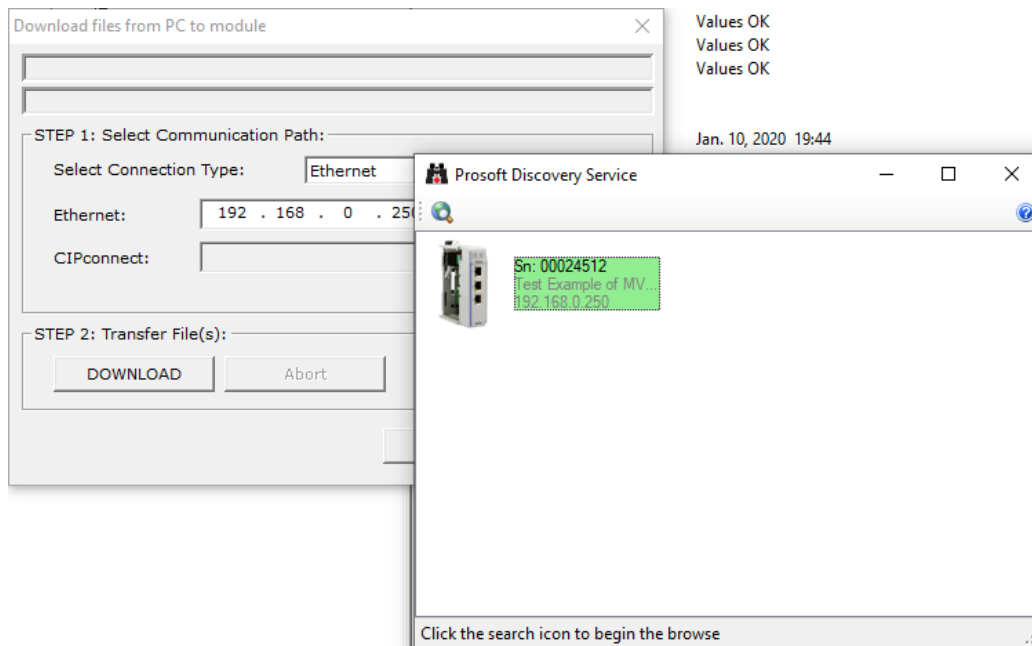
- 1 Right-click on the *MVI69-MBS* icon and select **DOWNLOAD FROM PC TO DEVICE**.



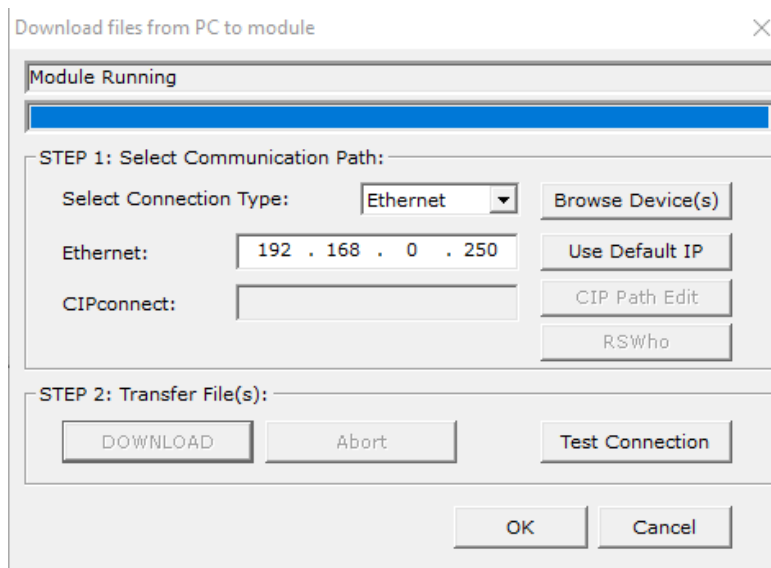
- 2 In the *Download files from PC to module* dialog, click on the **BROWSE DEVICE(S)** button. The ProSoft Discovery Service Utility searches for ProSoft devices on the network.



- 3 Double-click on the module icon.



- 4 Click **DOWNLOAD**. When complete, the 'Module Running' message is displayed.



- 5 Once complete, the MVI69E-MBS in Legacy Mode will operate similarly to the MVI69-MCM.

6.4 Optional Add-On Instruction

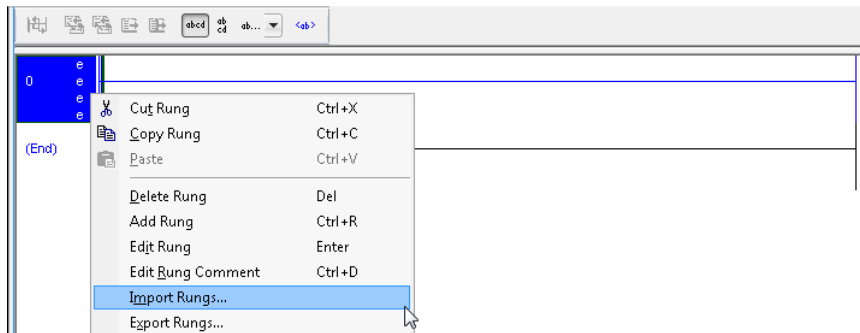
The Optional AOI supports the following optional features:

- Read/Write IP Address
- Read/Write Date Time

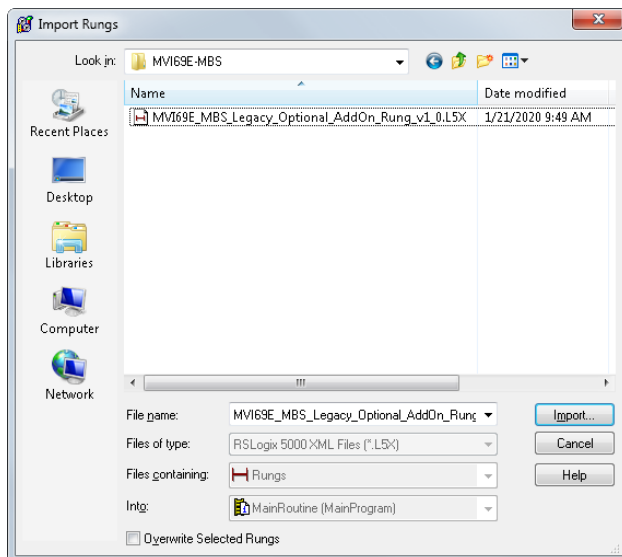
Using controller tags, the Optional AOI allows you to request and set the module's IP address, date, and time. These optional features are not supported by the MVI69E-MCM legacy module.

Note: The Optional AOI may be added to an existing legacy MVI69E-MBS application to add the new functionality during a module replacement.

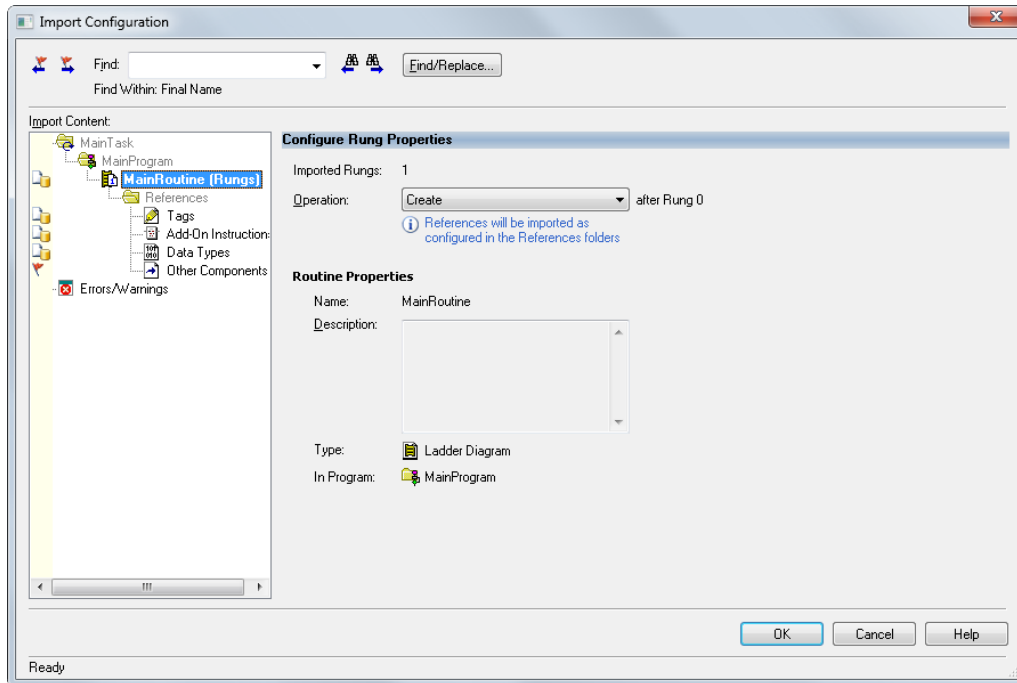
- 1 Add a new rung to the existing processor ladder logic. Right-click on the new rung and select *Import Rungs...*



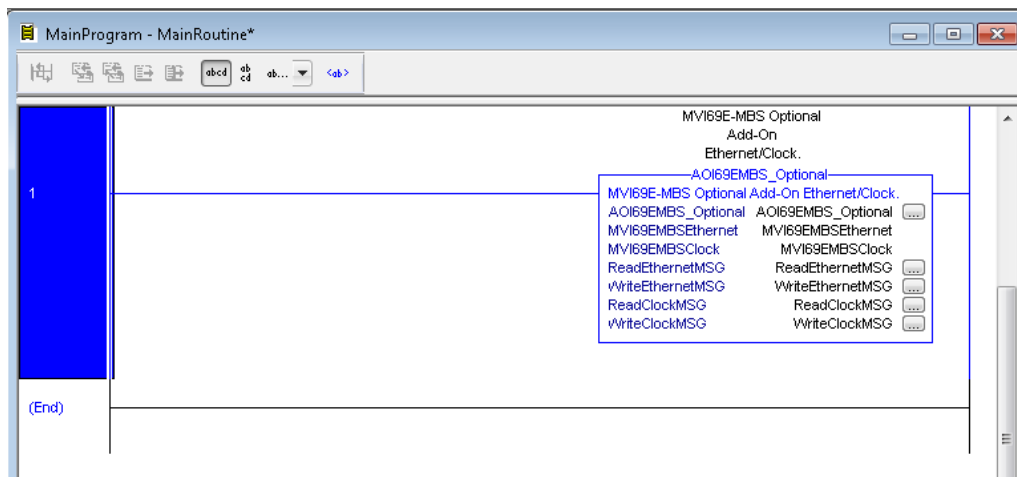
- 2 Select the Optional AOI file: *MVI69E_MBS_Optional_AddOn_Rung.L5X*



- 3 At the *Import Configuration* window, select the *Operation* parameter to **CREATE**. Then click **OK**.

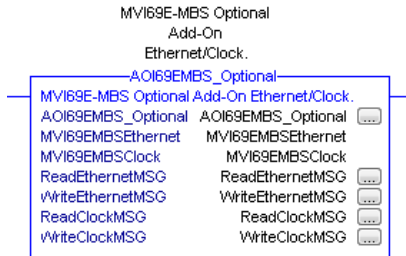


- 4 The imported AOI rung is now in place.

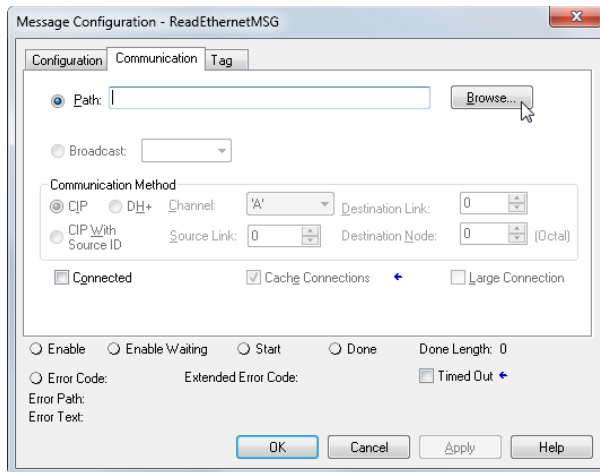


6.4.1 Setting Up the Optional AOI

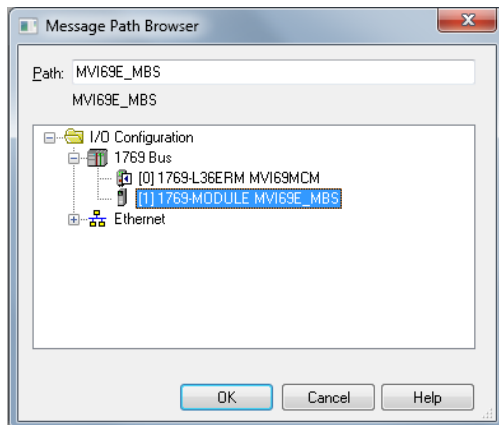
- 1 Click on the *ReadEthernetMSG*  icon to configure the message route:



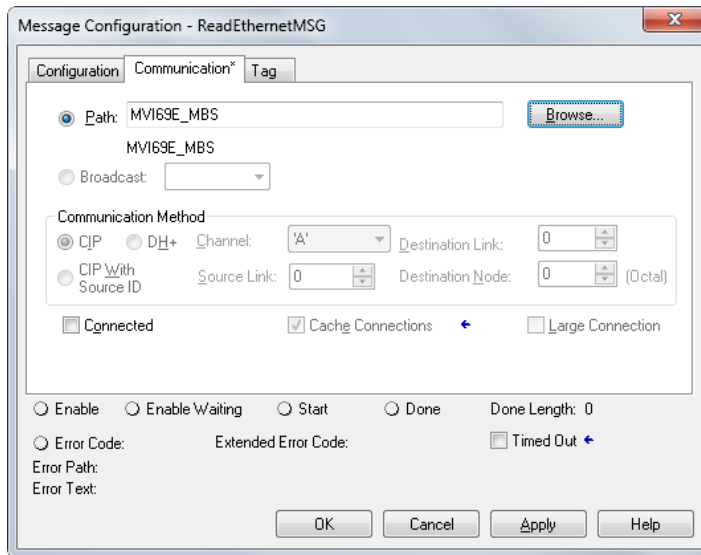
- 2 In the *Message Configuration* dialog, under the *Communication* tab, select the **BROWSE** button.



- 3 In the *Message Path Browser* dialog, select the MVI69E-MBS module under the 1769 Bus and click at **OK**.



- 4 The module name is displayed in the *Path* field. Click **OK** to confirm the route configuration.



- 5 Repeat the same procedure to set the route for the remaining messages:

- *WriteEthernetMSG* ...
- *ReadClockMSG* ...
- *WriteClockMSG* ...

6.4.2 Synchronizing the IP Settings from the MVI69E-MBS to the Processor

This section covers the process to read the IP settings from the MVI69E-MBS, and implement them in the processor.

- 1 To trigger the IP settings read operation, set the *MVI69EMBSEthernet.Read* bit to '1'.

[-] MVI69EMBSEthernet	{ ... }
[-] MVI69EMBSEthernet.Read	[1]

- 2 Once the operation is concluded, the tag will automatically reset to '0'.

[-] MVI69EMBSEthernet	{ ... }
[-] MVI69EMBSEthernet.Read	0

- 3 The data is stored in the *MVI69EMBSEthernet.Config* tags (IP, Netmask, Gateway) as follows:

[-] MVI69EMBSEthernet.Config	{ ... }
[-] MVI69EMBSEthernet.Config.IP	{ ... }
+ MVI69EMBSEthernet.Config.IP[0]	192
+ MVI69EMBSEthernet.Config.IP[1]	168
+ MVI69EMBSEthernet.Config.IP[2]	0
+ MVI69EMBSEthernet.Config.IP[3]	250
[-] MVI69EMBSEthernet.Config.Netmask	{ ... }
+ MVI69EMBSEthernet.Config.Netmask[0]	255
+ MVI69EMBSEthernet.Config.Netmask[1]	255
+ MVI69EMBSEthernet.Config.Netmask[2]	255
+ MVI69EMBSEthernet.Config.Netmask[3]	0
[-] MVI69EMBSEthernet.Config.Gateway	{ ... }
+ MVI69EMBSEthernet.Config.Gateway[0]	192
+ MVI69EMBSEthernet.Config.Gateway[1]	168
+ MVI69EMBSEthernet.Config.Gateway[2]	0
+ MVI69EMBSEthernet.Config.Gateway[3]	1

6.4.3 Synchronizing the IP Settings from the Processor to the MVI69E-MBS

This section covers the process to send the IP settings from the processor to the MVI69E-MBS.

- 1 Populate the IP settings in the *MVI69EMBSEthernet.Config* tag:

[-] MVI69EMBSEthernet.Config	{ ... }
[-] MVI69EMBSEthernet.Config.IP	{ ... }
+ MVI69EMBSEthernet.Config.IP[0]	192
+ MVI69EMBSEthernet.Config.IP[1]	168
+ MVI69EMBSEthernet.Config.IP[2]	0
+ MVI69EMBSEthernet.Config.IP[3]	250
[-] MVI69EMBSEthernet.Config.Netmask	{ ... }
+ MVI69EMBSEthernet.Config.Netmask[0]	255
+ MVI69EMBSEthernet.Config.Netmask[1]	255
+ MVI69EMBSEthernet.Config.Netmask[2]	255
+ MVI69EMBSEthernet.Config.Netmask[3]	0
[-] MVI69EMBSEthernet.Config.Gateway	{ ... }
+ MVI69EMBSEthernet.Config.Gateway[0]	192
+ MVI69EMBSEthernet.Config.Gateway[1]	168
+ MVI69EMBSEthernet.Config.Gateway[2]	0
+ MVI69EMBSEthernet.Config.Gateway[3]	1

- 2 Set the *MVI69EMBSEthernet.Write* bit to '1' to trigger the IP settings write operation.

[-] MVI69EMBSEthernet	{ ... }
- MVI69EMBSEthernet.Read	0
- MVI69EMBSEthernet.Write	[1]

- 3 The *MVI69EMBSEthernet.Write* bit will automatically reset to '0' once the operation is concluded.

[-] MVI69EMBSEthernet	{ ... }
- MVI69EMBSEthernet.Read	0
- MVI69EMBSEthernet.Write	0

6.4.4 Reading the Date/Time from the MVI69E-MBS to the Processor

- 1 Toggle the MVI69EMBSClock.Read bit to '1' to toggle the date/time read operation.

[-] MVI69EMBSClock	{ ... }
[-] MVI69EMBSClock.Read	1
[-] MVI69EMBSClock.Write	0

- 2 The MVI69EMBSClock.Read bit will automatically reset to '0' once the operation is concluded.

[-] MVI69EMBSClock	{ ... }
[-] MVI69EMBSClock.Read	0
[-] MVI69EMBSClock.Write	0

- 3 The date and time read from the MVI69E-MBS is stored at the MVI69EMBSClock.Config tag.

[-] MVI69EMBSClock	{ ... }
[-] MVI69EMBSClock.Read	0
[-] MVI69EMBSClock.Write	0
[-] MVI69EMBSClock.Config	{ ... }
+ MVI69EMBSClock.Config.Year	2020
+ MVI69EMBSClock.Config.Month	1
+ MVI69EMBSClock.Config.Day	6
+ MVI69EMBSClock.Config.Hour	8
+ MVI69EMBSClock.Config.Minute	3
+ MVI69EMBSClock.Config.Seconds	10

6.4.5 Writing the Date/Time from the Processor to the MVI69E-MBS

- 1 Populate date and time values in the *MVI69EMBSClock.Config* tag.

[-] MVI69EMBSClock.Config	{...}
+ MVI69EMBSClock.Config.Year	2020
+ MVI69EMBSClock.Config.Month	2
+ MVI69EMBSClock.Config.Day	14
+ MVI69EMBSClock.Config.Hour	4
+ MVI69EMBSClock.Config.Minute	45
+ MVI69EMBSClock.Config.Seconds	22

- 2 Toggle the *MVI69EMBSClock.Write* bit to '1' to trigger the write date/time operation.

[-] MVI69EMBSClock	{...}
- MVI69EMBSClock.Read	0
- MVI69EMBSClock.Write	1

- 3 The *MVI69EMBSClock.Write* tag will be automatically reset to '0' once the write date/time operation is concluded.

[-] MVI69EMBSClock	{...}
- MVI69EMBSClock.Read	0
- MVI69EMBSClock.Write	0

For further information concerning the MVI69-MCM, please download the MVI69-MCM User Manual from www.prosoft-technology.com.

7 Diagnostics and Troubleshooting

The MVI69E-MBS provides diagnostics and troubleshooting in the following forms:

- LED status indicators on the front of the module provides general status information.
- View status data contained in the module through the Ethernet port, using the troubleshooting and diagnostic capabilities of *ProSoft Configuration Builder (PCB)*.
- Transfer status data values from the module to processor memory and can monitor them in the processor manually or by customer-created logic.

7.1 LED Status Indicators

The LEDs indicate the module's operating status.

ETH	CFG
P1	BP
P2	OK

LED	Status	Description
ETH	Green	Application is running and Ethernet is ready
	Off	Possible causes: <ul style="list-style-type: none"> • Network communication has not started yet • Physical ethernet connection is down • Ethernet communication is disabled as the system shuts down
P1	Green	Data is being transferred
	Red	Not used
	Off	No serial communication activity
P2	Green	Data is being transferred
	Red	Not used
	Off	No serial communication activity
CFG	Red	Possible causes: <ul style="list-style-type: none"> • Configuration failure during start-up or module boot • Module detected a bad or unreadable configuration file; problem opening the configuration file or the file is corrupted
	Green	Configuration is valid
	Amber	Possible causes: <ul style="list-style-type: none"> • During initialization, the configuration process has started and is being validated • Configuration file has been received and is being processed
	Off	Application is not running, or backplane has failed. Possible causes: <ul style="list-style-type: none"> • Communication with the backplane has failed • Backplane process is not up and running; no link to the controller so configuration status cannot be confirmed • Nothing to configure; module is waiting or idle. Cannot open file for writing or if no configuration file is received from the processor • Backplane data failure counter (<i>Backplane Fail Count</i> parameter in the Module configuration) exceeded the limit • Module is in shutdown function; configuration is no longer active once the system is shutting down

BP	Red	<p>Possible causes:</p> <ul style="list-style-type: none"> • Backplane disruption or communication error (PLC may not be in RUN mode). • Initialization failed; backplane communication took too long. Block timeout occurred during module initialization • Backplane data failure counter (<i>Backplane Fail Count</i> parameter in the Module configuration) has exceeded the limit
	Green	Backplane transfers are successful; communications with PLC are operational
	Amber	Initialization state: Module is in the process of establishing communication with the controller.
	Off	<p>Possible causes:</p> <ul style="list-style-type: none"> • During power-up, no communication with the backplane has started yet • Module is in shutdown function; backplane link is no longer active once the system is shutting down
OK	Red	<p>Possible causes:</p> <ul style="list-style-type: none"> • Module is powering up • Module has encountered a fatal error • Module is in shutdown function
	Green	The module has properly initialized and is running

7.2 Ethernet LED Indicators

The Ethernet LEDs indicate the module's Ethernet port status.

LED	State	Description
100 Mbit	Off	Ethernet connected at 10Mbps duplex speed
	Amber Solid	Ethernet connected at 100Mbps duplex speed
LINK/ACT	Off	No physical network connection is detected. No Ethernet communication is possible. Check wiring and cables.
	Green Solid or Blinking	Physical network connection detected. This LED must be On solid for Ethernet communication to be possible.

7.3 Clearing a Fault Condition

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

To clear the condition, follow these steps:

- 1 Turn off power to the rack.
- 2 Remove the card from the rack.
- 3 Verify that all jumpers are set correctly.
- 4 If the module requires a Compact Flash card, verify it is installed correctly.
- 5 Re-insert the card in the rack and turn the power back on.
- 6 Verify correct configuration data is being transferred to the module from the CompactLogix or MicroLogix 1500-LRP controller.

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

7.4 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.

7.4.1 Processor Errors

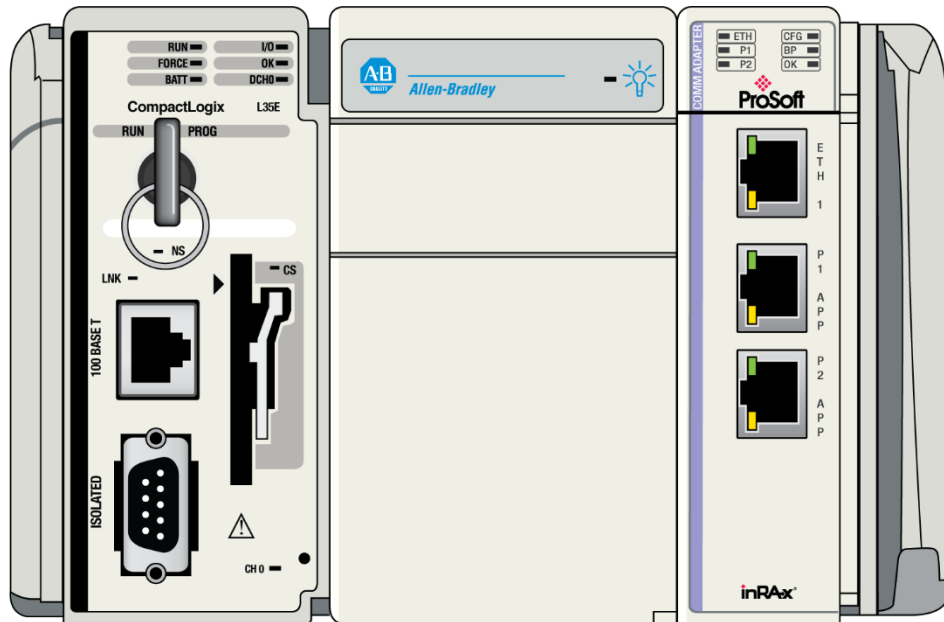
Problem description	Steps to Take
Processor fault	Verify that the module is securely plugged into the slot that has been configured for the module in the I/O Configuration in RSLogix. Verify that the slot location in the rack has been configured correctly in the ladder logic.
Processor I/O LED flashes	This indicates a problem with backplane communications. A problem could exist between the processor and any installed I/O module, not just the MVI69E-MBS. Verify that all modules in the rack are correctly configured.

7.4.2 Module Errors

Problem description	Steps to take
BP ACT LED (not present on MVI56E modules) remains OFF or blinks slowly	This indicates that backplane transfer operations are failing. Connect to the module's Configuration/Debug port to check this. To establish backplane communications, verify the following items: The processor is in RUN or REM RUN mode.
MVI69 modules with scrolling LED display: <Backplane Status> condition reads ERR	The backplane driver is loaded in the module. The module is configured for read and write data block transfer. The ladder logic handles all read and write block situations. The module is properly configured in the processor I/O configuration and ladder logic.
OK LED remains RED	The program has halted or a critical error has occurred. Connect to the Configuration/Debug (or Communication) port to see if the module is running. If the program has halted, turn off power to the rack, remove the card from the rack, then re-insert it, and then restore power to the rack.

7.5 Connecting the PC to the Module's Ethernet Port

With the module securely mounted, connect one end of the Ethernet cable to the **ETH1** Port, and the other end to an Ethernet hub or switch accessible from the same network as the PC. Or, connect directly from the Ethernet Port on the PC to the **ETH 1** Port on the module.

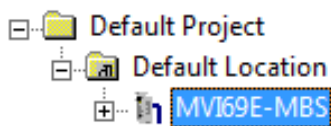


7.5.1 Setting Up a Temporary IP Address

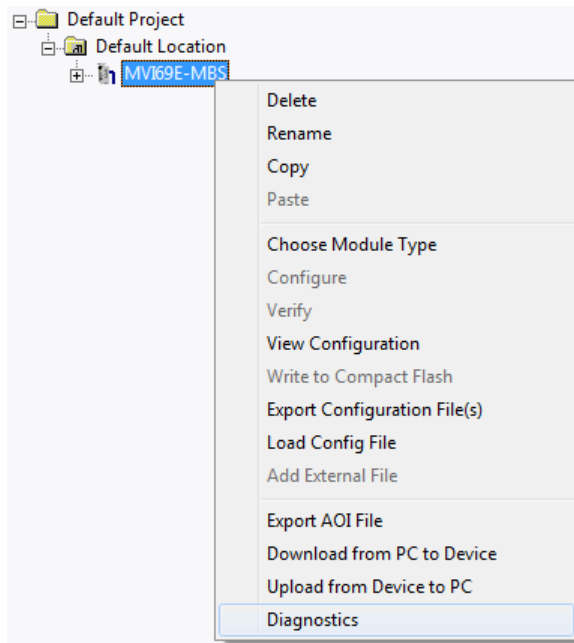
Important: ProSoft Configuration Builder (PCB) locates MVI69E-MBS modules through UDP broadcast messages. These messages may be blocked by routers or layer 3 switches. In that case, **ProSoft Discovery Service** is unable to locate the modules.

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

- 1 In the tree view in **ProSoft Configuration Builder (PCB)**, select the **MVI69E-MBS** module. (For instructions on opening and using a project in PCB, please refer to *Configuring the MVI69E-MBS Using PCB* (page 35).



- 2 Right-click the module icon in the tree and choose **DIAGNOSTICS**.

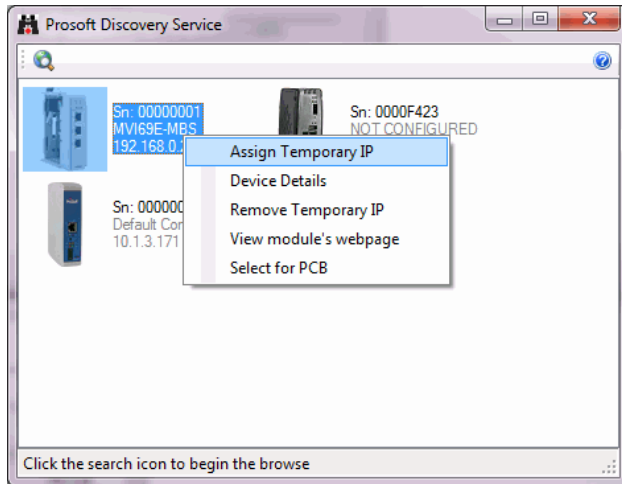


- 3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.

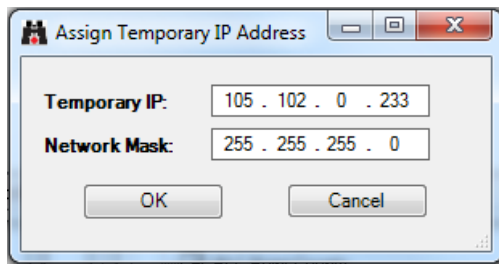


Click to set up connection

- 4 In the *Connection Setup* dialog box, click **BROWSE DEVICE(S)** to start *ProSoft Discovery Service*. Right-click the module and choose **ASSIGN TEMPORARY IP**.



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



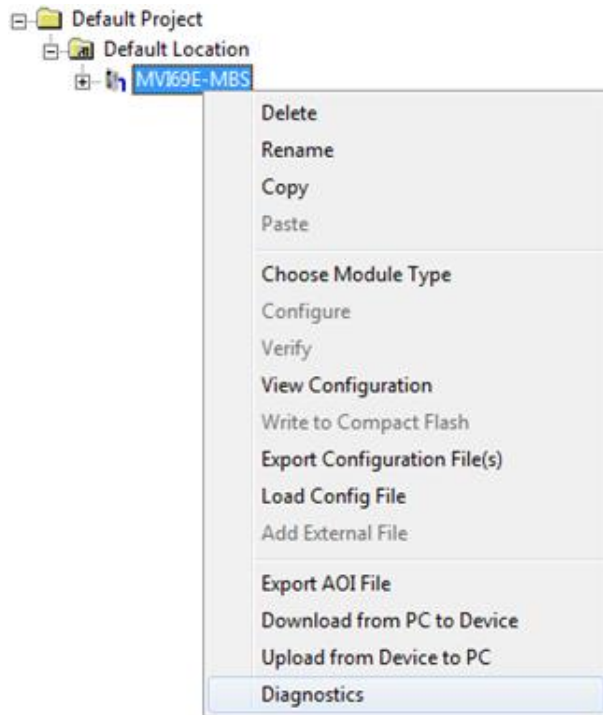
Important: The temporary IP address is only valid until the next time the module is initialized. For information on how to set the module's permanent IP address, see *Ethernet 1* (page 46).

- 6 Close the *ProSoft Discovery Service* window. Enter the temporary IP address in the **ETHERNET ADDRESS** field of the *Connection Setup* dialog box, then click **TEST CONNECTION** to verify that the module is accessible with the current settings.
- 7 If the *Test Connection* is successful, click **CONNECT**. The *Diagnostics* window is now accessible.

7.6 Using the Diagnostics Menu in PCB

ProSoft Configuration Builder (PCB) provides diagnostic menus for debugging and troubleshooting.

- 1 In the tree view in ProSoft Configuration Builder, right-click the **MVI69E-MBS** module and then choose **DIAGNOSTICS**. For instructions on opening and using a project in PCB, please refer to *Configuring the MVI69E-MBS Using PCB* (page 35).

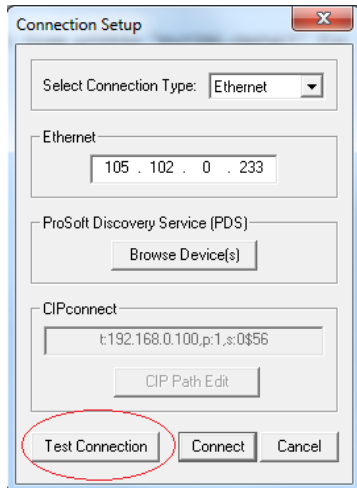


- 2 After the *Diagnostics* window opens, click the **SET UP CONNECTION** button to browse for the module's IP address.

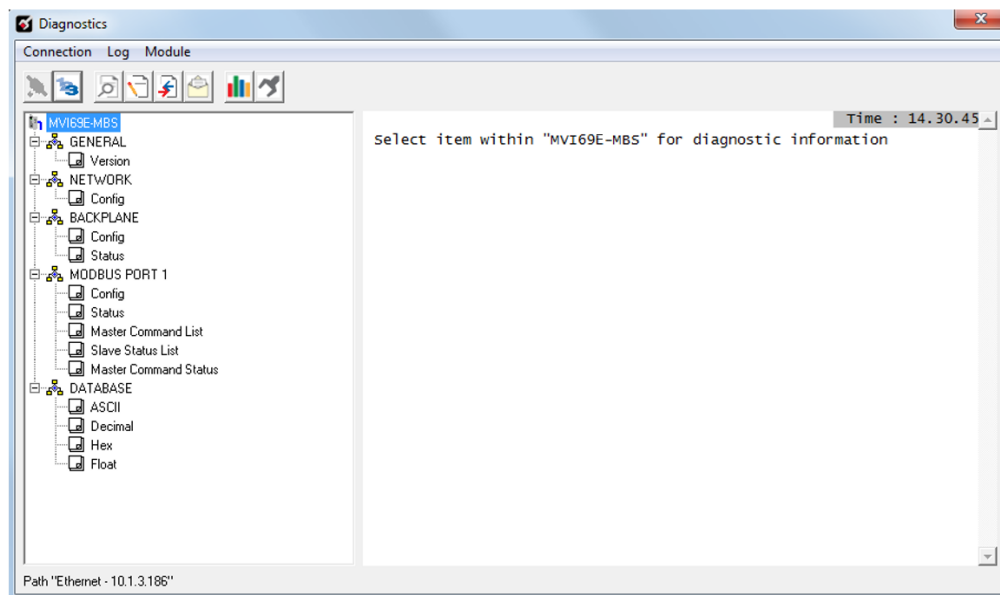


Click to set up connection

- 3 In the *Ethernet* field of the *Connection Setup* dialog box, enter the current IP address, whether it is temporary or permanent. Click **TEST CONNECTION** to verify that the module is accessible with the current settings.

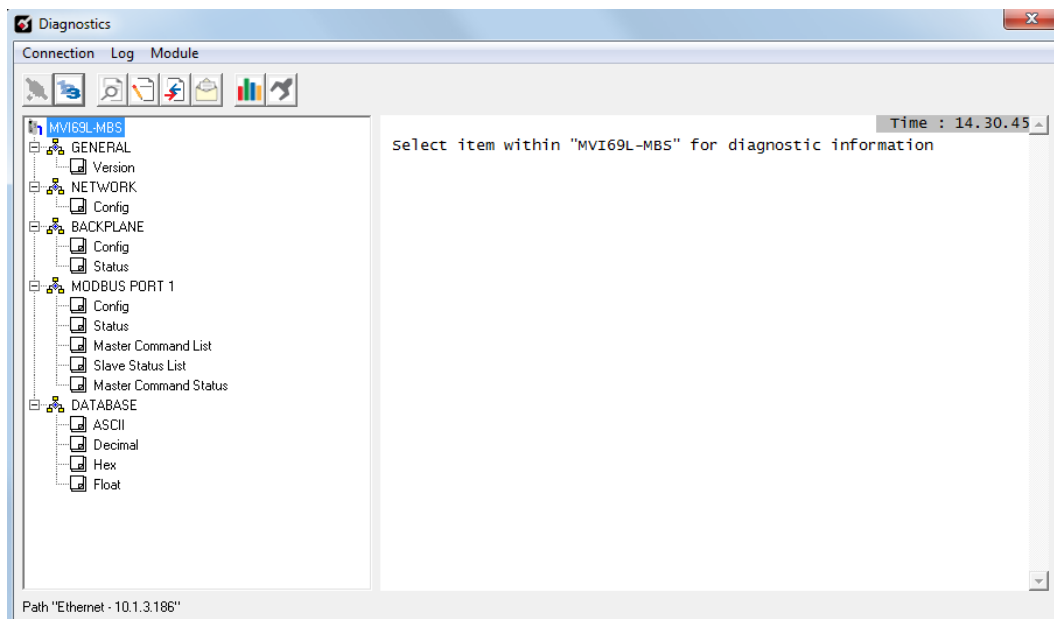


- 4 If the **TEST CONNECTION** is successful, click **CONNECT**. The *Diagnostics* window is now visible.



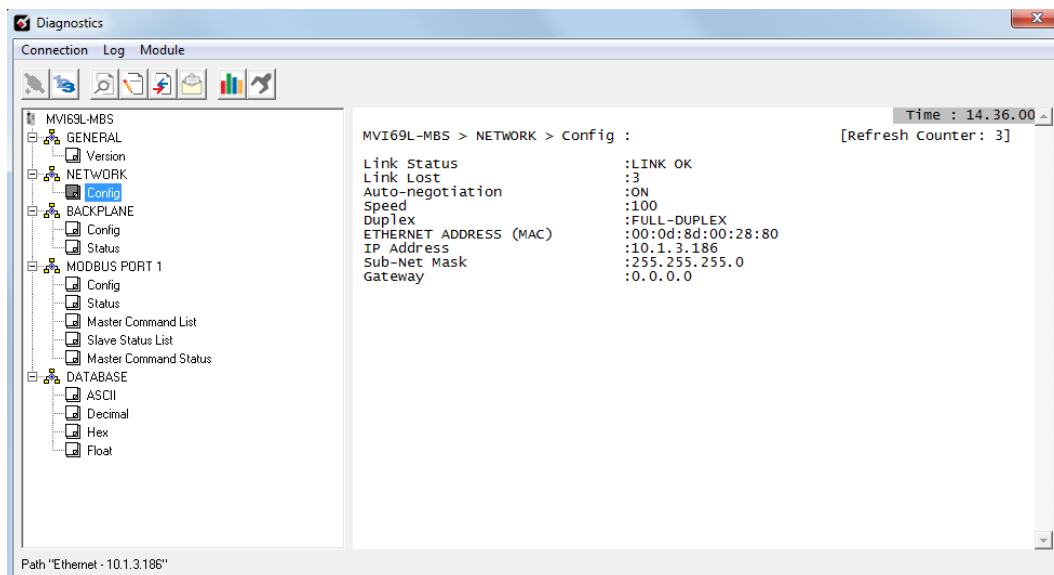
7.6.1 Diagnostics Menu

In the *Diagnostics* window in ProSoft Configuration Builder, the Diagnostics menu is available through the Ethernet configuration port. The menu is arranged as a tree structure.



7.6.2 Monitoring Network Configuration Information

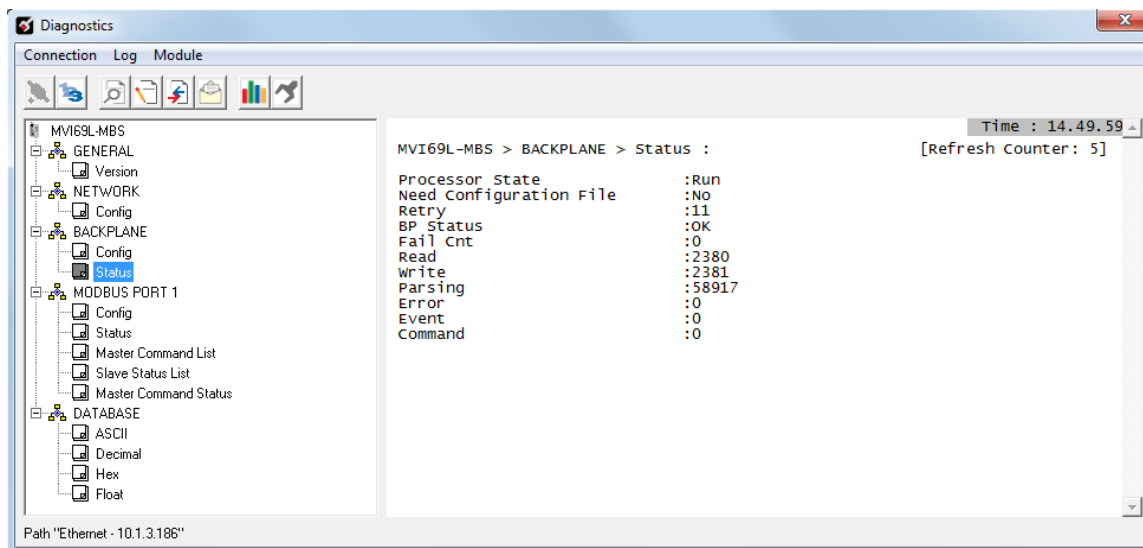
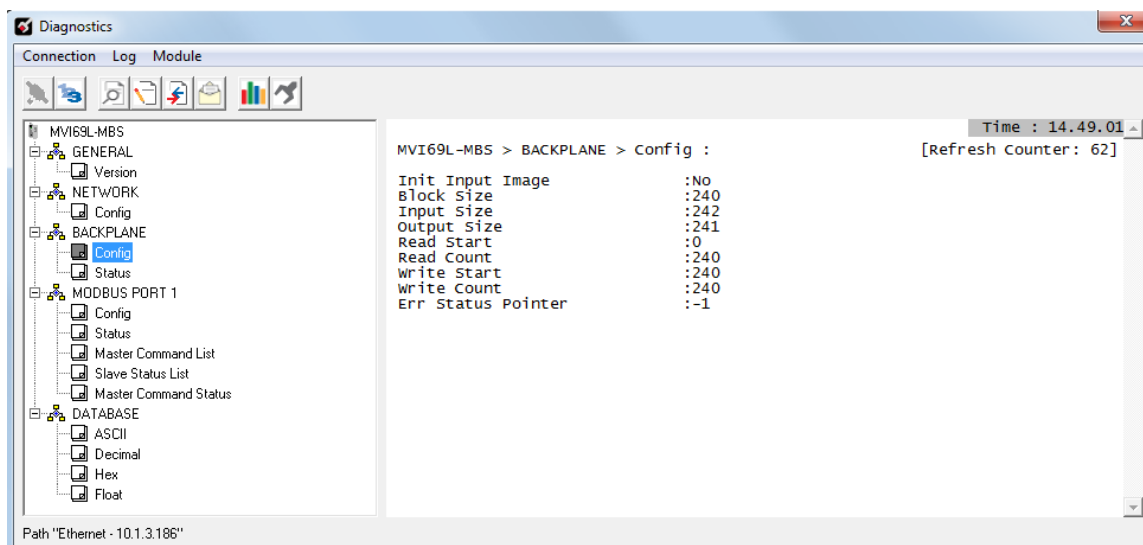
In the *Diagnostics* window in Prosoft Configuration Builder, click **NETWORK** and then click **CONFIG** to view the Ethernet network configuration information.



7.6.3 Monitoring Backplane Information

In the *Diagnostics* window in ProSoft Configuration Builder, click **BACKPLANE** to view the backplane information. This menu has two submenus:

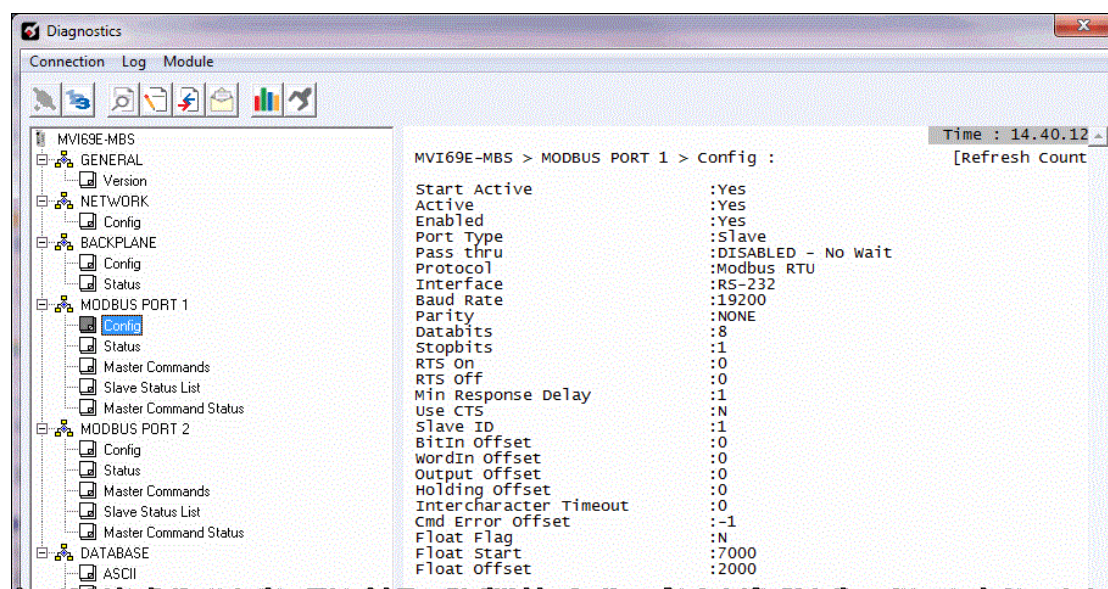
- **CONFIGURATION**
- **STATUS**



7.6.4 Port x Module Information

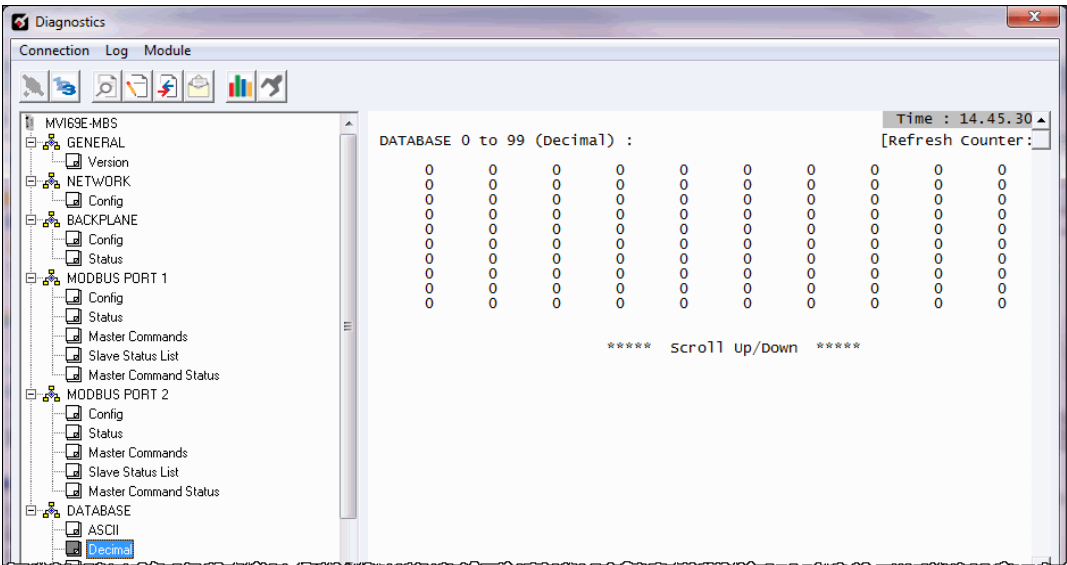
The **MODBUS PORT 1** and **MODBUS PORT 2** menus include the following submenus:

- Configuration
- Status (General status for the port)
- Master Commands (Used when port is configured as a Modbus master)
- Slave Status List (Status of each slave on the network, used when port is configured as a Modbus master)
- Master Command Status (Status code for each master command, used when port is configured as a Modbus master)



7.6.5 Monitoring Data Values in the Module’s Database

In the *Diagnostics* window in ProSoft Configuration Builder, click **DATABASE** and then click **DECIMAL** to view the contents of the MVI69E-MBS module’s internal database. You can view data values in ASCII, Hexadecimal, and Float format.



7.7 Communication Error Codes

Note: If an error code is reported that is not listed below, check with the documentation of the Modbus device(s) on the module's application ports. Modbus devices can produce device-specific error codes.

7.7.1 Standard Modbus Protocol Exception Code Errors

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

7.7.2 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

7.7.3 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

7.8 Connecting to the Module's Webpage

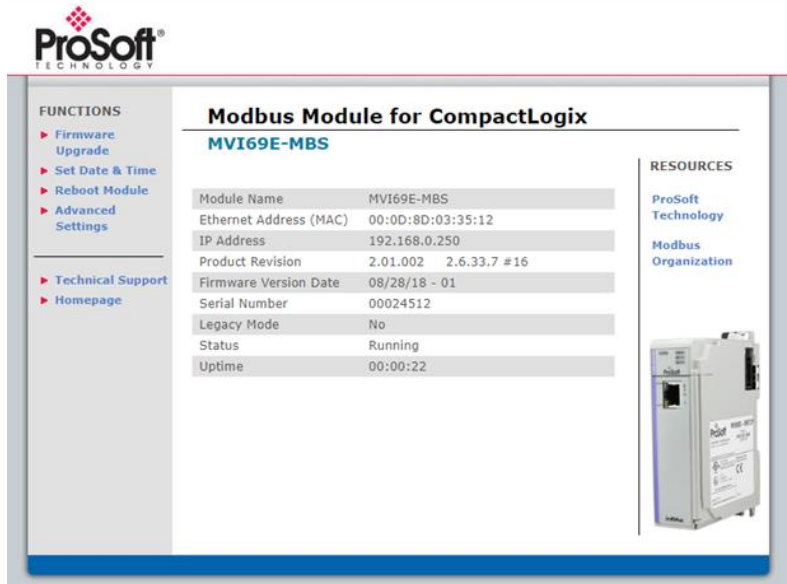
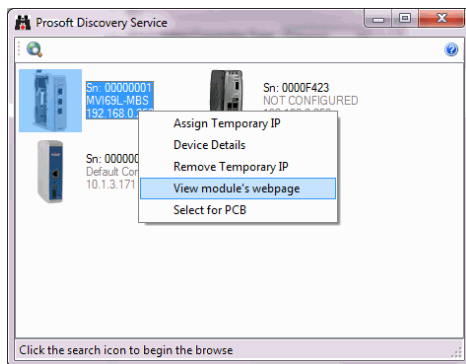
The module's internal web server provides access to module version and status information, as well as the ability to set the date and time, reboot the module, and download firmware upgrade to the module. Enter the assigned IP address of the module into a web browser or use the following steps in PCB.

- 1 In the PCB Diagnostics window, click the **SET UP CONNECTION** button.



Click to set up connection

- 2 In the *Connection Setup* dialog box, click **BROWSE DEVICE(S)** to start *ProSoft Discovery Service*.
- 3 Right-click the module icon and choose **VIEW MODULE'S WEBPAGE** to launch your default browser and display the module's webpage.



8 Reference

8.1 Product Specifications

The MVI69E-MBS allows Rockwell Automation® CompactLogix or MicroLogix 1500-LRP® I/O compatible processors to interface easily with other Modbus protocol compatible devices.

The module acts as an input/output communications module between the Modbus network and the CompactLogix or MicroLogix 1500-LRP backplane. The data transfer from the CompactLogix or MicroLogix 1500-LRP processor is asynchronous from the actions on the Modbus network. Databases are user-defined and stored in the module to hold the data required by the protocol.

- Single-slot, 1769 backplane-compatible
- The module is recognized as an Input/Output module and has access to processor memory for data transfer between processor and module.
- Ladder Logic is used for data transfer between module and processor. Sample Add-On Instruction file included.
- Configuration data obtained from and stored in the processor.
- Supports CompactLogix or MicroLogix 1500-LRP processors with 1769 I/O bus capability and at least 500 mA of 5 VDC backplane current available.

8.1.1 Hardware Specifications

Specification	Description
Dimensions	Standard 1769 Single-slot module
Current Load	500 mA max @ 5 VDC Power supply distance rating of 4 (L43 and L45 installations on first 4 slots of 1769 bus)
Operating Temp.	32° F to 140° F (0° C to 60° C)
Storage Temp.	-40° F to 185° F (-40° C to 85° C)
Relative Humidity	5% to 95% (with no condensation)
LED Indicators	Module OK Status Backplane Activity Ethernet Port Activity Configuration Activity
CFG Port (ETH)	Diagnostics over Ethernet connection
App Ports (P1, P2)	RS-232, RS-485 or RS-422 (jumper selectable) RJ45 Port (DB-9F with supplied cable) 500V Optical isolation from backplane
Shipped with Unit	RJ45 to DB-9M cables for each application port

8.1.2 General Specifications - Modbus Master/Slave

Specification	Description														
Communication Parameters	Baud rate: 110 to 115K baud Stop bits: 1 or 2 Data size: 7 or 8 bits Parity: None, Even, Odd RTS timing delays: 0 to 65535 milliseconds														
Modbus Modes	RTU mode (binary) with CRC-16 ASCII mode with LRC error checking														
Floating-Point Data	Floating-point data movement supported, including configurable support for Enron, Daniel®, and other implementations														
Modbus Function Codes Supported	<table> <tr> <td>1: Read Coil Status</td><td>15: Force(Write) Multiple Coils</td></tr> <tr> <td>2: Read Input Status</td><td>16: Preset (Write) Multiple Holding Registers</td></tr> <tr> <td>3: Read Holding Registers</td><td></td></tr> <tr> <td>4: Read Input Registers</td><td>17: Report Slave ID (Slave Only)</td></tr> <tr> <td>5: Force (Write) Single Coil</td><td>22: Mask Write Holding Register (Slave Only)</td></tr> <tr> <td>6: Preset (Write) Single Holding Register</td><td>23: Read/Write Holding Registers (Slave Only)</td></tr> <tr> <td>8: Diagnostics (Slave Only, Responds to Subfunction 00)</td><td></td></tr> </table>	1: Read Coil Status	15: Force(Write) Multiple Coils	2: Read Input Status	16: Preset (Write) Multiple Holding Registers	3: Read Holding Registers		4: Read Input Registers	17: Report Slave ID (Slave Only)	5: Force (Write) Single Coil	22: Mask Write Holding Register (Slave Only)	6: Preset (Write) Single Holding Register	23: Read/Write Holding Registers (Slave Only)	8: Diagnostics (Slave Only, Responds to Subfunction 00)	
1: Read Coil Status	15: Force(Write) Multiple Coils														
2: Read Input Status	16: Preset (Write) Multiple Holding Registers														
3: Read Holding Registers															
4: Read Input Registers	17: Report Slave ID (Slave Only)														
5: Force (Write) Single Coil	22: Mask Write Holding Register (Slave Only)														
6: Preset (Write) Single Holding Register	23: Read/Write Holding Registers (Slave Only)														
8: Diagnostics (Slave Only, Responds to Subfunction 00)															

8.2 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 MVI69E-MBS module can work as a Master and as a Slave.

The MVI69E-MBS module also works as an input/output module between itself and the Rockwell Automation backplane and CompactLogix or MicroLogix 1500-LRP processor. The module uses an internal database to pass data and commands between the processor and Master and Slave devices on Modbus networks.

8.2.1 Modbus Master

A port configured as a virtual Modbus Master actively issues Modbus commands to other nodes on the Modbus network, supporting up to 250 commands on each port. The Master ports have an optimized polling characteristic that polls slaves with communication problems less frequently.

Parameter	Description
Command List	Up to 250 commands 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.

8.2.2 Modbus Slave

A port configured as a Modbus slave permits a remote Master to interact with all data contained in the module. This data can be derived from other Modbus slave devices on the network, through a Master port, or from the CompactLogix or MicroLogix 1500-LRP processor.

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

8.2.3 Function Codes Supported by the Module

The format of each command in the list depends on the Modbus Function Code being executed. The following table lists the Function Codes supported by the MVI69E-MBS module.

Function Code	Definition	Supported as Master	Supported as Slave
1	Read Coil Status 0x	X	X
2	Read Input Status 1x	X	X
3	Read Holding Registers 4x	X	X
4	Read Input Registers 3x	X	X
5	Set Single Coil 0x	X	X
6	Single Register Write 4x	X	X
8	Diagnostics		X
15	Multiple Coil Write 0x	X	X
16	Multiple Register Write 4x	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.

8.2.4 Read Coil Status (Function Code 01)

8.2.4.1 Query

This function allows you to obtain the ON/OFF status of logic coils (Modbus 0x range) used to control discrete outputs from the addressed 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 are 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 (37 coils) from slave device number 11.

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

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

8.2.4.2 Response

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

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

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

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

8.2.5 Read Input Status (Function Code 02)

8.2.5.1 Query

This function allows you to obtain the ON/OFF status of discrete inputs (Modbus 1x range) 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 are 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 from zero; (input 10001 = zero, input 10002 = one, input 10003 = two, and so on, for a 584).

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

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

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

8.2.5.2 Response

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

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

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

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

Due to the quantity of input statuses requested, the last data field which is shown as 35 HEX = 0011 0101 (binary) contains the status of only 6 inputs (10213 to 10218) instead of 8 inputs. The two left-most bits are provided as zeros to fill the 8-bit format.

8.2.6 Read Holding Registers (Function Code 03)

8.2.6.1 Query

This function allows you to retrieve the contents of holding registers 4xxxx (Modbus 4x range) 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 retrieving up to 125 registers at each request; however, the specific slave device may have restrictions that lower this maximum quantity. The registers are numbered from zero (40001 = zero, 40002 = one, and so on). The broadcast mode is not allowed.

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

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

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

8.2.6.2 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 reflect the register content at the end of the scan. Some slaves limit the quantity of register content provided each scan; thus for large register quantities, multiple transmissions are made using register content from sequential scans.

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

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

8.2.7 Read Input Registers (Function Code 04)

8.2.7.1 Query

This function retrieves the contents of the controller's input registers from the Modbus 3x range. These locations receive their values from devices connected to the I/O structure and can only be referenced, not altered from within the controller. The addressing allows retrieving up to 125 registers at each request; however, the specific 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 30009 in slave number 11.

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

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

8.2.7.2 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 reflect the register content at the end of the scan. Each PC limits the quantity of register contents provided each scan; thus for large register quantities, multiple PC scans are required, and the data provided is from sequential scans.

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

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

8.2.8 Force Single Coil (Function Code 05)

8.2.8.1 Query

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

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

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

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

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

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

8.2.8.2 Response

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

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

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

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

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

8.2.9 Preset Single Register (Function Code 06)

8.2.9.1 Query

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

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

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

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

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

8.2.9.2 Response

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

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

8.2.10 Diagnostics (Function Code 08)

This function provides a series of tests for checking the communication system between a master device and a slave, or for checking various internal error conditions within a slave.

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

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

A server device can, however, be forced into 'Listen Only Mode' in which it monitors 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.

8.2.10.1 Sub-function Codes Supported

Only Sub-function 00 is supported by the MVI69E-MBS module.

8.2.10.1.1 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.

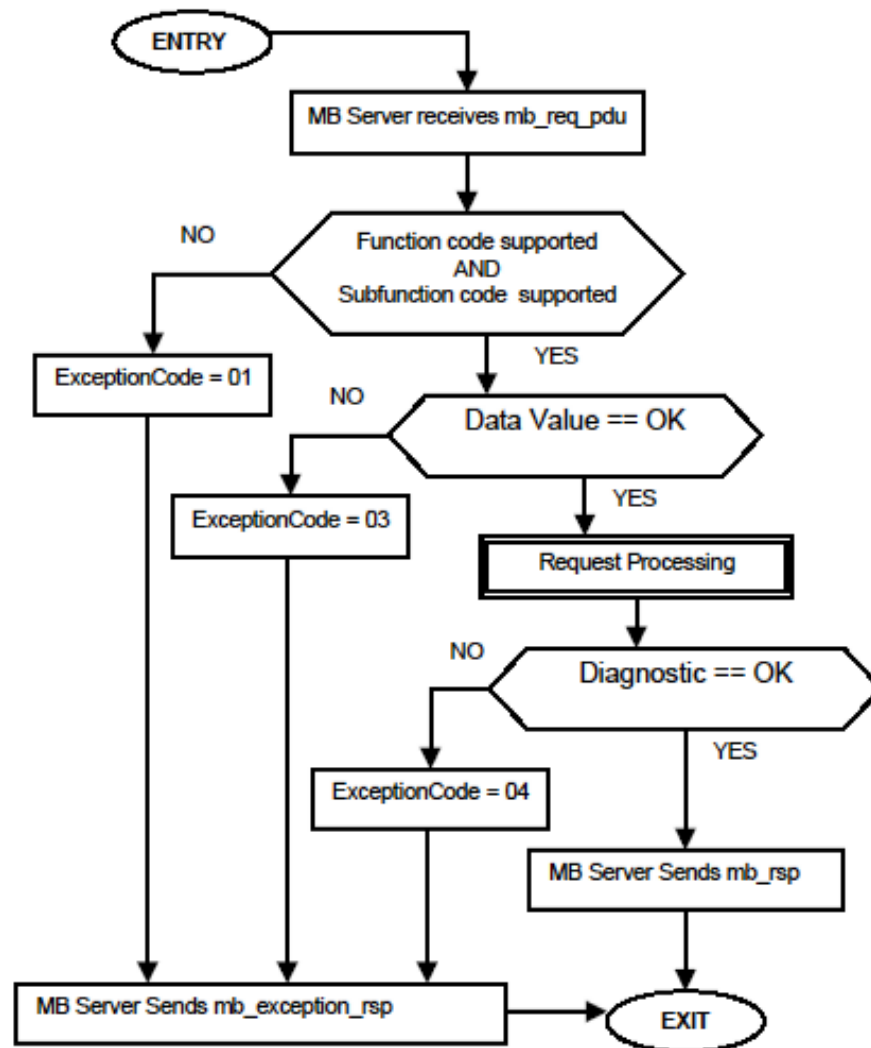
Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

8.2.10.1.2 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.



8.2.11 Force Multiple Coils (Function Code 15)

8.2.11.1 Query

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

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

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

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

8.2.11.2 Response

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

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

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

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

8.2.12 Preset Multiple Registers (Function Code 16)

8.2.12.1 Query

This Function Code allows you to modify the contents of a Modbus 4x range in the slave. This writes up to 125 registers at time. Since the controller is actively scanning, it also can alter the content of any holding register at any time.

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

The example below is a request to write 2 registers starting at register 40002 in slave 11.

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

Node Address	Func Code	Data Start Address High	Data Start Address Low	Number of Points High	Number of Points Low	Byte Count	Data High	Data Low	Data High	Data Low	Error Check Field (2 bytes)
0B	10	00	01	00	02	04	00	0A	01	02	CRC

8.2.12.2 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.

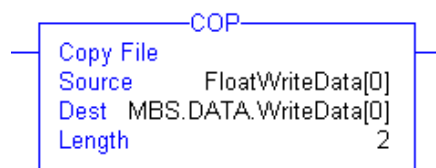
Node Address	Func Code	Data Start Address High	Data Start Address Low	Number of Points High	Number of Points Low	Error Check Field (2 bytes)
0B	10	00	01	00	02	CRC

8.3 Floating-Point Support

You can easily move floating point data between the MBS module and other devices as long as the device supports IEEE 754 Floating Point format. This IEEE format is a 32-bit single-precision floating-point format.

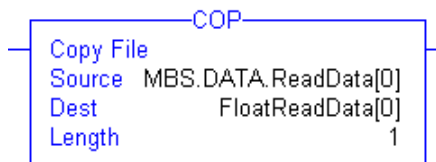
The logic necessary to move the floating-point data takes advantage of the COP instruction in Studio 5000. The COP instruction is unique for data movement commands in that it is an untyped function, meaning that no data conversion is done when data is moved between controller tags with different data types (that is, it is an image copy, not a value copy).

The COP instruction to move data from a floating-point controller tag into an integer controller tag (something you would do to move floating-point values to the module) is shown below.



This instruction moves one floating-point value in two 16-bit integer images to *MBS.DATA.WriteData[0]*, which is an integer tag. For multiple floating-point values increase the *Length* field by a factor of 2 per floating-point value.

The COP instruction to move data from *MBS.DATA.ReadData[0]*, which is an integer tag, to a floating-point tag (something you would do to receive floating-point values from the module) is shown below.



This instruction moves two 16-bit integer registers containing one floating point value image into the floating-point tag. For multiple values increase the *Length* field.

8.3.1 Enron Floating Point Support

Many manufacturers have implemented special support in their drivers for what is commonly called the Enron version of the Modbus protocol. In this implementation, addresses greater than 7000 are presumed to contain floating-point values. The significance to this is that the count descriptor for a data transfer now denotes the number of floating-point values to transfer, instead of the number of words.

8.3.2 Configuring the Floating-Point Data Transfer

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

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

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

8.3.2.1 Example #1

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

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

If the slave device uses addressing as shown in Example #2, then you do not need to do anything with the *Float Flag* or *Float Start* parameters, as this addressing scheme uses two Modbus addresses to represent each floating-point value:

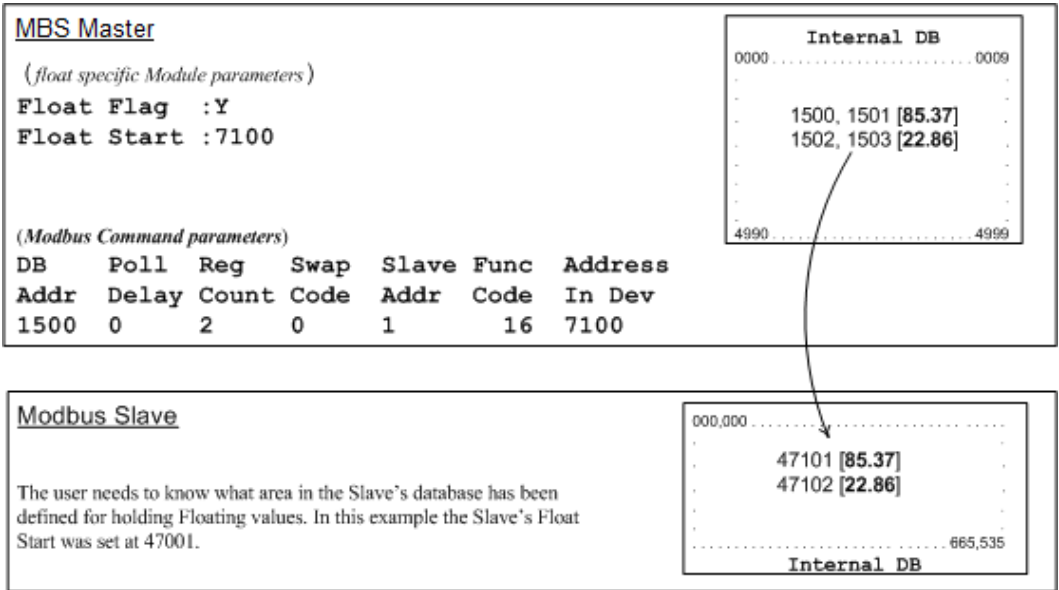
8.3.2.2 Example #2

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

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

8.3.2.3 Specific Examples:

8.3.2.3.1 Example #1: Master is issuing Modbus command with FC 16 (with Float Flag: Yes) to transfer Float data to slave



(Float specific module parameters)

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

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

(Modbus Command parameters)

DB Addr - Tells the master where in its data memory is the beginning of data to obtain and write out to the slave device.

Reg Count - Tells the master how many data points to send to the slave. Two counts mean two floating points with Float Flag: Y and the *Addr in Dev* greater than or equal to the *Float Start* Parameter.

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

Func Code - Tells the master to write the float values to the slave. FC16.

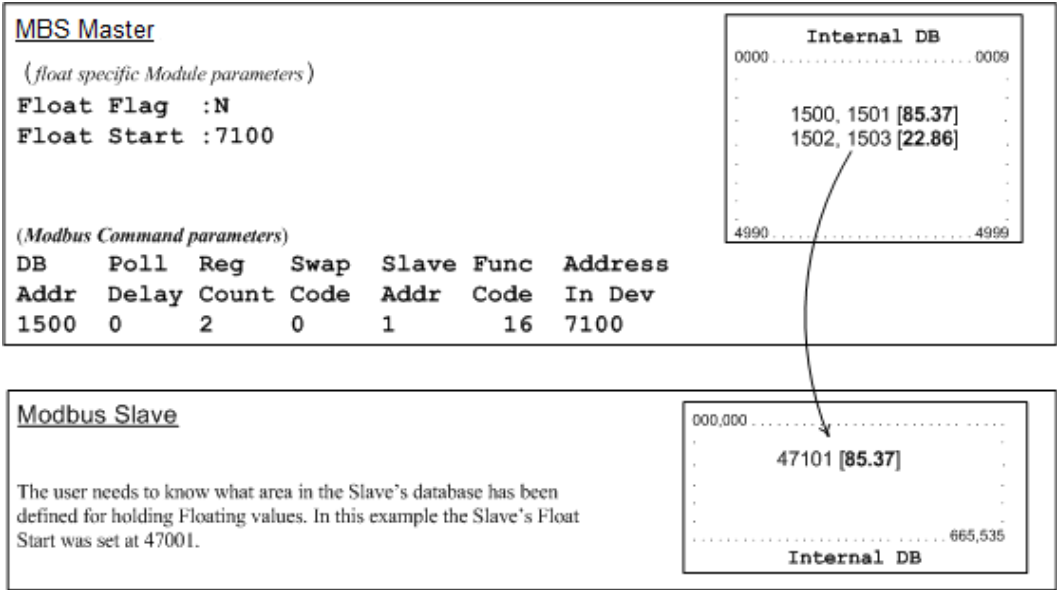
Addr in Dev - Tells the master where in the slave's database to locate the data.

In the above example, the master's Modbus command to transmit inside the Modbus packet is as follows.

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

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

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



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

Float Start: Ignored.

DB Addr - same as when Float Flag: Y.

Reg Count - Tells the master how many data points to send to the slave.

Swap Code - same as when Float Flag: Y.

Func Code - same as when Float Flag: Y.

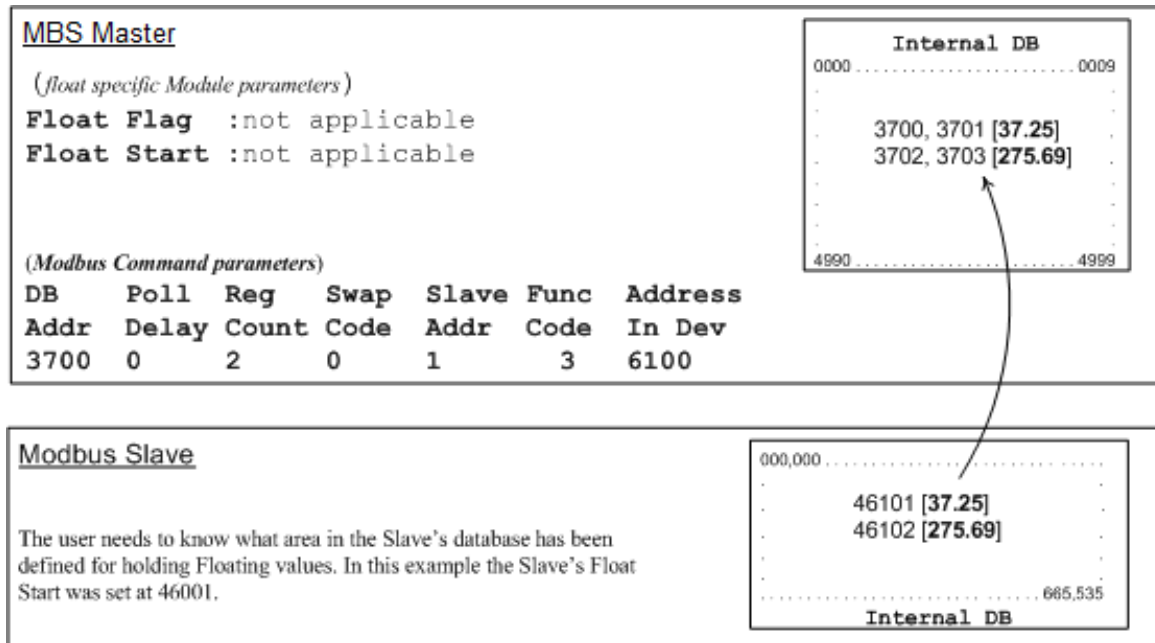
Addr in Dev - same as when Float Flag: Y as long as the slave's Float Flag = Y.

In the above example, the master's Modbus command to transmit inside the Modbus packet is as follows.

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

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

8.3.2.3.3 Example #3: Master is issuing Modbus command with FC 3 to transfer Float data from slave



Float Flag: Not applicable with Modbus Function Code 3.

Float Start: Not applicable with Modbus Function Code 3.

DB Addr - Tells the master where in its data memory to store the data obtained from the slave.

Reg Count - Tells the master how many registers to request from the slave.

Swap Code - Same as above.

Func Code - Tells the master to read the register values from the slave. FC3.

Addr in Dev - Tells the master where in the slave's database to obtain the data.

In the above example, the master's Modbus command to transmit inside the Modbus packet is as follows:

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

The (Enron/Daniel supporting) slave's Modbus command to transmit inside the Modbus packet is as follows:

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

The (a NON-Enron/Daniel supporting) slave's Modbus command that is transmitted inside the Modbus packet is as follows:

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

8.4 Function Blocks

Data contained in this database is paged through the input and output images by coordination of the CompactLogix or MicroLogix 1500-LRP ladder logic and the MVI69E-MBS module's program. Each block transferred from the module to the processor or from the processor to the module contains a block identification code that describes the content of the block.

Block ID Range	Description
-1000 to -1166	Get input image data for initialization
-1 to -999	Dummy block
0	Read or write data for small data sets
1 to 167	Read or write data
1000 to 1255	Event Command Port 1
2000 to 2255	Event Command Port 2
3000 to 3001	Port 1 slave polling control
3002 to 3006	Port 1 slave status
3100 to 3101	Port 2 slave polling control
3102 to 3106	Port 2 slave status
5001 to 5006	Port 1 Command Control
5101 to 5106	Port 2 Command Control
8000	Add Event with data for Port 1
8001	Add Event with data for Port 2
8100	Get Event with data status
9000 or -9000	Specifications of configuration file data from the processor to the module
9001 or -9001	Get configuration file from the processor to the module (continued)
9250	Get general module status data
9500	Set port and command active bits
9501	Get port and command active bits
9956	Pass-through formatted block for functions 6 and 16 with word data
9957	Pass-through formatted block for functions 6 and 16 with float data
9958	Pass-through formatted block for function 5
9959	Pass-through formatted block for function 15
9961	Pass-through formatted block for function 23
9970	Pass-through block for function 99
9972	Set module time using received time
9973	Pass module time to processor
9997	Reset status block
9998	Warm-boot control block
9999	Cold-boot control block

8.4.1 Event Command Blocks (1000 to 1255, 2000 to 2255)

Blocks 1000 to 1255: Event Port 1

Blocks 2000 to 2255: Event Port 2

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

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

8.4.1.1 Blocks 1000 to 1255 or 2000 to 2255: Request from Processor to Module

Offset	Description
0	Write Block ID: 1000 to 1255 for a Port 1 command or 2000 to 2255 for a Port 2 command. The last 3 digits of the command specify the slave address to use for the command.
1	Internal address in the module to be used with the command.
2	Count parameter that determines the number of digital points or registers to associate with the command.
3	Swap type for the data.
4	Modbus Function Code to be associated with the command.
5	Modbus address in the slave device to be used in the command.
6 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.1.2 Blocks 1000 to 1255 or 2000 to 2255: Response from Module to Processor

Offset	Description
0	Read Block ID: 1000 to 1255 or 2000 to 2255 requested by the processor.
1	Write Block ID: To be used by the processor in its next Write block
2	Result of the event request. 1 = the command was placed in the command queue; 0 = no room was found in the command queue.
3	Number of commands in the command queue for the specified port.
4 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.2 Slave Polling Disable Blocks

Block 3000: Port 1 Slave Polling Disable

Block 3100: Port 2 Slave Polling Disable

These blocks allow the processor to disable polling for specific slaves.

8.4.2.1 Block 3000 or 3100: Request from Processor to Module

Offset	Description
0	Write Block ID: 3000 for Port 1 and 3100 for Port 2 slave polling disable request.
1	Number of slaves listed in the block (1 to 60).
2 to 61	Slave indexes to disable in the command list for the selected port. The number of slaves to process is set in Word 1 of the block.

8.4.2.2 Block 3000 or 3100: Response from Module to Processor

Offset	Description
0	Read Block ID: 3000 or 3100 requested by the processor.
1	Write Block ID: To be used by the processor in its next Write block.
2	Number of slaves processed in the last request. This number should match the value passed in Word 1 of the request block.
3 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.3 Slave Polling Enable Blocks (3001, 3101)

Block 3001: Port 1 Slave Polling Enable

Block 3101: Port 2 Slave Polling Enable

These blocks allow the processor to enable polling for specific slaves.

8.4.3.1 Block 3001 or 3101: Request from Processor to Module

Offset	Description
0	Write Block ID: 3001 for Port 1 or 3101 for Port 2 slave polling enable request.
1	Number of slaves listed in the block (1 to 60).
2 to 61	Slave indexes to enable in the command list for the selected port. The number of slaves to process is set in Word 1 of the block.

8.4.3.2 Block 3001 or 3101: Response from Module to Processor

Offset	Description
0	Read Block ID: 3000 to 3101 or 3100 to 3101 requested by the processor.
1	Write Block ID: To be used by the processor in its next Write block.
2	Number of slaves processed in the last request. This number should match the value passed in Word 1 of the request block.
3 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.4 Slave Polling Status Blocks (3002 to 3006, 3102 to 3106)

Blocks 3002 to 3006: Port 1 Slave Status

Blocks 3102 to 3106: Port 2 Slave Status

Two arrays are allocated in the module's primary object to hold the polling status of each slave on the Master ports. You can use this status data to determine which slaves are currently active on the port, in communication error, or have their polling suspended and disabled.

8.4.4.1 Block 3002 to 3006 or 3102 to 3106: Request from Processor to Module

Offset	Description
0	Write Block ID: 3002 to 3006 for Port 1 and 3102 to 3106 for Port 2 slave polling status request.
1 to (n-1)	Spare

Where n = 60, 120, or 240 depending on the Block Transfer Size parameter.

Block 3002 to 3006 or 3102 to 3106: Response from Module to Processor

Offset	Description
0	Read Block ID: 3002 to 3006 or 3102 to 3106 requested by the processor.
1	Write Block ID: To be used by the processor in its next Write block.
2	Slave ID offset: Index of first slave in block
3	Number of slaves in this block
4 to 61	Slave polling status data
62 to (n-1)	Spare

Slave Status values

Value	Description
0	OK
1	Exceeded retry count and in error delay count mode
2	Block 3000 or 3100

8.4.5 Command Control Blocks (5001 to 5006, 5101 to 5106)

Blocks 5001 to 5006: Port 1 Command Control

Blocks 5101 to 5106: Port 2 Command Control

If the CompactLogix or MicroLogix 1500-LRP processor sends a command control block, the module places the commands referenced in the block in the command queue. Commands placed in the queue with this method need not have their enable bit set. Only valid commands are placed in the queue.

Up to 6 commands can be enabled and placed in the command queue with one write request from the CompactLogix or MicroLogix 1500-LRP processor.

8.4.5.1 Blocks 5001 to 5006 or 5101 to 5106: Request from Processor to Module

Offset	Description
0	Write Block ID: 5001 to 5006 for Port 1 or 5101 to 5106 for Port 2. The last digit indicates how many commands are to be placed in the command queue by this block.
1	Index in the command list for the first command to be entered into the command queue (applies to blocks 5001 to 5006 and 5101 to 5106).
2	Index for the second command (applies to blocks 5002 to 5006 and 5102 to 5106).
3	Index for the third command (applies to blocks 5003 to 5006 and 5103 to 5106).
4	Index for the fourth command (applies to blocks 5004 to 5006 and 5104 to 5106).
5	Index for the fifth command (applies to blocks 5005 to 5006 and 5105 to 5106).
6	Index for the sixth command (applies to blocks 5006 and 5106).
7 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.5.2 Blocks 5001 to 5006 or 5101 to 5106: Response from Module to Processor

Offset	Description
0	Read Block ID: 5001 to 5006 or 5101 to 5106 requested by the processor.
1	Write Block ID: To be used by the processor in its next Write block.
2	Number of commands in the block placed in the command queue.
3	Number of commands in the command queue for the specified port.
4 to (n-1)	Spare

8.4.6 Add Event with Data Blocks (8000, 8001)

Block 8000: Add Event with Data for Port 1

Block 8001: Add Event with Data for Port 2

The 8000-series blocks are similar to the 1000 and 2000-series blocks. The 8000-series blocks source the command data from the processor, instead of from the module's database.

8.4.6.1 Block 8000 or 8001: Request from Processor to Module

Offset	Description
0	Write Block ID: 8000 for Port 1 or 8001 for Port 2 event command with data request
1	Slave address of Modbus device to reach with the command request
2	Modbus function code to use with command (5, 6, 15 or 16)
3	Modbus address in slave device
4	Count value for operation- bit count for function 15 (1 to 1968 points) and word count for function 16 (1 to 50 words or 1 to 25 float values). For functions 5 and 6, the count is assumed to be 1.
5 to 54	Data to be used by command
55 to (n-1)	Spare

Where n = 60, 120, or 240 depending on the Block Transfer Size parameter.

8.4.6.2 Block 8000 or 8001: Response from Module to Processor

Offset	Description
0	Read Block ID: 8000 for Port 1 or 8001 for Port 2 event command with data request
1	Write Block ID: To be used by the processor in its next Write block.
2	Error Code for request: 0=no error -1=port is not enabled -2=port is not a master port -3=port is not active (enabled) -4=port busy with previous event command -5=invalid Modbus command -6=invalid point count for command
3 to (n-1)	Spare

8.4.7 Get Event with Data Status Block (8100)

Block 8100: Get Event with Data Status

This block requests status data for Event with Data Commands.

8.4.7.1 Block 8100: Request from Processor to Module

Offset	Description
0	Write Block ID: 8100 status data request for Event with Data Commands.
1 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.7.2 Block 8100: Response from Module to Processor

Offset	Description
0	Read Block ID: 8100 status data for Event with Data Commands.
1	Write Block ID: To be used by the processor in its next Write block.
2	Event command status for Port 1: 0 = No message active 1 = Waiting to execute command 2 = Command complete
3	Error code for last command executed for Port 1
4	Event command status for Port 2: 0 = No message active 1 = Waiting to execute command 2 = Command complete
5	Error code for last command executed for Port 2
6 to (n-1)	Spare

8.4.8 Get Configuration File Information Block (9000 or -9000)

Block 9000 or -9000: Get Configuration File Information

This block requests information from the processor about the configuration file, in preparation for transferring the configuration file from the processor to the module. It specifies the location in the configuration file to start copying and sending the information.

8.4.8.1 Block 9000 or -9000: Request from Module to Processor

Offset	Description
0	Read Block ID: 9000 or -9000 request for configuration file information from processor
1	Write Block ID: 9000 or -9000 to be used by the processor in its next Write block.

8.4.8.2 Block 9000 or -9000: Response from Processor to Module

Offset	Description
0	Write Block ID: 9000 or -9000 configuration file information
1	Module's slot number.
2	Size of the module's Input image to the processor.
3	Size of the module's Output image from the processor.
4	Status of configuration file.
5-6	These two registers contain the size of the configuration file in bytes.
7-8	These two registers contain the CRC for the configuration file.

8.4.9 Get Configuration File Block (9001 or -9001)

Block 9001 or -9001: Get Configuration File Information

This block requests the configuration file from the processor. The module returns the requested contents of the configuration file.

8.4.9.1 Block 9001 or -9001: Request from Module to Processor

Offset	Description
0	Read Block ID: 9001 or -9001 request for configuration file from processor
1	Write Block ID: 9001 or -9001 to be used by the processor in its next Write block.
2-3	File offset: Offset of the first register in the configuration file to begin transferring data from. If the size of the configuration file exceeds the block transfer size, the file is transferred in multiple blocks, and the file offset tells the processor which part of the configuration file is being requested by the individual block.
4-5	Number of bytes of the configuration file to include in next block
6-7	Copy of the data contained in registers 2 to 3.

8.4.9.2 Block 9001 or -9001: Response from Processor to Module

Offset	Description
0	Write Block ID: 9001 or -9001 configuration file data
1-2	File offset: Same as registers 2-3 of the previous request block
3-4	Data length: Same as registers 4-5 of the previous request block
5 to (n-1)	Contents of configuration file. If the size of the configuration file exceeds the block transfer size, this information is transferred in multiple blocks.

Where n = 60, 120, or 240 depending on the Block Transfer Size parameter.

8.4.10 Get General Module Status Data Block (9250)

Block 9250: Get General Module Status Data

This block requests the general module status.

8.4.10.1 Block 9250: Request from Processor to Module

Offset	Description
0	Write Block ID: 9250 request for general module status
1 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.10.2 Block 9250: Response from Module to Processor

Offset	Description
0	Read Block ID: 9250 requested by processor.
1	Write Block ID: To be used by the processor in its next Write block.
2	Program Scan Count: This value is incremented when a complete program cycle occurs.
3-4	Product Code: These two registers contain the product code of "MB6E" for the MVI69E-MBS.
5-6	Product Version: These two registers contain the product version for the current running software.
7-8	Operating System: These two registers contain the month and year values for the program operating system.
9-10	Run Number: These two registers contain the run number value for the currently running software.
11	Port 1 Command List Requests: Number of requests made from this port to slave devices on the network.
12	Port 1 Command List Response: Number of slave response messages received on the port.
13	Port 1 Command List Errors: Number of command errors processed on the port. These errors could be due to a bad response or command.
14	Port 1 Requests: Total number of messages sent out of the port.
15	Port 1 Responses: Total number of messages received on the port.
16	Port 1 Errors Sent: Total number of message errors sent out of the port.
17	Port 1 Errors Received: Total number of message errors received on the port.
18	Port 2 Command List Requests: Number of requests made from this port to slave devices on the network.
19	Port 2 Command List Response: Number of slave response messages received on the port.
20	Port 2 Command List Errors: Number of command errors processed on the port. These errors could be due to a bad response or command.
21	Port 2 Requests: Total number of messages sent out of the port.
22	Port 2 Responses: Total number of messages received on the port.
23	Port 2 Errors Sent: Total number of message errors sent out of the port.
24	Port 2 Errors Received: Total number of message errors received on the port.
25	Read Block Count: Total number of read blocks transferred from the module to the processor.
26	Write Block Count: Total number of write blocks transferred from the processor to the module.
27	Parse Block Count: Total number of blocks successfully parsed that were received from the processor.
28	Event Command Block Count: Total number of Event Command blocks received from the processor.
29	Command Control Block Count: Total number of Command Control blocks received from the processor.
30	Error Block Count: Total number of block errors recognized by the module.
31	Port 1 Current Error: For a slave port, this field contains the value of the current error code returned. For a master port, this field contains the index of the currently executing command.

Offset	Description
32	Port 1 Last Error: For a slave port, this field contains the value of the last error code returned. For a master port, this field contains the index of the command with an error.
33	Port 2 Current Error: For a slave port, this field contains the value of the current error code returned. For a master port, this field contains the index of the currently executing command.
34	Port 2 Last Error: For a slave port, this field contains the value of the last error code returned. For a master port, this field contains the index of the command with an error.
35 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.11 Set Port and Command Active Bits Block (9500)

Block 9500: Set Port and command active bits

This block enables and disables ports, as well as individual Master commands for a port.

8.4.11.1 Block 9500: Request from Processor to Module

Offset	Description
0	Write Block ID: 9500 to set port and command enable/disable state
1	Port 1 active state: 0=disabled, 1=enabled
2 to 21	Command enable bits for Port 1 commands (0=disabled, 1=enabled)
22	Port 2 active state: 0=disabled, 1=enabled
23 to 42	Command enable bits for Port 2 commands (0=disabled, 1=enabled)
43 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.11.2 Block 9500: Response from Module to Processor

Offset	Description
0	Read Block ID: 9500 requested by processor.
1	Write Block ID: To be used by the processor in its next Write block.
2 to (n-1)	Spare

8.4.12 Get Port and Command Active Bits Block (9501)

Block 9501: Get Port and command active bits

This block requests the enabled/disabled status of ports and Master commands.

8.4.12.1 Block 9501: Request from Processor to Module

Offset	Description
0	Write Block ID: 9501 to get port and command enable/disable state
1 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.12.2 Block 9501: Response from Module to Processor

Offset	Description
0	Read Block ID: 9501 requested by processor.
1	Write Block ID: To be used by the processor in its next Write block.
2	Port 1 active state: 0=disabled, 1=enabled
3 to 22	Command enable bits for Port 1 commands (0=disabled, 1=enabled)
23	Port 2 active state: 0=disabled, 1=enabled
24 to 43	Command enable bits for Port 2 commands (0=disabled, 1=enabled)
44 to (n-1)	Spare

8.4.13 Pass-through Formatted Word Data Block for Functions 6 & 16 (9956)

Block 9956: Pass-Through Formatted Block for Functions 6 and 16 with Word Data Block

If one or more of the slave ports on the module are configured for formatted Pass-Through mode, the module sends input image blocks with identification codes of 9956, 9957, 9958 or 9959 to the processor for each write command received. Any incoming Modbus Function 5, 6, 15 or 16 command is passed from the port to the processor using a block identification number that identifies the Function Code received in the incoming command.

The MBS Add-On Instruction handles the receipt of all Modbus write functions and to respond to commands issued by the remote Modbus Master device.

Note: Mutual exclusion on Pass-Through Block IDs 9956, 9957, 9958, and 9959 from both ports - If both ports are configured as slave ports, when both of the slave ports receive write commands with the same Function Code, which would need to use the same block identifier from the above list, the module will process the command from the port which first received the command and will return an Exception Code error code 6 (node is busy - retry command later error) from the other port that received the command last. The Master retries the command on the busy port after a short delay. This prevents Pass-Through blocks on both ports from overwriting each other.

8.4.13.1 Block 9956: Request from Module to Processor

Offset	Description
0	Read Block ID: 9956
1	Write Block ID: 9956
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to (n+1)	Data (Length in words = n - 2)

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

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

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

8.4.13.2 Block 9956: Response from Processor to Module

Offset	Description
0	Write Block ID: 9956
1 to n	Spare (Length in words = n - 2)

8.4.14 Pass-through Formatted Float Data Block for Functions 6 & 16 (9957)

Block 9957: Pass-Through Formatted Block for Functions 6 and 16 with Float Data Block

8.4.14.1 Block 9957: Request from Module to Processor

Offset	Description
0	Read Block ID: 9957
1	Write Block ID: 9957
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to (n+1)	Data (Length in words = n - 2)

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

The ladder logic is 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 block with a write block with the following format.

8.4.14.2 Block 9957: Response from Processor to Module

Offset	Description
0	Write Block ID: 9957
1 to n	Spare (Length in words = n - 2)

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

8.4.15 Pass-through Formatted Block for Function 5 (9958)

Block 9958: Pass-Through Formatted Block for Function 5

8.4.15.1 Block 9958: Request from Module to Processor

Offset	Description
0	Read Block ID: 9958
1	Write Block ID: 9958
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to (n+1)	Data (Length in words = n - 2)

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

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

8.4.15.2 Block 9958: Response from Processor to Module

Offset	Description
0	Write Block ID: 9958
1 to n	Spare (Length in words = n - 2)

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

8.4.16 Pass-through Formatted Block for Function 15 (9959)

Block 9959: Pass-Through Formatted Block for Function 15

When the module receives a function code 15 in Pass-Through mode, the module writes the data using block ID 9959 for multiple-bit data. First the bit mask clears the bits to be updated. This is accomplished in Studio 5000 by ANDing the inverted mask with the existing data.

Next, the new data ANDed with the mask is ORed with the existing data. This protects the other bits in the INT registers from being affected. This function can only be used if the Block Transfer Size parameter is set to 120 or 240 words.

8.4.16.1 Block 9959: Request from Module to Processor

Offset	Description
0	Read Block ID: 9959
1	Write Block ID: 9959
2	Length in words
3	Data address
4 to 28	Modbus Data
29 to 53	Bit mask to use with the data set. Each bit to be considered with the data set has a value of 1 in the mask. Bits to ignore in the data set has a value of 0 in the mask.
54 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

The ladder logic is 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.

8.4.16.2 Block 9959: Response from Processor to Module

Offset	Description
0	Write Block ID: 9959
1 to (n-1)	Spare

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

8.4.17 Pass-through Formatted Block for Function 23 (9961)

Block 9961: Pass-Through Formatted Block for Function 23

8.4.17.1 Block 9961: Request from Module to Processor

Offset	Description
0	Read Block ID: 9961
1	Write Block ID: 9961
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to (n+1)	Data (Length in words = n - 2)

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

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

8.4.17.2 Block 9961: Response from Processor to Module

Offset	Description
0	Write Block ID: 9961
1 to n	Spare (Length in words = n - 2)

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

8.4.18 Pass-through Block for Function 99 (9970)

Block 9970: Pass-Through Block for Function 99

8.4.18.1 Block 9970: Request from Module to Processor

Offset	Description
0	Read Block ID: 9970
1	Write Block ID: 9970
2	1
3	0
4 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

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

8.4.18.2 Block 9970: Response from Processor to Module

Offset	Description
0	Write Block ID: 9970
1 to n	Spare (Length in words = $n - 2$)

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

8.4.19 Pass Module Time to Processor Block (9973)

Block 9973: Pass Module Time to Processor Block

This block uses the time information from the module to set the processor time.

8.4.19.1 Block 9973: Request from Processor to Module

Offset	Description
0	Write Block ID: 9973
1 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.19.2 Block 9973: Response from Module to Processor

Offset	Description
0	Read Block ID: 9973
1	Write Block ID: To be used by the processor in its next Write block.
2	Year (0-9999)
3	Month (1-12)
4	Day (1-31)
5	Hour (0-23)
6	Minutes (0-59)
7	Seconds (0-59)
8	Milliseconds
9 to (n-1)	Spare

8.4.20 Reset Status Block (9997)

Block 9997: Reset Status Block

This block resets the module, port 1, and/or port 2 status.

8.4.20.1 Block 9997: Request from Processor to Module

Offset	Description
0	Write Block ID: 9997
1	Reset Module status (0=no, else yes)
2	Reset Port 1 status (0=no, else yes)
3	Reset Port 2 status (0=no, else yes)
4 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.4.20.2 Block 9997: Response from Module to Processor

Offset	Description
0	Read Block ID: 9997
1	Write Block ID: To be used by the processor in its next Write block.
2 to (n-1)	Spare

8.4.21 Cold-boot Control Block (9999)

Block 9999: Cold-boot Control Block

If the CompactLogix or MicroLogix 1500-LRP processor sends a block number 9999, the firmware performs a cold-boot operation. The firmware reloads the configuration file from the processor to the module and resets all MBS memory, error and status data.

8.4.21.1 Block 9999: Request from Processor to Module

Offset	Description
0	Write Block ID: 9999
1 to (n-1)	Spare

Where $n = 60, 120, \text{ or } 240$ depending on the Block Transfer Size parameter.

8.5 Ethernet Port Connection

8.5.1 Ethernet Cable Specifications


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

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

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

8.5.1.1 Ethernet Cable Configuration

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

Crossover Cable		Pin #1	Straight-Through Cable	
RJ-45 PIN	RJ-45 PIN		RJ-45 PIN	RJ-45 PIN
1 Rx+	3 Tx+		1 Tx+	
2 Rx-	6 Tx-		2 Tx-	
3 Tx+	1 Rx+		3 Rx+	
6 Tx-	2 Rx-		6 Rx-	

8.5.1.2 Ethernet Performance

Ethernet performance in the MVI69E-MBS module can be affected in the following way:

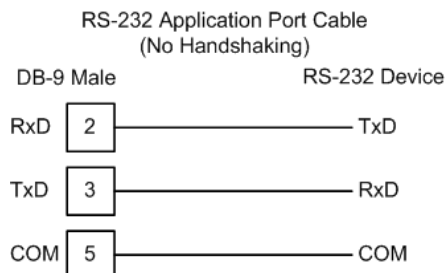
- Accessing the web interface (refreshing the page, downloading files, and so on) may affect performance
- Also, high Ethernet traffic may impact MBS performance, so consider one of these options:
 - Use managed switches to reduce traffic coming to module port
 - Use CIPconnect for these applications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

8.6 Modbus Application Port Connection

The module supports RS-232, RS-422, and RS-485 wiring to remote devices.

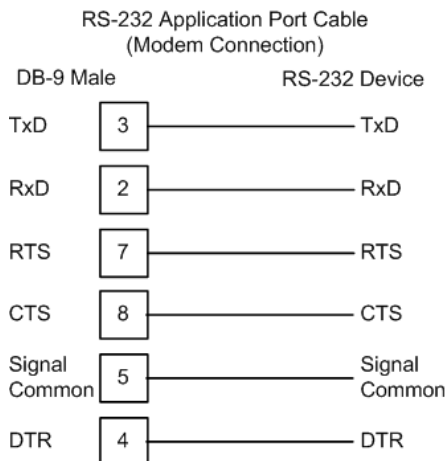
8.6.1 RS-232 Wiring

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 is used, here are the cable pin-outs to connect to the port.



8.6.1.1 RS-232: Modem Connection (Hardware Handshaking Required)

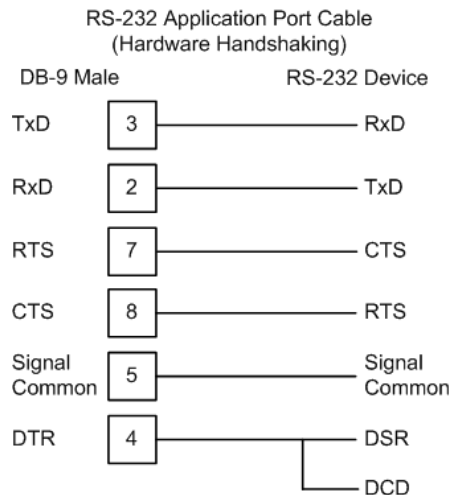
This type of connection is required between the module and a modem or other communication device.



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

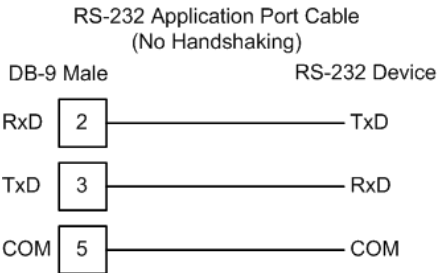
8.6.1.2 RS-232: Null Modem Connection (Hardware Handshaking)

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

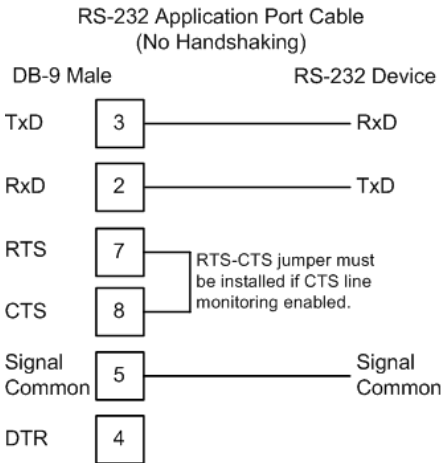


8.6.1.3 RS-232: Null Modem Connection (No Hardware Handshaking)

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

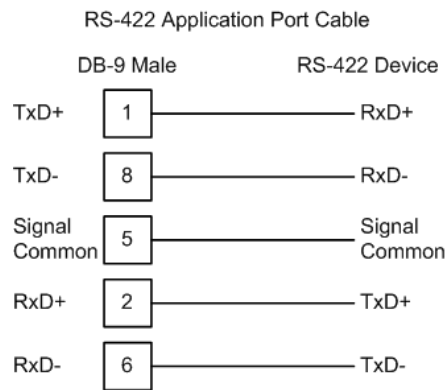


Note: For most null modem connections where hardware handshaking is not required, the *Use CTS Line* parameter should be set to **N** and no jumper is required between Pins 7 (RTS) and 8 (CTS) on the connector. If the port is configured with the *Use CTS Line* set to **Y**, then a jumper is required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



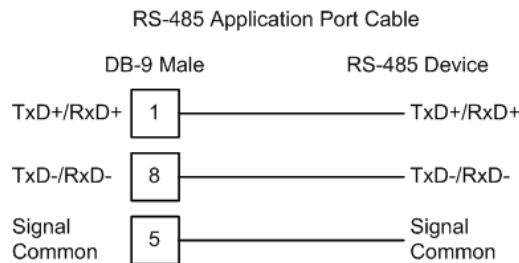
8.6.2 RS-422 Wiring

The RS-422 interface requires a single four or five wire cable. The Common connection is optional, depending on the RS-422 network devices used.



8.6.3 RS-485 Wiring

The RS-485 interface requires a single two or three wire cable. The Common connection is optional, depending on the RS-485 network devices used.



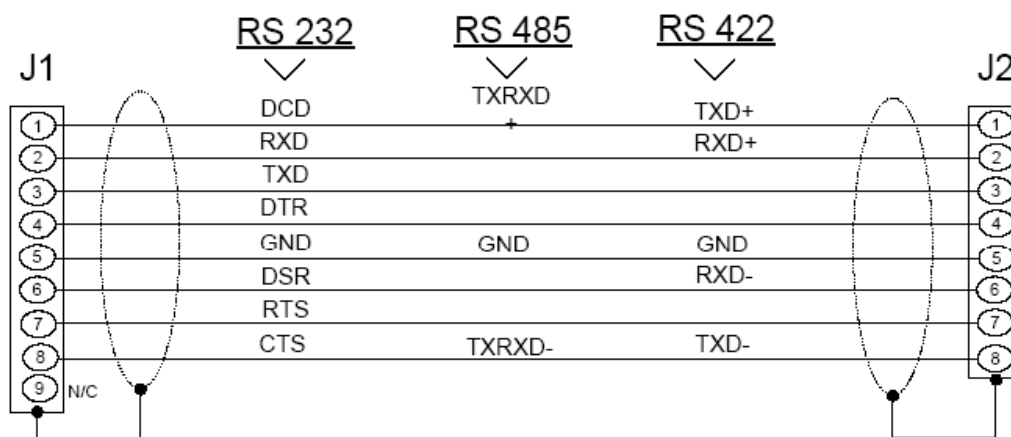
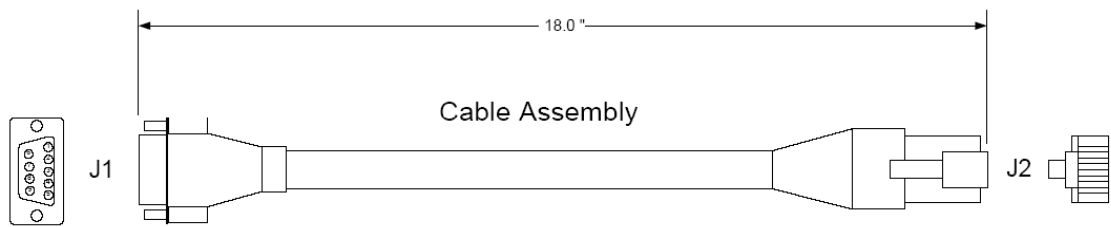
Note: This type of connection is commonly called a RS-485 half-duplex, 2-wire connection. If you have RS-485 4-wire, full-duplex devices, they can be connected to the gateway's serial ports by wiring together the TxD+ and RxD+ from the two pins of the full-duplex device to Pin 1 on the gateway and wiring together the TxD- and RxD- from the two pins of the full-duplex device to Pin 8 on the gateway. As an alternative, you could try setting the gateway to use the RS-422 interface and connect the full-duplex device according to the RS-422 wiring diagram. For additional assistance, please contact ProSoft Technical Support.

Note: Depending upon devices on the network, if there are problems in RS-485 communication that can be attributed to the signal echoes or reflections, then consider adding 120 OHM terminating resistors at both ends of the RS-485 line.

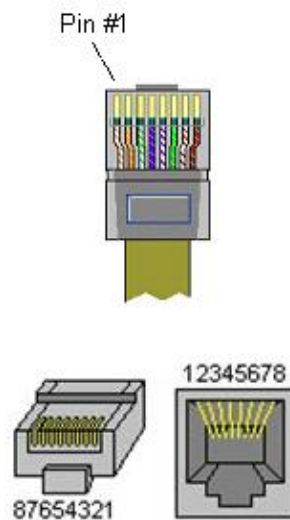
8.6.3.1 RS-485 and RS-422 Tip

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

8.6.4 DB9 to RJ45 Adaptor (Cable 14)



Wiring Diagram



9 Support, Service, and Warranty

9.1 Contacting Technical Support

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

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

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

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

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

For additional ProSoft Technology contacts in your area, please see:

www.prosoft-technology.com/About-Us/Contact-Us

9.2 Warranty Information

For details regarding ProSoft Technology's legal terms and conditions, please see:

www.prosoft-technology.com/ProSoft-Technology-Legal-Terms-and-Conditions

For Return Material Authorization information, please see:

www.prosoft-technology.com/Services-Support/Return-Material-Instructions