ProSoft®
TECHNOLOGY

Where Automation Connects.

MVI69L-MBS

CompactLogix Platform

Modbus Serial Lite Communication Module

August 3, 2015

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

MVI69L-MBS User Manual

August 3, 2015

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.

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

Content Disclaimer

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

North America: +1 (661) 716-5100
Asia Pacific: +603.7724.2080
Europe, Middle East, Africa: +33 (0) 5.3436.87.20
Latin America: +1.281.298.9109

Important Safety Information

North America Warnings

A   This Equipment is Suitable For Use in Class I, Division 2, Groups A, B, C, D or Non-Hazardous Locations Only.
B   Warning – Explosion Hazard – Substitution of Any Components May Impair Suitability for Class I, Division 2.
C   Warning – Explosion Hazard – Do Not Disconnect Equipment Unless Power Has Been Switched Off Or The Area is Known To Be Non-Hazardous.
D   The subject devices are powered by a Switch Model Power Supply (SMPS) that has regulated output voltage of 5 VDC.

ATEX/IECEx Warnings and Conditions of Safe Usage:

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

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

Agency Approvals and Certifications

Agency
ATEX; Category 3, Zone 2
CE Mark
CSA; CB Safety
Environmental
KCC
RoHS Compliant
UL/cUL; Class 1, Div. 2 Groups A, B, C, D

Contents

1   Start Here

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

* **Rockwell Automation® RSLogix™ software:** launch the program, configure ladder logic, and transfer the ladder logic to the processor
* **Microsoft Windows®:** install and launch programs, execute menu commands, navigate dialog boxes, and enter data
* **Hardware installation and wiring:** install the module, and safely connect Modbus and CompactLogix devices to a power source and to the MVI69L-MBS module's application port(s)

1.1   System Requirements

The MVI69L-MBS module requires the following minimum hardware and software components:

* Rockwell Automation CompactLogix® processor (firmware version 10 or higher), with compatible power supply and one free slot in the rack, for the MVI69L-MBS module.

Important: The MVI69L-MBS module has a power supply distance rating of 4 (L43 and L45 installations on first 2 slots of 1769 bus). It consumes 450 mA at 5 Vdc.

* The module requires 450 mA of available 5 Vdc power
* Rockwell Automation RSLogix 5000 programming software version 16 or higher
* Rockwell Automation RSLinx® communication software version 2.51 or higher
* ProSoft Configuration Builder (PCB) (included)
* ProSoft Discovery Service (PDS) (included in PCB)
* Pentium® II 450 MHz minimum. Pentium III 733 MHz (or better) recommended

- Supported operating systems:
  - Microsoft Windows® 7
  - Microsoft Windows Vista
  - Microsoft Windows XP Professional with Service Pack 1 or 2
  - Microsoft Windows 2000 Professional with Service Pack 1, 2, or 3
  - Microsoft Windows Server 2003
- 128 Mbytes of RAM minimum, 256 Mbytes of RAM recommended
- 100 Mbytes of free hard disk space (or more based on application requirements)
- 256-color VGA graphics adapter, 800 x 600 minimum resolution (True Color 1024 x 768 recommended)
- DVD drive

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

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 MVI69L-MBS module occupy?

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

- Are the RSLogix 5000 and RSLinx software installed?

  RSLogix and RSLinx are required to communicate to the CompactLogix processor.

- How many words of data do you need to transfer in your application (from CompactLogix to Module / to CompactLogix from Module)?

1.3   Package Contents

The following components are included with your MVI69L-MBS module, and are all required for installation and configuration.

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

Qty.	Part Name	Part Number	Part Description
1	MVI69L-MBS Module	MVI69L-MBS	Modbus Serial Lite Communication Module
2	Adapter Cable	Cable #14	RJ45 to DB9 Male Adapter cable. For DB9 connection to module's serial application port
2	Screw Terminal Adapter	1454-9F	DB9 female to 9-pin screw terminal. Used for RS422 or RS485 connections to Port 1 of the module
1	Ethernet Cable	RL-CBL025	Ethernet straight-through cable
1	ProSoft Solutions DVD	DVD-001	Contains ProSoft Configuration Builder (PCB) and ProSoft Discovery Service (PDS) software. Files can also be found at www.prosoft-technology.com

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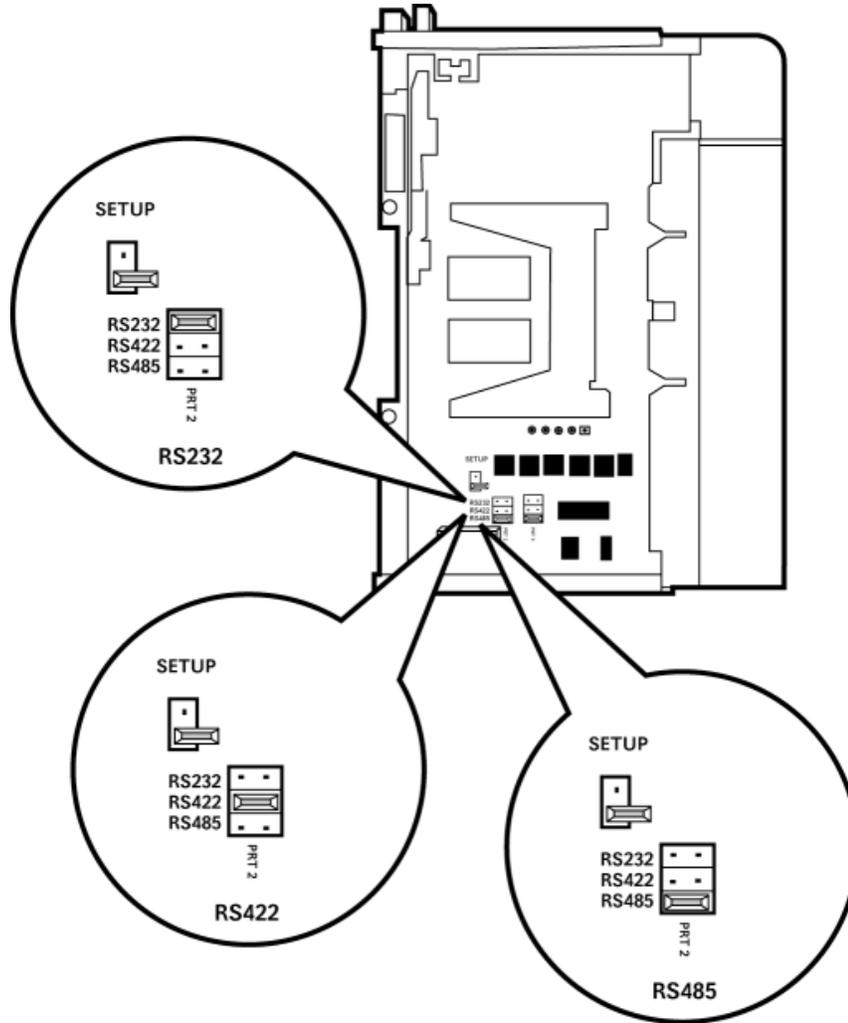
Note: Jumper pin placement on the circuit board may vary.

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 MVI69L-MBS jumper configuration, with the Setup Jumper OFF.
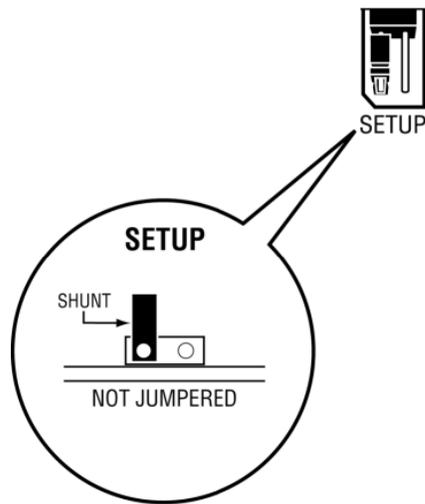
1.5   Installing the Module in the Rack

Make sure the processor and power supply are installed and configured before installing the MVI69L-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 MVI69L-MBS into the rack. Use the same technique recommended by Rockwell Automation to remove and install CompactLogix modules.

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

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

Upper Tongue and Groove

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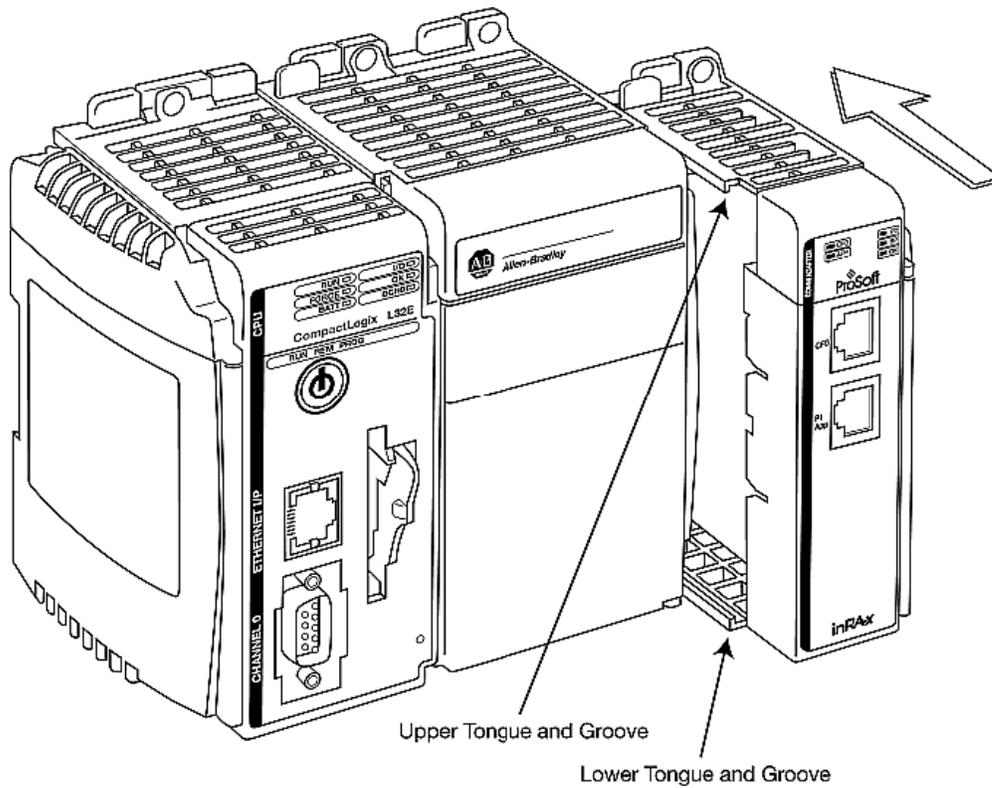
2 Move the module back along the tongue-and-groove slots until the bus connectors on the MVI69 module and the adjacent module line up with each other.

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.
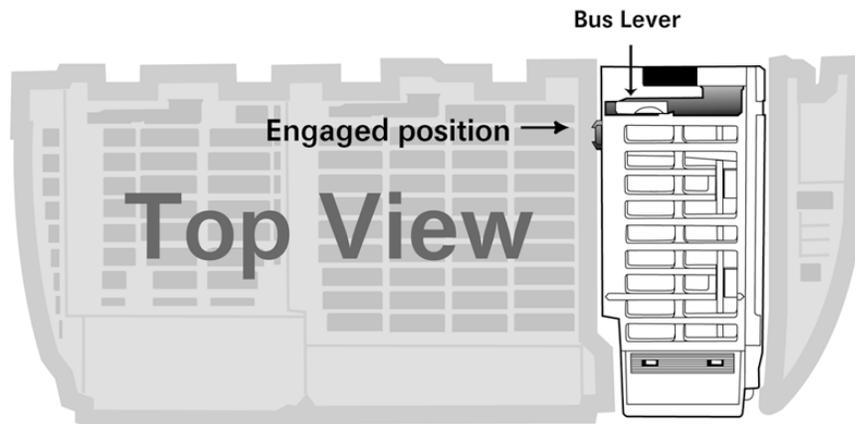
4 Close all DIN-rail latches.

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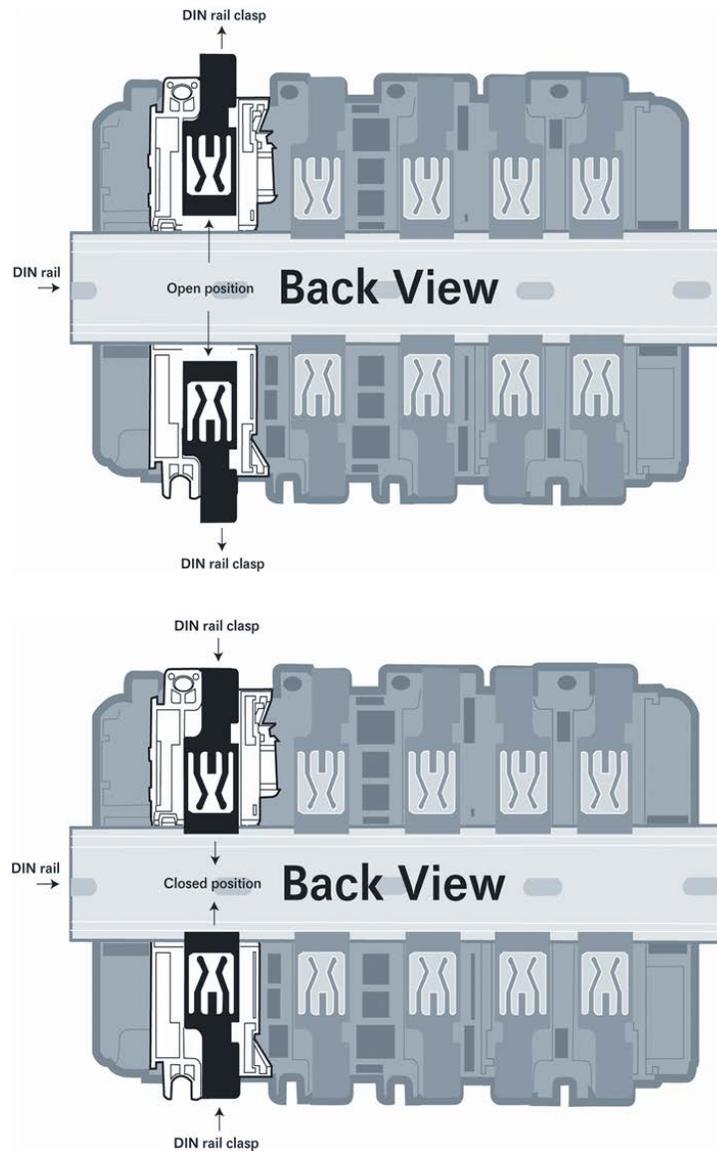
1.6 DVD Contents

The DVD contains all the necessary files for the module including the User Manual and the ProSoft Configuration Builder (PCB) software. It also may contain module-specific configuration files such as the RSLogix 5000 Add-On Profile and sample Add-On Instruction (where applicable to the module).

If the DVD is not present, please visit http://www.prosoft-technology.com for the latest files.

2   Adding the Module to RSLogix

To add the MVI69L-MBS module in RSLogix 5000, you must:

1 Create a new project in RSLogix 5000.
2 Add the module to the RSLogix 5000 project. There are two ways to do this:
  o You can use the Add-On Profile from ProSoft Technology. This is the preferred way, but requires RSLogix version 15 or later.
  o 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 an RSLogix 5000 compatible file (.L5X file).
4 Import the Add-On Instruction (the .L5X file) into RSLogix 5000.

The .L5X file contains the Add-On Instruction, user-defined data types, controller tags and ladder logic required to configure the MVI69L-MBS module.

2.1   Creating the Module in an RSLogix 5000 Project

In an RSLogix 5000 project, there are two ways you can add the MVI69L-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 18).
- 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.

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

Installing an Add-On Profile

Download the AOP file (MVI69x_RevX.X_AOP.zip) from the product webpage (found at http://www.prosoft-technology.com) or from the ProSoft Solutions DVD onto your local hard drive and then extract the files from the zip archive. Make sure you have shut down RSLogix 5000 and RSLinx before you install the Add-On Profile (AOP).

Run the **MPSetup.exe** file to start the Setup Wizard. Follow the Setup Wizard to install the AOP.
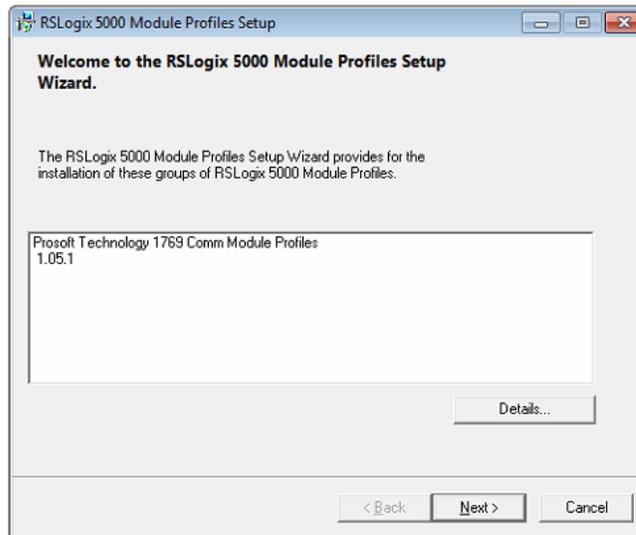
Continue to follow the steps in the wizard to complete the installation.
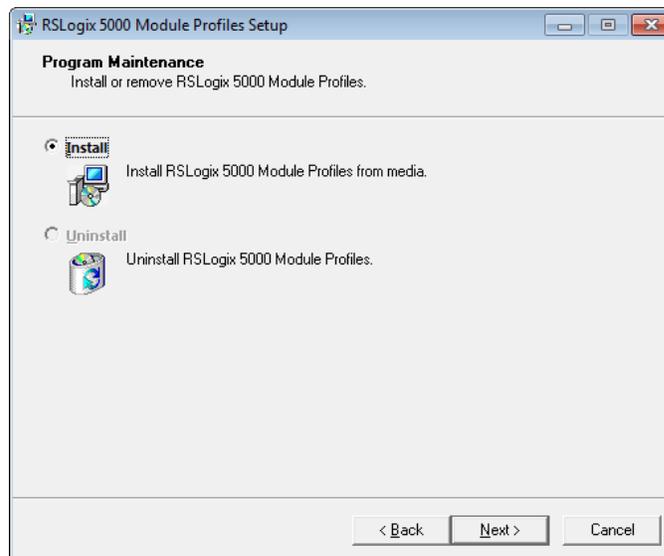
Click **FINISH** when complete**.** The AOP is now installed in RSLogix 5000. You do not need to reboot the PC.
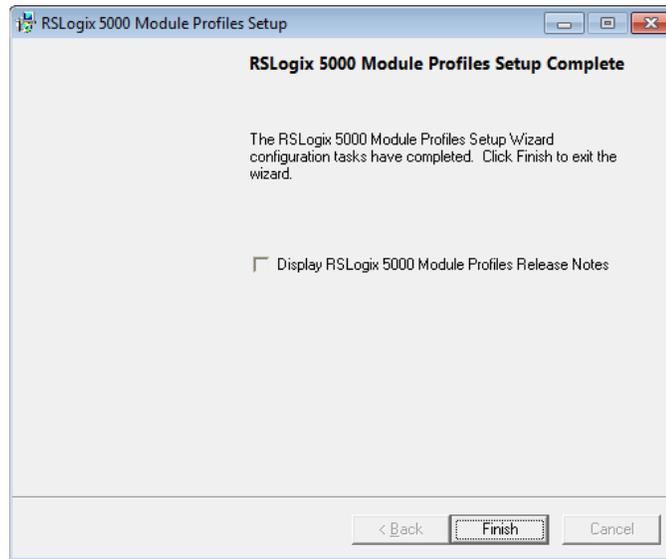
Using an Add-On Profile

1 In RSLogix 5000, expand the **I/O CONFIGURATION** folder in the Project tree. Right-click the appropriate communications bus, and then click **NEW MODULE**.
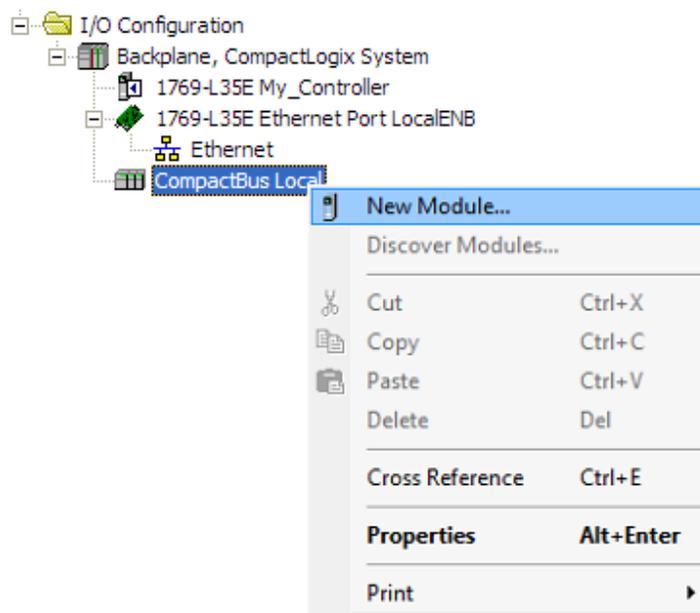
This opens the *Select Module Type* dialog box. In the *Module Type Vendor Filters* area, uncheck all boxes except the **PROSOFT TECHNOLOGY** box. A list of ProSoft Technology modules appears in the dialog box.
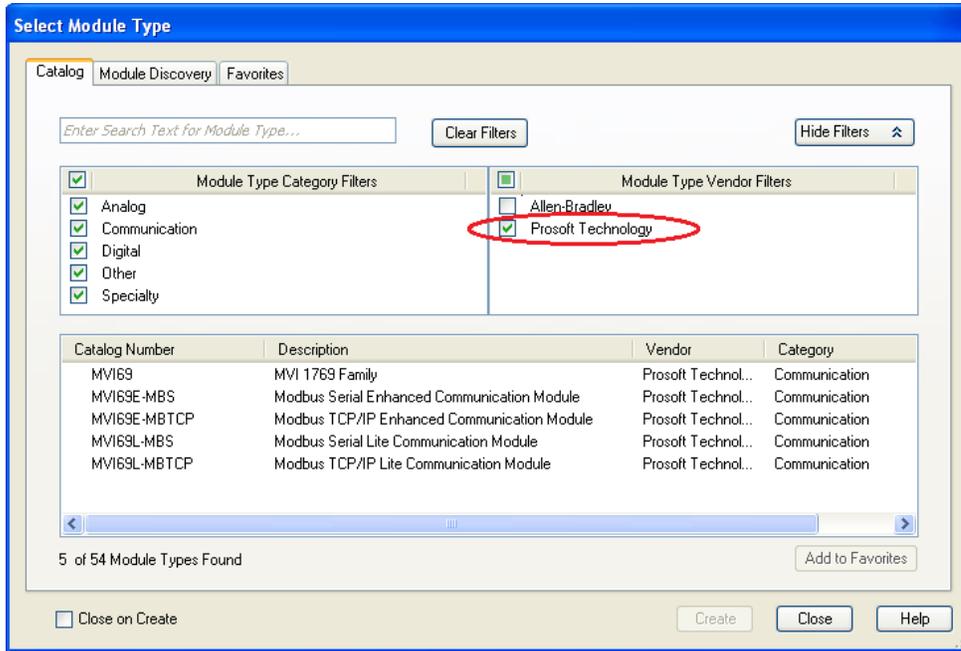
2 Select the **MVI69L-MBS** module in the list and click **CREATE**:
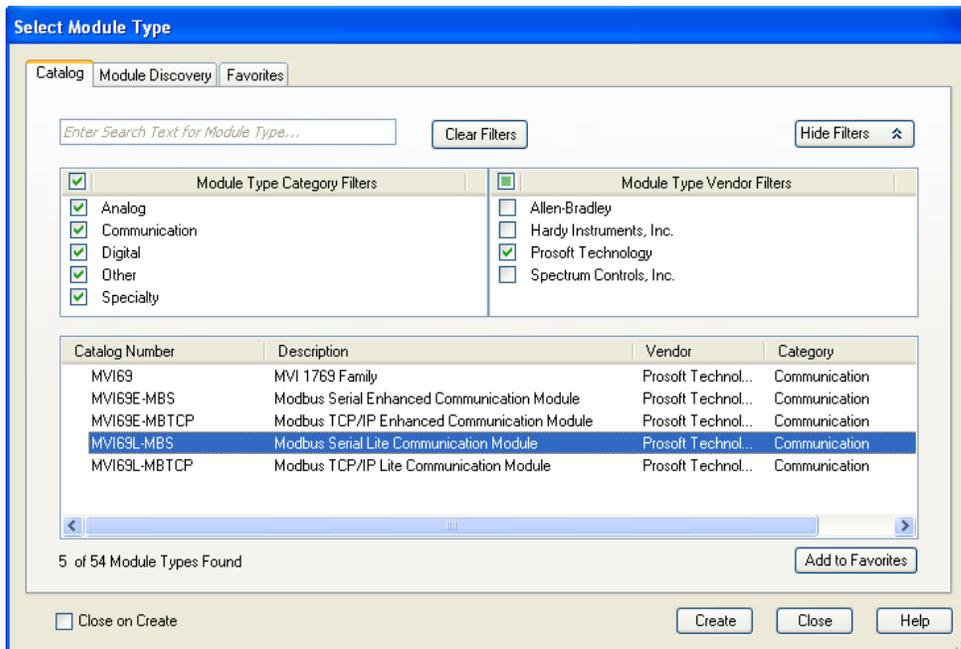
3 A *New Module* dialog box opens. Edit the **NAME** and **SLOT** for the module and click **OK**.
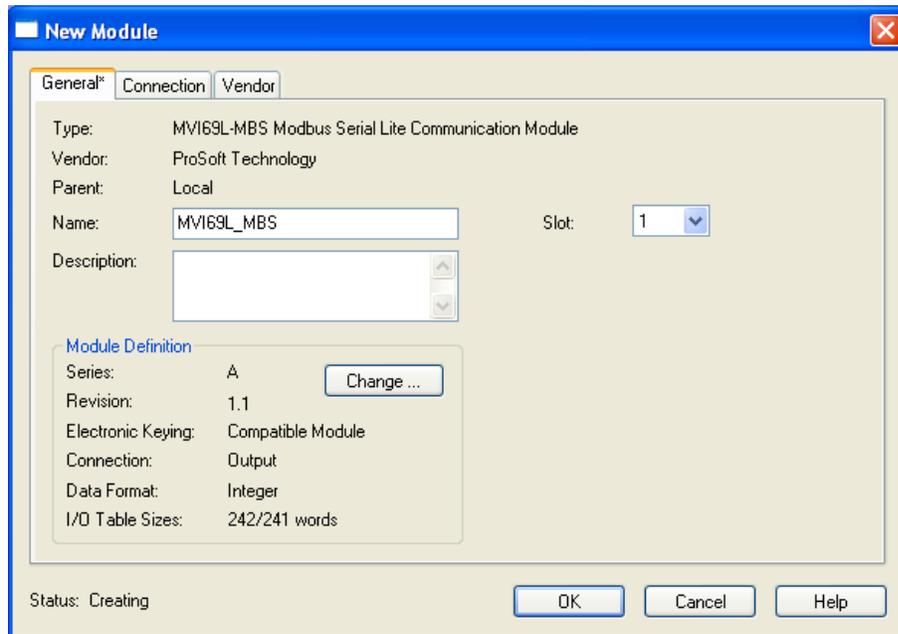
> **Note:** This module uses a block transfer size of 240 only. Therefore, it uses an **I/O TABLE SIZE** of 242/241 words.

The MVI69L-MBS module is now visible in the I/O Configuration tree.
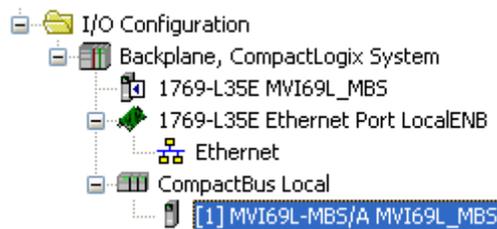
2.1.2 Creating a Module in the Project Using a Generic 1769 Module Profile

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**.
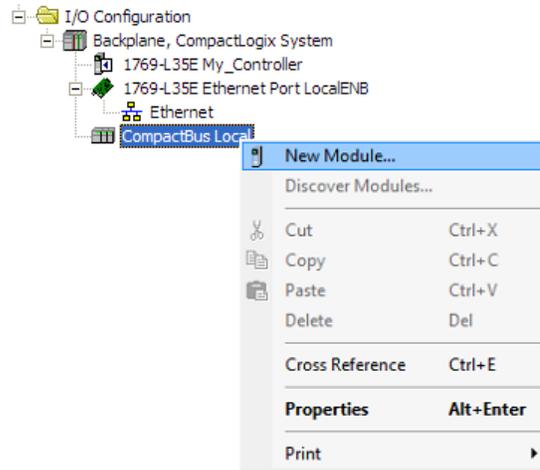
This opens the *Select Module Type* dialog box.

2 Enter **GENERIC** in the search text box and select the **GENERIC 1769 MODULE**. If you are using an earlier version of RSLogix, expand **OTHER** in the *Select Module* dialog box, and then select the **GENERIC 1769 MODULE**.
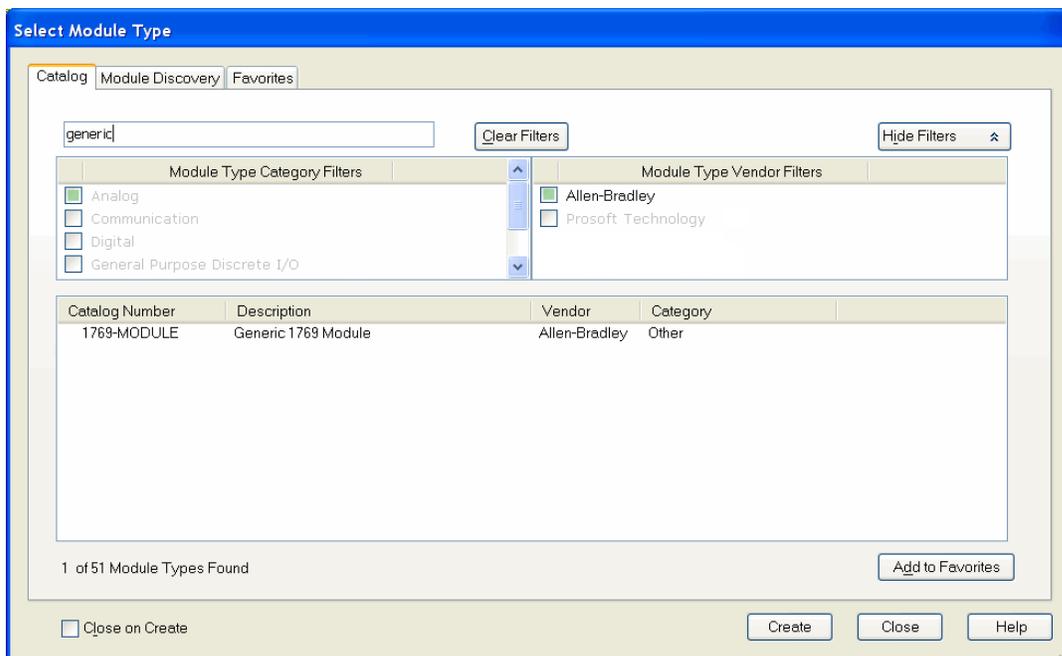
3 Set the *Module Properties* values as follows:

Parameter	Value
Name	Enter a module identification string. Example: MVI69LMBS
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	242
Output Assembly Instance	100
Output Size	241
Configuration Assembly Instance	102
Configuration Size	0

This module must be configured with a block transfer size of 240 words (input block size = 242 words, output block size = 241 words).
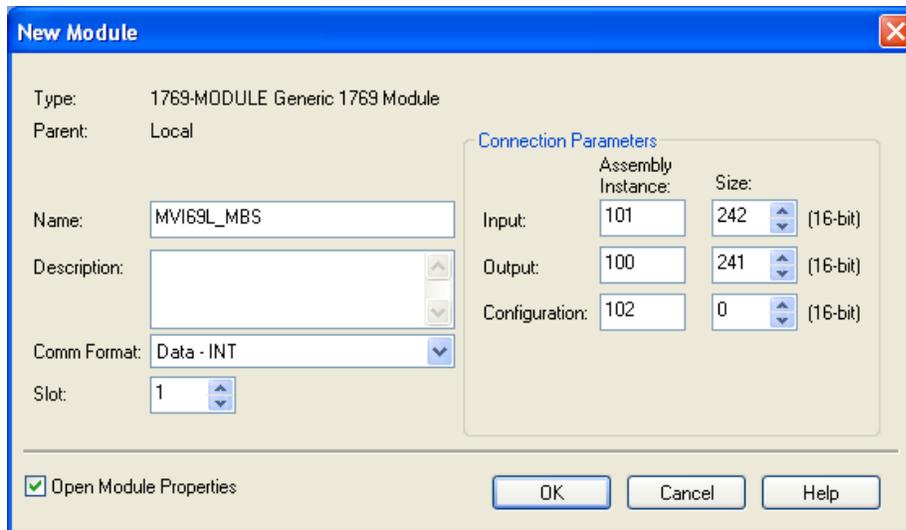
4 On the *Connection* tab, set the **REQUESTED PACKET INTERVAL** value for your project and click **OK**.
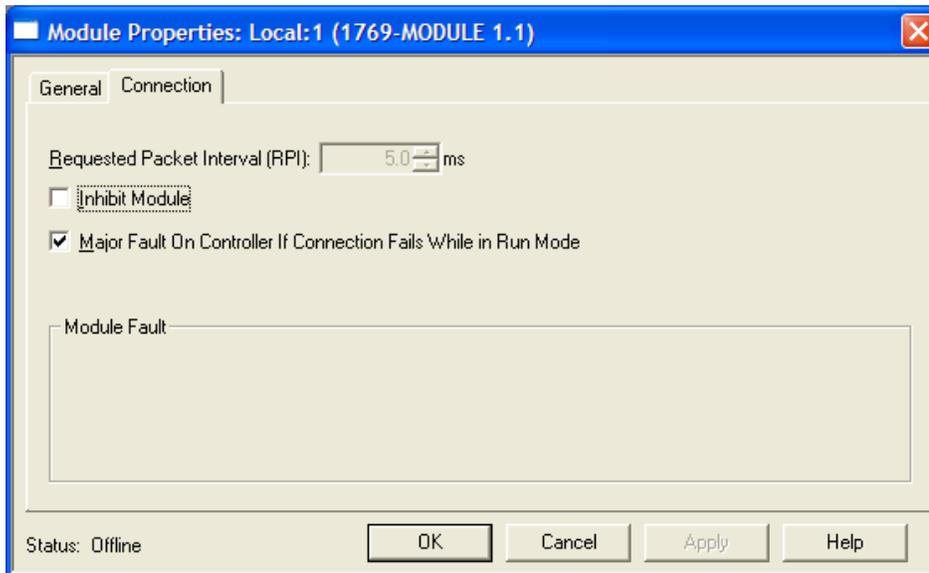
The MVI69L-MBS module is now visible in the I/O Configuration tree.
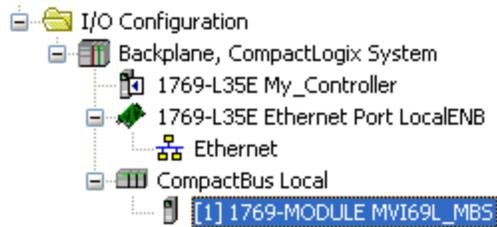
2.2   Installing ProSoft Configuration Builder

Use the ProSoft Configuration Builder (PCB) software to configure the module. You can find the latest version of the ProSoft Configuration Builder (PCB) on our web site: http://www.prosoft-technology.com, or you can install it from the ProSoft Solutions DVD. The installation filename contains the PCB version number. For example, **PCB_4.3.4.5.0238.EXE**.

If you are installing PCB from the ProSoft website:

1   Open a browser window and navigate to http://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 MVI69L-MBS, enter the model number into the ProSoft website search box and press the Enter key.

If you are installing PCB from the ProSoft Solutions DVD:

1   Insert the ProSoft Solutions DVD into your computer's DVD drive and wait for the ProSoft Installation program to start.
2   If the ProSoft Installation program does not start, open the Windows file Explorer, navigate to the DVD, and double-click the ProSoft_DVD.exe file.
3   Navigate to the MVI69L-MBS selection using the **PLATFORM** and **PRODUCT** selections.
4   Click **PROSOFT CONFIGURATION BUILDER**.
5   Follow the instructions that appear on the screen.

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

To begin, start PROSOFT CONFIGURATION BUILDER (PCB).
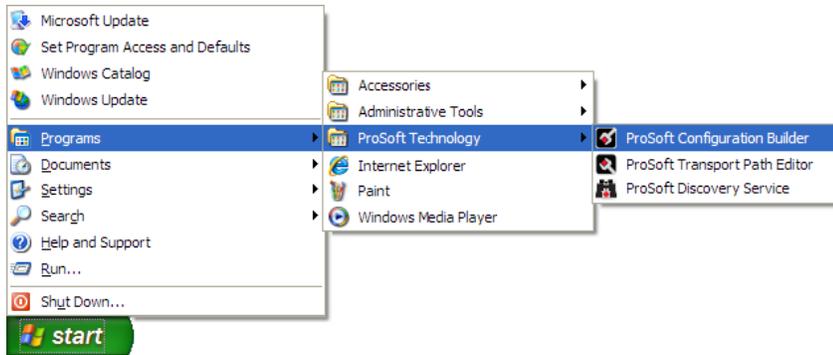
The *PCB* window consists of a tree view on the left, and an information pane and a configuration pane on the right side of the window. The tree view consists of folders for *Default Project* and *Default Location*, with a *Default Module* in the *Default Location* folder. The following illustration shows the *PCB* window with a new project.
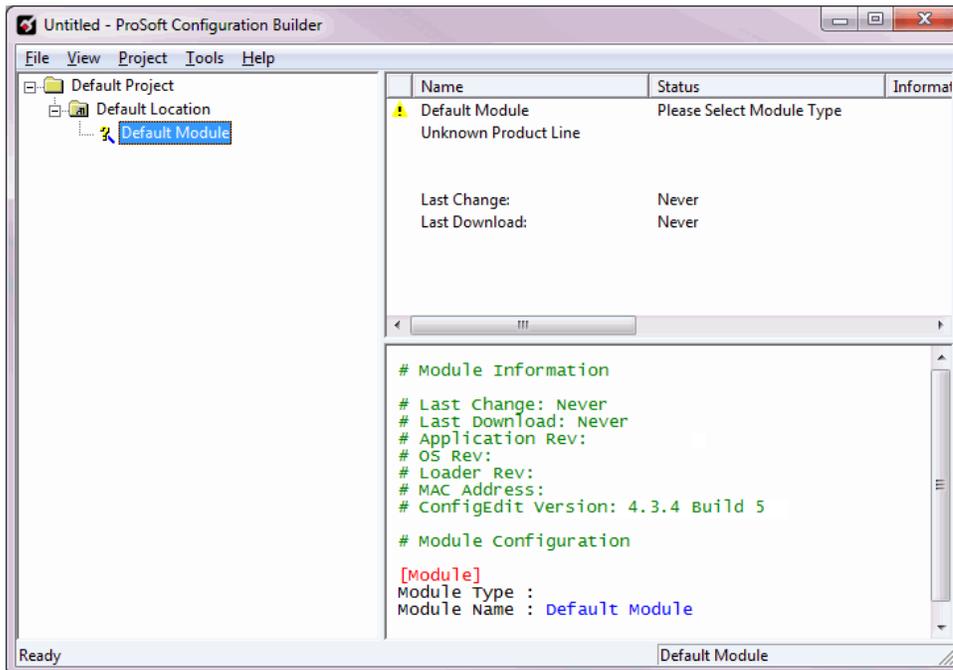
Your first task is to add the MVI69L-MBS module to the project.

1   In the Tree view, right-click **DEFAULT MODULE**, and then click **CHOOSE MODULE TYPE**. This opens the *Choose Module Type* dialog box.
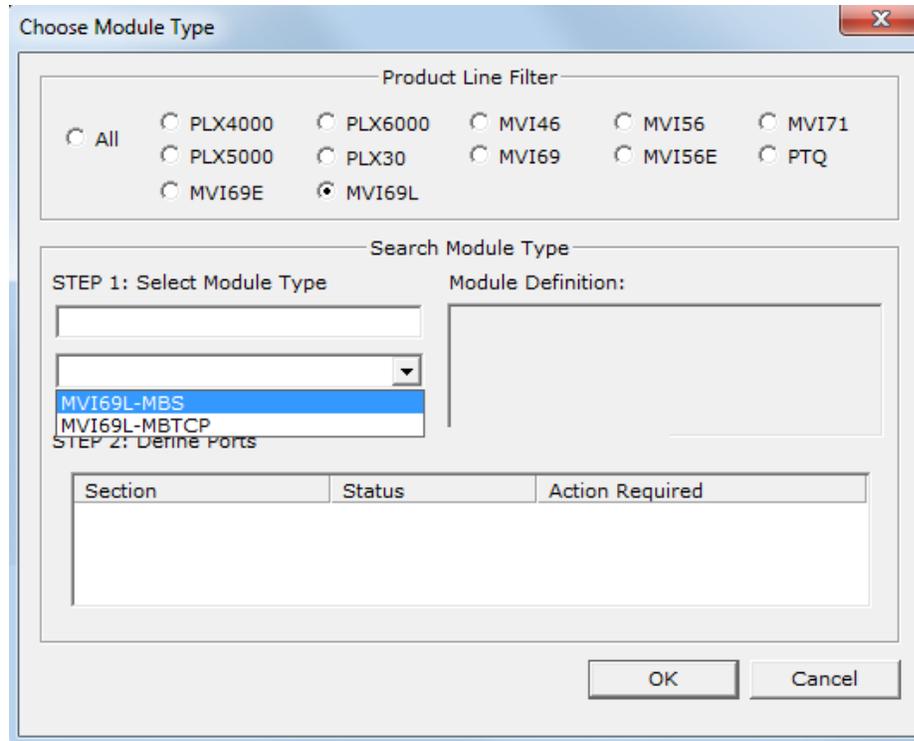
2   In the *Product Line Filter* area of the dialog box, click **MVI69**. In the *Select Module Type* dropdown list, click **MVI69L-MBS**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window. The MVI69L-MBS icon is now visible in the tree view.
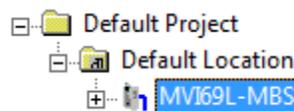
2.3.2 Creating and Exporting the .L5X File

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 **MVI69L-MBS** icon by clicking the **[+]** symbol beside it. Similarly, expand the ⊞ 🖧 Module icon. Double-click the 🅱 Module icon to open the *Edit - Module* dialog box.
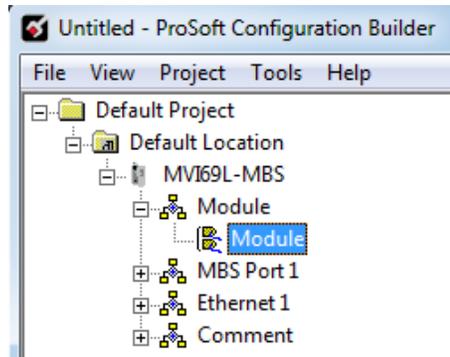
2 Edit the *Slot Number* indicating where the module is placed in the 1769 bus.
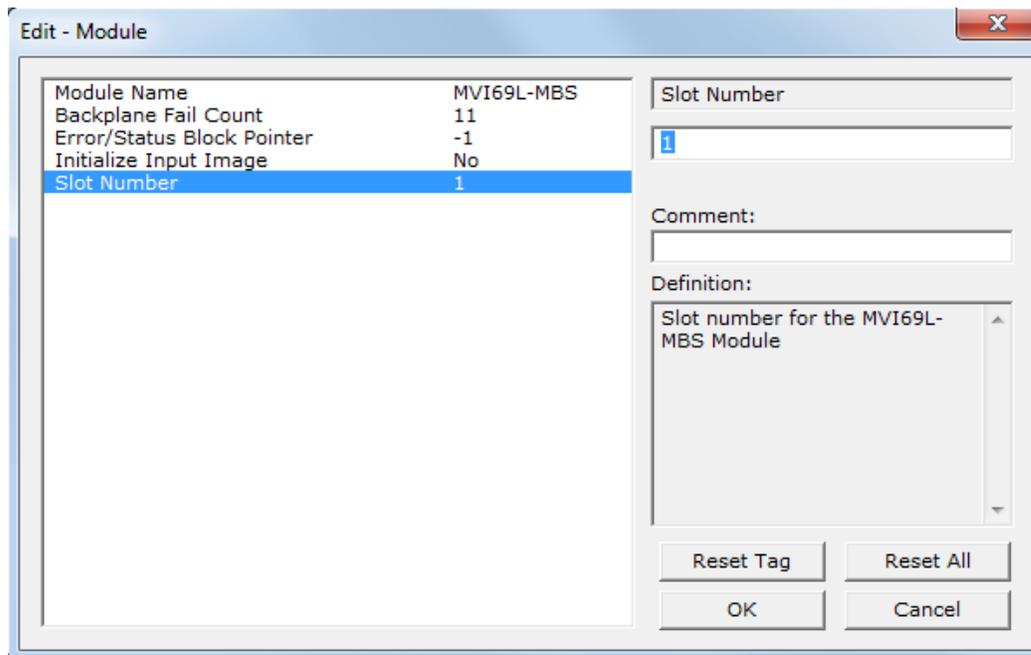
3 Click **OK** to close the *Edit – Module* dialog box. The .L5X file is now ready to be exported to the PC/Laptop.

4 Right-click the **MVI69L-MBS** icon in the project tree and then click **EXPORT AOI FILE**.
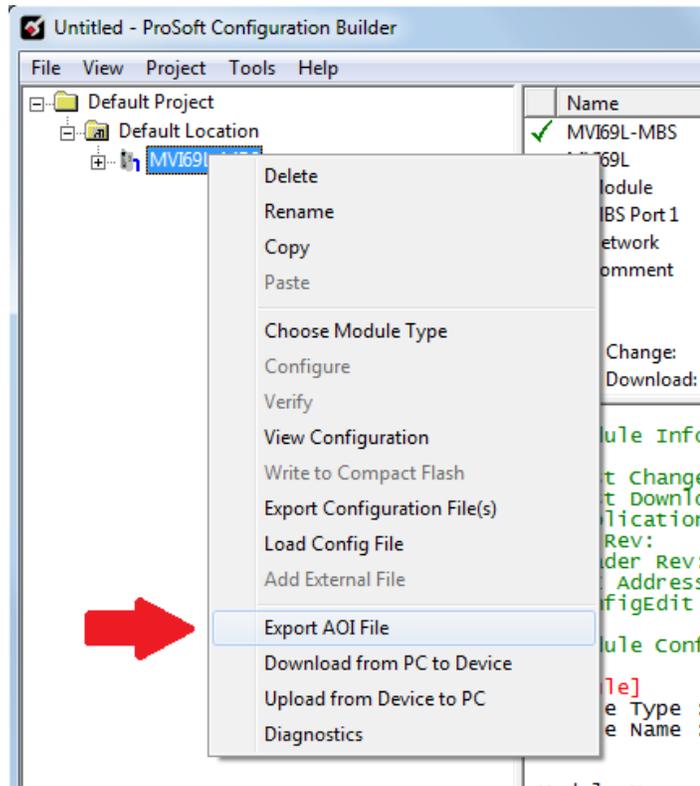
5 Save the .L5X file to the PC/Laptop in an easily found location, such as Windows Desktop.
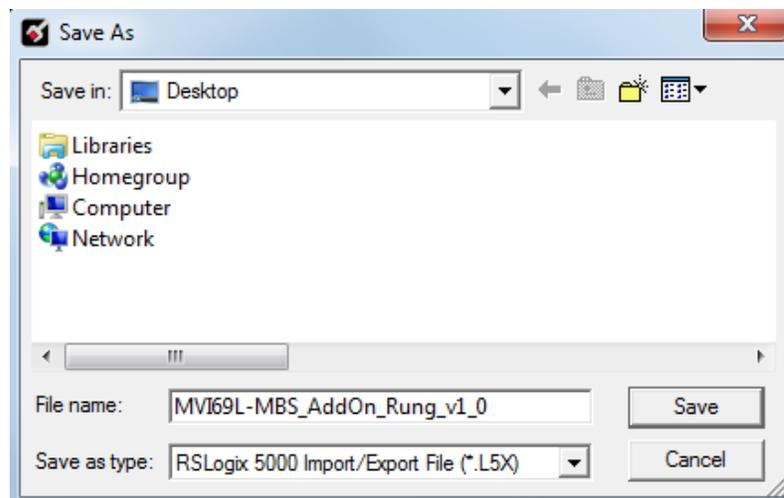
2.4   Creating a New RSLogix 5000 Project

1   Click the **FILE** menu and then choose **NEW**.
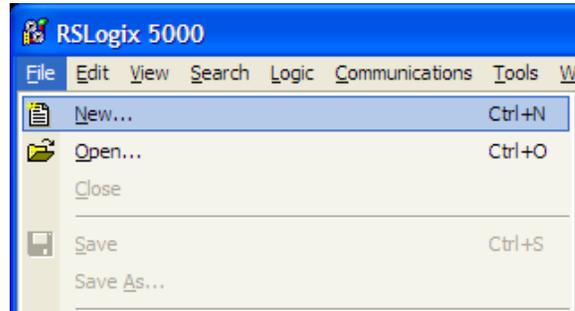
2   Select your CompactLogix controller model.
3   Select **REVISION 16** or newer.
4   Enter a name for your controller, such as *My_Controller*.
5   Select your CompactLogix chassis type.
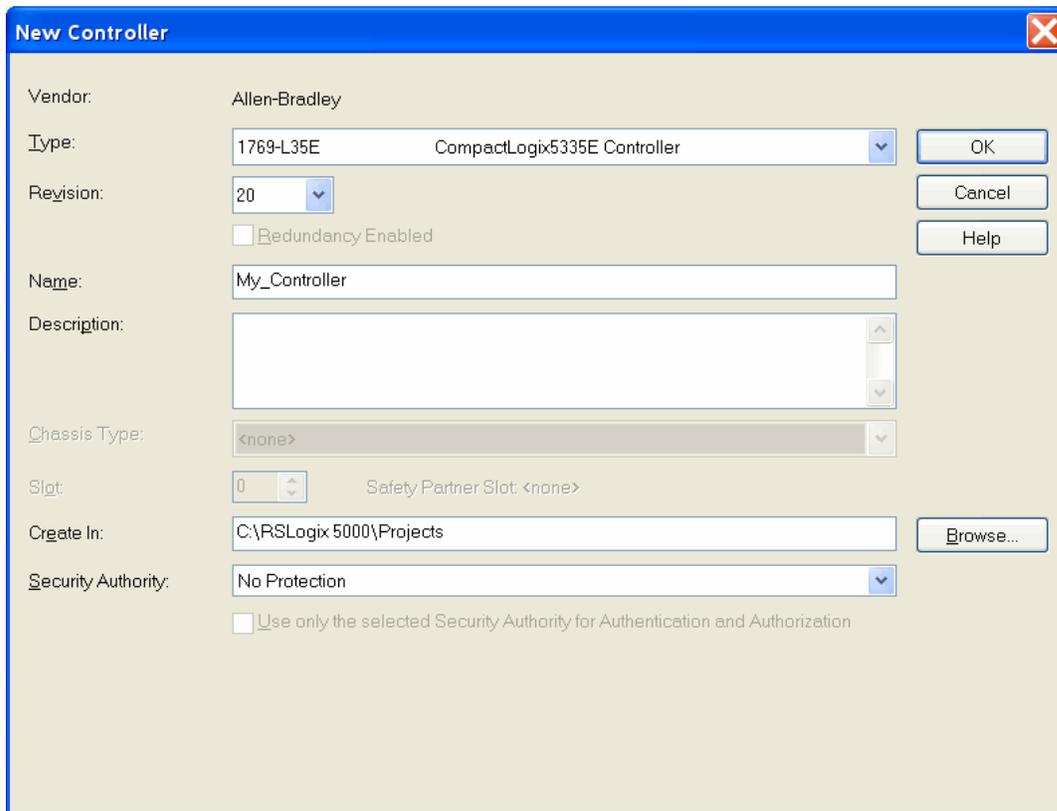
2.5 Importing the Add-On Instruction

1 Open the application in RSLogix 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**.
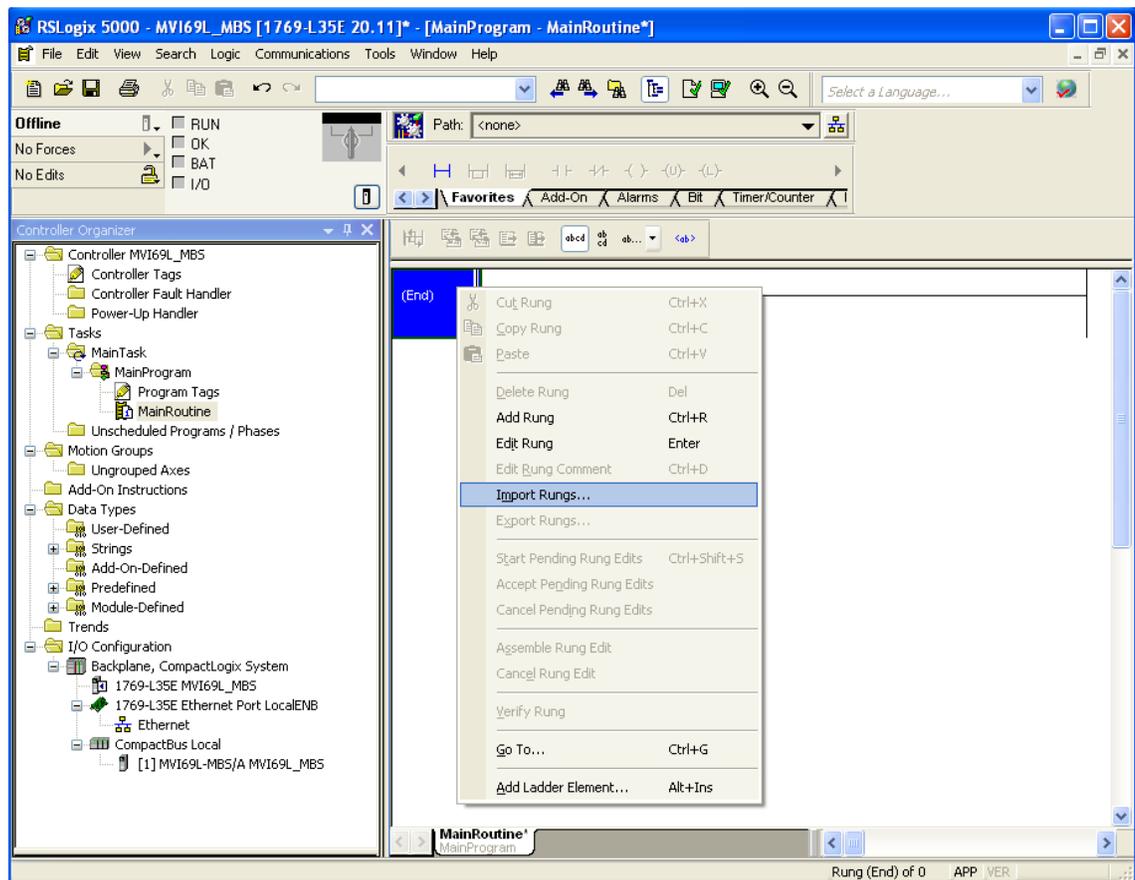
5 Select the **.L5X** file that you exported from PCB.
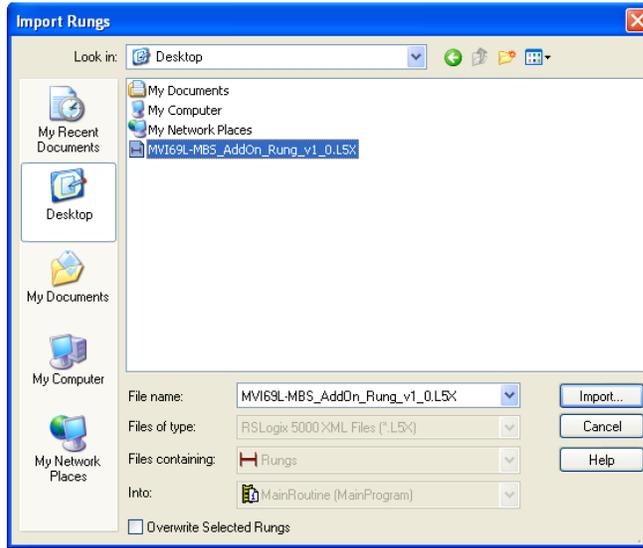
This opens the *Import Configuration* dialog box. Click **TAGS** under **MAINROUTINE** to display the controller tags in the Add-On Instruction.

Note: If you are using RSLogix version 16 or earlier, the *Import Configuration* dialog box does not contain the *Import Content* tree.
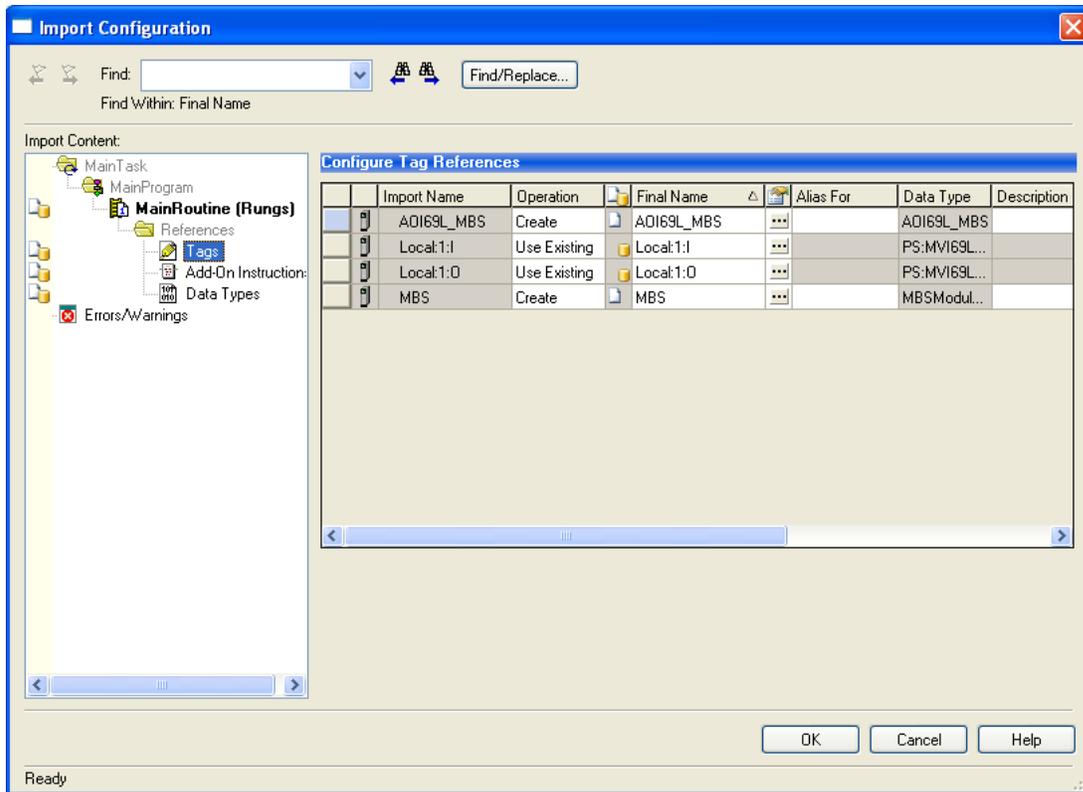
6 If the module is not located in the default slot (or is in a remote rack), edit the connection input and output variables that define the path to the module in the **FINAL NAME** column (**NAME** column for RSLogix version 16 or less). For example, if your module is located in slot 3, change *Local:1:I* in the **FINAL NAME** column to *Local:3:I.* Do the same for *Local:1:O.*

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

7 Click **OK** to confirm the import.
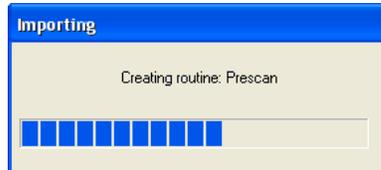
When the import is complete, the new Add-On Instruction rung is present.
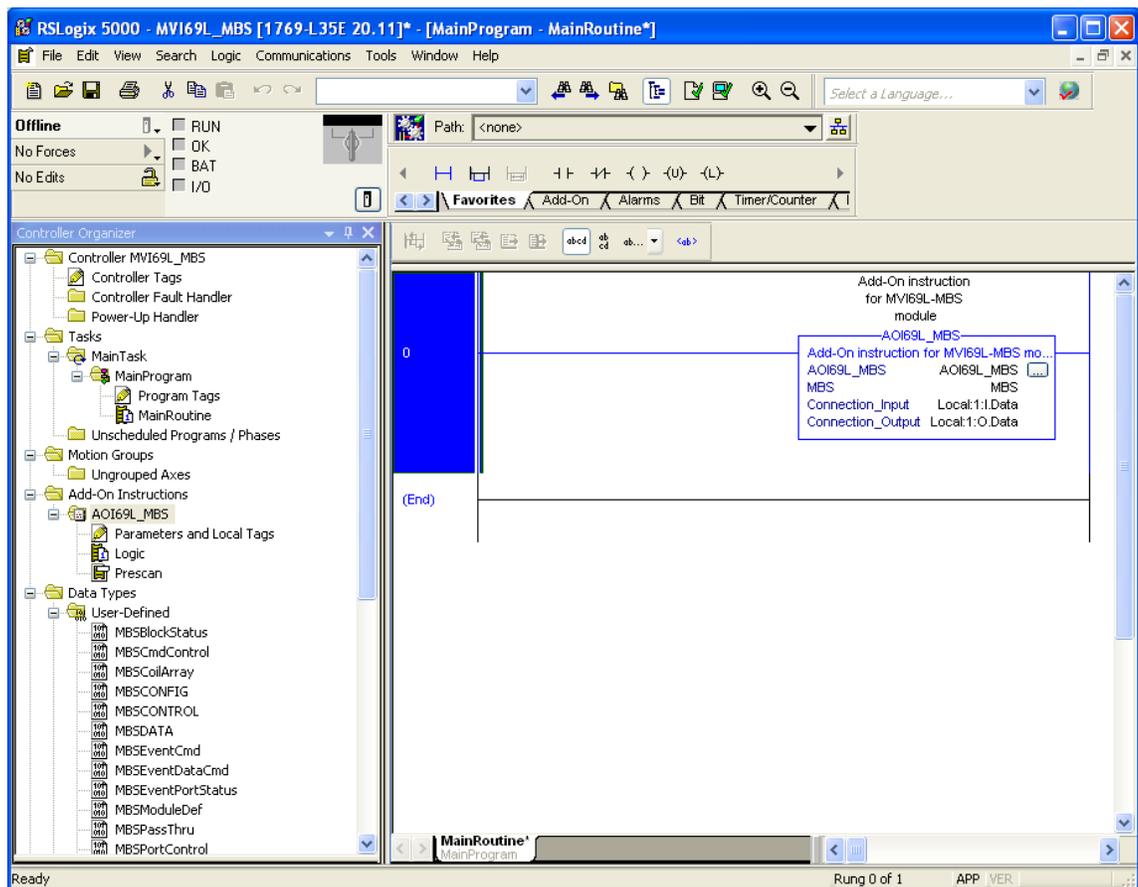
The procedure imports new user-defined data types, data objects and the Add-On instruction to be used in the project with the MVI69L-MBS module.

2.6   Adding Multiple Modules in the Rack (Optional)

Important: This procedure is for multiple MVI69L-MBS modules running in the same CompactLogix rack

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

1 Add a new MVI69L-MBS module to the ProSoft Configuration Builder (PCB) project.
2 Export the module configuration as an L5X file.
3 Add a new MVI69L-MBS to the RSLogix 5000 project.
4 Import the .L5X file into RSLogix 5000 for the new module as an Add-On Instruction.

2.6.1   Adding an Additional Module in PCB

1 Start ProSoft Configuration Builder.
2 Right click **DEFAULT LOCATION** (which you can rename) and choose **ADD MODULE**.
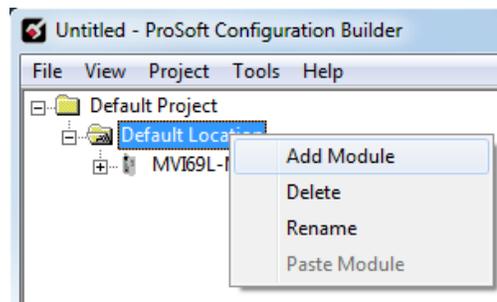
3 Right-click **NEW MODULE** and choose **CHOOSE MODULE TYPE**.
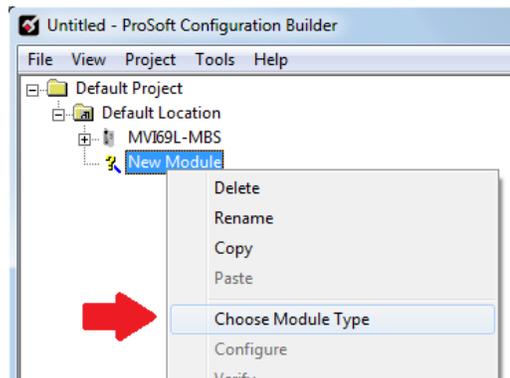
4 In the *Choose Module Type* dialog box, select **MVI69L** in the **PRODUCT LINE FILTER** area, and then select **MVI69L-MBS** as the **MODULE TYPE**. Click **OK**.

5 Select the **MVI69L-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 MVI69L-MBS module a unique name. The default name on a duplicate module appends a number to the end such as **MVI69L-MBS_000**, **MVI69L-MBS_001**, etc.
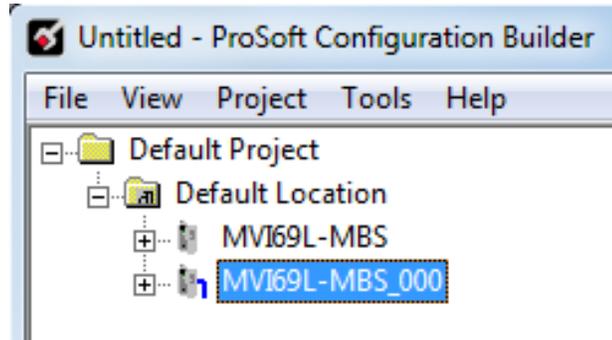
6 You can rename the module by right clicking the module and choosing **Rename**.
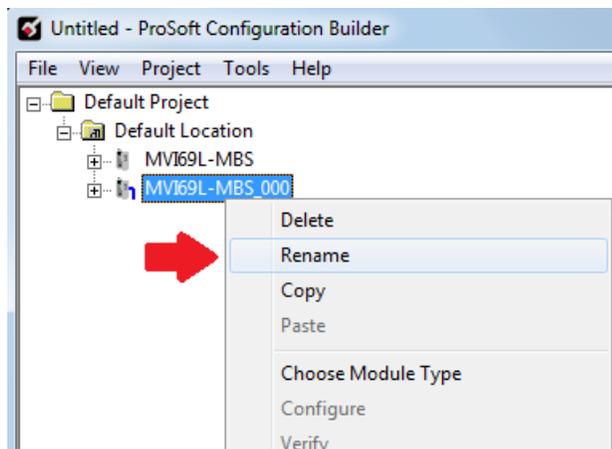
7 Configure the module parameters. See Module Configuration Parameters (page 47) and then export the AOI .L5X file for the new module (right-click the module and choose **EXPORT AOI FILE**. See Creating and Exporting the .L5X File.

2.6.2 Adding an Additional Module in RSLogix 5000

You can place multiple MVI69L-MBS modules in the same rack provided it does not exceed the power distance rating of the CompactLogix rack (see System Requirements). 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 RSLogix 5000 and open the project.
2 In RSLogix 5000, locate the **I/O CONFIGURATION** folder. Right click **COMPACTBUS LOCAL** and choose **NEW MODULE**.
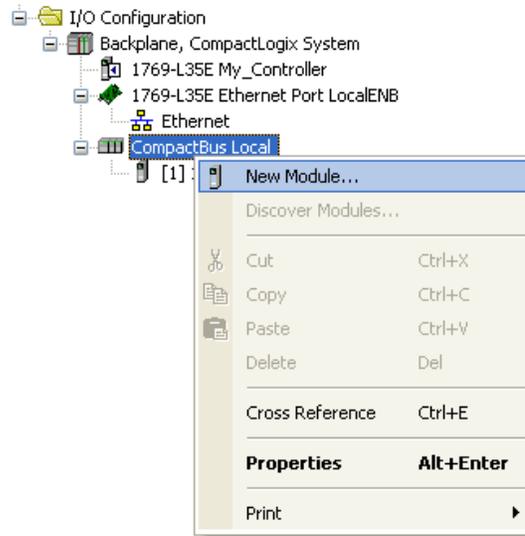
3 In the *Select Module Type* dialog box, select the MVI69L-MBS module.
   o If you are using an Add-On Profile (AOP), this adds the MVI69L-MBS module and configures the relevant parameters. You must be using RSLogix version 15 or later to to use an AOP.
   o If using an AOP is not an option, select **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.
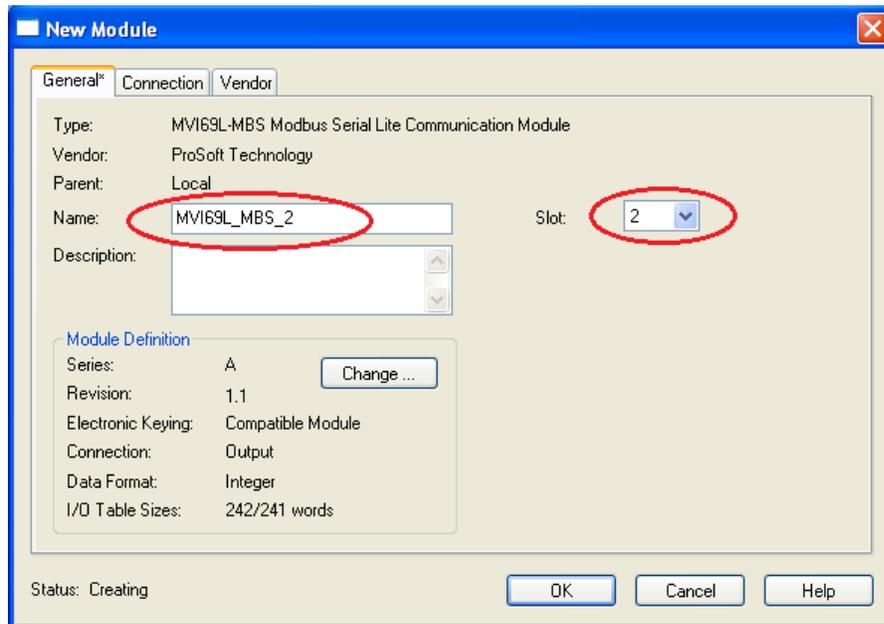
5 Click **OK**. The new module is now visible.
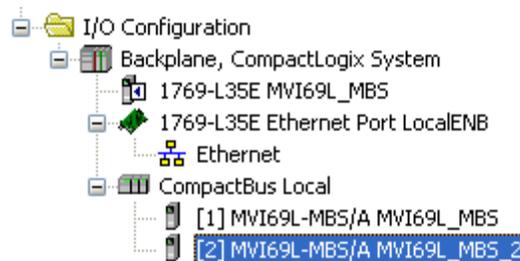
6 You must also import the Add-On Instruction (AOI) for the new module. In the *Controller Organizer* pane, double-click **MAINROUTINE** to open the ladder for the routine.

7 Right-click an empty rung in the routine, and then choose **IMPORT RUNGS…**
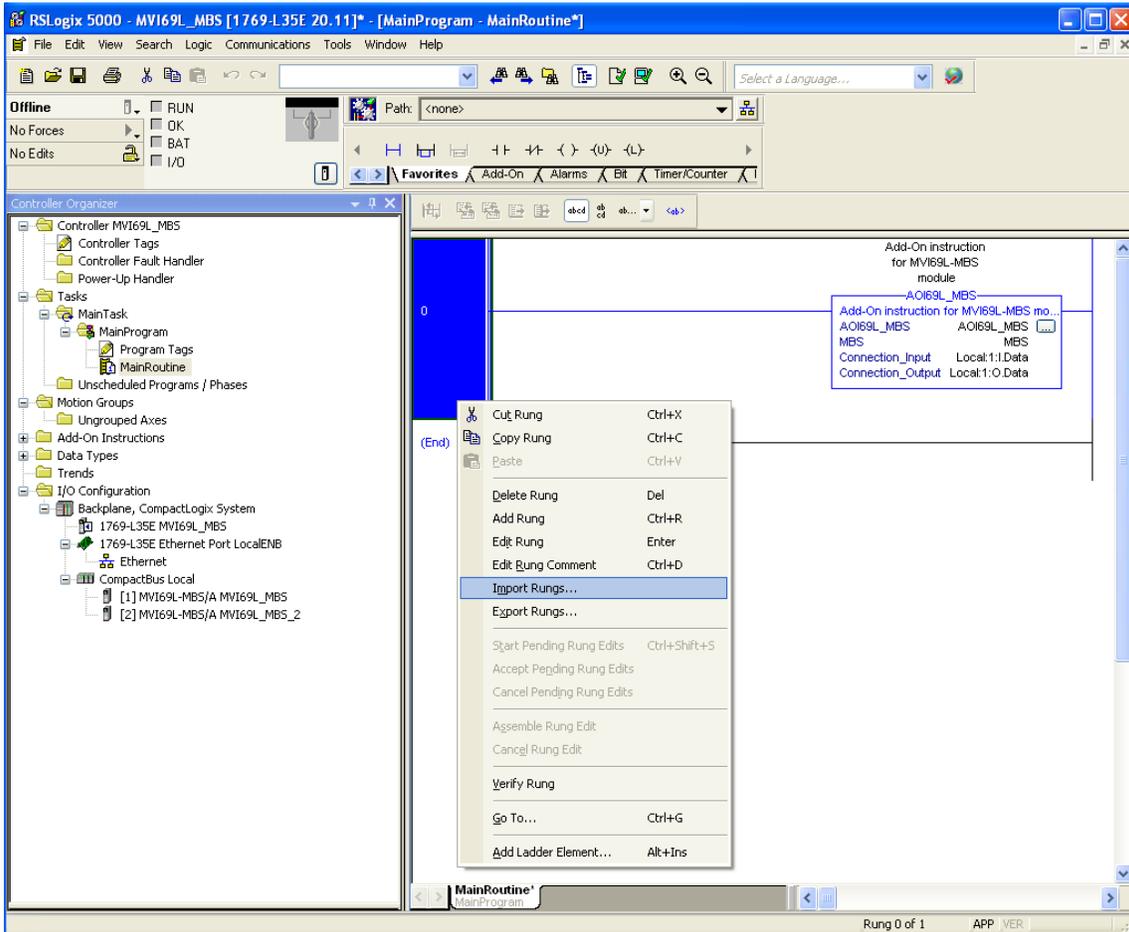
8   Select the .L5X file you created and exported for the new module, and click
IMPORT. Recall that the new .L5X file has a unique filename that is specific to
the new module.
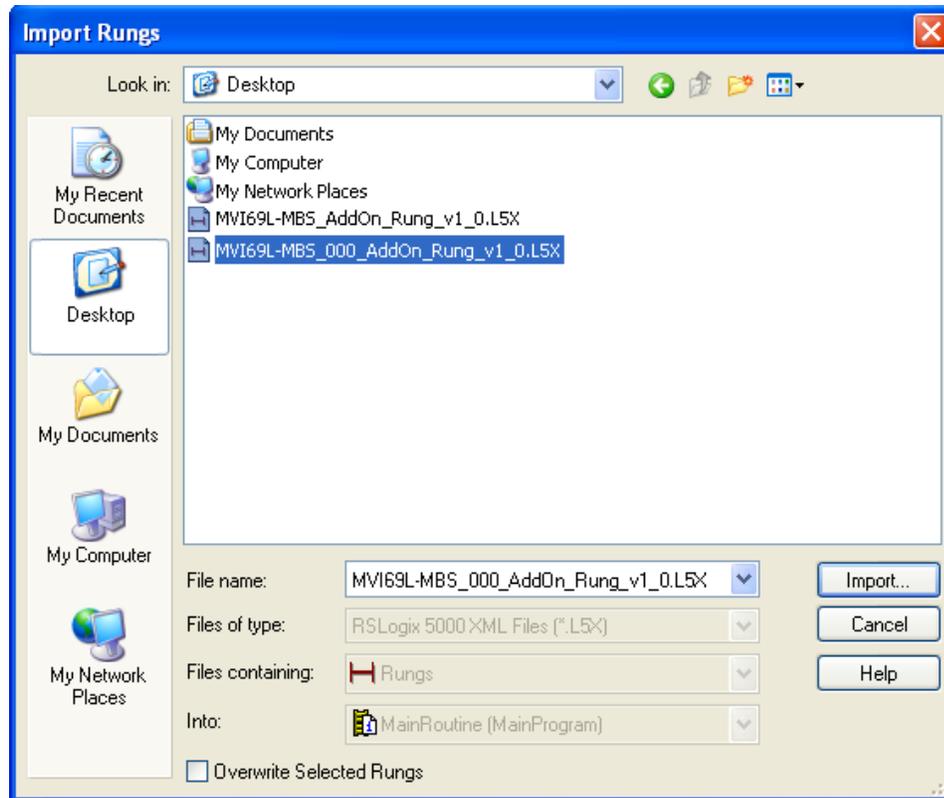
9 This opens the **IMPORT CONFIGURATION** dialog box. Click **TAGS** to show the controller tags in the AOI. You must edit the **FINAL NAME** column of the tags for the second module to make them unique.
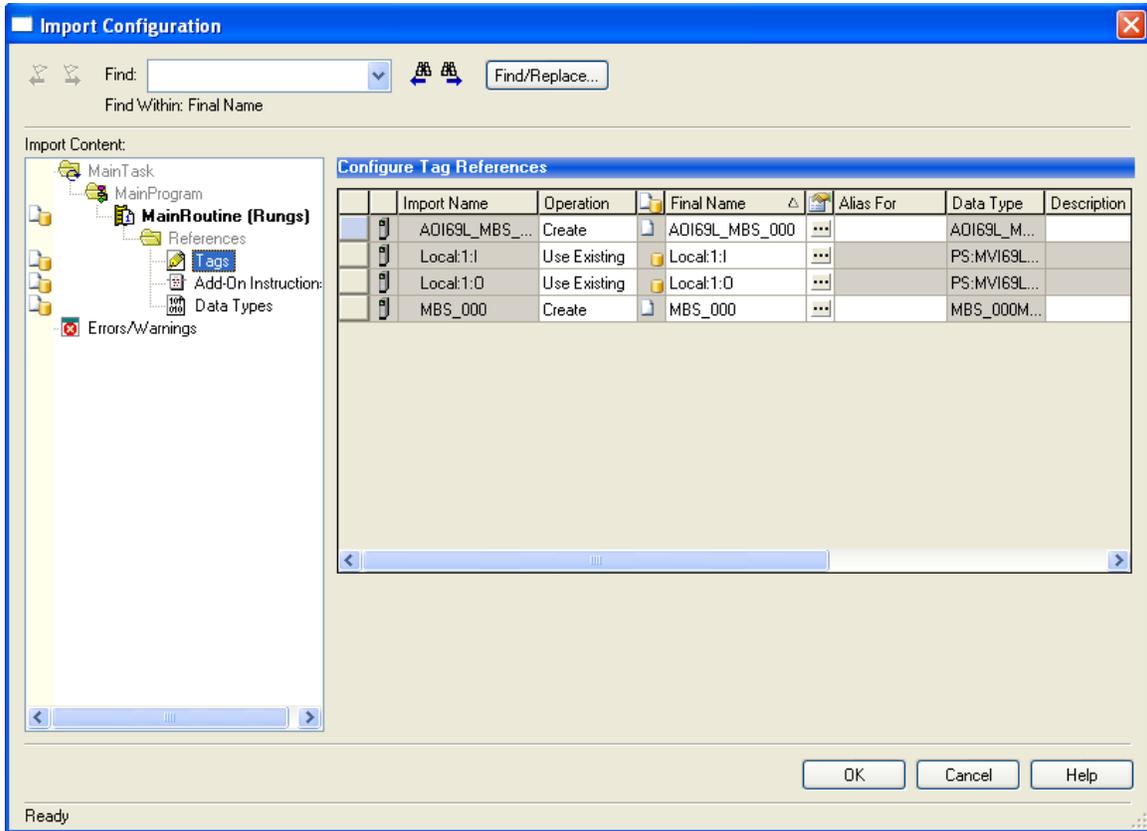
10 Associate the I/O connection variables to the correct module in the corresponding slot number. The default values are Local:1:I and Local:1:O. You must edit these values if the card is placed in a slot location other than slot 1 (Local:1:x means the card is located in slot 1). Since the second card 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.
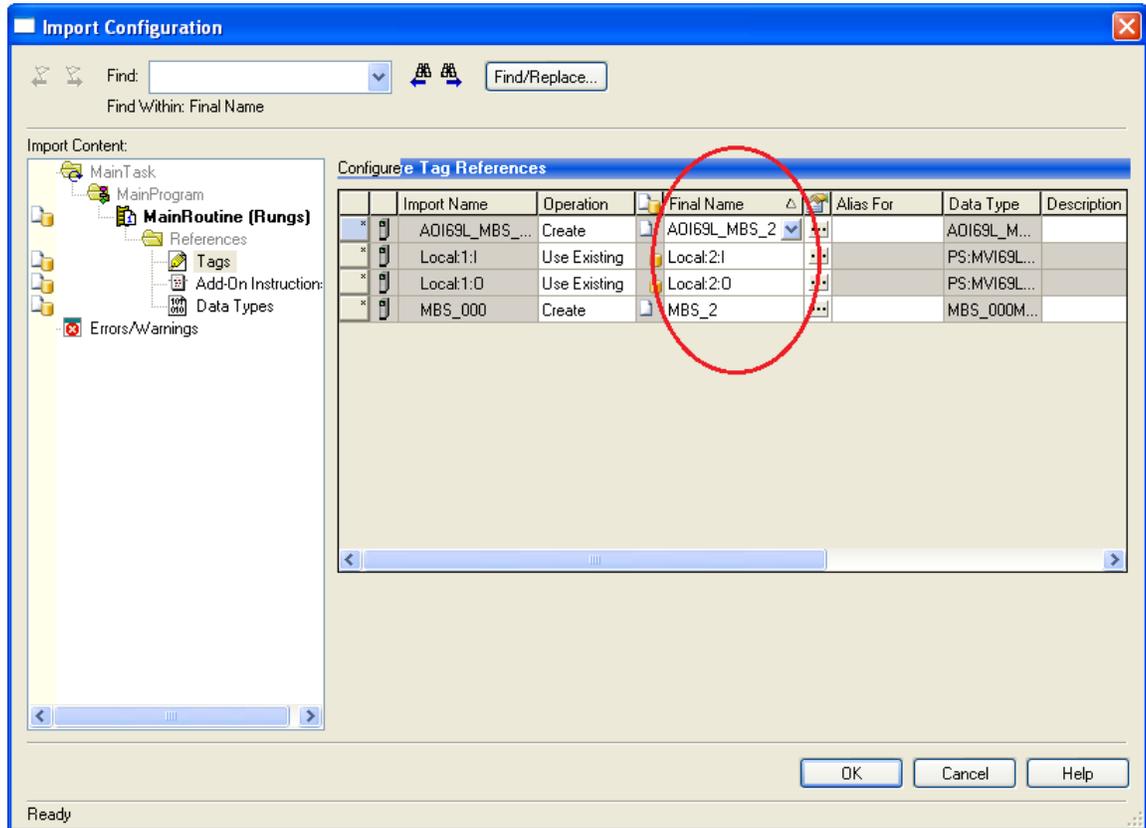
11 Click **OK**.
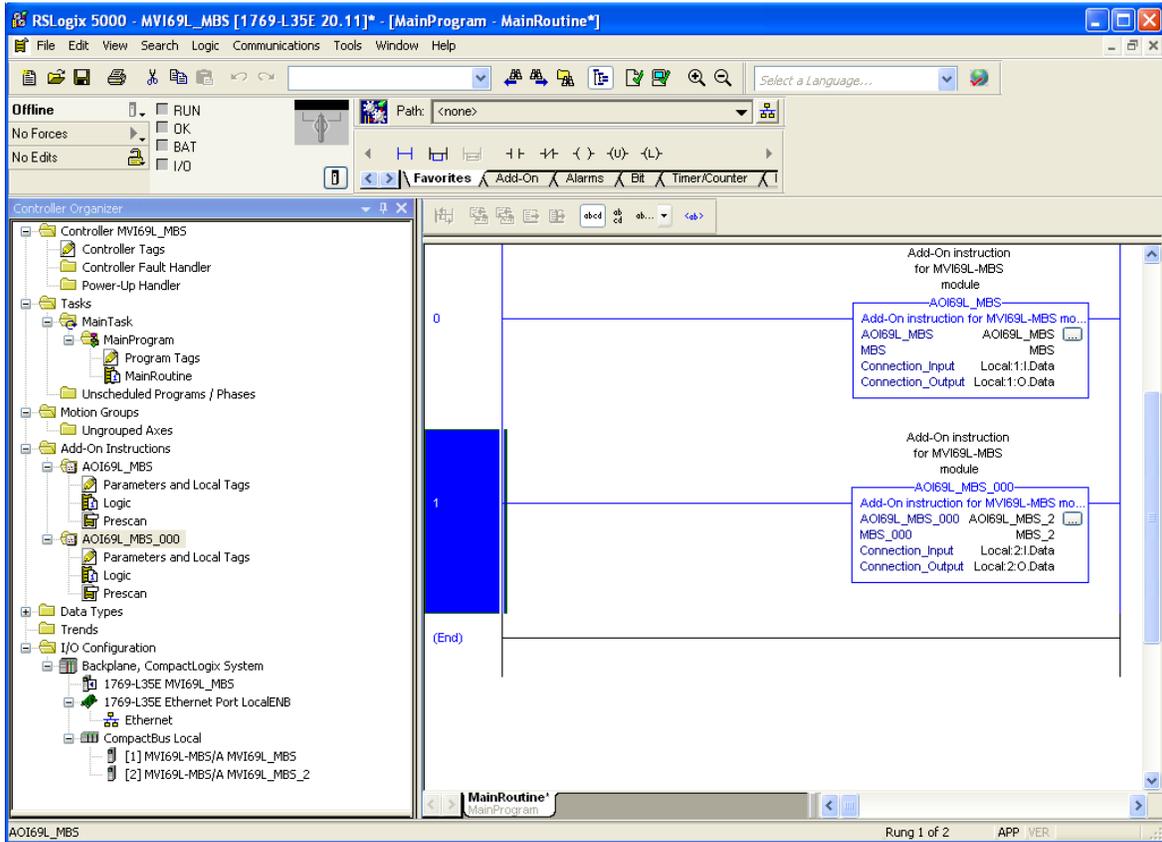
The setup procedure is now complete. Save the project. It is ready to download to the CompactLogix processor.

3   Configuring the MVI69L-MBS Using PCB

In This Chapter

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

You build and edit the module's configuration in ProSoft Configuration Builder. You use PCB to download the configuration file to the CompactLogix processor, where it is stored in the MBS.CONFIG controller tag generated by the previously exported AOI. See Creating and Exporting the .L5X File. When the MVI69L-MBS module boots up, it requests the processor to send the configuration over the backplane in special Configuration Blocks.

See the chapter Adding the Module to RSLogix (page 17) for the procedures to create a new PCB project and export a .L5X file for the processor. This chapter describes the module configuration parameters in detail, as well as how to download the configuration to the processor using PCB.

3.1 Basic PCB Functions

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

Please see the chapter Adding the Module to RSLogix (page 17).

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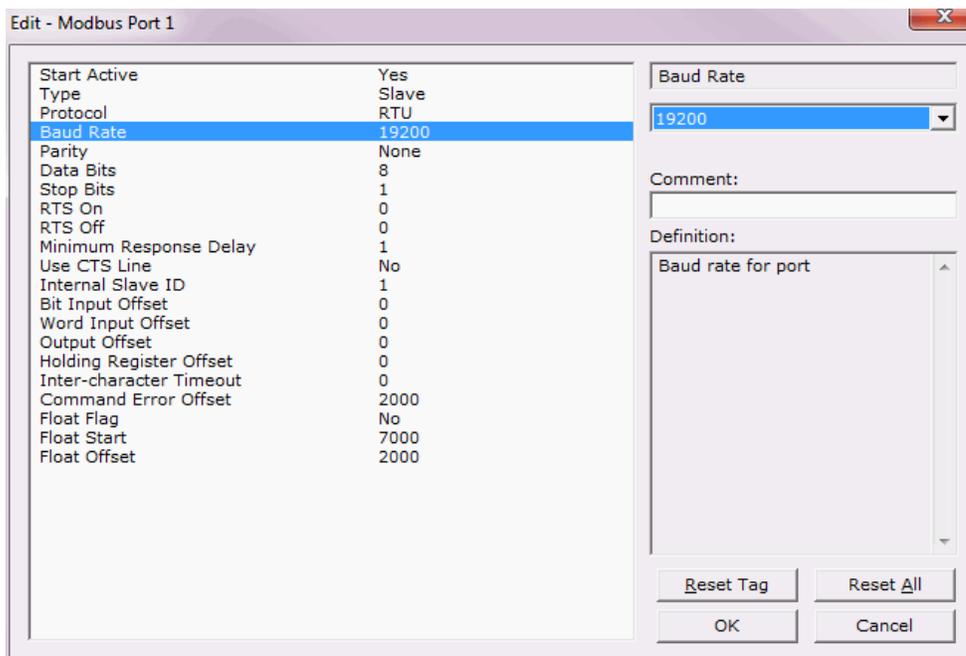
Note: Depending on the parameter, you must enter text, or a valid number, or select from a list of options.

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.
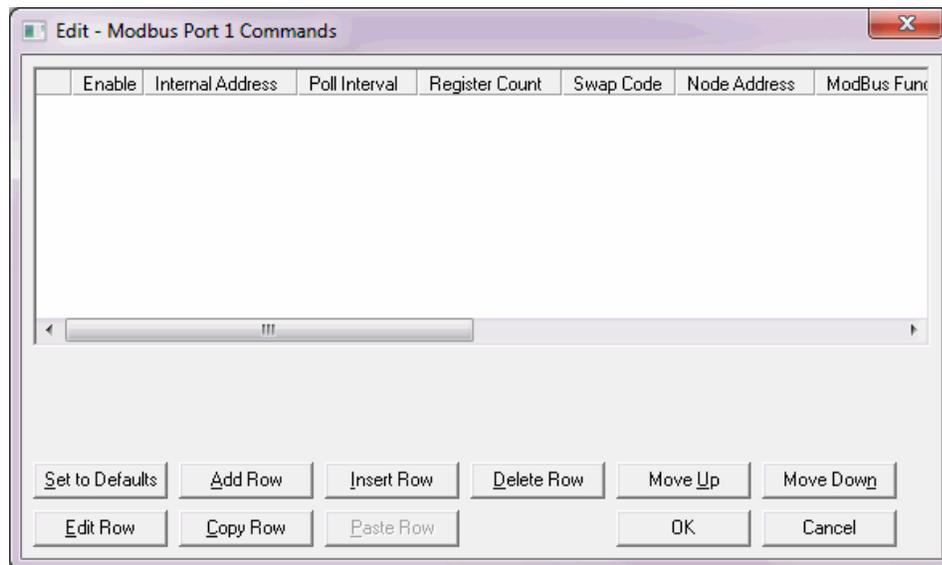
6 To add a row to the table, click **ADD ROW**.
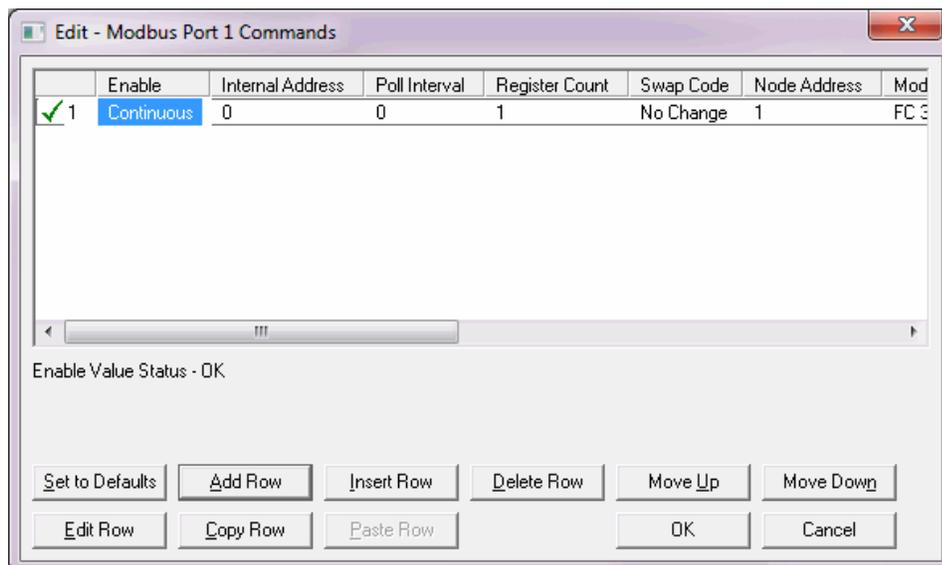
7 To edit the row, click **EDIT ROW**. This opens an *Edit* dialog box.
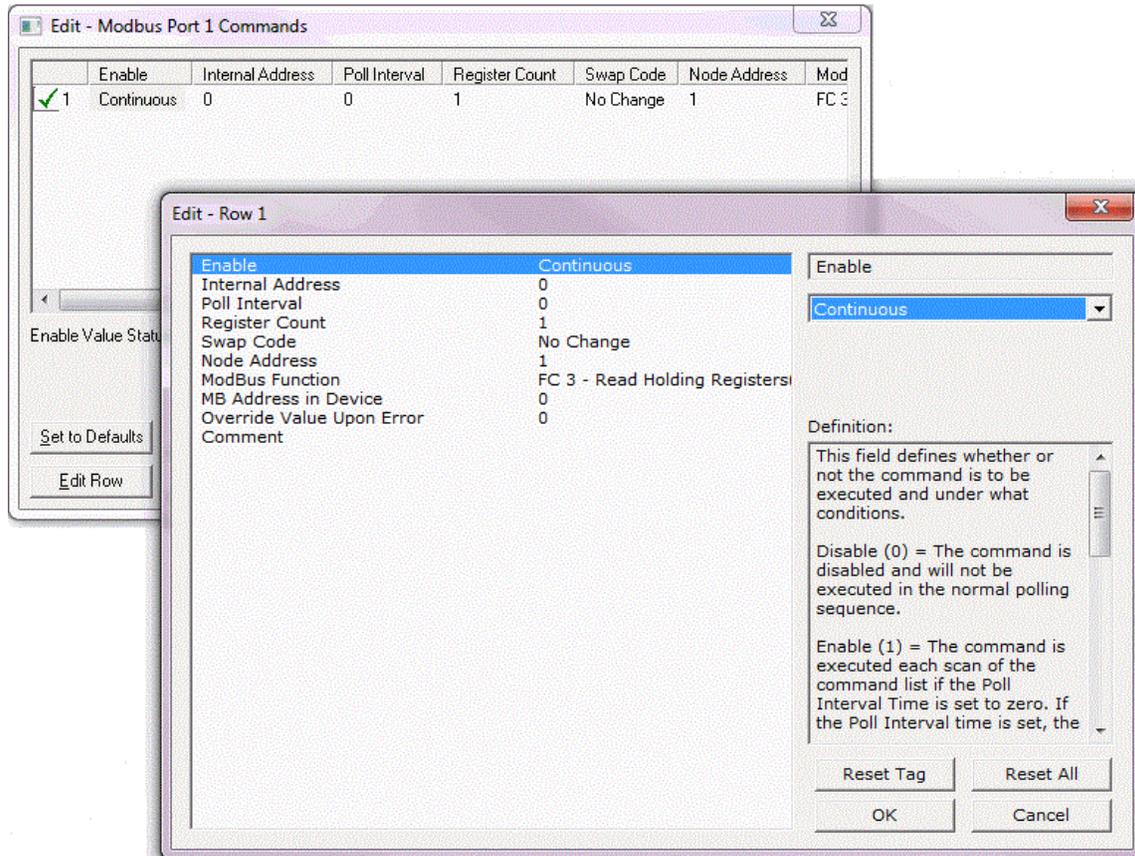
3.1.4 Printing a Configuration File

1 In the main PCB window, right-click the **MVI69L-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. The module uses 240 words of read data (user input data) and 240 words of write data (user output data).

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

Parameter	Value	Description
Module Name	ASCII characters (max. 80)	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.
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 239	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. All other valid values determine the starting location of the data. This data must be placed in the read data range of module memory. This data area includes the module version information and all server error/status data. Refer to MBS.STATUS 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. Using this option requires associated ladder logic to pass the data from the processor to the module.
Slot Number	1 to x	Specifies the slot in the CompactLogix rack for the module.

3.2.2 MBS Port 1 Parameters

In the ProSoft Configuration Builder tree view, double-click the **MODBUS PORT 1** icon.
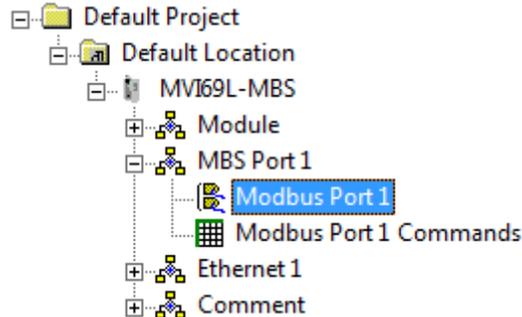
Configuration Parameters Common to Master and Slave

Parameter	Value	Description
Start Active	Yes or No	Specifies whether or not 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 Slave Mode (page 65) 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 to use. All devices communicating through 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 *Ready To Send* (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 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 *hardware handshaking*.
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.
		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. Example: Count = **10**, Slave driver sends 20 16-bit registers for 10 total 32-bit floating-point values.
		If, however, the remote Master sends a count of two (2) for each 32-bit floating-point value it expects to receive or send, or, if you do not plan to use floating-point data in your application, then set this parameter to **No**, which is the default setting.
		You also need to set the *Float Start* and *Float Offset* parameters to appropriate values whenever the *Float Flag* parameter is set to **YES**.
Float Start	0 to 479	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 Float Flag is enabled. For example, if you enter a value of 200, all requests for registers 200 and above are considered as floating-point data.
Float Offset	0 to 479	Defines the start register for floating-point data in the internal database. This parameter is used only if the Float Flag is enabled. For example, if you set the Float Offset value to 100 and the float start parameter to 200, data requests for register 200 use the internal Modbus register 100.

Additional Configuration Parameters as Master

The *Type* parameter must be **MASTER** to configure these parameters. See Configuration Parameters Common to Master and Slave (page 48).

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 *Response Timeout* 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 *slow poll* 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 239	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 230, the command errors are copied to the database as follows: 230: error code for command 0 231: error code for command 1 … An error code of 0 means that the command was successfully sent (no error).

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 48).

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 479	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	240 to 479	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 350, an address request of 0 returns the value at register 350 in the database.
Output Offset	0 to 479	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 479	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 250, a request for address 0 corresponds to the register 250 in the database.

3.2.3 Modbus Port 1 Commands

This section defines the master command list specifications for a Master port. In the ProSoft Configuration Builder tree view, double-click the **MODBUS PORT 1 COMMANDS** icon.
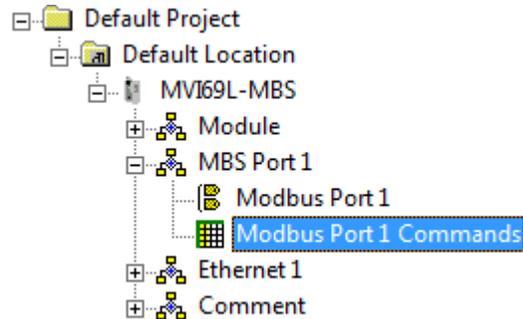
In order to interface the MVI69L-MBS module with Modbus slave devices, you must create a command list. The commands in the list specify the slave device to be addressed, the function to be performed (read or write), the data area in the device to interface with and the registers in the internal database to be associated with the device data.

The Master command list supports up to 30 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 MBS Modbus 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. For information on floating-point support, please see Floating-Point Support (page 108).

Parameter	Value	Description
Enable	0 to 4	This field defines whether the command is to be executed and under what conditions.
		Disabled (0) = The command is disabled and is not executed in the normal polling sequence.
		Continuous (1) = The command is executed each scan of the command list if the *Poll Interval* (see below) is set to zero. If the Poll *Interval* is set to a nonzero value, the command is executed when the interval timer expires.
		Conditional (2) = For write commands only. The command executes only if the internal data associated with the command changes.
		Bit/Word Override upon Error (3) = For read commands only. If a command error occurs, the module overrides the associated database area with the *Override Value Upon Error* parameter value.
		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 *Override Value Upon Error* parameter value.
Internal Address	0 to 479 (word-level) or 0 to 7679 (bit-level)	Specifies the module's internal database register to be associated with the command. Allowable range is 0 to 479 for Modbus Function Codes 3, 4, 6, or 16, and 0 to 7679 for Modbus Function Codes 1, 2, 5, or 15.
		If the command is a read function, the data read from the slave device is *stored* 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 *Read Register Start* and *Read Register Count* parameters in the Module section.
		If the command is a write function, the data to be written to the slave device is *sourced* 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 *Write Register Start* and *Write Register Count* parameters in the Module section.
		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 RSLogix 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.
Poll Interval	0 to 65535 seconds	Specifies the minimum interval between executions of continuous commands (*Enable* code = 1).
		Example: If you set the value to 10, the command executes no more frequently than once 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.
Register Count	1 to 125 (words) or 1 to 800 (coils)	Specifies the number of registers or digital points to be associated with the command. Function Codes 5 and 6 ignore this field as they only apply to a single data point.

		For 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.
		For Function Codes 3, 4 and 16, this parameter sets the number of 16-bit registers to be associated with the command.
Swap Code	0,1,2,3	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.
		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)
		Note: Each pair of characters is a byte. Ex: AB and CD. Two pairs of characters is 16-bit register Ex: ABCD.
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	Specifies the Modbus function code to be executed.
		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 9999	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.
		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.
		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).
Override Value Upon Error		This parameter is only applicable for *Enable Codes* 3 (Bit/Word Override) or 4 (Float Override).
		If an error occurs associated to a read command the module automatically populates the associated database area with this override value.

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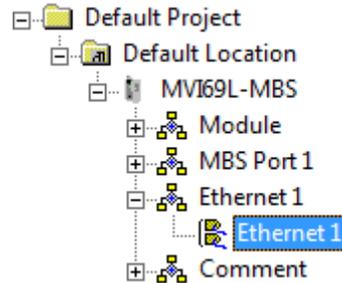
Parameter	Description
IP Address	Unique IP address assigned to the module
Netmask	Subnet mask of module
Gateway	Gateway (if used)

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**.
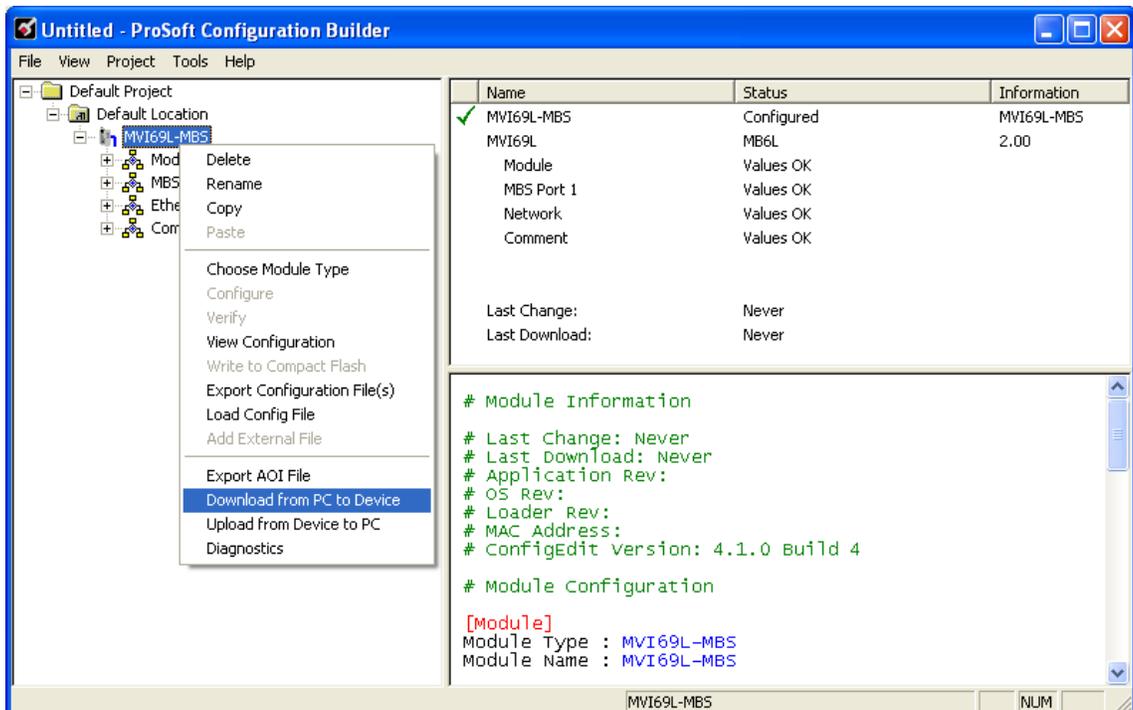
2 In the *Download Configuration File* dialog box, click **RSWHO**.
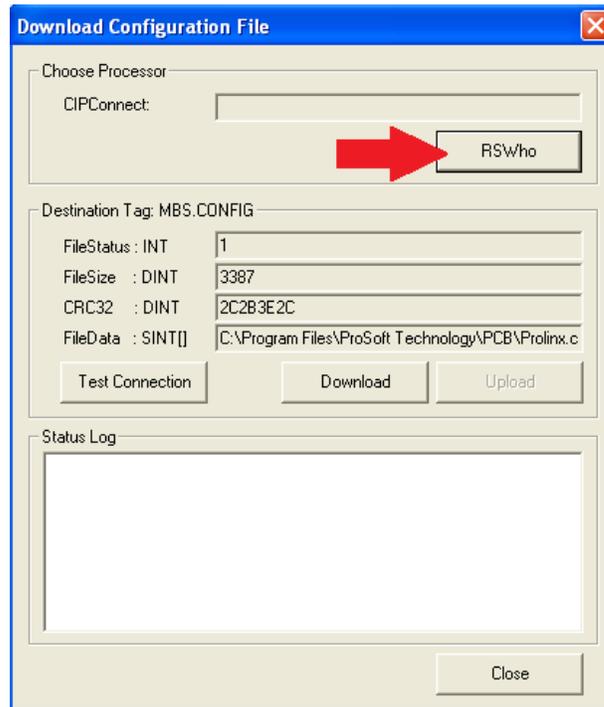
3 Browse to, and then highlight the CompactLogix processor and click **OK**.
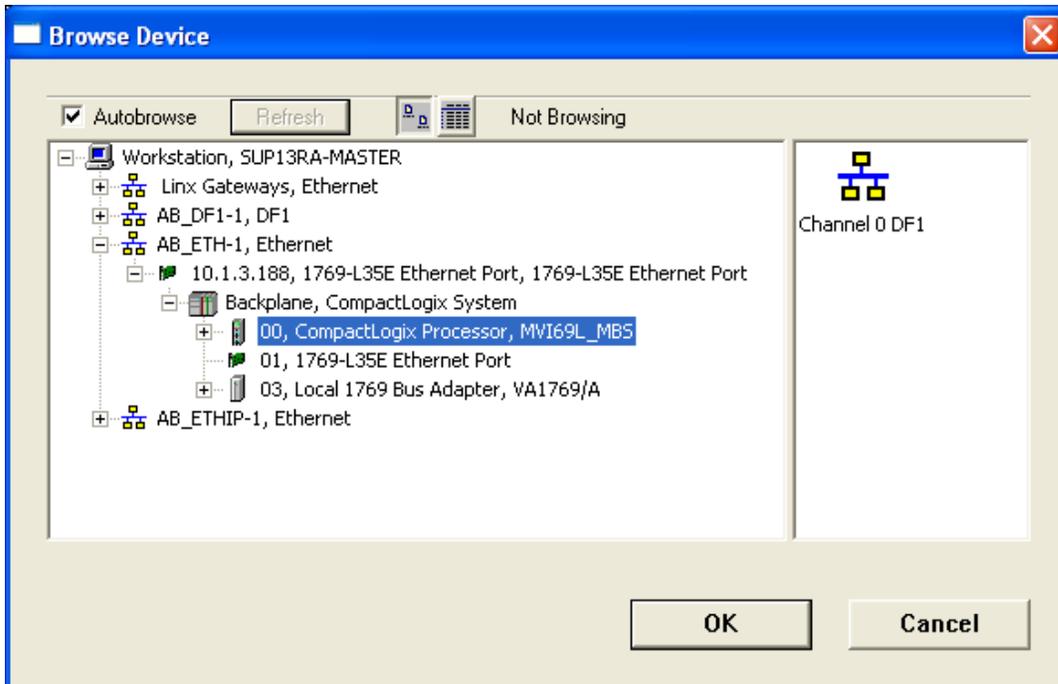
4 Notice the CIPConnect path has been updated in the *Download Configuration File* dialog box. Click **TEST CONNECTION** to verify the path is active and can successfully connect to the processor.
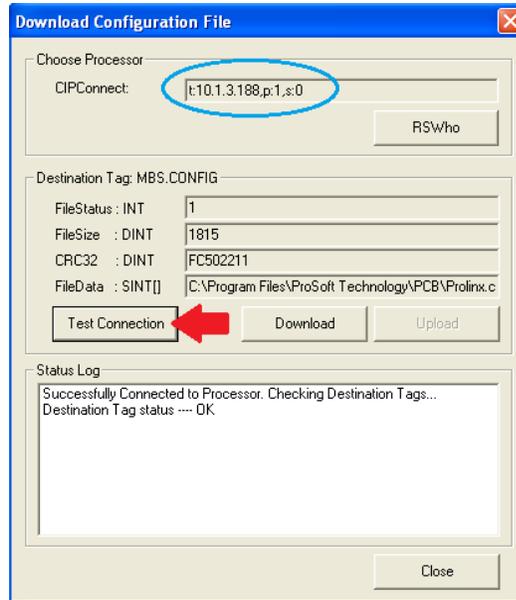
5 When ready, click **DOWNLOAD** to download the configuration file to the processor. Following the download process, the module reboots.
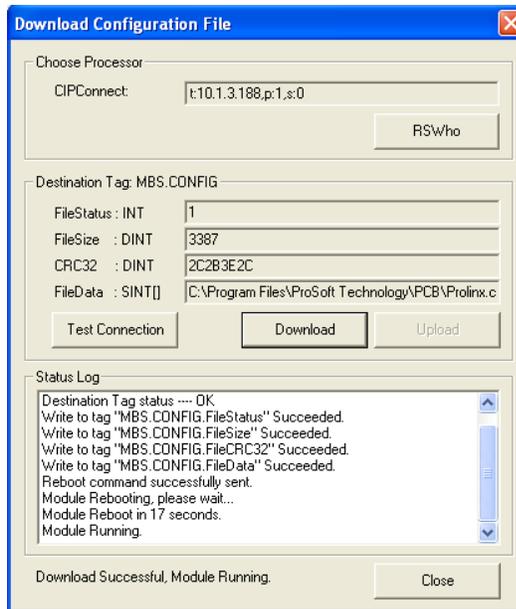
6 After rebooting, the ladder logic sends the configuration data from the processor to the module. When complete, the module starts Modbus communications.

3.4 Uploading the Configuration File from the Processor

1   In the ProSoft Configuration Builder tree view, right-click the **MVI69L-MBS** icon and choose **UPLOAD FROM DEVICE TO PC**.
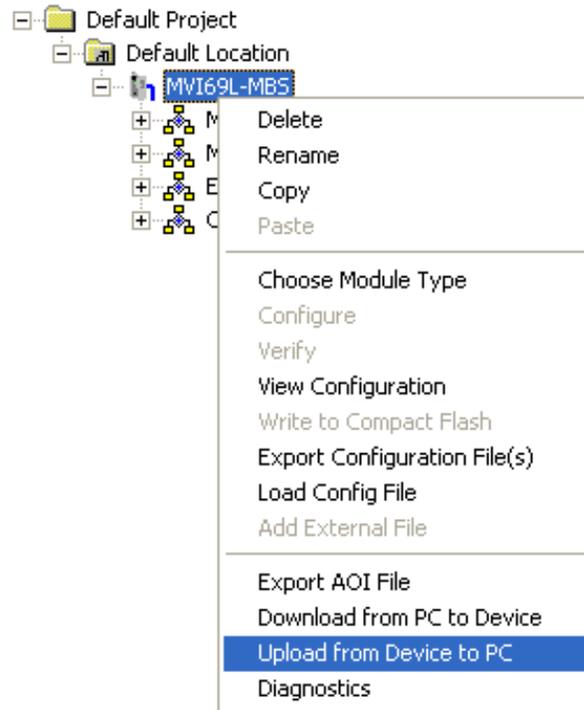
2 In the *Upload Configuration File* dialog box, the CIPConnect path should already be constructed if you have previously downloaded the configuration file from the same PC. If not, click **RSWHO**, browse to, and then select the CompactLogix Processor, and click **OK**.
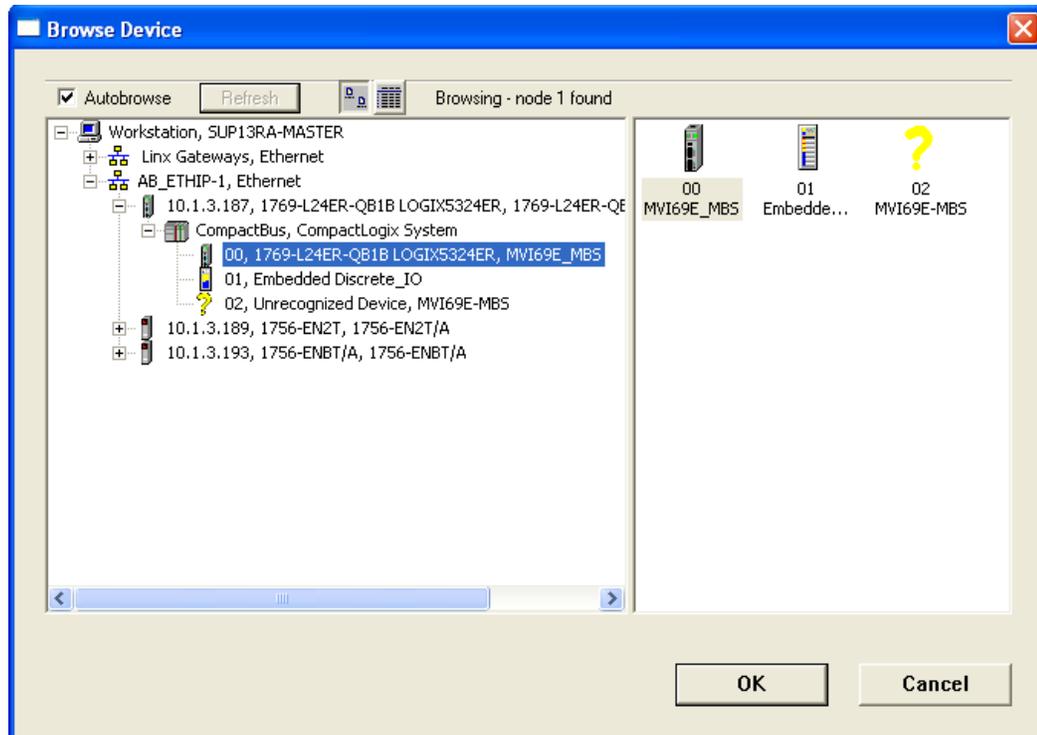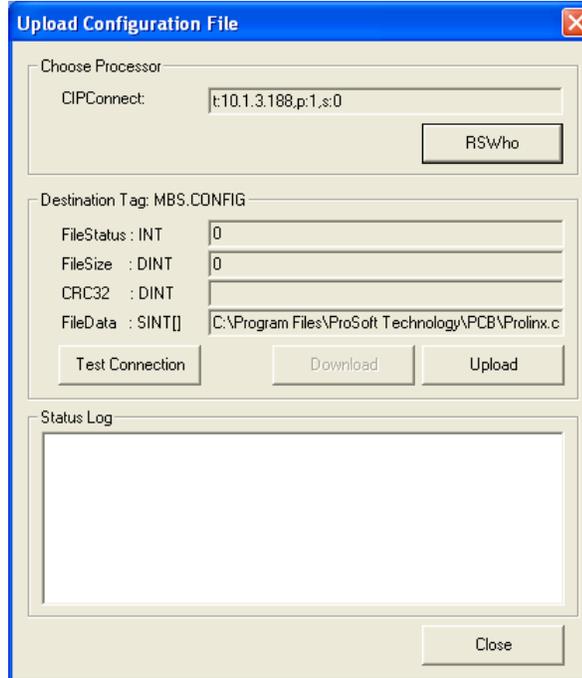
3 Click **TEST CONNECTION** to verify the path is active and can successfully connect to the processor.

4 When ready, click **UPLOAD**. When upload is complete, click **CLOSE.**
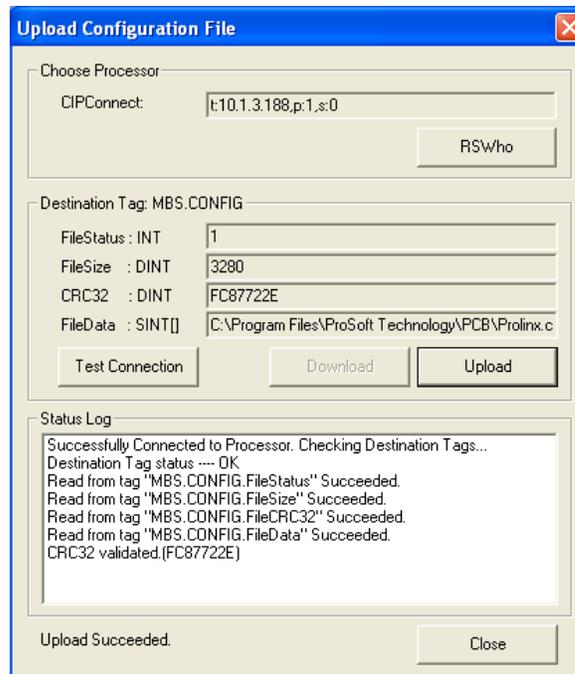
5 ProSoft Configuration Builder now displays the uploaded configuration file.

4   MVI69L-MBS Backplane Data Exchange

In This Chapter

4.1   General Concepts of the MVI69L-MBS Data Transfer

The MVI69L-MBS module uses ladder logic to communicate with the CompactLogix 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 MVI69L-MBS module. This is the order of operations on power-up:

1   The module begins the following logical functions:
- o   Initialize hardware components
- o   Initialize CompactLogix backplane driver
- o   Test and clear all RAM

2   Read configuration from the CompactLogix 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.

4.2   Backplane Data Transfer

The MVI69L-MBS module communicates directly over the CompactLogix backplane. Data is paged between the module and the CompactLogix 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 is 242 words. This data area permits fast throughput of data between the module and the processor.

The processor inserts data to the module's output image to transfer to the module. The module's program extracts the data and places it in the module's internal database. The output image for the module is 241 words.

The following illustration shows the data transfer method used to move data between the CompactLogix processor, the MVI69L-MBS module and the Modbus Network.
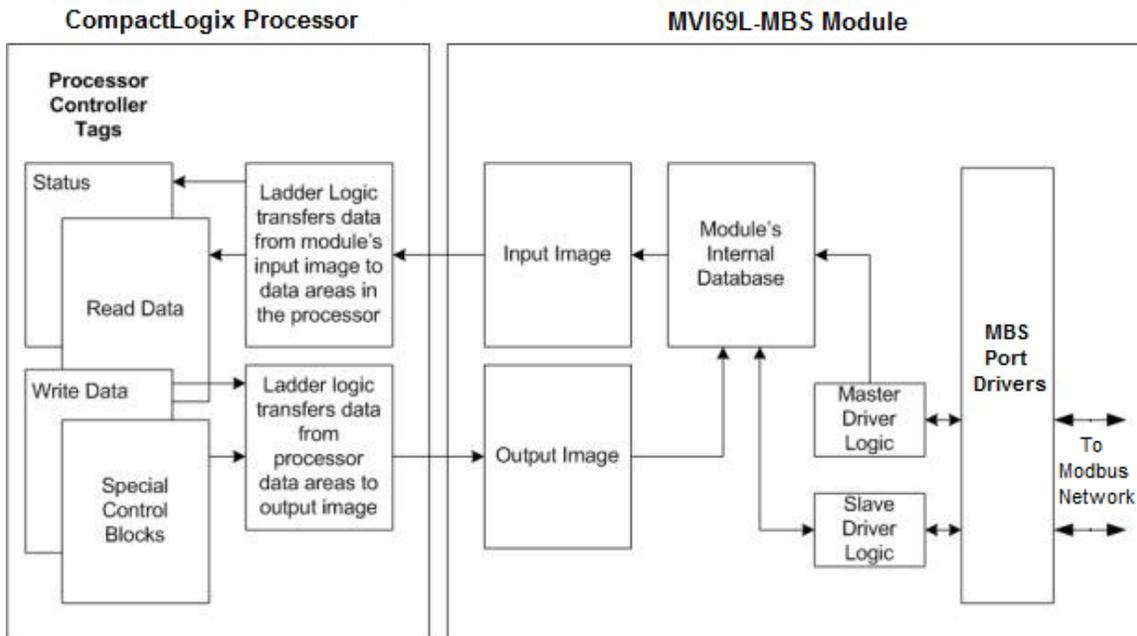
All data transferred between the module and the processor over the backplane is through the input and output images. Ladder logic in the CompactLogix processor interfaces the input and output image data with data defined in the Controller Tags. All data used by the module is stored in its internal database. This database is defined as virtual MBS data tables with addresses from 0 to 239 each.

4.3   Normal Data Transfer

4.3.1   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 240	Write Data	240

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

4.3.2   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 241	Read Data	241

4.3.3 Read and Write Block Transfer Sequences

There are 240 words of data transferred per block along the backplane between the module and the processor.

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. The application uses one read and one write block, the sequence is as follows:

$$R1W1 \rightarrow R1W1 \rightarrow R1W1 \rightarrow R1W1 \rightarrow ...$$

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 Ethernet port.

The backplane parameters are configured as follows:
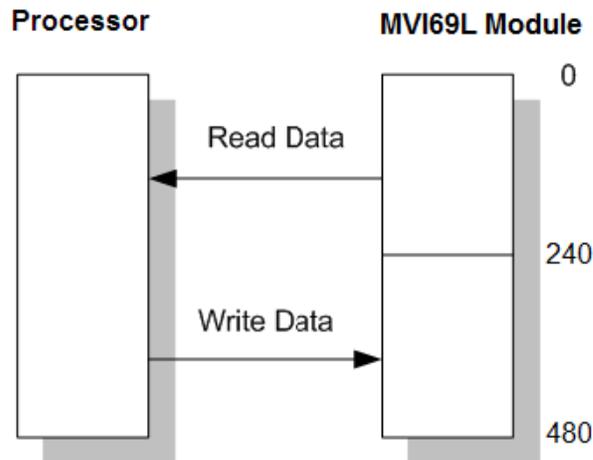
Database address 0 to 239 is continuously transferred from the module to the processor. Database address 240 to 479 is continuously transferred from the processor to the module.

4.4 Data Flow Between the Module and Processor

The following topics describe the flow of data between the two pieces of hardware (CompactLogix processor and MVI69L-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.

4.4.1 Slave Mode

In Slave Driver mode, the MVI69L-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 application 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	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.
	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.
	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 *WriteData* controller tag array at an address that corresponds to the Modbus Address received in the write command.
	d. During normal backplane communications, the data from the *WriteData* 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.
	Note: The *ReadData* array is not used in Pass-Through mode.
4	Same as normal mode.
5	Same as normal mode.

4.4.2 Master Mode

In Master mode, the MVI69L-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 1 Commands (page 52). 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 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 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 write to the same internal database address of the module, the results are invalid. All commands containing invalid data are ignored by the module.

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 1 Commands (page 52). This list contains up to 30 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 particular 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

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 92). You can view the command error codes through the Ethernet diagnostics port; refer to Diagnostics and Troubleshooting (page 79). 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 50).

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.

5   Using Controller Tags

Controller tags are a feature of the RSLogix software and are part of the MVI69L-MBS Add-On Instruction. Refer to the section Adding the Module to RSLogix (page 17) for information on importing the Add-On Instruction into RSLogix.

5.1   Controller Tags

Data related to the MVI69L-MBS is stored in the ladder logic in variables called controller tags. You use controller tags to manage communication between the MVI69L-MBS module and the CompactLogix 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 MVI69L-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 RSLogix 5000 window. This controller tag structure is arranged as a tree structure. Individual controller tags are found at the lowest level of the tree structure. Each individual controller tag is defined to hold data of a specific type, such as integer or floating-point data.

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 MVI69L-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.

5.1.1 MVI69L-MBS Controller Tags

The main MVI69L-MBS controller tag structure, *MBS*, is broken down into five lower-level controller tag structures.
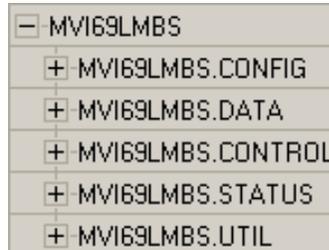
The five lower-level controller tag structures contain other controller tags and controller tag structures. Click the **[+]** sign next to any controller tag structure to expand it and view the next level in the structure.

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 240-element integer arrays.

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).

5.2   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 MVI69L-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.

5.2.1   MVI69L-MBS User-Defined Data Types

Twenty different UDTs are defined for the MVI69L-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.

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 240-element integer arrays.

Name:	MBSModuleDef

Description:	Main module definition

Members: Data Type Size: 67168

	Name	Data Type	Style	Description	External Access
	⊞ CONFIG	MBSCONFIG		Configuration file	Read/Write
	⊟ DATA	MBSDATA		Database data	Read/Write
	—ReadData	INT[240]	Decimal	Read from the module	Read/Write
	—WriteData	INT[240]	Decimal	Write to the module	Read/Write
	⊞ CONTROL	MBSCONTROL		Special tasks requeste	Read/Write
	⊞ STATUS	MBSSTATUS		Status	Read/Write
	⊞ UTIL	MBSUTIL		Tags used for internal l	Read/Write

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.

5.3   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)

5.3.1 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
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.
FileSize	Configuration file size (*MBS.CONFIG.FileData* array) in bytes.
FileCRC32	CRC checksum of the configuration file stored in the array.
FileStatus	Configuration file status. 0 = No file present, 1 = File present

5.3.2 MBS.DATA

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

Tag Name	Description
ReadData	Data area copied from the module to the processor. This 240 element array stores the Modbus data coming into the module from the Modbus network.
WriteData	Data area copied from the processor to the module. This 240 element array stores the outgoing data sent from the module to the Modbus network.

5.3.3 MBS.CONTROL

This array handles special tasks requested by the processor.

MBS.CONTROL.PortControl.

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

Tag Name	Range	Description
Set	0 or 1	Sends Port Control to module
Get	0 or 1	Reads Port Control from module
Port1	n/a	Definition of Port 1 Control
Port1.Active	0 or 1	Port Control: Disable = 0, Enable = 1
Port1.CmdEnableBits[x]	0 or 1	Index of command to be controlled. Example: Command 20 in port 1 command list can be controlled at CmdEnableBits[1].3 - This is the 20th bit offset.

MBS.CONTROL.CmdControl.

This array allows the processor to dynamically enable configured commands.

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

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
EventCmdTrigger	0 or 1	Toggle to send Event Command. 0 = Disable, 1 = Enable
PortNumber	1	Port number to be associated with command.
SlaveID	1 to 248	Slave ID of Modbus slave
InternalDBAddress	0 to 479 or 0 to 3839 (bit-level)	Used only if UseModuleDBAddress=1. Allowable range is 0 to 479 for Function Codes 3, 4, 6, or 16, and 0 to 3839 for Function Codes 1, 2, 5, or 15
PointCount	0 to 125	Number of bit/words used in this command.
SwapCode	0 to 3	Swap code 0 = no swap, 1 = word swap, 2 = words & byte swap, 3 = byte swap
ModbusFunctionCode	-	Modbus function code (1,2,3,4,5,6,15, or 16)
DeviceDBAddress	0 to 9999	Modbus address of the target slave database
EventCmdStatusReturned	-	Event status returned by the module

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
CmdTrigger	0 or 1	Toggle to send Event Command. 0 = Disable, 1 = Enable
GetStatusTrigger	0 or 1	Toggle to retrieve event status. 0 = Disable, 1 = Enable
PortNumber	1	Port number to be associated with command
SlaveAddress	1 to 248	Slave ID of Modbus slave
ModbusFunctionCode	-	Modbus function code (5,6,15, or 16)
DeviceDBAddress	0 to 9999	Modbus address of the target slave database
PointCount	0 to 125	Number of bit/words associated with this command.
Data[x]	0 to 49	Data values to be sent to the slave
EventCmdStatusReturned	-	Command status
Port1Status	-	Port 1 Status array
Port1Status.Status	-	Status code. See Communication Error Codes (page 92).
Port1Status.LastError	-	Last error code

MBS.CONTROL.SlavePoll.

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

Tag Name	Range	Description
Port1	-	Port 1 slave polling control array
Port1.EnableSlaves	0 or 1	Slave Poll request 0 = Disable, 1 = Enable
Port1.EnableSlaveCount	1 to 60	Number of slaves to be enabled
Port1.EnableSlavesIDs[x]	-	Data array associated to enable slave request where word x corresponds to slave ID x (0-based). 1 = Enable slave
Port1.DisableSlaves	0 or 1	Triggers disable slaves request 0 = Disable, 1 = Enable
Port1.DisableSlaveCount	1 to 60	Number of slaves to be disabled
Port1.DisableSlavesIDs[x]	-	Data array associated to disable slave request where word x corresponds to slave ID x (0-based). 1 = Disable slave
Port1.GetSlavesStatus	0 or 1	Triggers request to read slave status 0 = Disable, 1 = Enabled
Port1.SlavesStatus[x]	-	Data array with status

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

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

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

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

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

5.3.4 MBS.STATUS

This array contains status data for the module.

Tag Name	Description
PassCnt	Program cycle counter – this value is incremented each time a complete program cycle occurs in the module
Product	Product code
Rev	Firmware revision level number
OP	Operating level number
Run	Run number
Port1Stats	Port 1 status
Port1Stats.CmdListReq	Total number of requests made from port 1 to slave devices on the network
Port1Stats.CmdListResp	Total number of slave response messages received on port 1
Port1Stats.CmdListErr	Total number of command errors processed on port 1. These errors could be due to a bad response or command
Port1Stats.PortReq	Total number of messages sent out of port 1
Port1Stats.PortResp	Total number of messages received on port 1
Port1Stats.PortErrSent	Total number of message errors sent out of port 1
Port1Stats.PortErrRec	Total number of message errors received on port 1
MBS.STATUS.Port1Stats.CurrErr	Not used
MBS.STATUS.Port1Stats.LastErr	Not used
Block	Backplane transfer status
Block.Read	Total number of read blocks transferred from the module to the processor
Block.Write	Total number of write blocks transferred from the processor to the module
Block.Parse	Total number of blocks successfully parsed that were received from the processor
Block.Event	Total number of event command blocks received from the processor
Block.Cmd	Total number of command blocks received from the processor
Block.Err	Total number of block transfer errors recognized by the module
Port1LastErr	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.
Port1PreviousErr	For a slave port, this field contains the value of the last error code returned. For a master port, this field contains the index of the command with an error.

5.3.5 MBS.UTIL

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

Tag Name	Description
ReadDataSizeGet	Holds Read Data array size (240)
WriteDataSizeGet	Holds Write Data array size (240)
ReadDataBlkCount	Number of Read Data blocks (1)
WriteDataBlkCount	Number of Write Data blocks (1)
RBTSremainder	Not used
WBTSremainder	Not used
BlockIndex	Computed block offset for data
LastRead	Latest Read Block ID received from the module
LastWrite	Latest Write Block ID to be sent to the module
LastWriteInit	Latest Write Block ID used during initialization
ConfigFile	Holds variables for configuration file transfer
ConfigFile.WordLength	Length of configuration data to be included in block transfer
ConfigFile.BlockCount	Not used
ConfigFile.FileOffset	Offset in configuration file to use as a starting point for copying over configuration data
ConnectionInputSize	Holds size of the Connection Input array (240)
BlockTransferSize	Size of the backplane transfer blocks (240)
SlotNumber	Slot number of the module in the rack
EventBlockID	Holds Block ID for Event Command
EventCmdPending	Keeps an Event Command message from being sent to the module before the previous Event Command is completed
PollStatusOffset	Offset in slave status data array to use as a starting point for copying over slave status data
CmdsAddedToQueue	Number of Command Control messages added to the command queue
CmdControlBlockID	Holds Block ID for Command Control
CmdCntrlPending	Keeps a Command Control message from being sent to the module before the previous Command Control is completed
EventDataCmdPending	Keeps an Event Command with Data message from being sent to the module before the previous Event Command with Data is completed
BootTimer	Timer used to clear both cold and warm boot requests
PassThru[ ] Array	Holds variables used for processing pass-through messages

6 Diagnostics and Troubleshooting

In This Chapter

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

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

6.1   Ethernet LED Indicators

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

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

6.2   LED Status Indicators

The LEDs indicate the module's operating status.

LED	Status	Indication
ETH	On	Ethernet communications are ok
	Off	No Ethernet cable connected
P1	Green	Data is being transferred between the module and the Modbus network on Port 1
	Red	Communication error detected
	Off	No Modbus network activity detected
CFG	Green	Configuration is ok
	Yellow	Module is reading configuration
	Red	Error setting up Modbus protocol driver, failed startup, or module shutting down
	Off	Processor is in Program mode
BP	Green	The LED is on when the module is performing a write operation on the backplane. Under normal operation, the LED should blink rapidly on and off.
	Red	Major fault or module shutting down
OK	Green	Module is ok
	Red	The program has detected an error or is being configured. If the LED remains red for over 10 seconds, the program has probably halted.

During module configuration, the OK LED is red and the BP ACT LED is on. If the APP, BP ACT and OK LEDs blink at a rate of every one-second, this indicates a serious problem with the module. Call ProSoft Technology Technical Support to arrange for repairs.

6.2.1 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, 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 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.

6.2.2 Troubleshooting

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

Processor Errors

Problem description	Steps to take
Processor fault	Verify that the module is 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 MVI69L-MBS. Verify that all modules in the rack are correctly configured.

Module Errors

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

6.3 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.
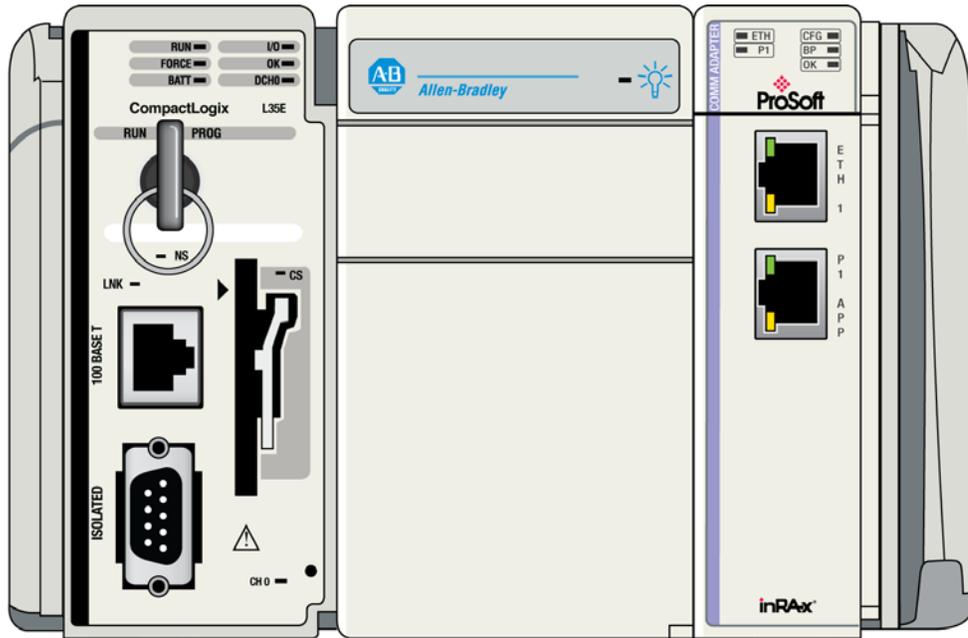
6.3.1 Setting Up a Temporary IP Address

Important: ProSoft Configuration Builder (PCB) locates MVI69L-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 **MVI69L-MBS** module. (For instructions on opening and using a project in PCB, please refer to Configuring the MVI69L-MBS Using PCB (page 43).
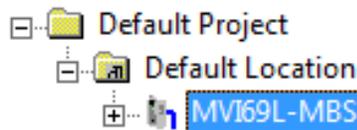
2 Right-click the module icon in the tree and choose **DIAGNOSTICS.**
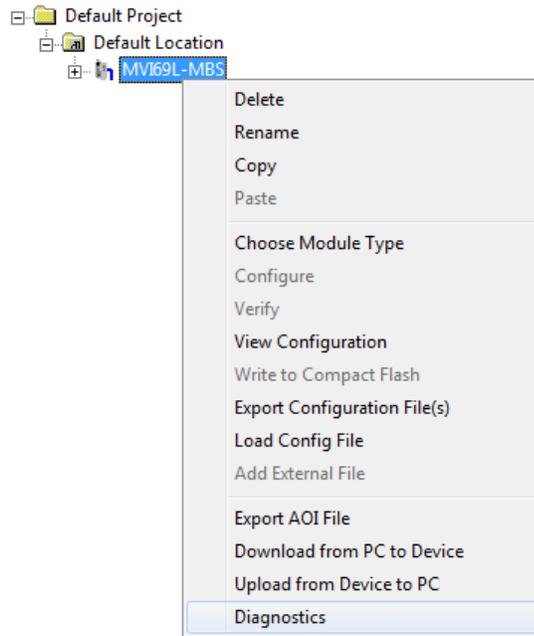
3 In the *Diagnostics* window, click the **SET UP CONNECTION** button.

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**.
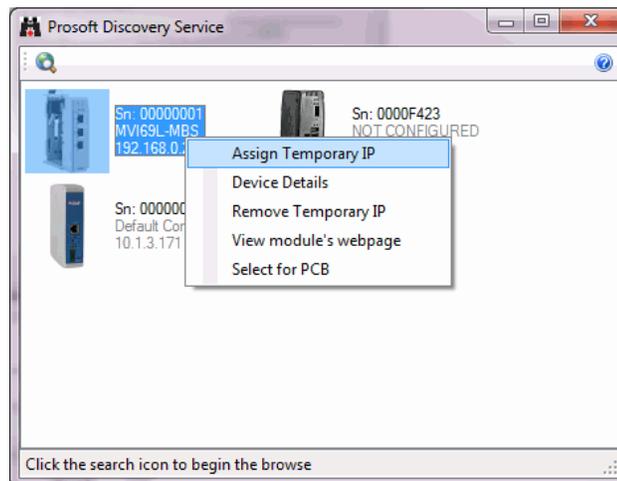
5 The module's default IP address is usually 192.168.0.250. Choose an unused IP within your subnet, and then click **OK**.
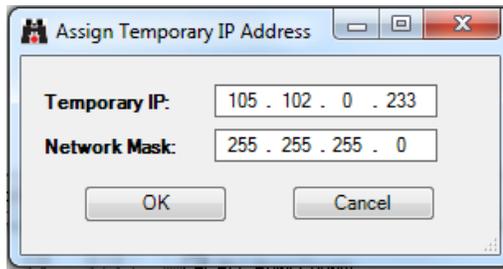
Important: The temporary IP address is only valid until the next time the module is initialized. For information on how to set the module's permanent IP address, see Ethernet 1 (page 55).

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.

6.4 Using the Diagnostics Menu in PCB

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

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

1 In the tree view in ProSoft Configuration Builder, right-click the **MVI69L-MBS** module and then choose **DIAGNOSTICS**. For instructions on opening and using a project in PCB, please refer to Configuring the MVI69L-MBS Using PCB (page 43).
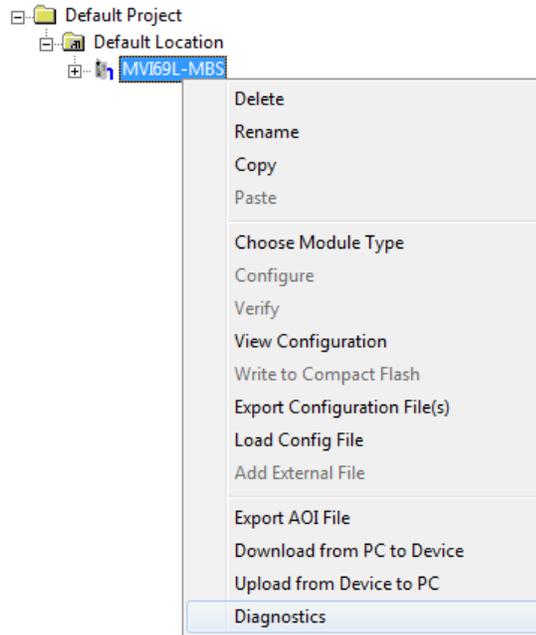
2 After the *Diagnostics* window opens, click the **SET UP CONNECTION** button to browse for the module's IP address.

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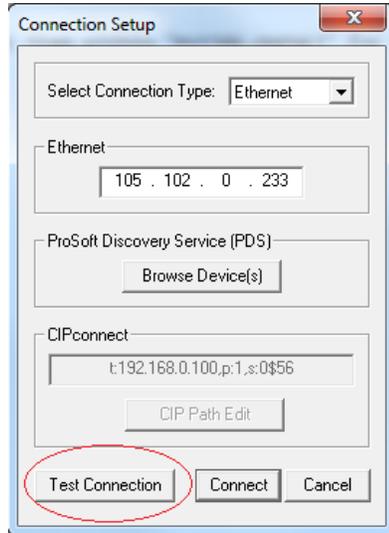
4 If the **TEST CONNECTION** is successful, click **CONNECT**. The *Diagnostics* window is now visible.
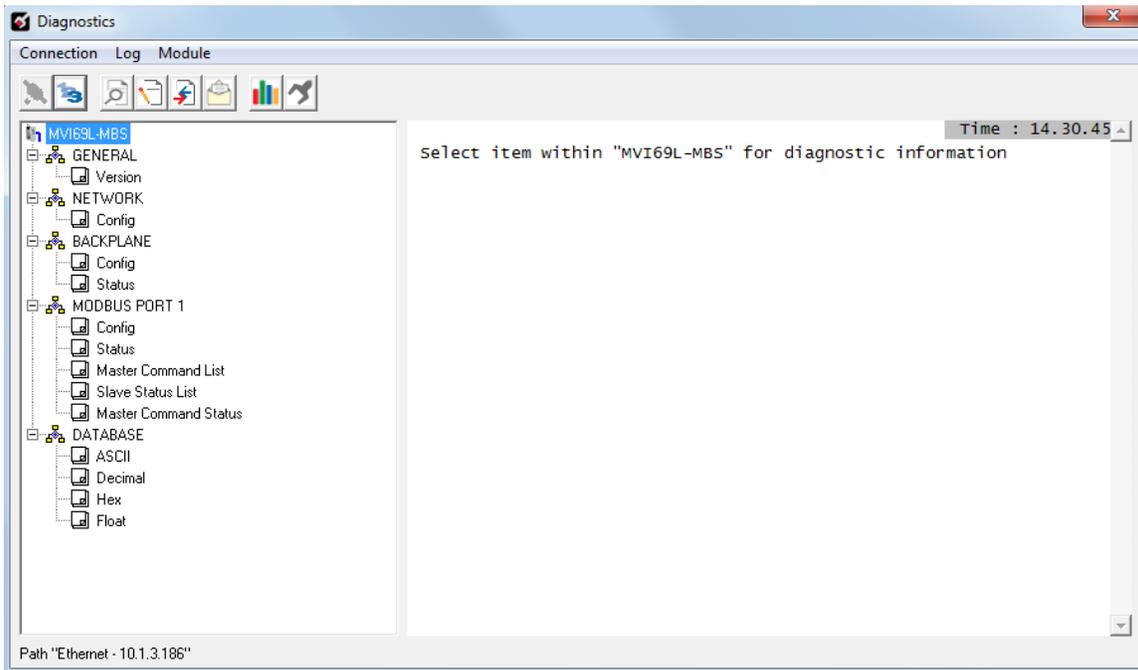
6.4.1 Diagnostics Menu

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.
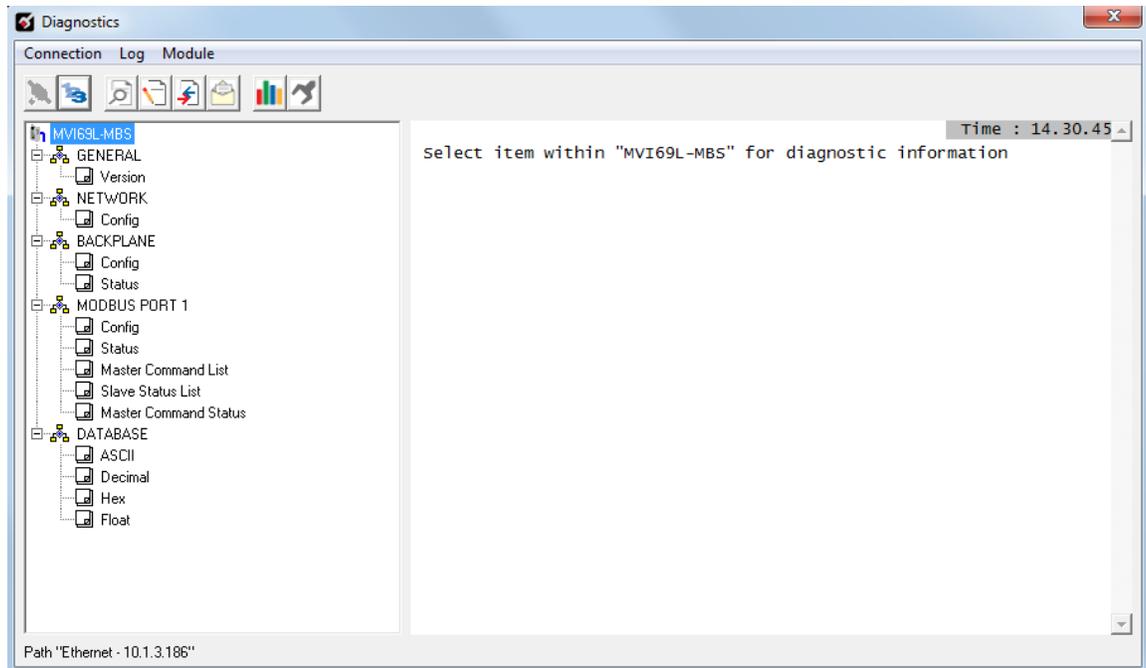
6.4.2 Monitoring General Information

In the *Diagnostics* window in ProSoft Configuration Builder, click **MODULE** and then click **VERSION** to view module version information.
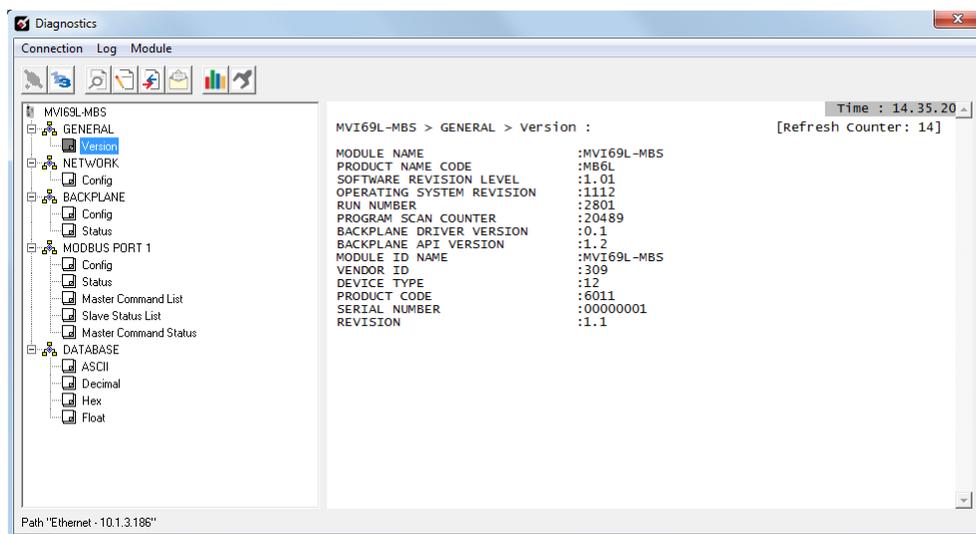
6.4.3 Monitoring Network Configuration Information

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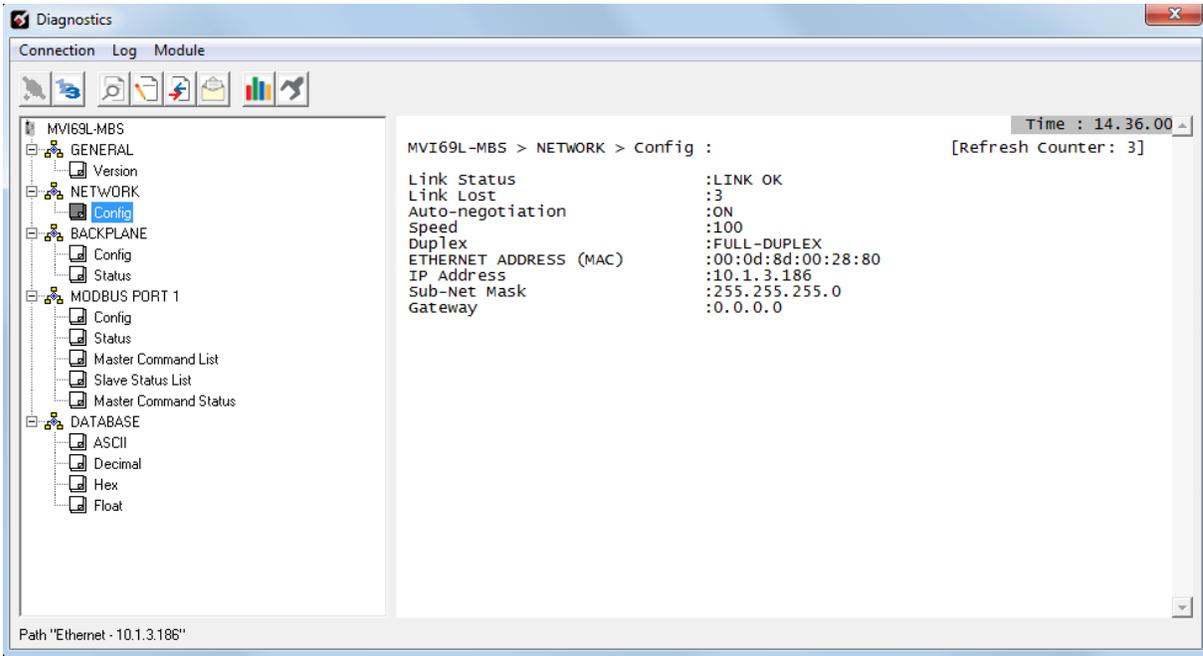
6.4.4 Monitoring Backplane Information

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

- **CONFIGURATION**
- **STATUS**
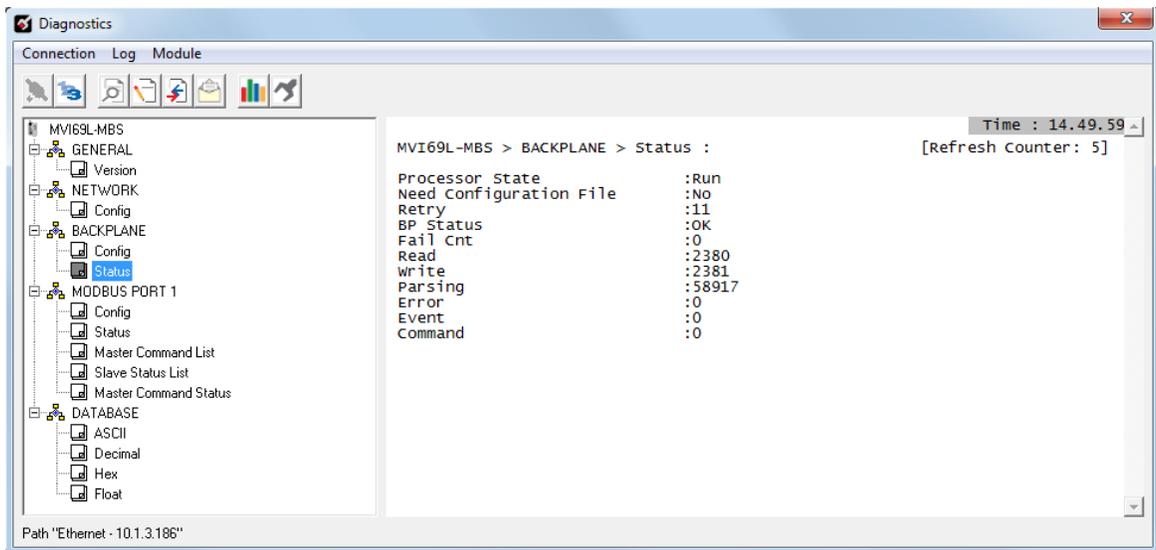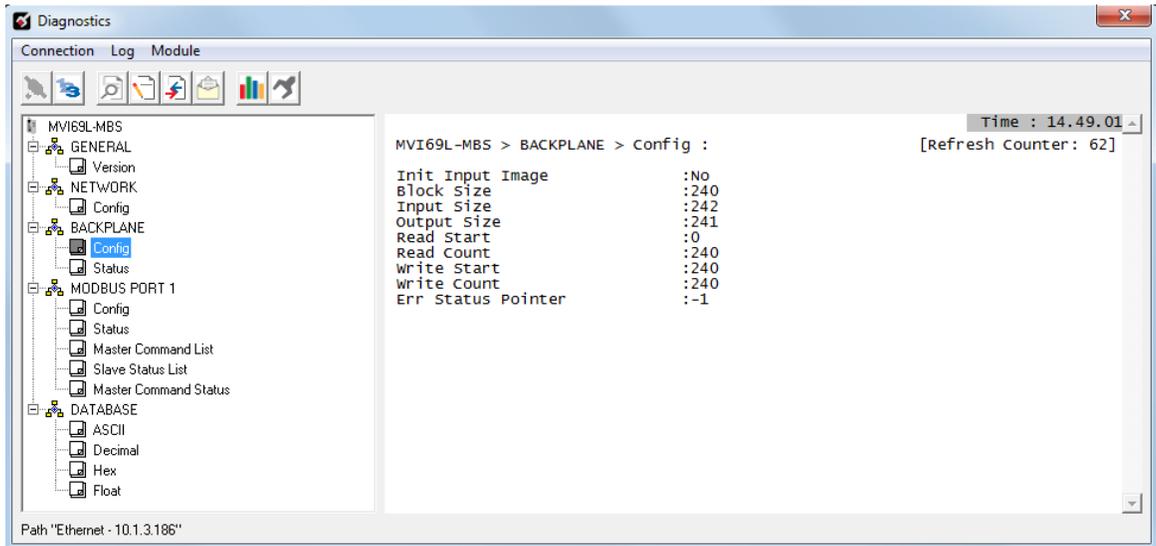
6.4.5 Port 1 Module Information

The **MODBUS PORT 1** menu includes 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)
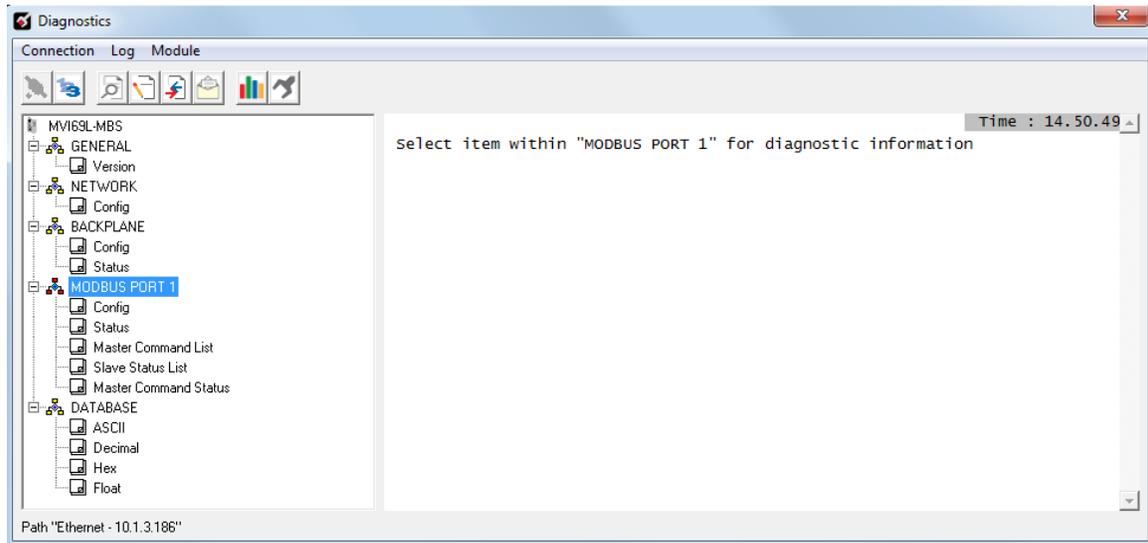
6.4.6 Monitoring Data Values in the Module's Database

In the *Diagnostics* window in ProSoft Configuration Builder, click **DATABASE** and then click **DECIMAL** to view the contents of the MVI69L-MBS module's internal database. You can view data values in ASCII, Hexadecimal, and Float format.
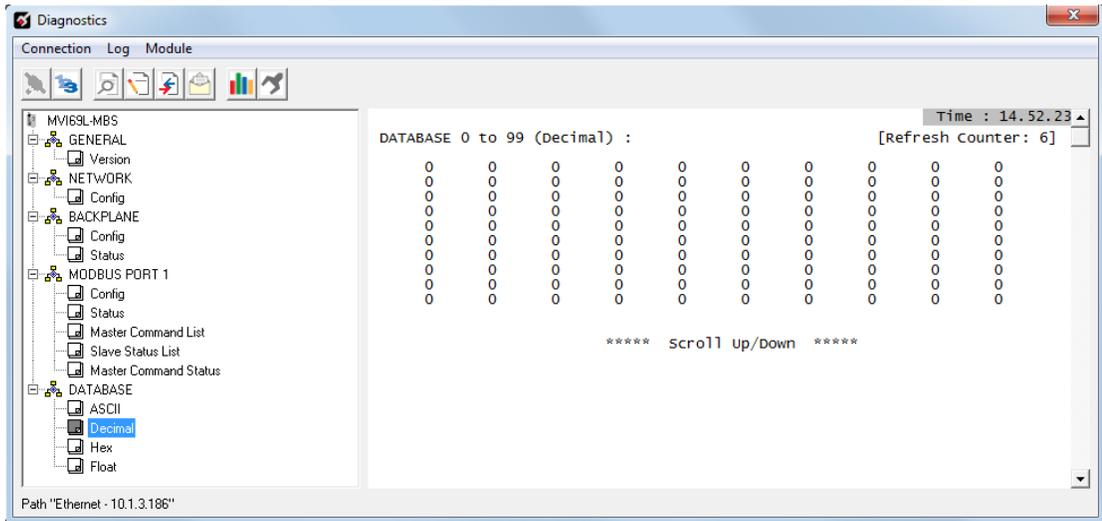
6.5 Communication Error Codes

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.

6.5.1 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

6.5.2 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

6.5.3 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

6.6 Connecting to the Module's Web Page

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.

Modbus Module for CompactLogix
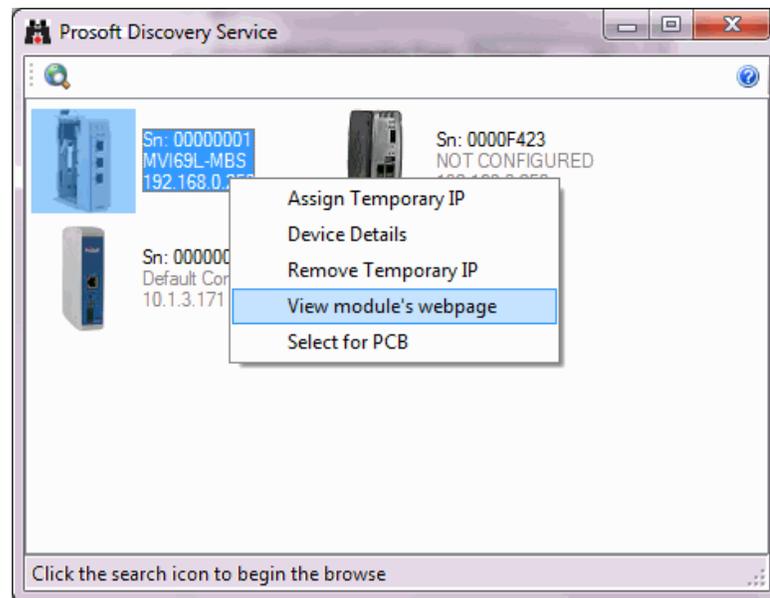
MVI69L-MBS

FUNCTIONS

► **Firmware Upgrade**
► **Set Date & Time**
► **Reboot Module**

► **Technical Support**
► **Homepage**

RESOURCES

ProSoft Technology

Modbus Organization

Module Name	MVI69L-MBS
Ethernet Address (MAC)	00:0D:8D:00:28:80
IP Address	10.1.3.186
Product Revision	1.01.006 2.6.33.7 #7
Firmware Version Date	11/28/12 - 01
Serial Number	00000001
Status	Running
Uptime	6 days 00:02:35

7   Reference

In This Chapter

7.1   Product Specifications

The MVI69L-MBS allows Rockwell Automation® CompactLogix® 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 backplane. The data transfer from the CompactLogix 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.

7.1.1   MVI69L General Specs

- 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 processors with 1769 I/O bus capability and at least 800 mA of 5 Vdc backplane current available.

7.1.2 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 2 slots of 1769 bus)
Operating Temp.	32° F to 140° F (0° C to 60°C)
Storage Temp.	-40° F to 185° F (-40° C to 85° C)
Relative Humidity	5% to 95% (with no condensation)
LED Indicators	Module OK Status
	Backplane Activity
	Ethernet Port Activity
	Configuration Activity
CFG Port (ETH)	Diagnostics over Ethernet connection
App Port (P1)	RS-232, RS-485 or RS-422 (jumper selectable)
	RJ45 Port (DB-9F with supplied cable)
	500V Optical isolation from backplane
Shipped with Unit	DVD-001 ProSoft Solutions DVD
	RJ45 to DB-9M cable for application port

7.1.3 General Specifications - Modbus Master/Slave

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

7.2   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 MVI69L-MBS module can work as a Master and as a Slave.

The MVI69L-MBS module also works as an input/output module between itself and the Rockwell Automation backplane and CompactLogix processor. The module uses an internal database to pass data and commands between the processor and Master and Slave devices on Modbus networks.

7.2.1 Modbus Master

A port configured as a virtual Modbus Master actively issues Modbus commands to other nodes on the Modbus network, supporting up to 30 commands on the port. The Master port has an optimized polling characteristic that polls slaves with communication problems less frequently.

Command List	Up to 30 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.

7.2.2 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 processor.

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

7.2.3 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 MVI69L-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.

7.2.4 Read Coil Status (Function Code 01)

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

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	OE	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.

7.2.5 Read Input Status (Function Code 02)

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 form zero; (input 10001 = zero, input 10002 = one, input 10003 = two, and so on, for a 584).

The following table is a sample read input status request to read inputs 10197 to 10218 (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

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 102180) instead of 8 inputs. The two left-most bits are provided as zeros to fill the 8-bit format.

7.2.6 Read Holding Registers (Function Code 03)

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 form 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

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

7.2.7 Read Input Registers (Function Code 04)

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

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

7.2.8 Force Single Coil (Function Code 05)

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

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".

7.2.9 Preset Single Register (Function Code 06)

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

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

7.2.10 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.

Sub-function Codes Supported

Only Sub-function 00 is supported by the MVI69L-MBS module.

00 Return Query Data

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

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

Example and State Diagram

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

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

The data fields in responses to other kinds of queries could contain error counts or other data requested by the sub-function code.
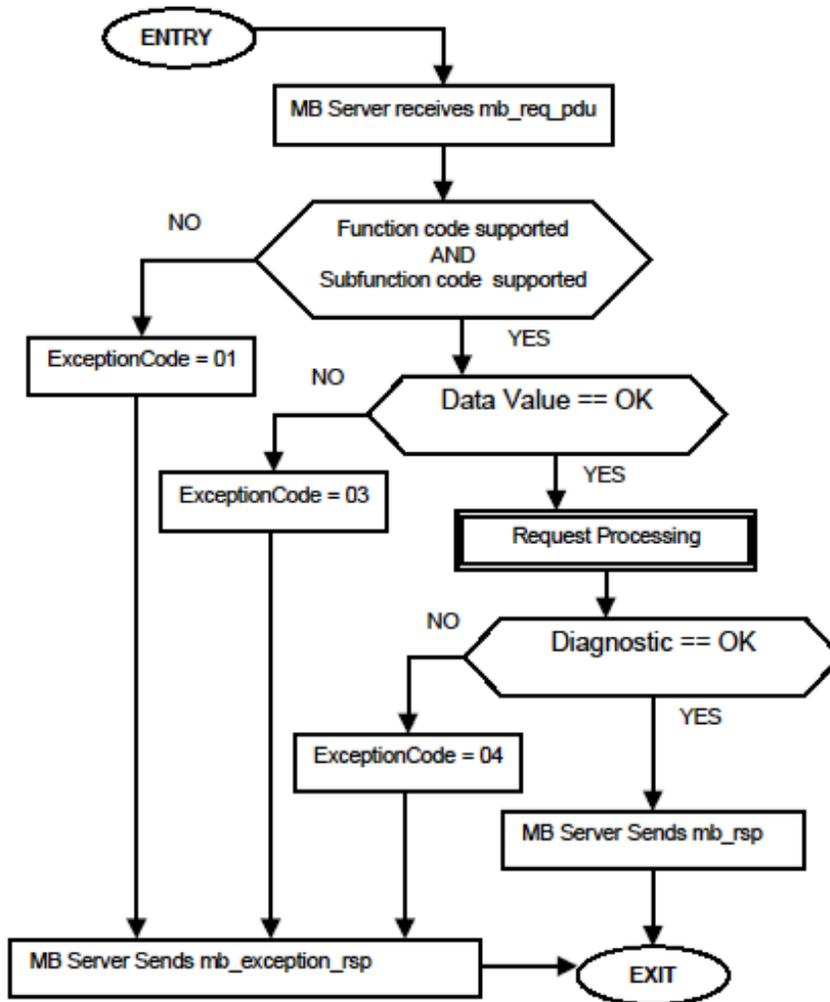
7.2.11 Force Multiple Coils (Function Code 15)

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

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.

7.2.12 Preset Multiple Registers (Function Code 16)

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

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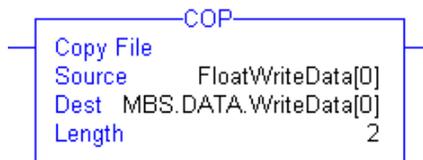
Node Address	Func Code	Data Start Address High	Data Start Address Low	Number of Points High	Number of Points Low	Error Check Field (2 bytes)
0B	10	00	01	00	02	CRC

7.3   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 RSLogix 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.

```
                  ─COP─
    ─┤ Copy File                          ├─
       Source        FloatWriteData[0]
       Dest   MBS.DATA.WriteData[0]
       Length                         2
```

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.

```
                  ─COP─
    ─┤ Copy File                          ├─
       Source  MBS.DATA.ReadData[0]
       Dest            FloatReadData[0]
       Length                         1
```
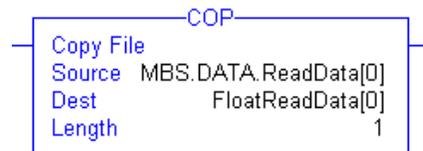
This instruction moves two 16-bit integer registers containing one floating point value image into the floating-point tag. For multiple values increase the *Length* field.

7.3.1 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.

7.3.2 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:

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:

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.

The next few pages show three specific examples:

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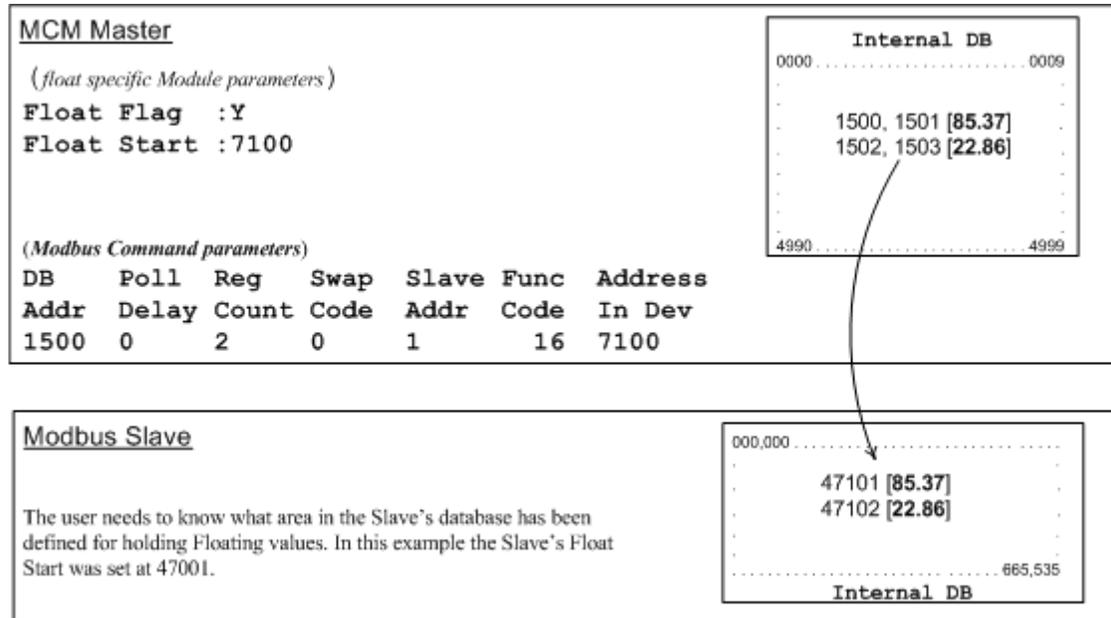
(Float specific module parameters)

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 <= 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

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

Example #2: Master is issuing Modbus command with FC 16 (with Float Flag: No) to transfer Float data.
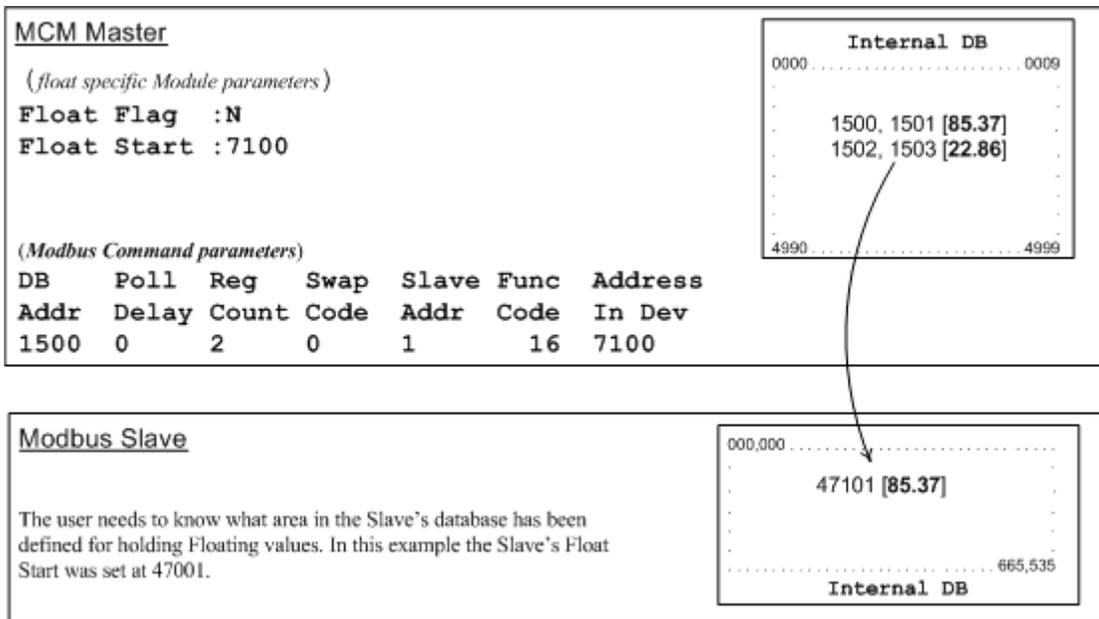
Float Flag: "N" tells the master to ignore the floating values and treat each register data as a data point composed of 1 word, 2 bytes or 16 bits.

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

In this example, the master's Modbus packet contains the data byte and data word counts that have NOT been doubled from the amount specified by Reg Count due to the Float Flag set to N. The slave looks for the byte count in the data packet to know the length of the data to read from the wire. Because of insufficient byte count, some slaves 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.

Example #3: master is issuing Modbus command with FC 3 to transfer Float data from slave.
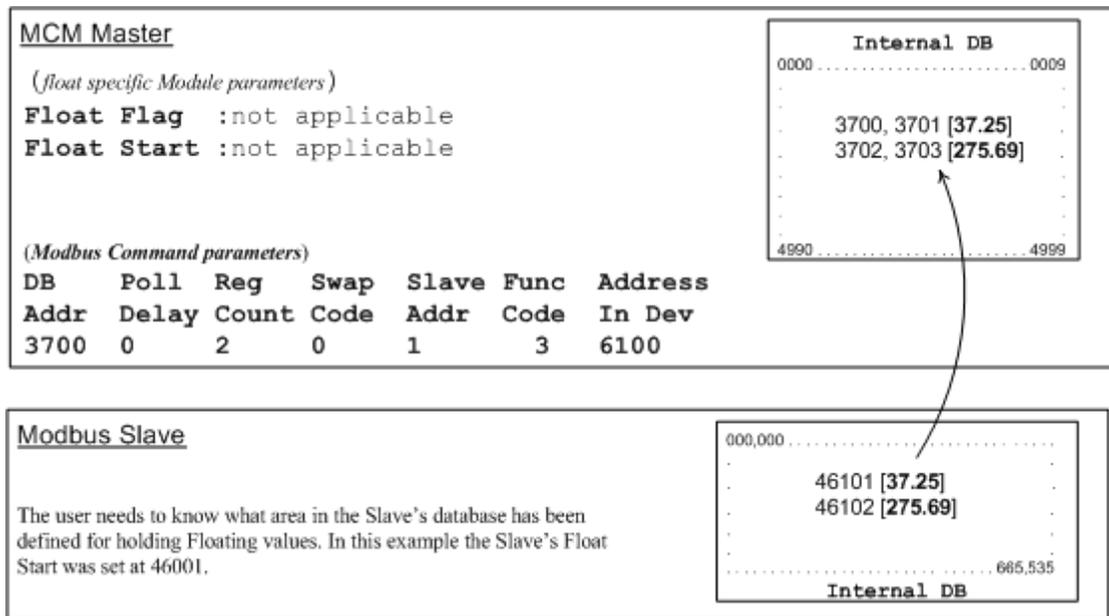
Float Flag: Not applicable with Modbus Function Code 3.

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

In the above example 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	

In the above example 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

7.4 Function Blocks

Data contained in this database is paged through the input and output images by coordination of the CompactLogix ladder logic and the MVI69L-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
3000 to 3001	Port 1 slave polling control
3002 to 3006	Port 1 slave status
5001 to 5006	Port 1 Command Control
8000	Add Event with data for Port 1
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
9960	Pass-through formatted block for function 22
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

7.4.1 Event Command Blocks (1000 to 1255)

Blocks 1000 to 1255: Event Port 1

Event Command blocks send Modbus commands directly from the ladder logic the Master port. 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.

Blocks 1000 to 1255: Request from Processor to Module

Offset	Description
0	Write Block ID: 1000 to 1255 for a Port 1 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 239	Spare

Blocks 1000 to 1255: Response from Module to Processor

Offset	Description
0	Read Block ID: 1000 to 1255 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 239	Spare

7.4.2   Slave Polling Disable Block (3000)

Block 3000: Port 1 Slave Polling Disable

This block allows the processor to disable polling for specific slaves.

Block 3000: Request from Processor to Module

Offset	Description
0	Write Block ID: 3000 for Port 1 slave polling disable request.
1	Number of slaves listed in the block (1 to 60).
2-61	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.

Block 3000: Response from Module to Processor

Offset	Description
0	Read Block ID: 3000 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 1of the request block.
3 to 239	Spare

7.4.3   Slave Polling Enable Blocks (3001)

Block 3001: Port 1 Slave Polling Enable

This block allows the processor to enable polling for specific slaves.

Block 3001: Request from Processor to Module

Offset	Description
0	Write Block ID: 3001 for Port 1 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.

Block 3001: Response from Module to Processor

Offset	Description
0	Read Block ID: 3000 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 1of the request block.
3 to 299	Spare

7.4.4 Slave Polling Status Block (3002 to 3006)

Blocks 3002 to 3006: Port 1 Slave Status

Two arrays are allocated in the module's primary object to hold the polling status of each slave on the Master port. 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.

Block 3002 to 3006: Request from Processor to Module

Offset	Description
0	Write Block ID: 3002 to 3006 for Port 1 slave polling status request.
1 to 239	Spare

Block 3002 to 3006: Response from Module to Processor

Offset	Description
0	Read Block ID: 3002 to 3006 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 239	Spare

Slave Status values

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

7.4.5 Command Control Blocks (5001 to 5006)

Blocks 5001 to 5006: Port 1 Command Control

If the CompactLogix 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 processor.

Blocks 5001 to 5006: Request from Processor to Module

Offset	Description
0	Write Block ID: 5001 to 5006 for Port 1. 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).
2	Index for the second command (applies to blocks 5002 to 5006).
3	Index for the third command (applies to blocks 5003 to 5006).
4	Index for the fourth command (applies to blocks 5004 to 5006).
5	Index for the fifth command (applies to blocks 5005 to 5006).
6	Index for the sixth command (applies to blocks 5006).
7 to 239	Spare

Blocks 5001 to 5006: Response from Module to Processor

Offset	Description
0	Read Block ID: 5001 to 5006 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 239	Spare

7.4.6 Add Event with Data Block (8000)

Block 8000: Add Event with Data for Port 1

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

Block 8000: Request from Processor to Module

Offset	Description
0	Write Block ID: 8000 for Port 1 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 800 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 239	Spare

Block 8000: Response from Module to Processor

Offset	Description
0	Read Block ID: 8000 for Port 1 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 239	Spare

7.4.7 Get Event with Data Status Block (8100)

Block 8100: Get Event with Data Status

This block requests status data for Event with Data Commands.

Block 8100: Request from Processor to Module

Offset	Description
0	Write Block ID: 8100 status data request for Event with Data Commands.
1 to 239	Spare

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
6 to 239	Spare

7.4.8 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.

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.

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.

7.4.9 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.

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.

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 239	Contents of configuration file. If the size of the configuration file exceeds the block transfer size, this information is transferred in multiple blocks.

7.4.10 Get General Module Status Data Block (9250)

Block 9250: Get General Module Status Data

This block requests the general module status.

Block 9250: Request from Processor to Module

Offset	Description
0	Write Block ID: 9250 request for general module status
1 to 239	Spare

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 each time a complete program cycle occurs in the module.
3-4	Product Code: These two registers contain the product code of "MB6E" for the MVI69L-MBS module.
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 to 24	Spare
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.

Offset	Description
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.
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 to 239	Spare

7.4.11 Set Port and Command Active Bits Block (9500)

Block 9500: Set Port and command active bits

This block enables and disables Port 1, as well as individual Master commands for a port.

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	Spare
23 to 42	Command enable bits for Port 2 commands (0=disabled, 1=enabled)
43 to 239	Spare

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 239	Spare

7.4.12 Get Port and Command Active Bits Block (9501)

Block 9501: Get Port and command active bits

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

Block 9501: Request from Processor to Module

Offset	Description
0	Write Block ID: 9501 to get port and command enable/disable state
1 to 239	Spare

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	Spare
24 to 43	Command enable bits for Port 2 commands (0=disabled, 1=enabled)
44 to 239	Spare

7.4.13 Pass-through Formatted Block for Functions 6 and 16 with Word Data Block

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

If the slave port on the module is 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 as expected to commands issued by the remote Modbus Master device.

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 239	Data

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.

Block 9956: Response from Processor to Module

Offset	Description
0	Write Block ID: 9956
1 to 239	Spare

7.4.14 Pass-through Formatted Block for Functions 6 and 16 with Float Data Block (9957)

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

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 *239*	Data

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.

Block 9957: Response from Processor to Module

Offset	Description
0	Write Block ID: 9957
1 to 239	Spare

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

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

Block 9958: Pass-Through Formatted Block for Function 5

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 239	Data

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.

Block 9958: Response from Processor to Module

Offset	Description
0	Write Block ID: 9958
1 to 239	Spare

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

7.4.16 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 RSLogix 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.

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 239	Spare

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.

Block 9959: Response from Processor to Module

Offset	Description
0	Write Block ID: 9959
1 to 239	Spare

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

7.4.17 Pass-through Formatted Block for Function 22 (9960)

Block 9960: Pass-Through Formatted Block for Function 22

Block 9960: Request from Module to Processor

Offset	Description
0	Read Block ID: 9960
1	Write Block ID: 9960
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to 239	Data

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.

Block 9960: Response from Processor to Module

Offset	Description
0	Write Block ID: 9960
1 to 239	Spare

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

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

Block 9961: Pass-Through Formatted Block for Function 23

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 239	Data

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.

Block 9961: Response from Processor to Module

Offset	Description
0	Write Block ID: 9961
1 to 239	Spare

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

7.4.19 Pass-through Block for Function 99 (9970)

Block 9970: Pass-Through Block for Function 99

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 239	Spare

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.

Block 9970: Response from Processor to Module

Offset	Description
0	Write Block ID: 9970
1 to 239	Spare

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

7.4.20 Set Module Time Using Received Time Block (9972)

Block 9972: Set Module Time Using Received Time Block

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

Block 9972: Request from Processor to Module

Offset	Description
0	Write Block ID: 9972
1	Year (0-9999)
2	Month (1-12)
3	Day (1-31)
4	Hour (0-23)
5	Minutes (0-59)
6	Seconds (0-59)
7	Milliseconds (0-999)
8 to 239	Spare

Block 9972: Response from Module to Processor

Offset	Description
0	Read Block ID: 9972
1	Write Block ID: To be used by the processor in its next Write block.
2	Return code 0=OK, -1=error
3 to 239	Spare

7.4.21 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.

Block 9973: Request from Processor to Module

Offset	Description
0	Write Block ID: 9973
1 to 239	Spare

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 239	Spare

7.4.22 Reset Status Block (9997)

Block 9997: Reset Status Block

This block resets the module and port 1 status.

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)
4 to 239	Spare

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 239	Spare

7.4.23 Warm-boot Control Block (9998)

Block 9998: Warm-boot Control Block

If the CompactLogix sends a block number 9998, the module performs a warm-boot operation. The module reconfigures the application port and reset the error and status counters.

Block 9998: Request from Processor to Module

Offset	Description
0	Write Block ID: 9998
1 to 239	Spare

7.4.24 Cold-boot Control Block (9999)

Block 9999: Cold-boot Control Block

If the CompactLogix 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.

Block 9999: Request from Processor to Module

Offset	Description
0	Write Block ID: 9999
1 to 239	Spare

7.5 Ethernet Port Connection

7.5.1 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.

Refer to Ethernet Cable Configuration (page 134) for a diagram of how to configure Ethernet cable.

Ethernet Cable Configuration

Note: The standard connector view shown is color-coded for a straight-through cable.
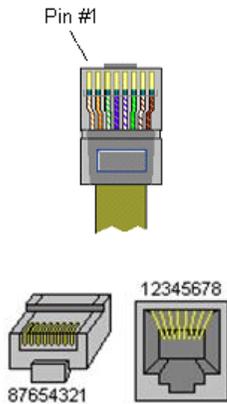
Crossover cable			Straight- through cable	
RJ-45 PIN	**RJ-45 PIN**	Pin #1	**RJ-45 PIN**	**RJ-45 PIN**
1 Rx+	3 Tx+		1 Rx+	1 Tx+
2 Rx-	6 Tx-		2 Rx-	2 Tx-
3 Tx+	1 Rx+		3 Tx+	3 Rx+
6 Tx-	2 Rx-		6 Tx-	6 Rx-

Ethernet Performance

Ethernet performance in the MVI69L-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:
  - o Use managed switches to reduce traffic coming to module port
  - o Use CIPconnect for these applications and disconnect the module Ethernet port from the network

7.6 Modbus Application Port Connection

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

7.6.1 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.
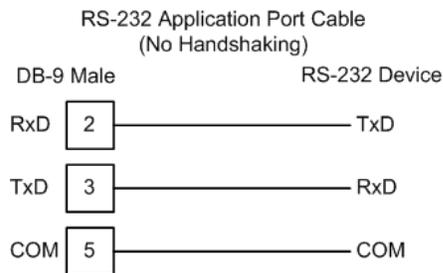
RS-232: Modem Connection (Hardware Handshaking Required)

This type of connection is required between the module and a modem or other communication device.
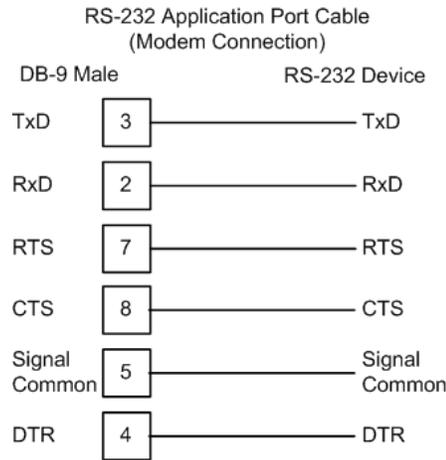
The "Use CTS Line" parameter for the port configuration should be set to 'Y' for most modem applications.

RS-232: Null Modem Connection (Hardware Handshaking)

This type of connection is used when the device connected to the module requires hardware handshaking (control and monitoring of modem signal lines).
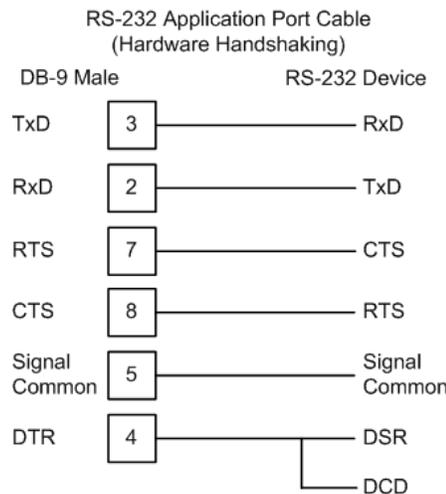
RS-232: Null Modem Connection (No Hardware Handshaking)

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

RS-232 Application Port Cable
(No Handshaking)

DB-9 Male		RS-232 Device
RxD	2	TxD
TxD	3	RxD
COM	5	COM
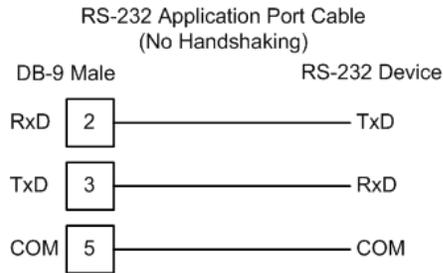
Note: For most null modem connections where hardware handshaking is not required, the *Use CTS Line* parameter should be set to **N** and no jumper is required between Pins 7 (RTS) and 8 (CTS) on the connector. If the port is configured with the *Use CTS Line* set to **Y**, then a jumper is required between the RTS and the CTS lines on the port connection.

RS-232 Application Port Cable
(No Handshaking)

DB-9 Male   RS-232 Device

TxD	3	RxD
RxD	2	TxD
RTS	7	RTS-CTS jumper must be installed if CTS line monitoring enabled.
CTS	8	
Signal Common	5	Signal Common
DTR	4	
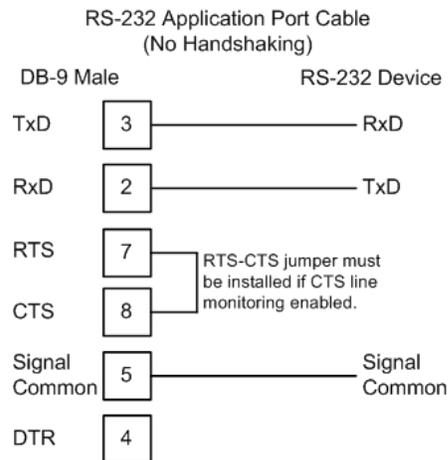
7.6.2 RS-422 Wiring

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. The cable required for this interface is shown below:

RS-422 Application Port Cable
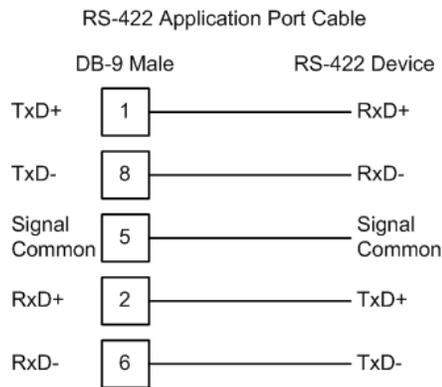
7.6.3 RS-485 Wiring

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. The cable required for this interface is shown below:

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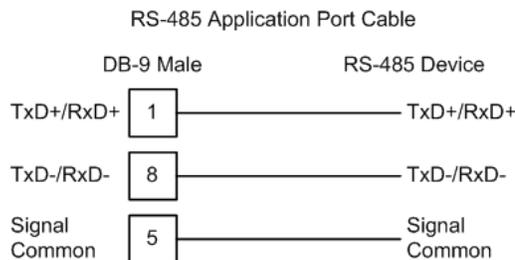
Note: Terminating resistors are generally not required on the RS-485 network, unless you are experiencing communication problems that can be attributed to signal echoes or reflections. In these cases, installing a 120-ohm terminating resistor between pins 1 and 8 on the module connector end of the RS-485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

RS-485 and RS-422 Tip

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

7.6.4   DB9 to RJ45 Adaptor (Cable 14)
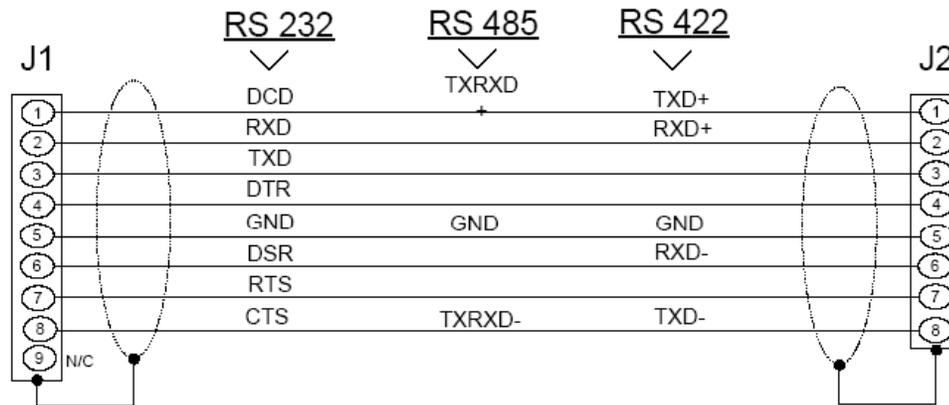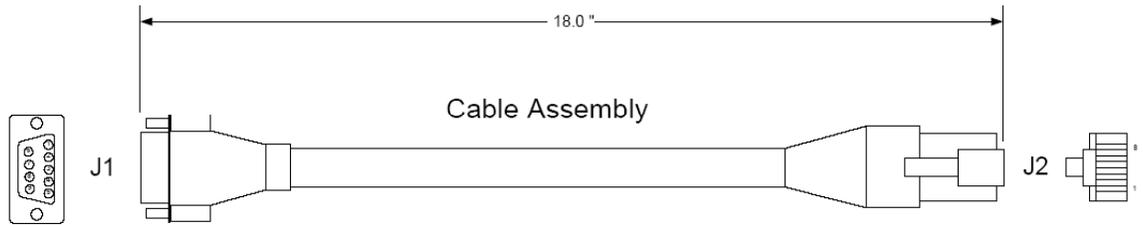
Wiring Diagram
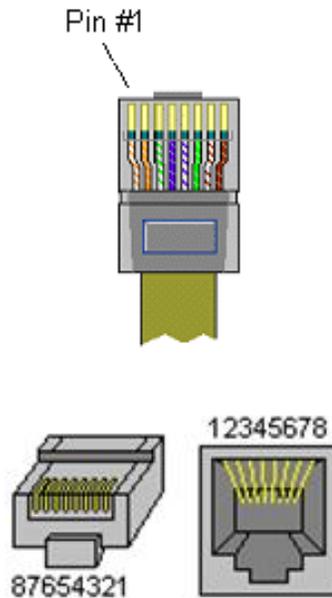
8   Support, Service & Warranty

8.1   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 serial, Ethernet or Fieldbus devices interfaced to the module, if any.

Note: *For technical support calls within the United States, ProSoft's 24/7 after-hours phone support is available for urgent plant-down issues. Detailed contact information for all our worldwide locations is available on the following page.*

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

8.2   Warranty Information

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

Documentation is subject to change without notice.

Index