ProSoft TECHNOLOGY

Where Automation Connects.

MVI69E-MBTCP

Modbus TCP/IP Enhanced Communication Module

September 29, 2025

USER MANUAL

Your Feedback Please

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

ProSoft Technology, Inc.
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com
ps.support@belden.com

MVI69E-MBTCP User Manual
For Public Use.

September 29, 2025

Content Disclaimer

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

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

For professional users in the European Union
If you wish to discard electrical and electronic equipment (EEE), please contact your dealer or supplier for further information.

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

Agency Approvals and Certifications

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

Open-Source Information

Open-Source Software used in the product

The product contains, among other things, Open-Source Software files, as defined below, developed by third parties and licensed under an Open-Source Software license. These Open-Source Software files are protected by copyright. Your right to use the Open-Source Software is governed by the relevant applicable Open-Source Software license conditions. Your compliance with those license conditions will entitle you to use the Open-Source Software as foreseen in the relevant license. In the event of conflicts between other ProSoft Technology, Inc. license conditions applicable to the product and the Open-Source Software license conditions, the Open-Source Software conditions shall prevail. The Open-Source Software is provided royalty-free (i.e. no fees are charged for exercising the licensed rights). Open-Source Software contained in this product and the respective Open-Source Software licenses are stated in the module webpage, in the link Open-Source.
If Open-Source Software contained in this product is licensed under GNU General Public License (GPL), GNU Lesser General Public License (LGPL), Mozilla Public License (MPL) or any other Open-Source Software license, which requires that source code is to be made available and such source code is not already delivered together with the product, you can order the corresponding source code of the Open-Source Software from ProSoft Technology, Inc. - against payment of the shipping and handling charges - for a period of at least 3 years since purchase of the product. Please send your specific request, within 3 years of the purchase date of this product, together with the name and serial number of the product found on the product label to:

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

Warranty regarding further use of the Open-Source Software

ProSoft Technology, Inc. provides no warranty for the Open-Source Software contained in this product, if such Open-Source Software is used in any manner other than intended by ProSoft Technology, Inc. The licenses listed define the warranty, if any, from the authors or licensors of the Open-Source Software. ProSoft Technology, Inc. specifically disclaims any warranty for defects caused by altering any Open-Source Software or the product's configuration. Any warranty claims against ProSoft Technology, Inc. in the event that the Open-Source Software contained in this product infringes the intellectual property rights of a third party are excluded. The following disclaimer applies to the GPL and LGPL components in relation to the rights holders:
"This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License and the GNU Lesser General Public License for more details."
For the remaining Open-Source components, the liability exclusions of the rights holders in the respective license texts apply. Technical support, if any, will only be provided for unmodified software.

Table of Contents

1 Start Here

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

- **Studio 5000 Logix Designer®:** launch the program, configure ladder logic, and transfer the ladder logic to the processor
- **Microsoft Windows:** install and launch programs, execute menu commands, navigate dialog boxes, and enter data
- **Hardware installation and wiring:** install the module, and safely connect Modbus and CompactLogix devices to a power source and to the MVI69E-MBTCP module's Ethernet port

1.1 System Requirements

The MVI69E-MBTCP 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 MVI69E-MBTCP module.

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

Important: For 1769-L23x processors, please make note of the following limitation:
1769-L23E-QBFC1B = 450 mA at 5 Vdc (No MVI69E module can be used with this processor.)

- The module requires 500 mA of available 5 Vdc power
- Rockwell Automation Studio 5000 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 10
  - Microsoft Windows 7 Professional (32-or 64-bit)
  - Microsoft Windows XP Professional with Service Pack 1 or 2
- 128 Mbytes of RAM minimum, 256 Mbytes of RAM recommended

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

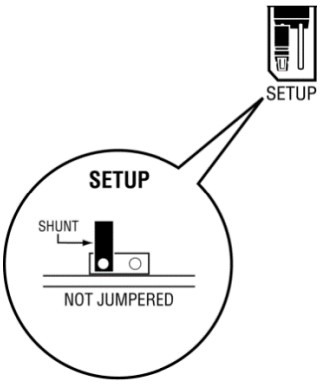
Before you begin to configure the module, consider the following questions. Your answers will help you determine the scope of your project, and the configuration requirements for a successful deployment.

- 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 MVI69E-MBTCP module occupy? For communication to occur, enter the correct slot number in the sample program.
- Are the Studio 5000 and RSLinx software installed?
- RSLogix and RSLinx are required to communicate to the CompactLogix processor.
- How many words of data do you need to transfer in your application (from CompactLogix to Module / to CompactLogix from Module)?
- Is this module replacing an existing legacy MVI69-MNET module (refer to Chapter 6 *Legacy Mode* on page 81)?

1.3 Setting Jumpers

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. Do not jumper the Setup pins together unless you are directed to do so by ProSoft Technical Support (or you want to update the module firmware).

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



Note: When installing the module in a remote rack, it may be preferable to leave the Setup pins jumpered. That way, module firmware updates can be done without physically accesssing the module.

1.4 Installing the Module in the Rack

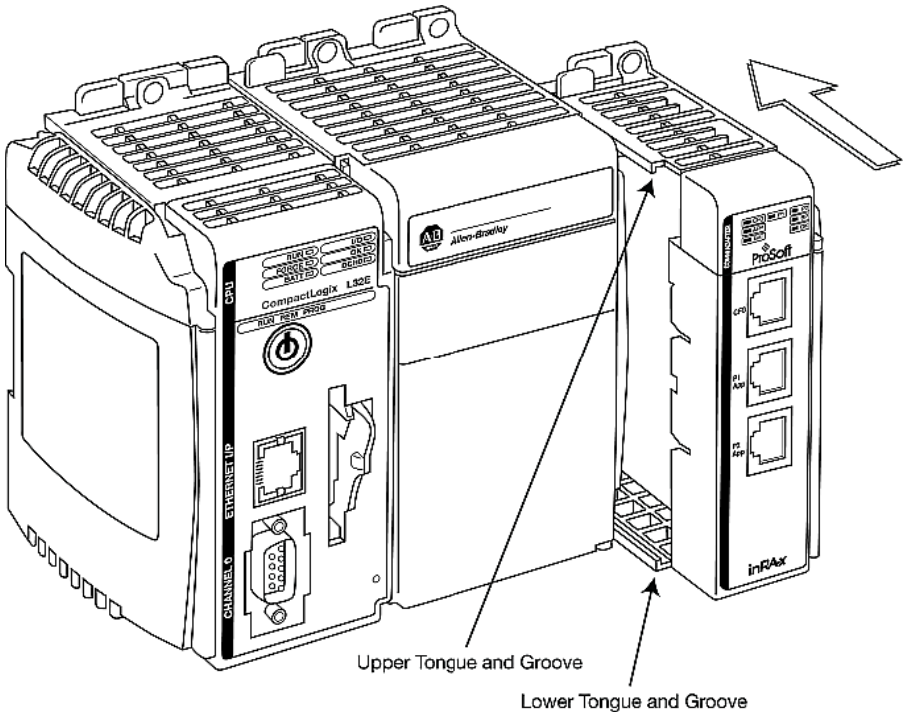
Make sure the processor and power supply are installed and configured before installing the MVI69E-MBTCP module. Refer to the Rockwell Automation product documentation for installation instructions.

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

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

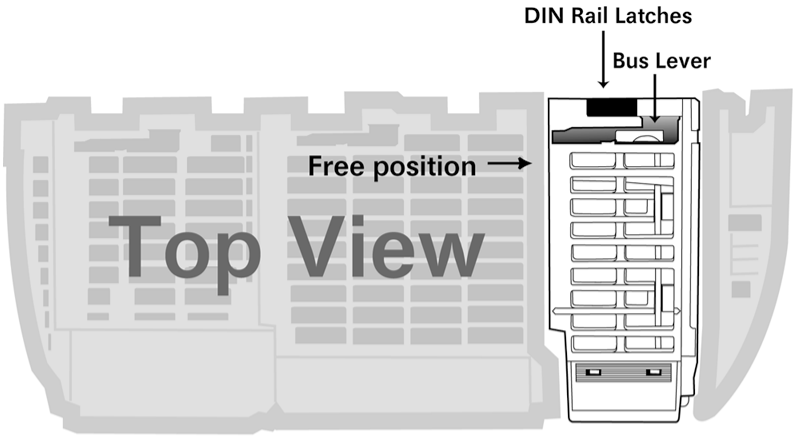
Warning: This module is not hot-swappable! Always remove power from the rack before inserting or removing this module, or damage may result to the module, the processor, or other connected devices.

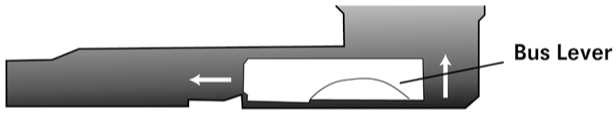
1. Align the module using the upper and lower tongue-and-groove slots with the adjacent module and slide forward in the direction of the arrow.



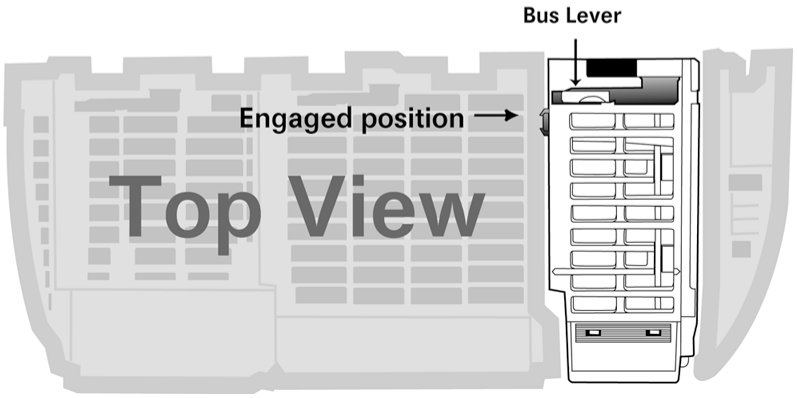
2. Move the module back along the tongue-and-groove slots until the bus connectors on the MVI69 module and the adjacent module line up with each other.

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



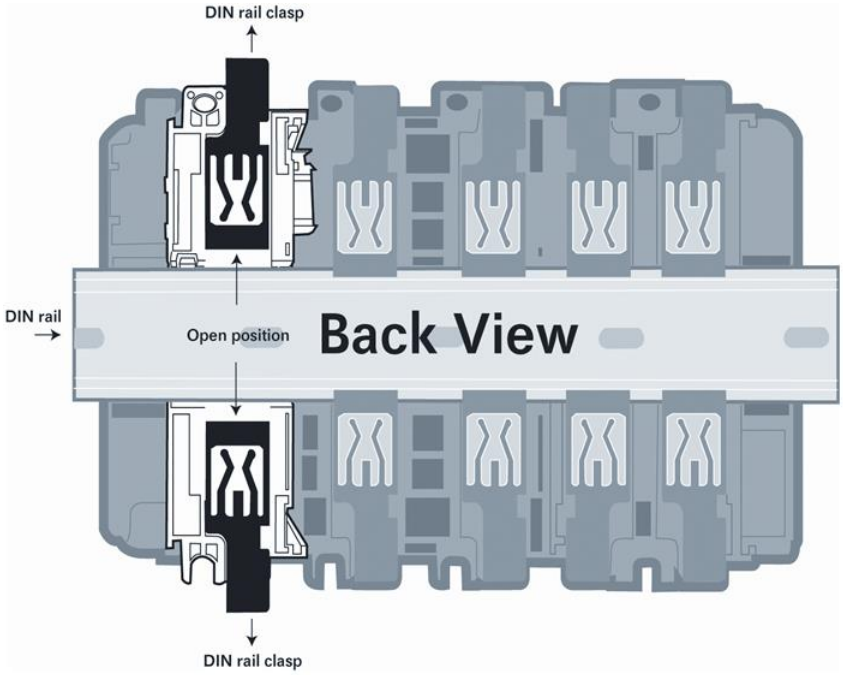


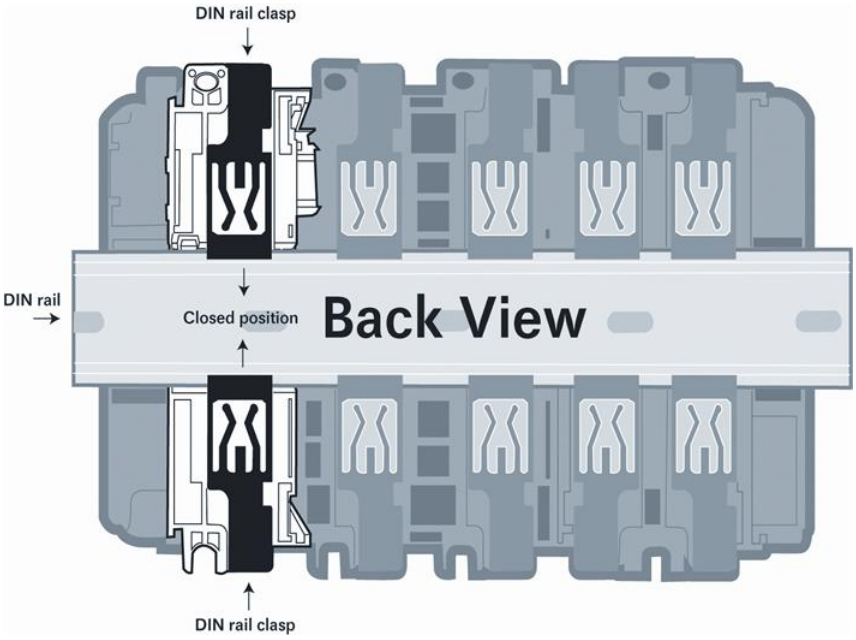
Move the Bus Lever to the left until it clicks



4 Close all DIN-rail latches.

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





1.5 Package Contents

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

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

Qty.	Part Name	Part Number	Part Description
1	MVI69E-MBTCP Module	MVI69E-MBTCP	Modbus TCP/IP Enhanced Communication Module

If any of these components are missing, please contact ProSoft Technology Technical Support for replacement parts. For the latest files, please visit www.prosoft-technology.com.

2 Adding the Module to RSLogix

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

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

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

2.1 Creating the Module in a Studio 5000 Project

In an Studio 5000 project, there are two ways to add the MVI69E-MBTCP module to the project.

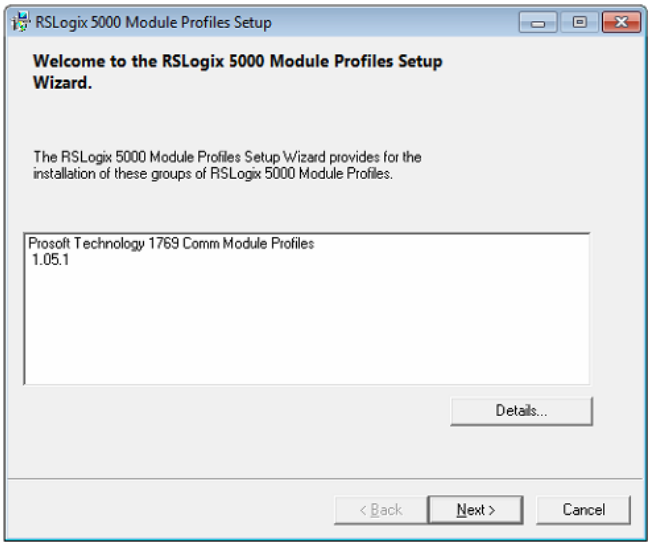
- Use an Add-On Profile (AOP) from ProSoft Technology. The AOP contains all the configuration information needed to add the module to the project. This is the preferred way, but requires RSLogix version 15 or later. Refer to *Creating a Module in the Project Using an Add-On Profile* (page 14).
- If using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

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

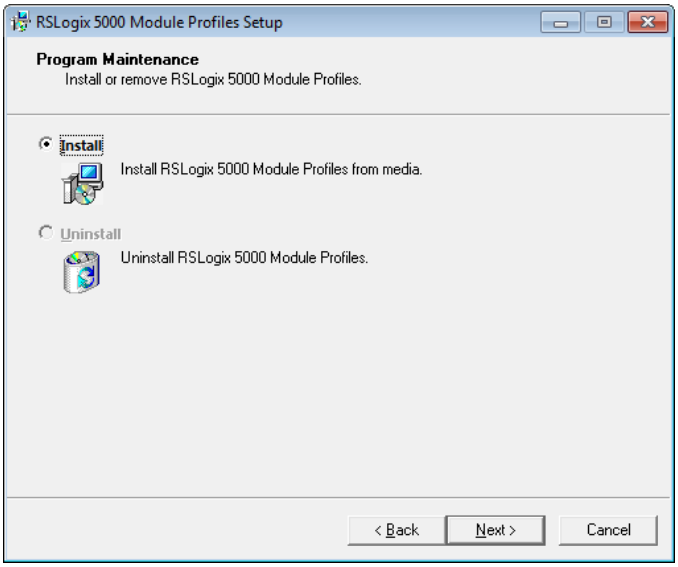
2.1.1.1 Installing an Add-On Profile

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

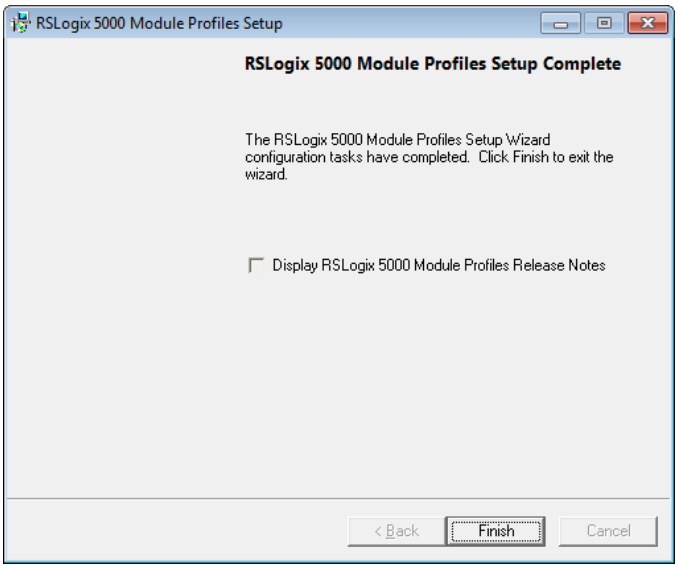
1 Run the **MPSetup.exe** file to start the Setup Wizard. Follow the Setup Wizard to install the AOP.



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

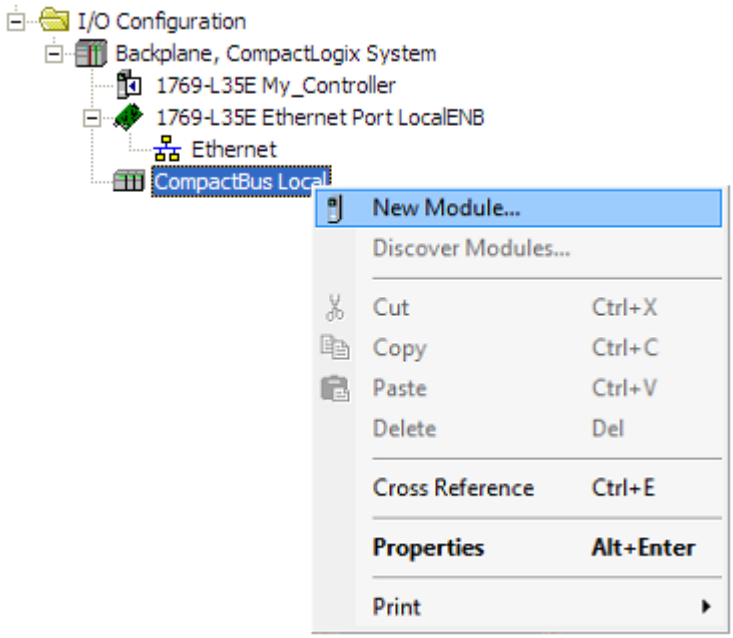


3 Click **FINISH** when complete**.** The AOP is now installed in Studio 5000.



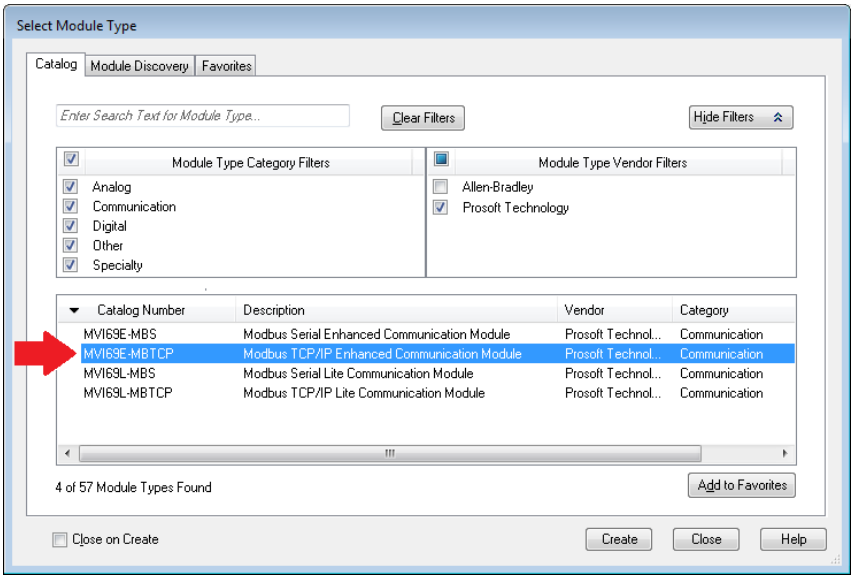
2.1.1.2 Using an Add-On Profile

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



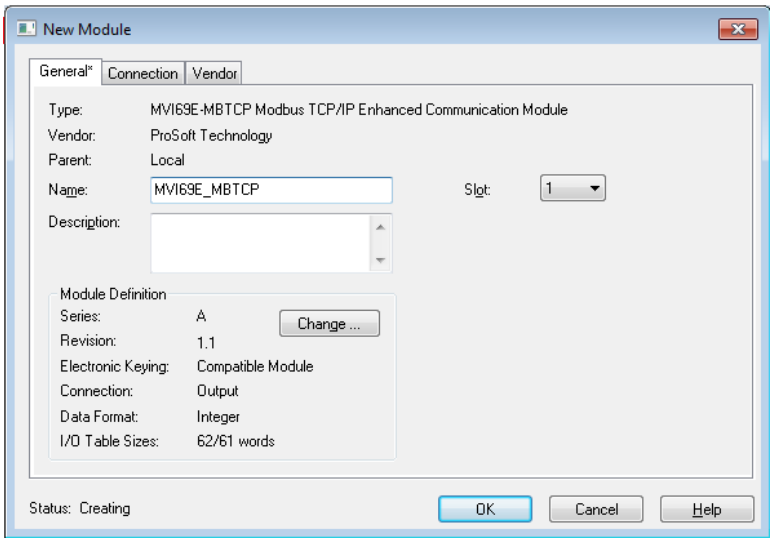
2 This opens the *Select Module Type* dialog box.

3 In the *Module Type Vendor Filters* area, uncheck all boxes except the **PROSOFT TECHNOLOGY** box. A list of ProSoft Technology modules appears.



4 Select the **MVI69E-MBTCP** module in the list and click **CREATE**.

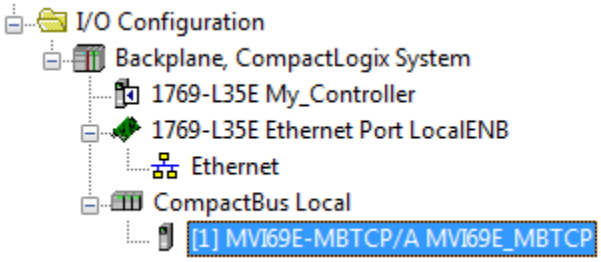
5 In the *New Module* dialog box, edit the **NAME** and **SLOT**. Click **OK**.



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

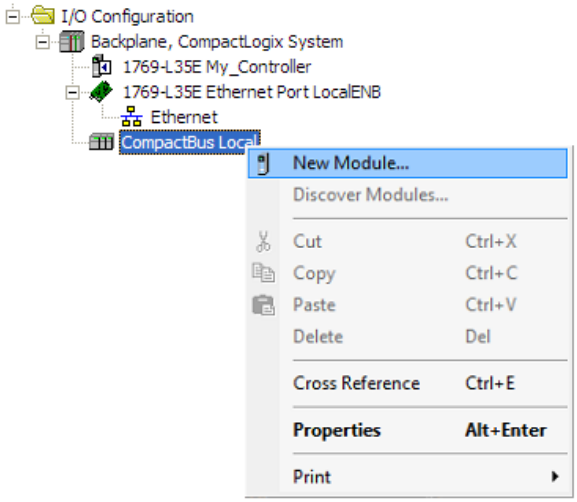
A *Block Transfer Size* of 60 uses an **I/O TABLE SIZE** of 62/61 words.
A *Block Transfer Size* of 120 uses an **I/O TABLE SIZE** of 122/121 words.
A *Block Transfer Size* of 240 uses an **I/O TABLE SIZE** of 242/241 words.

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



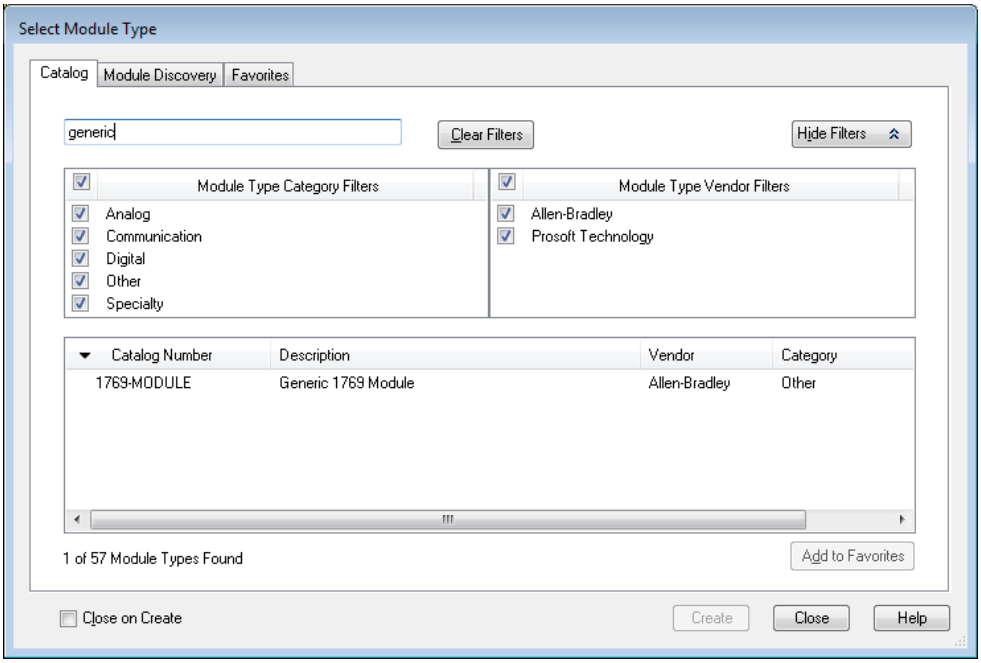
2.1.2 Creating a Module in the Project Using a Generic 1769 Module Profile

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



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

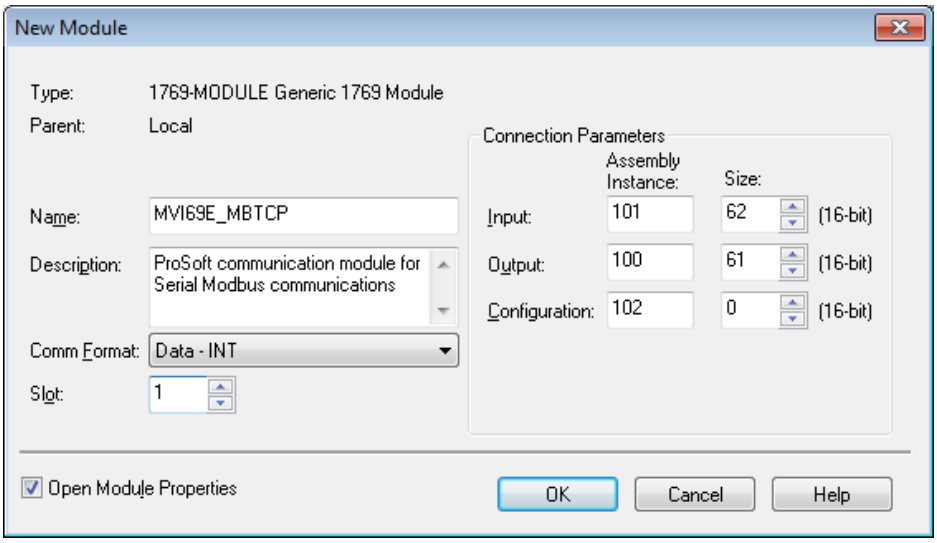
3 Enter **GENERIC** in the search text box and select the **GENERIC 1769 MODULE**. If you're using an earlier version of RSLogix, expand **OTHER** in the *Select Module* dialog box, and then select the **GENERIC 1769 MODULE**.



4 Set the *Module Properties* values as follows:

Parameter	Value
Name	Enter a module identification string. Example: **MVI69E_MBTCP**
Description	Enter a description for the module. Example: ProSoft communication module for Modbus TCP/IP communications.
Comm Format	Select **DATA-INT**
Slot	Enter the slot number in the rack where the MVI69E-MBTCP module is installed.
Input Assembly Instance	101
Input Size	62 / 122 / 242
Output Assembly Instance	100
Output Size	61 / 121 / 241
Configuration Assembly Instance	102
Configuration Size	0

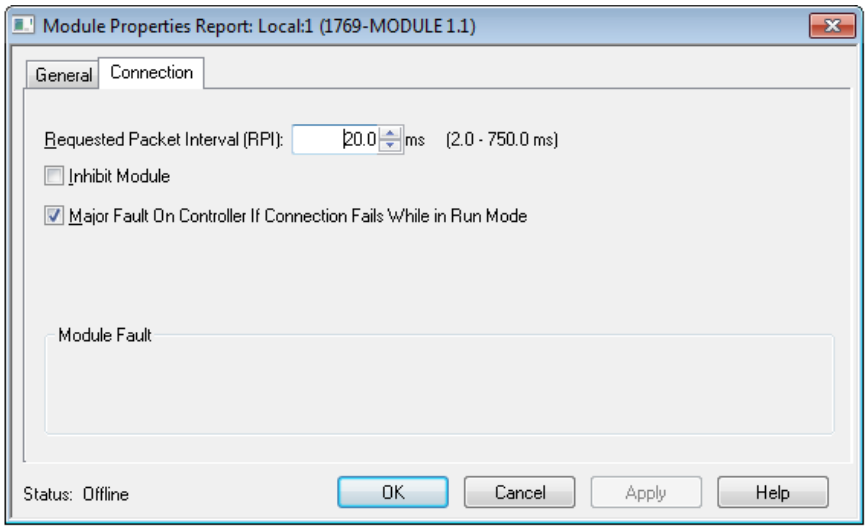
5 The following illustration shows an example where the module was configured for a block transfer size of 60 words (input block size = 62 words, output block size = 61 words):



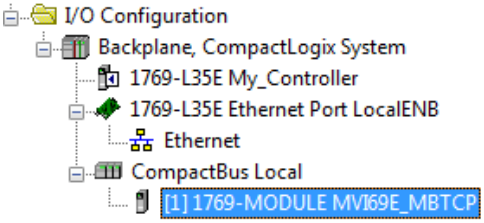
6 The following options are available:

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

7 On the *Connection* tab, set the **REQUESTED PACKET INTERVAL** value for your project and click **OK**. (10 to 30 ms is recommended).



8 The MVI69E-MBTCP 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 website: www.prosoft-technology.com. The installation filename contains the PCB version number. For example, **PCB_4.3.4.5.0238.EXE**.

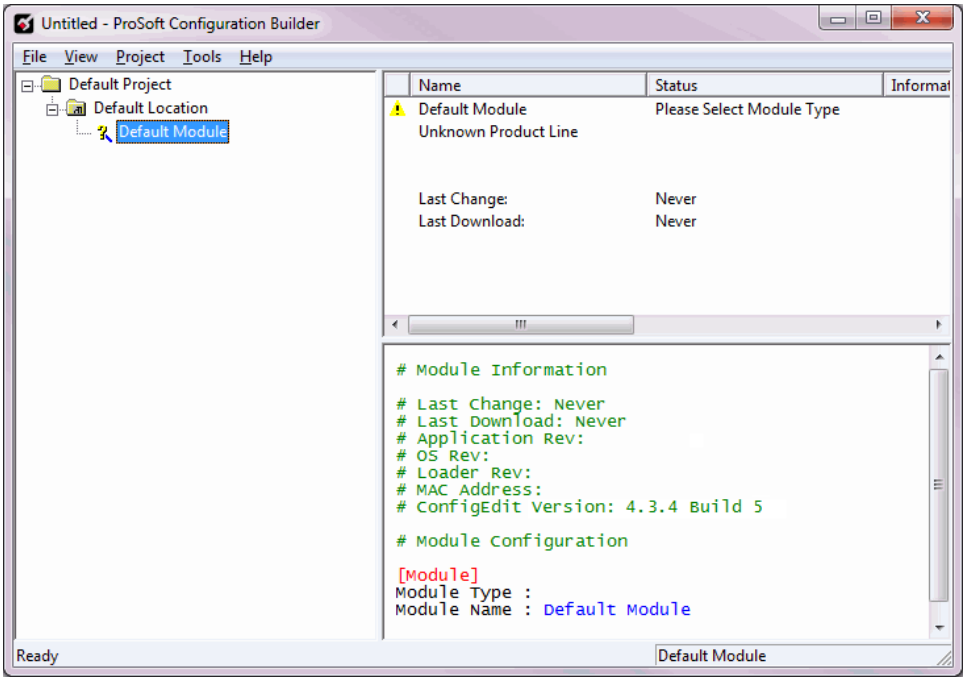
1 Open a browser window and navigate to **www.prosoft-technology.com**.
2 Perform a search for 'pcb' in the Search bar. Click on the ProSoft Configuration Builder search result.
3 On the PCB page, click the download link for ProSoft Configuration Builder, and save the file to your Windows desktop.
4 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.
5 If you want to find additional software specific to your MVI69E-MBTCP, enter the model number into the ProSoft website search box and press the Enter key.

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

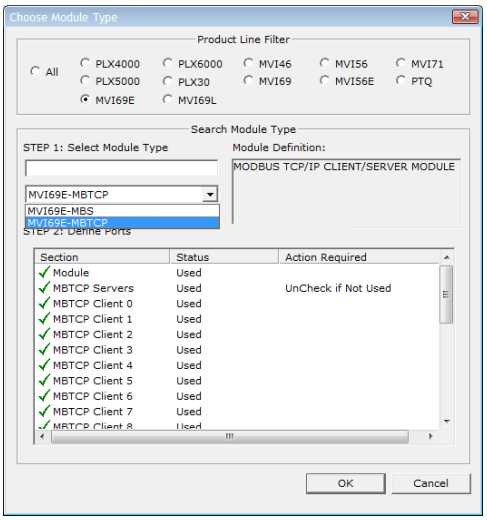
The following sections describe the steps required to set up a new configuration project in ProSoft Configuration Builder (PCB), and to export the .L5X file for the project.

2.3.1 Setting Up the Project in PCB

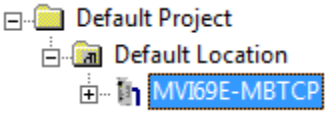
1. Start **PROSOFT CONFIGURATION BUILDER** (PCB).
2. The *PCB* window consists of a tree view on the left, and an information pane and a configuration pane on the right side of the window. The tree view consists of folders for *Default Project* and *Default Location*, with a *Default Module* in the *Default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



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



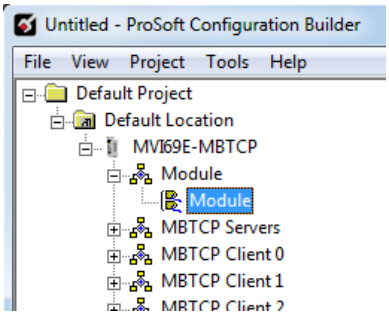
4 In the *Product Line Filter* area of the dialog box, click **MVI69**. In the *Select Module Type* dropdown list, click **MVI69E-MBTCP**, and then click **OK** to save your settings and return to the *ProSoft Configuration Builder* window. The MVI69E-MBTCP icon is now visible in the tree view.



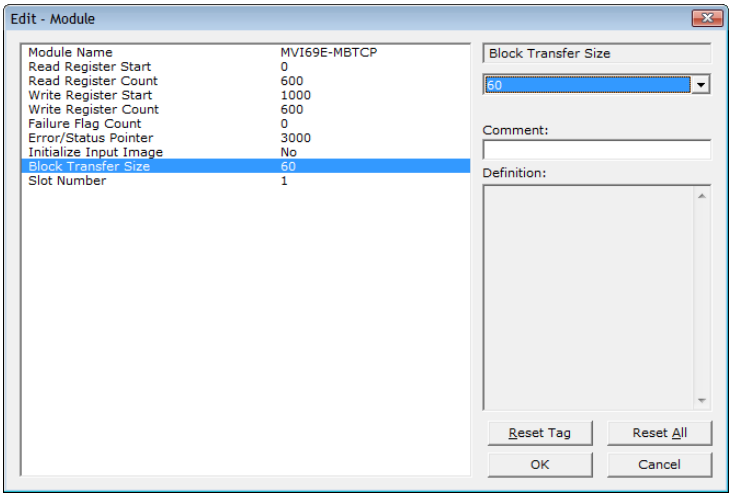
2.3.2 Creating and Exporting the .L5X File

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

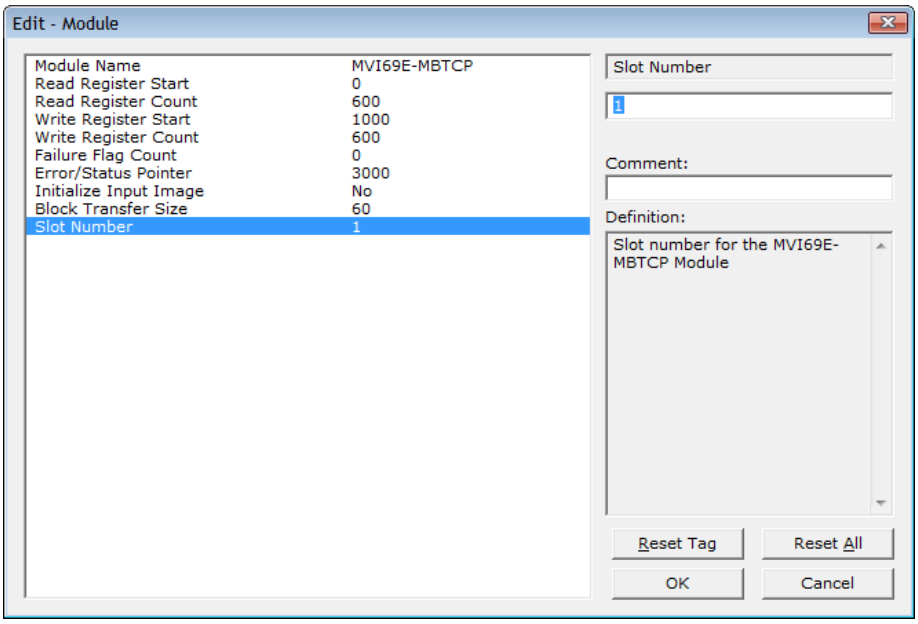
1. Expand the **MVI69E-MBTCP** icon by clicking the **[+]** symbol beside it. Similarly, expand the Module icon.



2. Double-click the Module icon to open the *Edit - Module* dialog box.
3. Set the *Block Transfer Size* to the desired size of the data blocks transferred between the module and processor (60, 120 or 240 words). You can find block transfer size information in *Normal Data Transfer* (page 73).

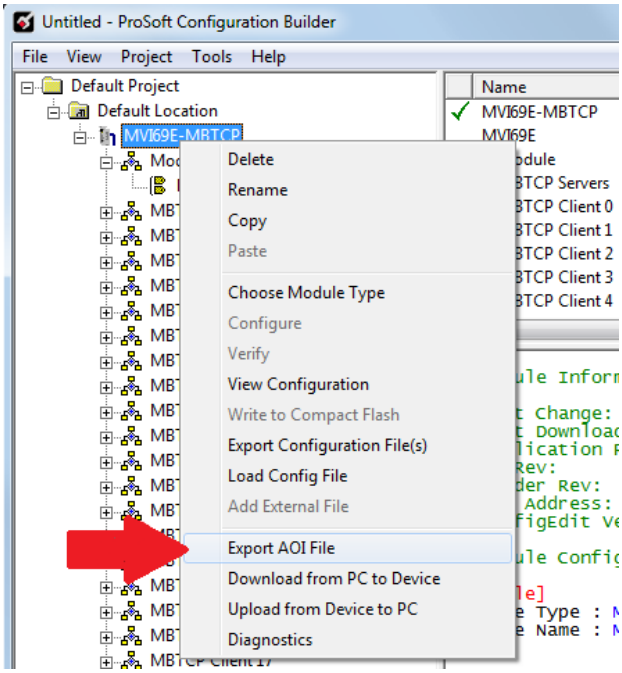


4 Edit the *Slot Number* indicating where the module is located in the 1769 bus.

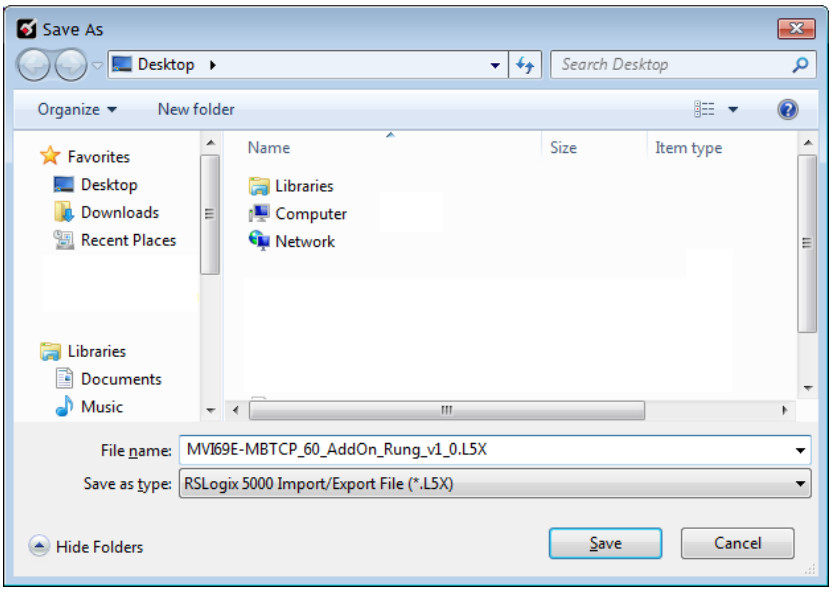


5 Click **OK** to close the *Edit – Module* dialog box. The .L5X file is now ready to export to the PC/Laptop.

6 Right-click the **MVI69E-MBTCP** icon in the project tree and choose **EXPORT AOI FILE**.



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

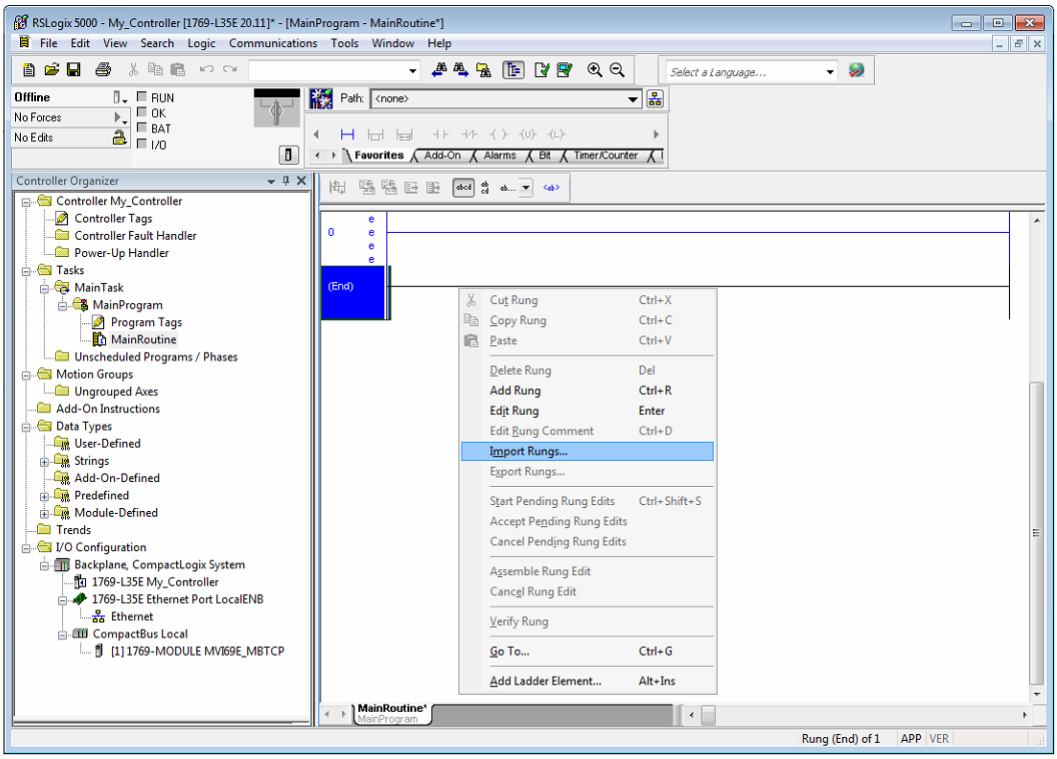


2.4 Importing the Add-On Instruction

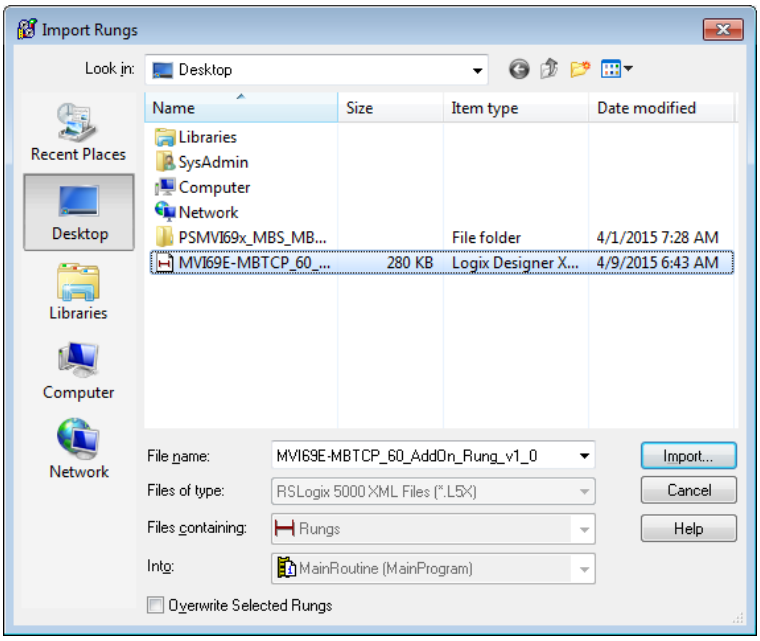
1. Open the application in Studio 5000.
2. Expand the **TASKS** folder, and expand the **MAINTASK** folder.
3. Expand the **MAINPROGRAM** folder and then double-click the **MAINROUTINE** icon to display the Routine Editor. The MainRoutine contains rungs of logic. The very last rung in this routine is blank. This is where you can import the Add-On Instruction (AOI).

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**.

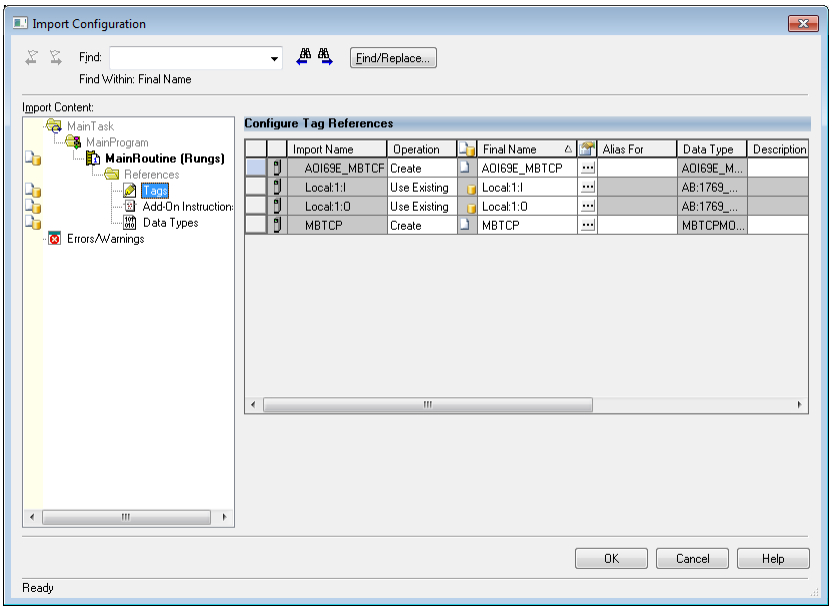


5 Select the **.L5X** file that you exported from ProSoft Configuration Builder. See *Creating and Exporting the .L5X File* (page 23).



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

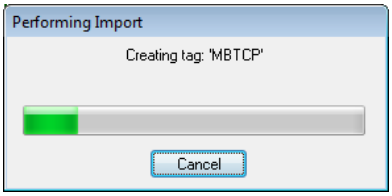
Note: If you are using RSLogix version 16 or earlier, the *Import Configuration* dialog box does not contain the *Import Content* tree.



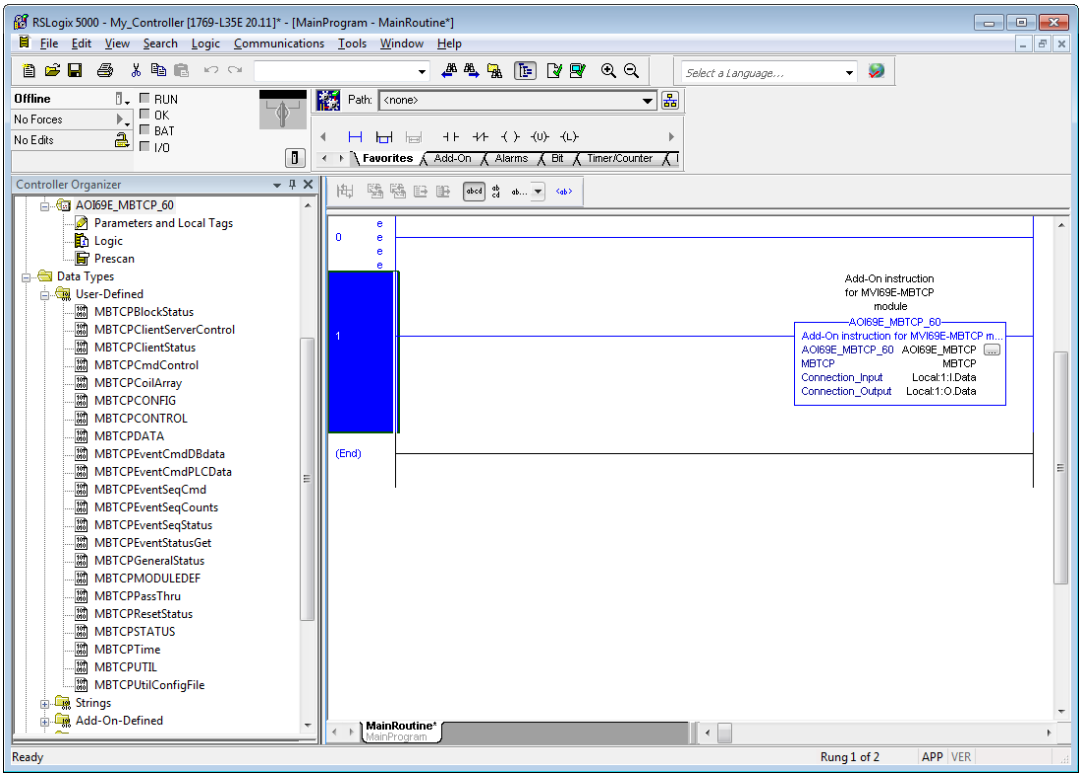
7 If the module is not located in the default slot (or is in a remote rack), edit the connection input and output variables that define the path to the module in the **FINAL NAME** column (**NAME** column for RSLogix version 16 or less). For example, if your module is located in slot 3, change *Local:1:I*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

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

8 Click **OK** to confirm the import. RSLogix indicates that the import is in progress:



9 When the import is completed, the new rung with the Add-On Instruction is visible as shown in the following image.



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

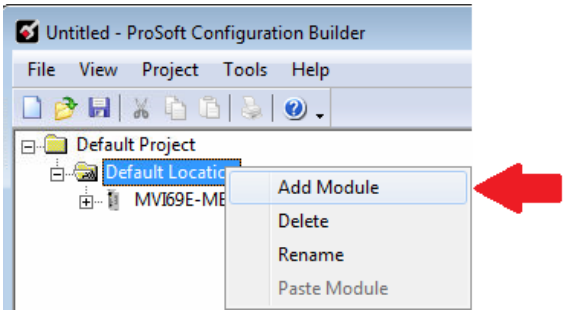
2.5 Adding Multiple Modules in the Rack (Optional)

The following procedure is for running multiple MVI69E-MBTCP modules running in the same CompactLogix rack.

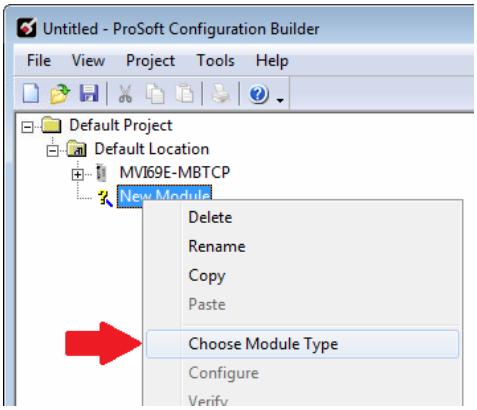
1. Add a new MVI69E-MBTCP module to the ProSoft Configuration Builder (PCB) project.
2. Export the module configuration as an L5X file.
3. Add a new MVI69E-MBTCP to the Studio 5000 project.
4. Import the .L5X file into Studio 5000 for the new module as an Add-On Instruction.

2.5.1 Adding an Additional Module in PCB

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



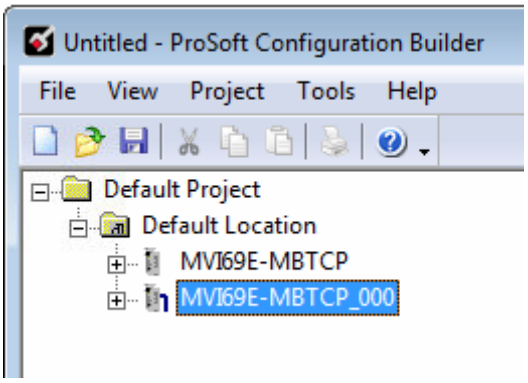
3. Right-click **NEW MODULE** and choose **CHOOSE MODULE TYPE**.



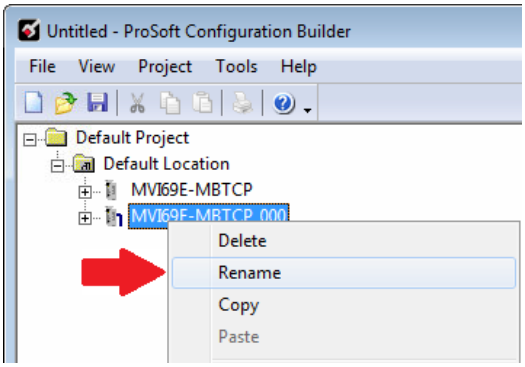
4. In the *Choose Module Type* dialog box, select **MVI69E** in the **PRODUCT LINE FILTER** area, and then select **MVI69E-MBTCP** as the **MODULE TYPE**. Click **OK**.

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

Note: Give each MVI69E-MBTCP module a unique name. The default name on a duplicate module appends a number to the end such as **MVI69E-MBTCP_000**, **MVI69E-MBTCP_001**, etc.



6 Rename the module by right clicking the module and selecting **Rename**.

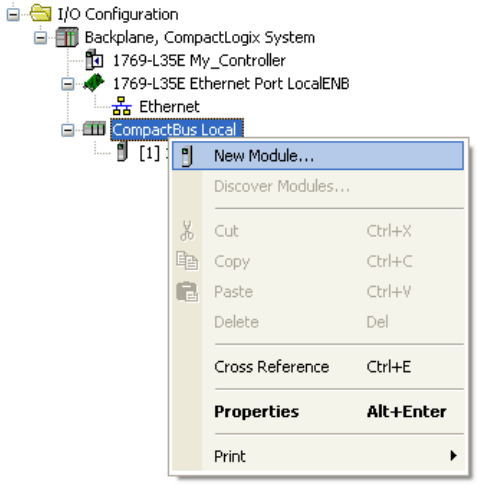


7 Configure the module parameters. See *Module Configuration Parameters* (page 38), and then export the AOI .L5X file for the new module (right-click the module and then choose **EXPORT AOI FILE**). See *Creating and Exporting the .L5X File* (page 23).

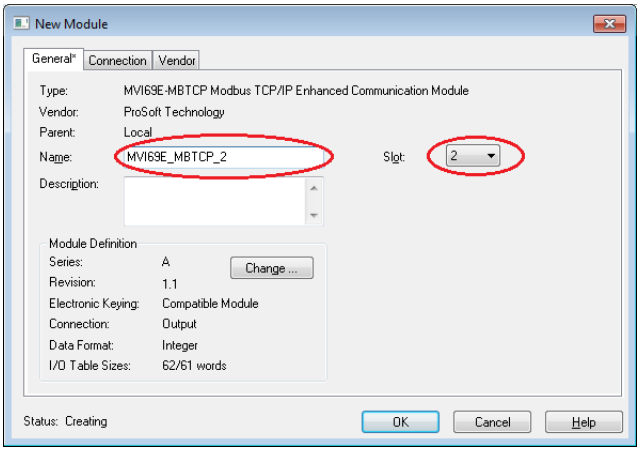
2.5.2 Adding an Additional Module in Studio 5000

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

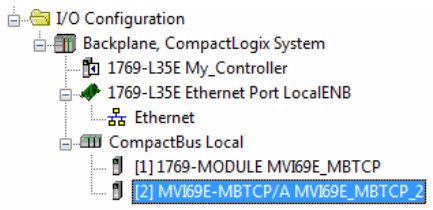
1. Start Studio 5000 and open the project.
2. In Studio 5000, locate the **I/O CONFIGURATION** folder. Right click **COMPACTBUS LOCAL** and choose **NEW MODULE**.



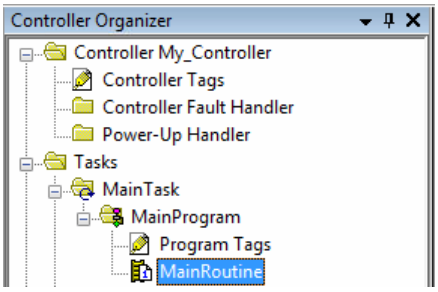
3. In the *Select Module Type* dialog box, select the MVI69E-MBTCP module.
   - For an Add-On Profile (AOP), it adds the MVI69E-MBTCP module and configures the relevant parameters. RSLogix version 15 or later is required to use an AOP.
   - If using an AOP is not an option, select **GENERIC 1769 MODULE** and click **CREATE**.
4. The *New Module* dialog box appears. Enter a unique name for the new module, and confirm the slot number of the new module.



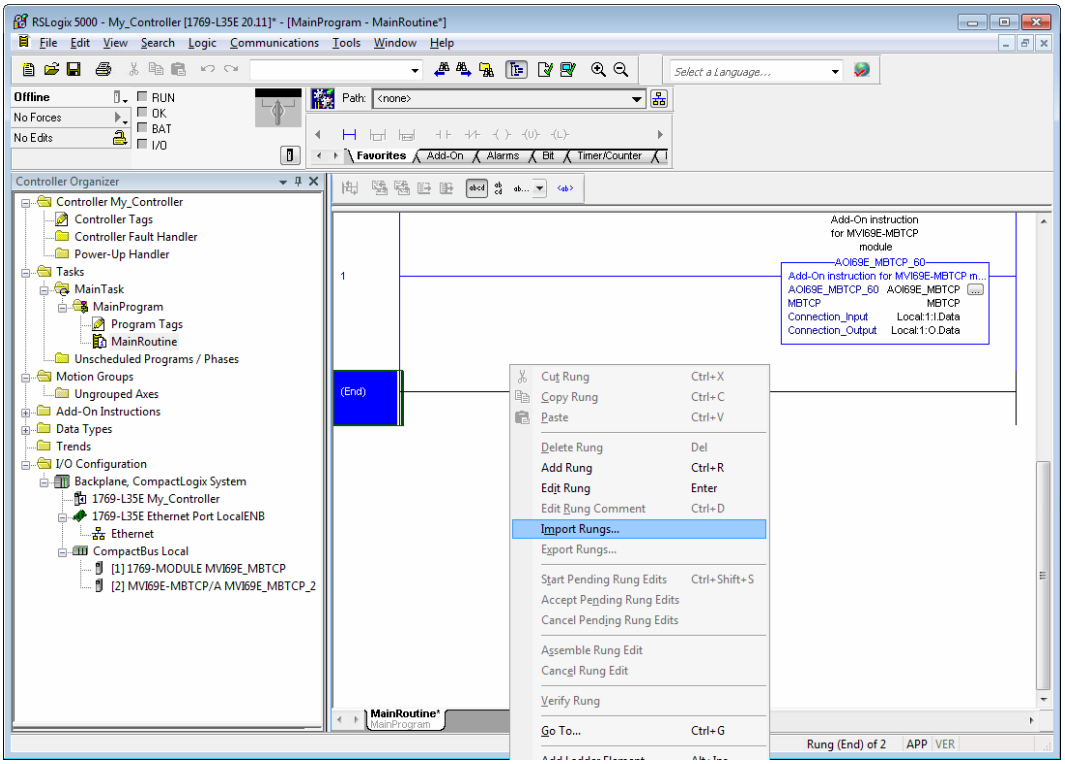
5 Click **OK**. The new module is now visible.



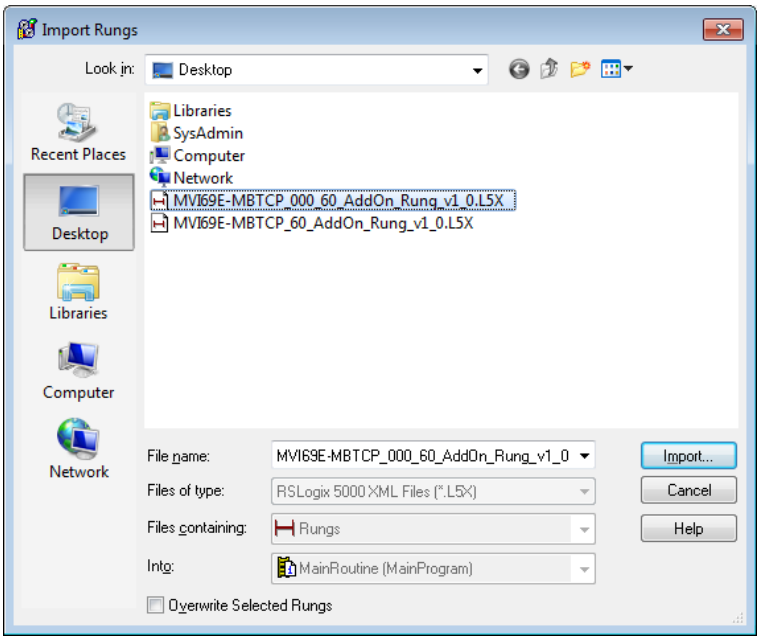
6 Import the Add-On Instruction(AOI) for the new module (see Adding another module in PCB). In the *Controller Organizer* pane, double-click **MAINROUTINE** to open the ladder for the routine.



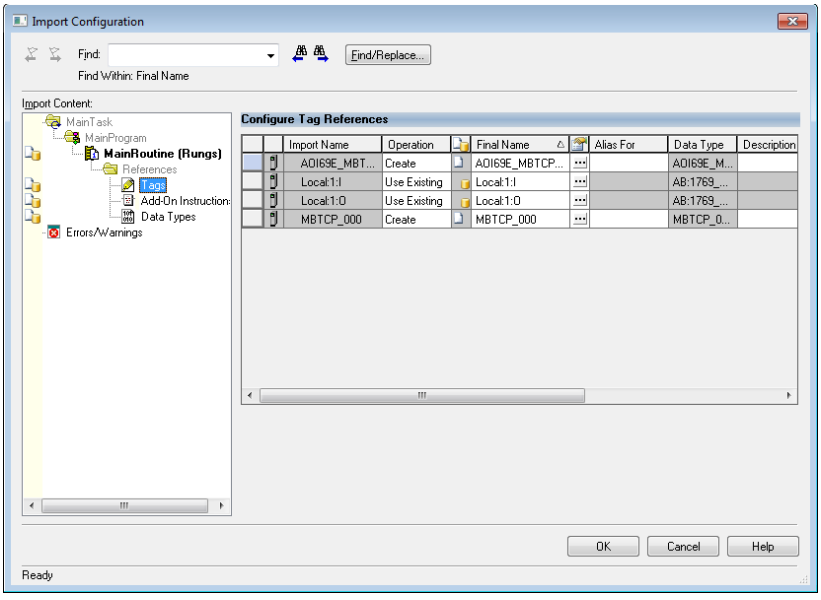
7 Right-click an empty rung in the routine and then choose **IMPORT RUNGS…**



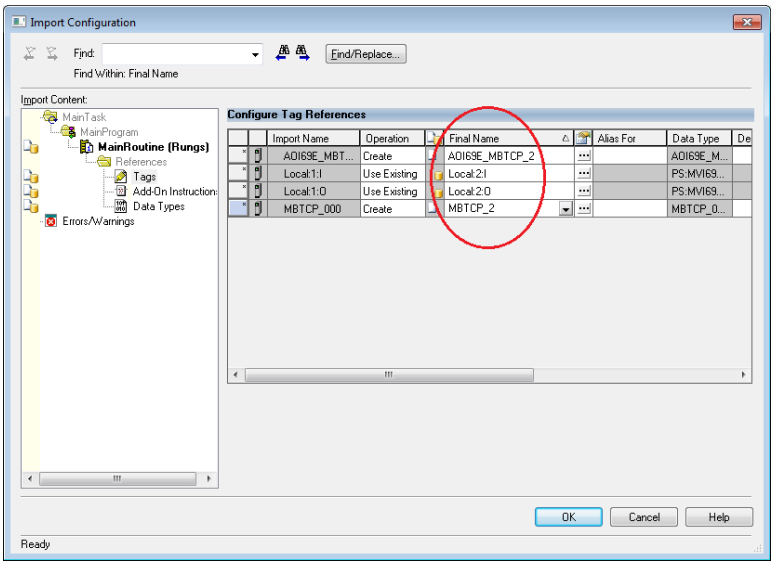
8 Select the .L5X file you created and exported for the new module, and click **IMPORT.** Recall that the new .L5X file has a unique filename that is specific to the new module.

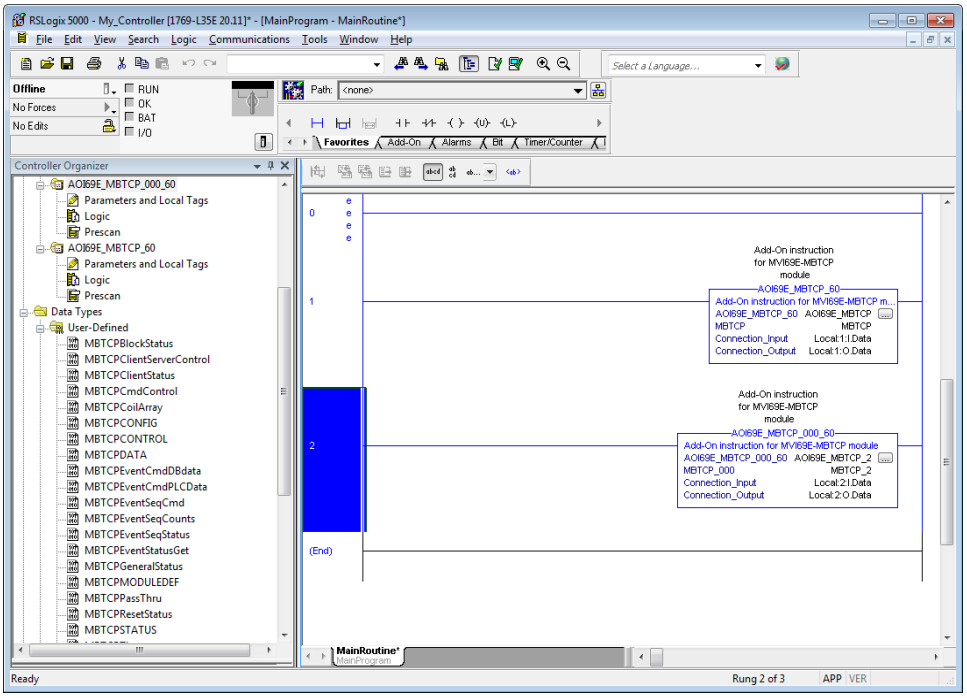


9 This opens the *Import Configuration* dialog box. Click **TAGS** to show the controller tags in the AddOn Instruction. Edit the **FINAL NAME** column of the tags for the second module to make them unique.



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. 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_MBTCP' and 'MBTCP' arrays as shown below. Then click **OK**.





11 The setup procedure is now complete. Save the project. It is ready to download to the CompactLogix processor.

3 Configuring the MVI69E-MBTCP Using PCB

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 *MBTCP.CONFIG* controller tag generated by the previously exported AOI. See *Creating and Exporting the .L5X File* (page 23). When the MVI69E-MBTCP 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 13) 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 13).

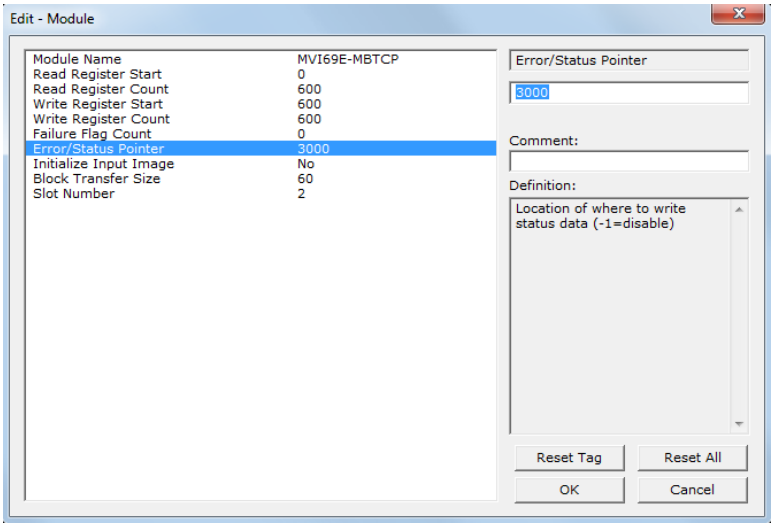
3.1.2 Renaming PCB Objects

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

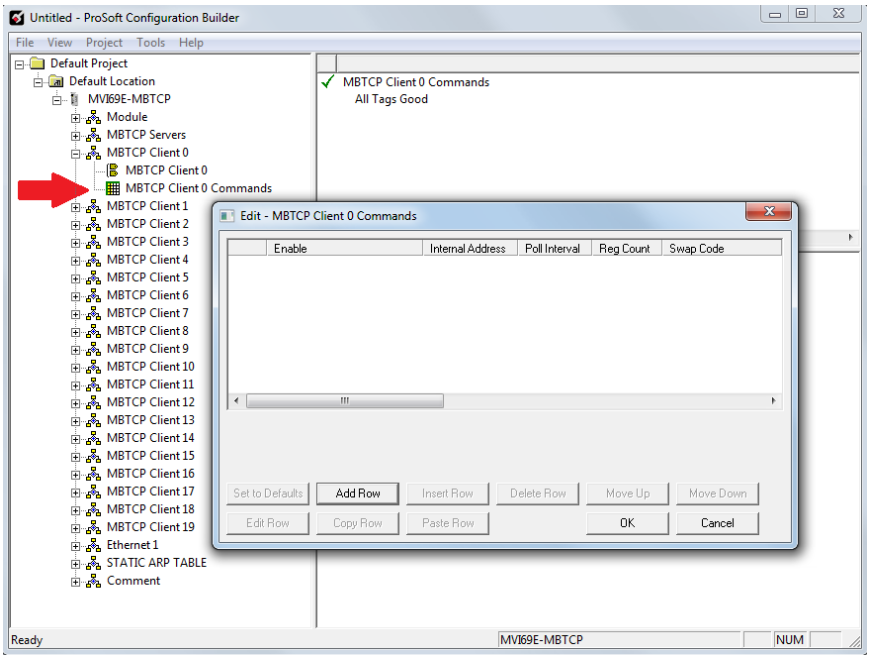
1. Right-click the object you want to rename and then choose **RENAME.**
2. Type the new name for the object and press **Enter**.

3.1.3 Editing Configuration Parameters

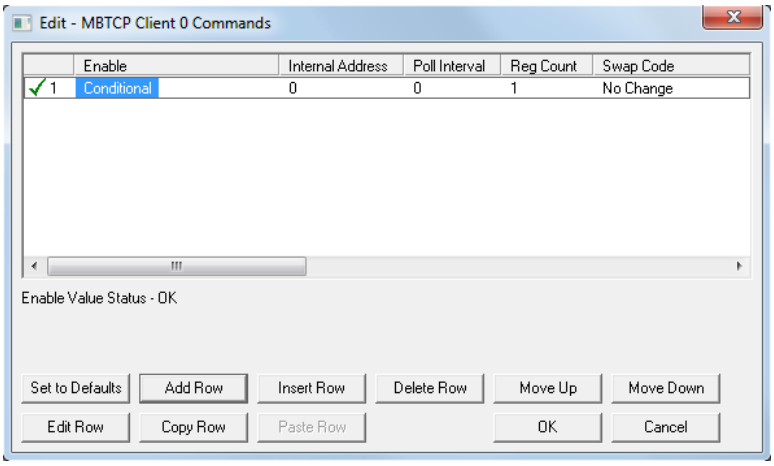
1. Click the **[+]** sign next to the **MVI69E-MBTCP** icon to expand module information.
2. Click the **[+]** sign next to any icon to view information and configuration options.
3. Double-click any icon to open an *Edit* dialog box. To edit a parameter, click the parameter in the left pane and then make your changes in the right pane.



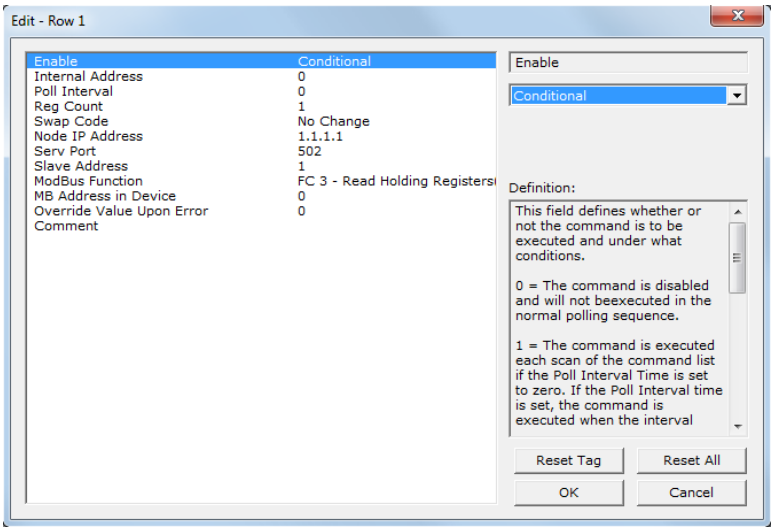
4. Click **OK** to save changes.
5. Double-click any icon to open an *Edit* dialog box to build and edit Modbus Client commands.



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



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



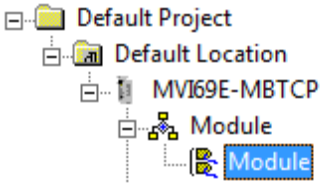
3.1.4 Printing a Configuration File

1 In the main PCB window, right-click the **MVI69E-MBTCP** 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

This section contains general module configuration parameters. In the ProSoft Configuration Builder (PCB) tree view, expand the **MVI69E-MBTCP** icon, then expand **MODULE**, and then double-click the **MODULE** icon.

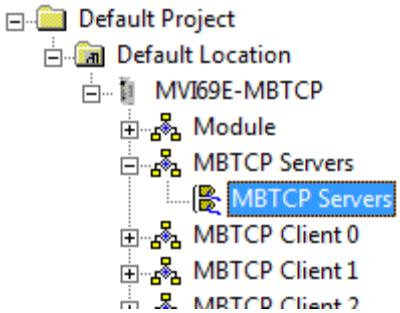


Parameter	Value	Description
Module Name	ASCII characters (max. 38)	Assigns a name to the module that can be viewed using the configuration/debug port. Use this parameter to identify the module and the configuration file.
Read Register Start	0 to 9999	Specifies the starting address of the module's Read Data area. Data in this area is transferred from the module to the processor.
Read Register Count	0 to 10,000	Specifies the size of the Read Data area.
Write Register Start	0 to 9999	Specifies the start of the Write Data area in module memory. Data in this area is transferred in from the processor.
Write Register Count	0 to 10,000	Specifies the size of the Write Data area.
Failure Flag Count	0 to 65535	Specifies the number of consecutive backplane transfer failures that can occur before Modbus communications are halted.
Error/Status Block Pointer	-1 to 9955	The starting MVI69E-MBTCP database location to store server error/status data. If a value of -1 is entered, the error/status data will not be placed in the database. This feature returns 8 server error/status values. The descriptions of these values start at the *MBTCP.STATUS.GeneralStatus.MNETRequestCount* controller tag. Refer to the General Status description on page 67 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.
Block Transfer Size	60, 120 or 240	Specifies the number of words in each block transferred between the module and processor.
Slot Number	1 to x	Specifies the slot in the CompactLogix rack for the module.

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

3.2.2 MBTCP Servers

This section applies to configuring the MVI69E-MBTCP Server (Slave) Driver. In the ProSoft Configuration Builder tree view, double-click the **MBTCP SERVERS** icon.

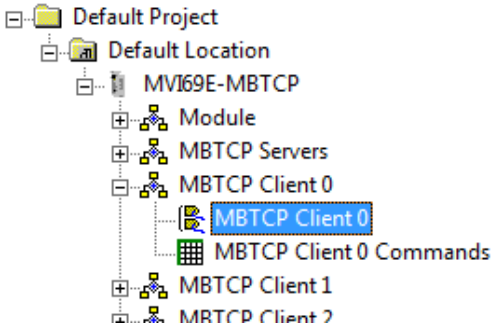


Parameter	Value	Description
Start Active	Yes or No	Specifies whether the port and commands are active upon module boot-up.
Pass-Through Mode	Client, Server, or Server with Pass-Through	Specifies which device type the port emulates. Refer to the section *Data Flow Between the Module and Processor* (page 77) for more information on the server with Pass-Through option.
Float Flag	Yes or No	Specifies how the Server driver responds to Function Code 3, 6, and 16 commands (read and write Holding Registers) from a remote client when it is moving 32-bit floating-point data. If the remote client expects to receive or send one complete 32-bit floating-point value for each count of one (1), then set this parameter to **YES**. When set to **YES**, the Server 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 client for write commands. Example: Count = **10**, Server driver sends 20 16-bit registers for 10 total 32-bit floating-point values. If, however, the remote client 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** (the default setting). You also must set the *Float Start* and *Float Offset* parameters to appropriate values whenever the *Float Flag* parameter is set to **YES**.
Float Start	0 to 9998	Defines the first register of floating-point data. All requests with register values greater-than or equal to this value are considered floating-point data requests. This parameter is only used if the *Float Flag* is enabled. For example, if you enter a value of 7000, all requests for registers 7000 and above are considered as floating-point data.
Float Offset	0 to 9998	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 3000 and set the float start parameter to 7000, data requests for register 7000 use the internal Modbus register 3000.
Output Offset	0 to 9999	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.
Bit Input Offset	0 to 9999	Specifies the offset address into the internal Modbus database for network requests for Modbus function 2 commands. For example, if you set the value to 150, an address request of 0 returns the value at register 150 in the database.

Holding Register Offset	0 to 9999	Specifies the offset address in the internal Modbus database for network requests for Modbus function 3, 6, or 16 commands. For example, if you enter a value of 50, a request for address 0 corresponds to the register 50 in the database.
Word Input Offset	0 to 9999	Specifies the offset address into the internal Modbus database for network requests for Modbus function 4 commands. For example, if you set the value to 150, an address request of 0 returns the value at register 150 in the database.
Connection Timeout	0 to 1200	Specifies the server's timeout period if it is not receiving any new data in the number of seconds preset.

3.2.3 MBTCP Client x

This section defines the general configuration for MBTCP Client x. You can configure up to 20 MBTCP clients, each using the parameters below. In the ProSoft Configuration Builder tree view, double-click the **MBTCP CLIENT X** icon.



Parameter	Value	Description
Enabled	Yes or No	Enables this client
Start Active	Yes or No	Specifies whether to start with commands active on boot up
Error/Status Pointer	-1 to 9990	The starting MVI69E-MBTCP database location to store Client x's error/status data. If a value of -1 is entered, the error/status data will not be placed in the database. This feature returns 8 Client x error/status data values. The descriptions of these values start at the *MBTCP.STATUS.ClientStatus.CommandRequests* controller tag. Refer to the Client Status description on page 65 for more information.
Command Error Pointer	-1 to 9984	Specifies the address in the module's database where the command error data is placed. If you set the value to -1, the data is not transferred to the database. This data should be placed within the read data range of module memory.
Minimum Command Delay	0 to 65535 milliseconds	Specifies the number of milliseconds to wait between receiving the end of a server'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 servers can be ready to receive them. It does not affect retries of a command, as retries are issued when a command failure is recognized.
Response Timeout	1 to 65535 milliseconds	Specifies the command response timeout period in 1 millisecond increments. The client waits for a response from the addressed server within the timeout period before re-transmitting the command (Retries) or skipping to the next command in the Command List. The value 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.
Float Flag	Yes or No	Specifies if the Daniel/ENRON-specific floating-point data access functionality is to be implemented. If you set the *Float Flag* to Y, Modbus functions 3, 6 and 16 interpret floating point values for registers as specified by the two following parameters (*Float Start*, *Float Offset*). Note: You do not need to enable this parameter for most applications using floating-point data.

Float Start	0 to 9998	Specifies the first register of floating-point data. All requests with register values greater-than or equal to this value are considered floating-point data requests. This parameter is only used if the *Float Flag* is enabled.
Float Offset	0 to 9998	Specifies the start register for floating-point data in the internal database. This parameter is used only if the *Float Flag* is enabled.
ARP Timeout	1 to 60 seconds	Specifies the number of seconds to wait for an ARP reply after a request is issued. If the value is out of range, the module uses the default value of 5.
Command Error Delay	0 to 300	Specifies the number of 100 millisecond intervals to turn off a command in the error list after an error is recognized for the command. If you set this parameter to 0, there is no delay.
MBAP Port Override	Yes or No	Override default port settings. **NO** = Use standard server Port 502 with MBAP format messages. All other server Port values use encapsulated Modbus message format (RTU via TCP). **YES** = Use MBAP format messages for all server Port values. RTU via TCP is not used.

3.2.4 MBTCP Client x Commands

In order to interface the MVI69E-MBTCP module with Modbus server devices, you must create a command list. The commands in the list specify the server 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.

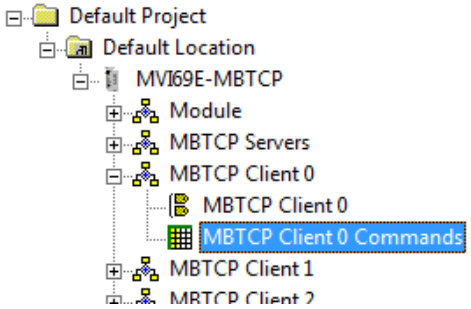
Each of the 20 Client Command Lists supports up to 16 commands each. The command list is processed from top (Command #0) to bottom.

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.

Note: The first command in the Client x Command list cannot be disabled.

The MBTCP Modbus client (and server) communication drivers support several data read and write commands. When you configure a command, you need to consider the type of data (bit, 16-bit integer, 32-bit float, etc), and the level of Modbus support in the server equipment.

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



Parameter	Value	Description
Enable	Disable, Enable, Conditional, Bit/Word Override, Float Override	Specifies whether the command is executed and under what conditions. **DISABLE** (0) = The command is disabled and is not executed in the normal polling sequence. **ENABLE** (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** (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** (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 9999 (word-level)	Specifies the module's internal database register to be associated with the command.

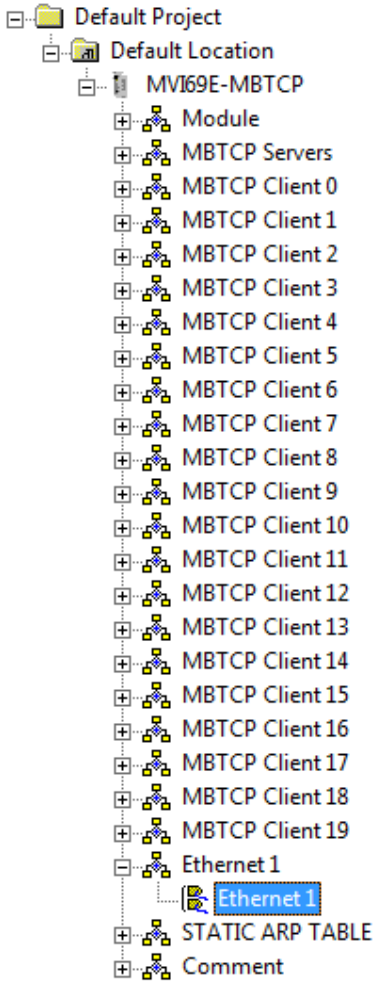
	or 0 to 159,999 (bit-level)	For Modbus Function Codes 3, 4, 6, or 16, the allowable range is 0 to 9999. For Modbus Function Codes 1, 2, 5, or 15, the allowable range is 0 to 159,999. **Note**: This bit address range is available with ProSoft Configuration Builder (PCB) v4.6 or later. Previous versions have a range of 0 to 65535. If the command is a read function, the data read from the server 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 server 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.
Poll Interval	0 to 65535 (1/10 second)	Specifies the minimum interval between executions of continuous commands (**Enable** code = 1). Example: The parameter is entered in 1/10th of a second. Therefore, if a value of 100 is entered, the command executes no more frequently than every 10 seconds. When the command reaches the top of the command queue and 10 seconds has not elapsed, it is skipped until the poll interval has expired.
Register Count	1 to 125 (words) or 1 to 2000 (coils)	Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point. For Modbus Function Codes 1, 2, and 15, this parameter sets the number of single bit digital points (inputs or coils) to be associated with the command. **Note:** Up to 2000 coils are supported for Modbus Function Codes 1 and 2. Up to 1968 coils are supported for Modbus Function Code 15. For Modbus Function Codes 3, 4, and 16, this parameter sets the number of 16-bit registers to be associated with the command.
Swap Code	No Change, Word Swap, Word and Byte Swap, Byte Swap	Defines if the data received from the Modbus server is to be ordered differently than received from the server device. This parameter is helpful when dealing with floating-point or other multi-register values, as there is no standard method of storing these data types in server devices. You can set this parameter to order the register data received in an order useful by other applications. **NO CHANGE** = No change is made in the byte ordering (ABCD = ABCD) **WORD SWAP** = The words are swapped (ABCD= CDAB) **WORD AND BYTE SWAP** = The words are swapped, then the bytes in each word are swapped (ABCD=DCBA) **BYTE SWAP** = The bytes in each word are swapped (ABCD=BADC) **Note:** Each pair of characters is a byte (example: AB and CD). Two pairs of characters is a 16-bit register (example: ABCD).
Node IP Address	xxx.xxx.xxx.xxx	Specifies the IP address of the target device being addressed by the command.

Service Port	1 to 65535	Use a value of 502 when addressing Modbus TCP/IP servers which are compatible with the Schneider Electric MBAP specifications (most devices). If a server implementation supports another service port, enter the value here. Service Port 2000 is common for encapsulated format messages.
Slave Address	0 to 255	Mainly used for Modbus TCP/IP to serial conversion. This specifies the Modbus slave node address on the serial network to be considered. If a Modbus TCP/IP server device does not have or need a slave address, use a value of '1'. 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 to be executed by the command. These function codes are defined in the Modbus protocol. **1** = Read Coil Status (0xxxx) **2** = Read Input Status (1xxxx) **3** = Read Holding Registers (4xxxx) **4** = Read Input Registers (3xxxx) **5** = Force (Write Single) Coil (0xxxx) **6** = Force (Write Single) Holding Register (4xxxx) **15** = Preset (Write) Multiple Coils (0xxxx) **16** = Preset (Write) Multiple Registers (4xxxx)
MB Address in Device	0 to 65535	Specifies the register or digital point address offset within the Modbus server device. The MBTCP client reads or writes from/to this address within the server. Refer to the documentation of each Modbus server device for their register and digital point address assignments. **Note:** The value you enter 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:** When using a Modbus Function Code 3 to read from address 40010 in the server, enter a value of '9' for 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 when the *Enable* parameter is 3 (Bit/Word Override) or 4 (Float Override). If an error occurs associated with a read command, the module automatically populates the associated database area with this override value.

3.2.5 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.



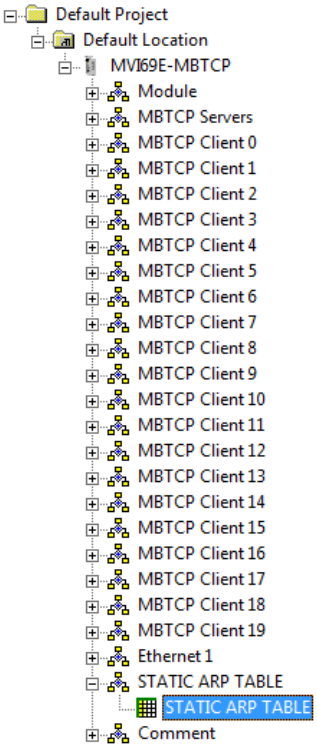
Parameter	Description
IP Address	Unique IP address assigned to the module
Netmask	Subnet mask of module
Gateway	Gateway (if used)

3.2.6 Static ARP Table

This section defines a list of static IP addresses that the module uses when an ARP (Address Resolution Protocol) is required. The module accepts up to 40 static IP/MAC Address data sets.

Use the Static ARP table to reduce the amount of network traffic by specifying IP addresses and their associated MAC (hardware) addresses that the MVI69E-MBTCP module communicates with regularly.

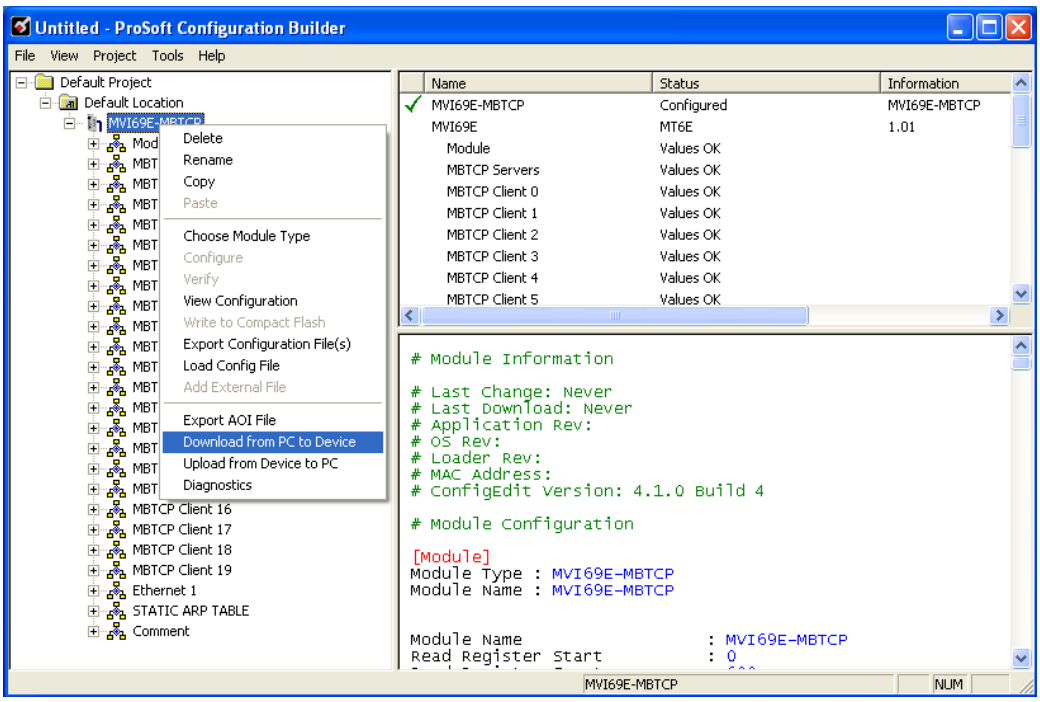
In ProSoft Configuration Builder tree view, double-click the **STATIC ARP TABLE** icon.



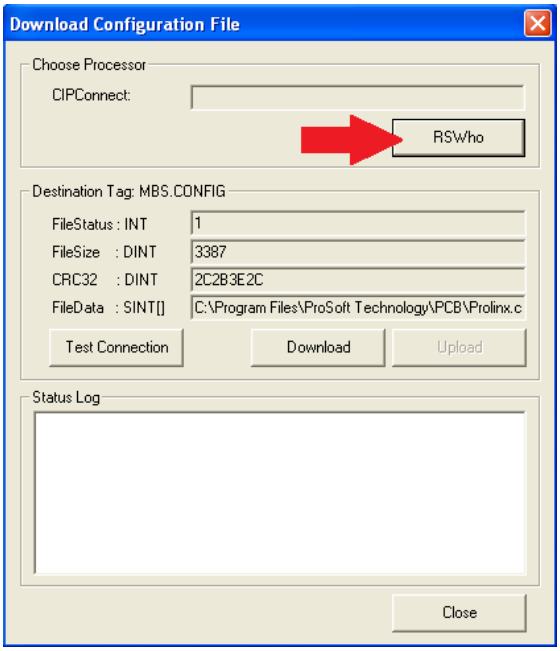
Parameter	Value	Description
IP Address	xxx.xxx.xxx.xxx	This table contains a list of static IP addresses that the module uses when an ARP is required. The module accepts up to 40 static IP/MAC address data sets. **Important:** If the device in the field is changed, this table must be updated to contain the new MAC address for the device and downloaded to the module. If the MAC is not changed, there is no communication with the module.
Hardware MAC Address	FF.FF.FF.FF.FF.FF	This table contains a list of static MAC addresses that the module uses when an ARP is required. The module accepts up to 40 static IP/MAC address data sets. **Important:** If the device in the field is changed, this table must be updated to contain the new MAC address for the device and downloaded to the module. If the MAC is not changed, there is no communication with the module.

3.3 Downloading the Configuration File to the Processor

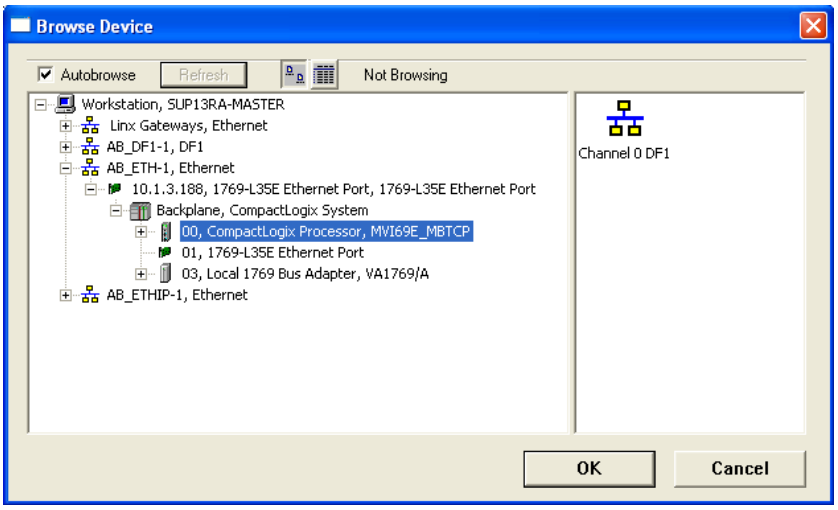
1 In the ProSoft Configuration Builder tree view, right-click the module icon and then click **DOWNLOAD FROM PC TO DEVICE**.



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

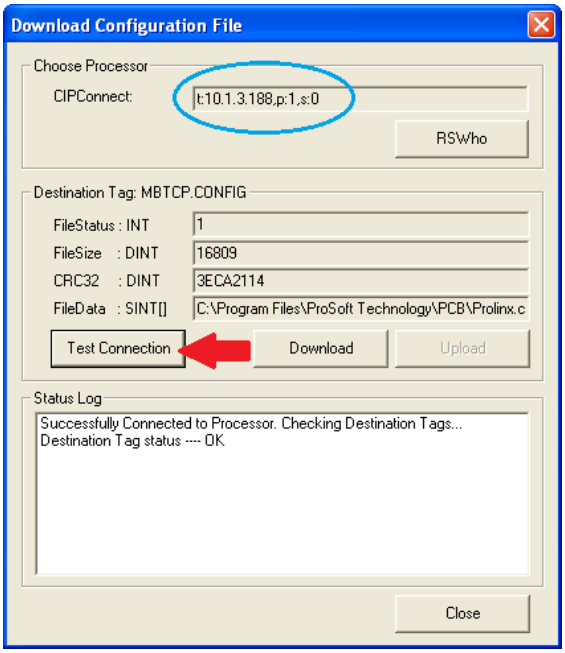


3 Browse to, and then click the CompactLogix processor and click **OK**.

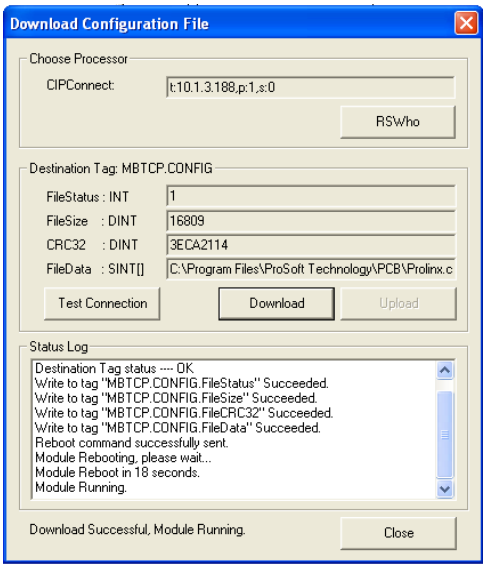


Note: DF1 serial download via CIPConnect is not supported. Only use Ethernet or EtherNet/IP drivers via RSWho.

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



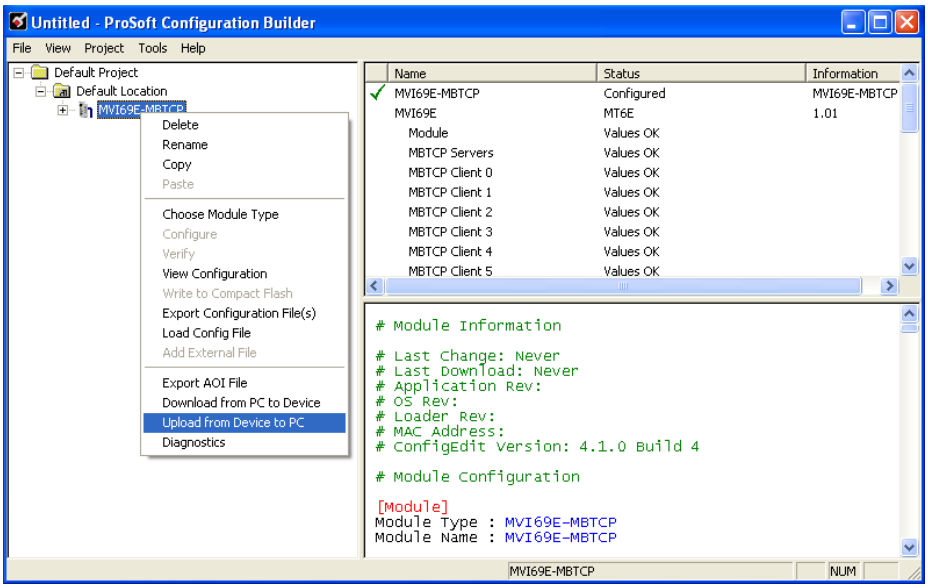
5 When ready, click **DOWNLOAD** to download the configuration file to the processor. Following the download process, the module is automatically rebooted.



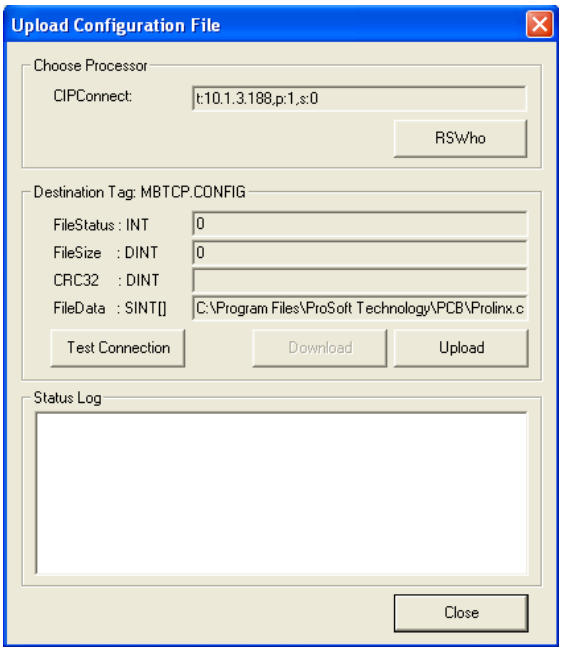
6 After rebooting, the ladder logic sends the configuration data from the processor to the module. When that is complete, the module starts Modbus communications.

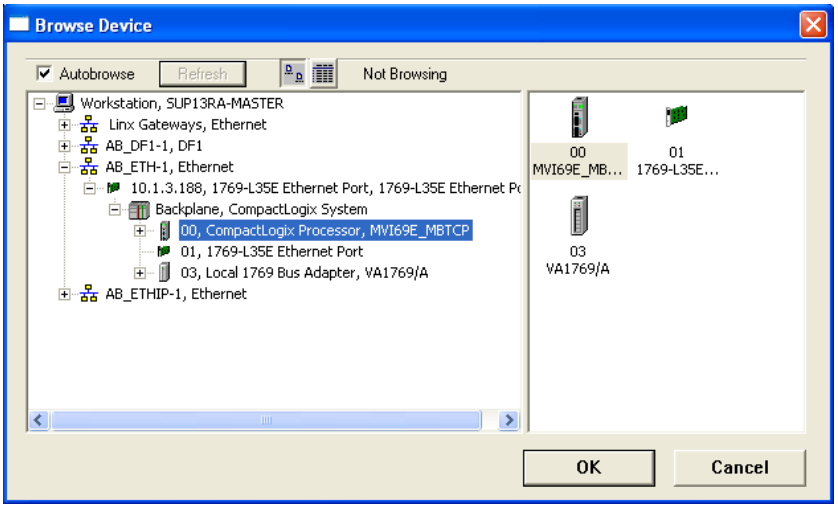
3.4 Uploading the Configuration File from the Processor

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

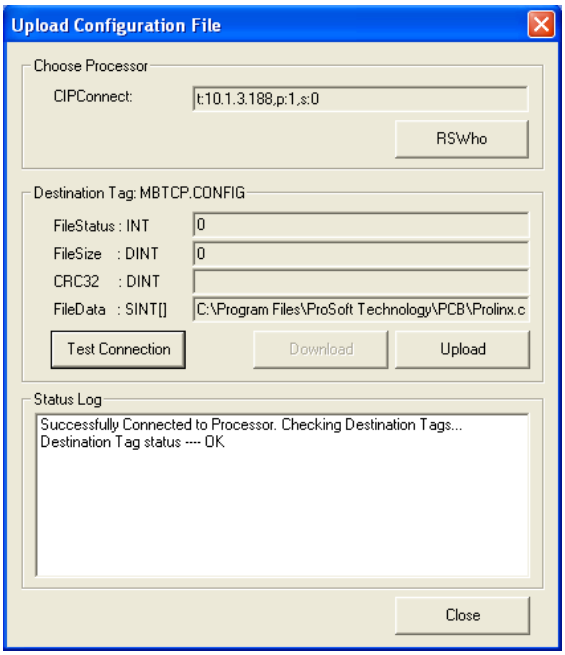


2 In the *Upload Configuration File* dialog box, the CIPConnect path should be constructed if you have previously downloaded the configuration file from the same PC. If not, click **RSWHO**, browse to, and then select the CompactLogix Processor, and click **OK**.

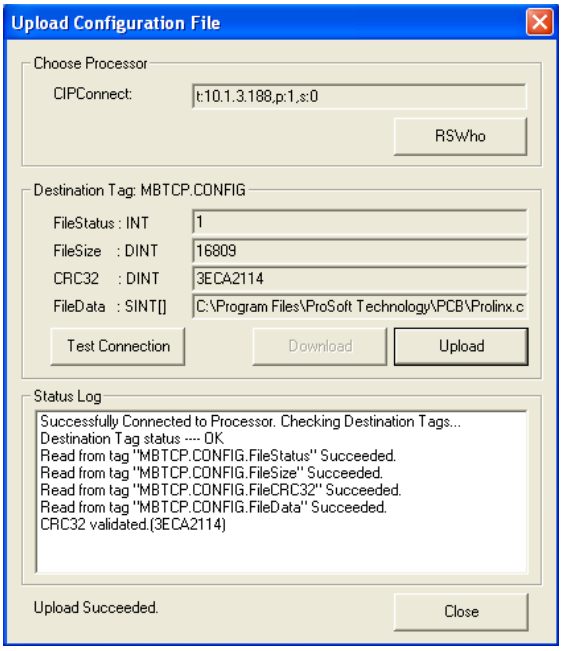




3 Click **TEST CONNECTION** to verify the path is active and can successfully connect to the processor.



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



5 PCB now displays the uploaded configuration file.

4 Using Controller Tags

Ladder logic is required for managing communication between the MVI69E-MBTCP module and the CompactLogix processor. The ladder logic handles tasks such as:

- Module backplane data transfer
- Special block handling
- Status data receipt

Additionally, a power-up handler may be needed to initialize the module's database and may clear some processor fault conditions.

The sample Import Rung with 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 Import Rung with Add-On Instruction works without needing any modification.

4.1 Controller Tags

Data related to the MVI69E-MBTCP is stored in the ladder logic in variables called controller tags. Individual controller tags can be grouped into collections of controller tags called controller tag structures. A controller tag structure can contain any combination of:

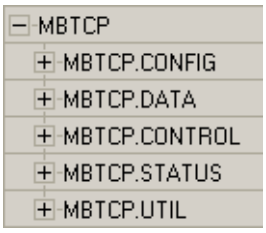
- Individual controller tags
- Controller tag arrays
- Lower-level controller tag structures

The controller tags for the module are pre-programmed into the Add-On Instruction Import Rung ladder logic. After you import the Add-On Instruction, you can find the controller tags in the *Controller Tags* subfolder, located in the *Controller* folder in the *Controller Organizer* pane of the main Studio 5000 window.

This controller tag structure is arranged as a tree structure. Individual controller tags are found at the lowest level of the tree structure. Each individual controller tag is defined to hold data of a specific type, such as integer or floating-point data. Controller tag structures are declared with user-defined data types (UDTs), which are collections of data types.

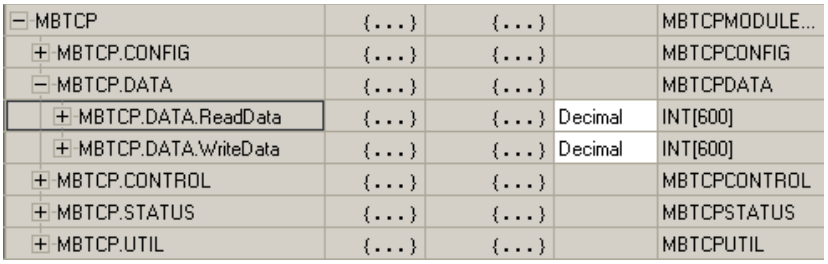
4.1.1 MVI69E-MBTCP Controller Tags

The main controller tag structure, *MBTCP*, 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 *MBTCP.DATA* controller tag structure, you see that it contains two controller tag arrays, *MBTCP.DATA.ReadData* and *MBTCP.DATA.WriteData*, which are 600-element integer arrays by default.



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

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

4.2 User-Defined Data Types (UDTs)

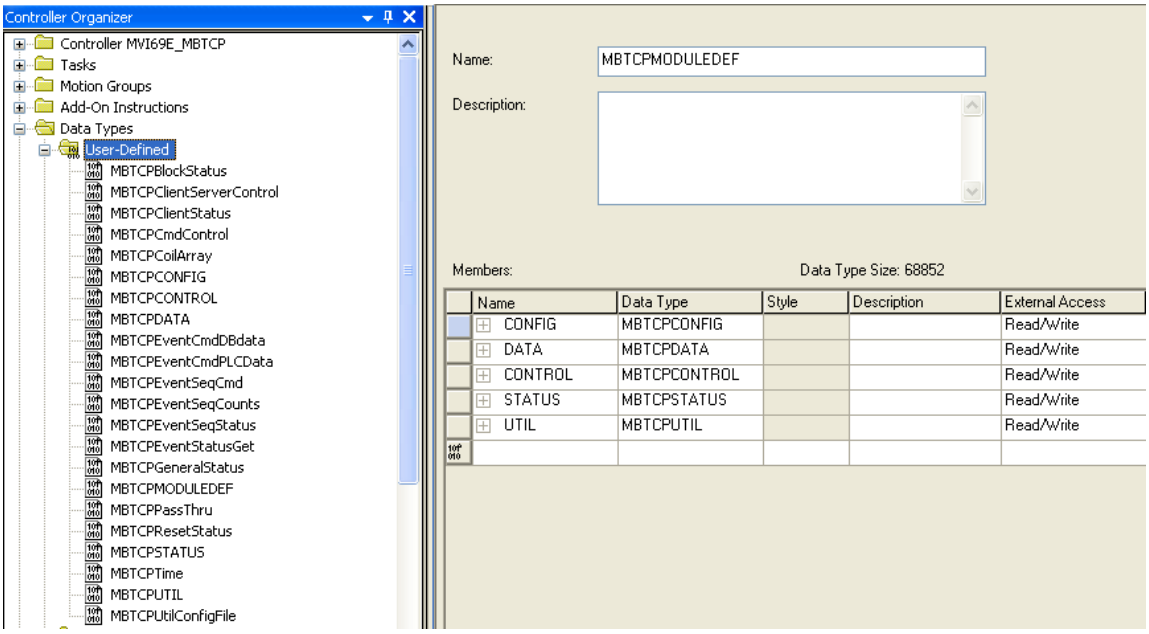
User-defined data types (UDTs) allow you to organize collections of data types into groupings. You can use these groupings, or data type structures, to declare the data types for controller tag structures. Another advantage of defining a UDT is that you may reuse it in other controller tag structures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

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

4.2.1 MVI69E-MBTCP User-Defined Data Types

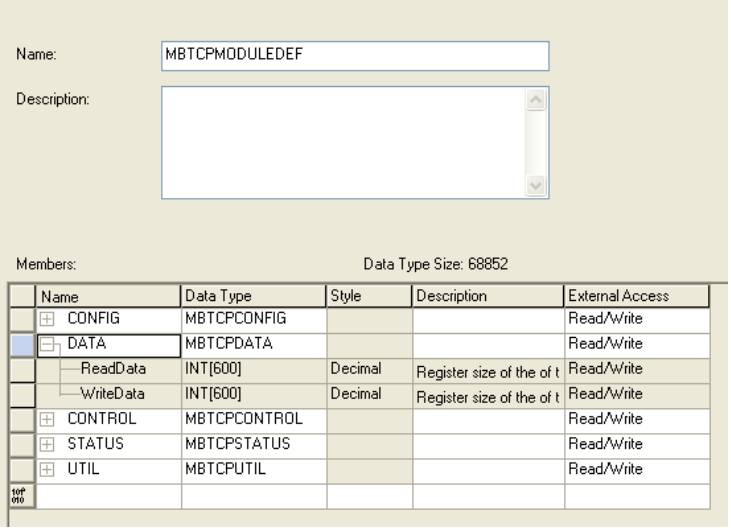
Multiple UDTs are defined for the MVI69E-MBTCP Add-On Instruction.

The main UDT, *MBTCPMODULEDEF*, contains all the data types for the module and was used to create the main controller tag structure, *MBTCP*. There are five UDTs one level below *MBTCPMODULEDEF*. These lower-level UDTs were used to create the *MBTCP.CONFIG*, *MBTCP.DATA*, *MBTCP.CONTROL*, *MBTCP.STATUS*, and *MBTCP.UTIL* controller tag structures.



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

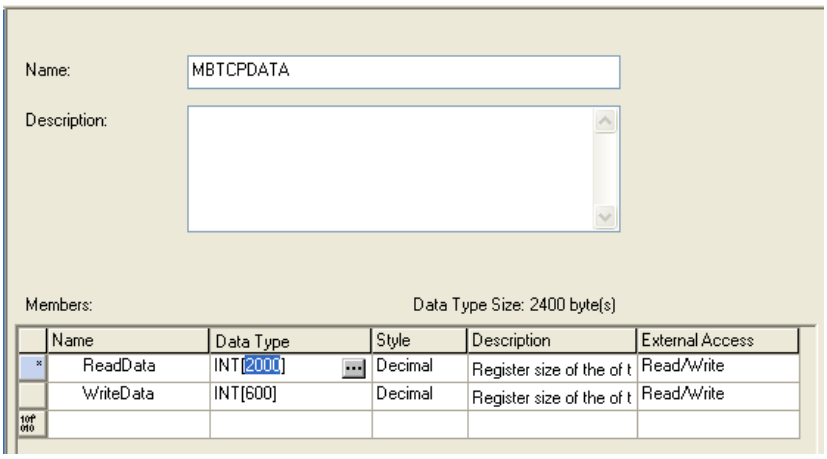
For example, if you expand *MBTCP.DATA*, it contains two UDTs, *ReadData* and *WriteData*. Both of these are 600-element integer arrays by default.



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

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

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



4.3 MBTCP Controller Tag Overview

Use controller tags to:

- View the read and write 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)

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

The following sections describe each of these controller tag structures in more detail.

4.3.1 MBTCP.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 MBTCP.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 do not match the calculated CRC.

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

4.3.2 MBTCP.DATA

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

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

4.3.3 MBTCP.CONTROL

This array handles special tasks requested by the processor.

4.3.3.1 MBTCP.CONTROL

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

Tag Name	Range	Description
MBTCP.CONTROL. CommandControl.Trigger	0 or 1	Command Control: Disable = 0, Enable = 1
MBTCP.CONTROL. CommandControl.CommandID	1 to 16	This value represents the quantity of commands to be requested in the Command Control block (1 to 16). The ladder logic uses this value to generate the Command Control Block ID. The rightmost digits of the Command Control Block ID are the number of commands requested by the block.
MBTCP.CONTROL. CommandControl.ClientID	0 to 19	Client ID associated with the command to be executed. There are 20 MBTCP clients available.
MBTCP.CONTROL. CommandControl.CommandIndex	0 to 15	This array stores the command index within the client ID. It can be determined by command row number minus 1. Up to 16 command indexes can be stored here
MBTCP.CONTROL. CommandControl.CmdsAddedToQue		This value is returned from the module. This is the number of commands added to the queue. -1 = Client not enabled and active. -2 = Client index not valid.
MBTCP.CONTROL. CommandControl.CmdInQue		Number of commands in the queue waiting to be executed

4.3.3.2 MBTCP.CONTROL.EventCommand_DBData

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

Tag Name	Range	Description
MBTCP.CONTROL.EventCommand_DB Data.Trigger	0 or 1	Toggle to send Event Command. 0 = Disable, 1 = Enable
MBTCP.CONTROL.EventCommand_DB Data.ClientID	0 to 19	Client ID associated with the command to be executed
MBTCP.CONTROL.EventCommand_DB Data.ServerIPaddress	xxx.xxx.xxx.xxx	IP address of target Modbus server
MBTCP.CONTROL.EventCommand_DB Data.ServicePort	502 or 2000	Service port of target Modbus server
MBTCP.CONTROL.EventCommand_DB Data.SlaveAddress	1 to 255	Slave address of target Modbus TCP/IP to serial device, if applicable
MBTCP.CONTROL.EventCommand_DB Data.InternalDBaddress	0 to 9999 (word-level) or 0 to 159,999* (bit-level)	Specifies the module's internal database register to be associated with the command. For Modbus Function Codes 3, 4, 6, or 16, the allowable range is 0 to 9999. For Modbus Function Codes 1, 2, 5, or 15, the allowable range is 0 to 159,999. **Note**: This bit address range is available with ProSoft Configuration Builder (PCB) v4.6.0.0 or later. Previous versions have a range of 0 to 65535.
MBTCP.CONTROL.EventCommand_DB Data.RegisterCount	1 to 125 (words) or 1 to 2000 (coils)	Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point.
MBTCP.CONTROL.EventCommand_DB Data.SwapCode	0, 1, 2, 3	Specifies if the data received from the Modbus server is to be ordered differently than received from the server 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 server devices.
MBTCP.CONTROL.EventCommand_DB Data.ModbusFC	1, 2, 3, 4, 5, 6, 15, 16	Specifies the Modbus function to be executed by the command.
MBTCP.CONTROL.EventCommand_DB Data.DeviceModbusAddress	0 to 9999	Specifies the register or digital point address offset within the Modbus server device. The MBTCP client reads or writes from/to this address within the server.
MBTCP.CONTROL.EventCommand_DB Data.StatusReturned		0 = Fail 1 = Success -1 = Client is not Enabled and Active
MBTCP.CONTROL.EventCommand_DB Data.CmdInQue		Number of commands in the queue waiting to be executed

4.3.3.3 MBTCP.CONTROL.EventCommand_PLCData

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

Tag Name	Range	Description
MBTCP.CONTROL.EventCommand_PLCData.Trigger	0 or 1	Toggle to send Event Command. 0 = Disable, 1 = Enable
MBTCP.CONTROL.EventCommand_PLCData.ClientID	0 to 19	Client ID associated with the command to be executed
MBTCP.CONTROL.EventCommand_PLCData.ServerIPaddress	xxx.xxx.xxx.xxx	IP address of target Modbus server
MBTCP.CONTROL.EventCommand_PLCData.ServicePort	502 or 2000	Service port of target Modbus server
MBTCP.CONTROL.EventCommand_PLCData.SlaveAddress	1 to 255	Slave address of target Modbus TCP/IP to serial device, for backwards compatibility
MBTCP.CONTROL.EventCommand_PLCData.ModbusFunctionCode	1, 2, 3, 4, 5, 6, 15, 16	Specifies the Modbus function to be executed by the command.
MBTCP.CONTROL.EventCommand_PLCData.DeviceDBAddress	0 to 9999	Specifies the register or digital point address offset within the Modbus server device. The MBTCP client reads or writes from/to this address within the server.
MBTCP.CONTROL.EventCommand_PLCData.PointCount	1 to 125 (words) or 1 to 2000 (coils)	Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point.
MBTCP.CONTROL.EventCommand_PLCData.Data		Data values associated with the command
MBTCP.CONTROL.EventCommand_PLCData.ErrorStatus		Command status after execution

4.3.3.4 MBTCP.CONTROL.EventSequenceCommand

This tag array contains the values needed to build one Modbus TCP/IP command, have it sent to a specific client on the module, and control the processing of the returned response block.

Tag Name	Range	Description
MBTCP.CONTROL.EventSequenceCommand.Trigger	0 or 1	Toggle to send Event Sequence Command. 0 = Disable, 1 = Enable
MBTCP.CONTROL.EventSequenceCommand.ClientID	0 to 19	Client ID associated with the command to be executed
MBTCP.CONTROL.EventSequenceCommand.ServerIPaddress	xxx.xxx.xxx.xxx	IP address of target Modbus server
MBTCP.CONTROL.EventSequenceCommand.ServicePort	502 or 2000	Service port of target Modbus server
MBTCP.CONTROL.EventSequenceCommand.SlaveAddress	1 to 255	Slave address of target Modbus TCP/IP to serial device, if applicable
MBTCP.CONTROL.EventSequenceCommand.InternalDBaddress	0 to 9999 (word-level) or 0 to 159,999 (bit-level)	Specifies the module's internal database register to be associated with the command. For Modbus Function Codes 3, 4, 6, or 16, the allowable range is 0 to 9999. For Modbus Function Codes 1, 2, 5, or 15, the allowable range is 0 to 159,999. **Note**: This bit address range is available with ProSoft Configuration Builder (PCB) v4.6.0.0 or later. Previous versions have a range of 0 to 65535.
MBTCP.CONTROL.EventSequenceCommand.RegisterCount	1 to 125 (words) or 1 to 2000 (coils)	Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point.
MBTCP.CONTROL.EventSequenceCommand.SwapCode	0, 1, 2, 3	Specifies if the data received from the Modbus server is to be ordered differently than received from the server 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 server devices.
MBTCP.CONTROL.EventSequenceCommand.ModbusFC	1, 2, 3, 4, 5, 6, 15, 16	Specifies the Modbus function to be executed by the command.
MBTCP.CONTROL.EventSequenceCommand.DeviceModbusAddress	0 to 9999	Specifies the register or digital point address offset within the Modbus server device. The MBTCP client reads or writes from/to this address within the server.
MBTCP.CONTROL.EventSequenceCommand.SequenceNumber		Event Sequence Command Number
MBTCP.CONTROL.EventSequenceCommand.StatusReturned		Event Sequence Command Returned 0 = Fail 1 = Success -1 = Client disabled /inactive
MBTCP.CONTROL.EventSequenceCommand.CmdInQue		Number of Event Sequence commands in queue

4.3.3.5 MBTCP.CONTROL.Time

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

Tag Name	Range	Description
MBTCP.CONTROL.Time.SetTime	0 or 1	Sends the PLC time to the module 0 = Disable, 1 = Enable
MBTCP.CONTROL.Time.GetTime	0 or 1	Retrieves the time from the module to PLC 0 = Disable, 1 = Enable
MBTCP.CONTROL.Time.Year	0 to 9999	Four digit year value. Example: 2014
MBTCP.CONTROL.Time.Month	1 to 12	Month
MBTCP.CONTROL.Time.Day	1 to 31	Day
MBTCP.CONTROL.Time.Hour	0 to 23	Hour
MBTCP.CONTROL.Time.Minute	0 to 59	Minute
MBTCP.CONTROL.Time.Second	0 to 59	Second
MBTCP.CONTROL.Time.Milliseconds	0 to 999	Millisecond
MBTCP.CONTROL.Time.Error	0 or -1	0 = OK -1 = Error present

4.3.3.6 MBTCP.CONTROL.ClientServerControl

This array allows the control and retrieval of driver command active bits.

Tag Name	Range	Description
MBTCP.CONTROL.ClientServerControl.Trigger	0 or 1	Toggle client/server control 0 = Disable, 1 = Enable
MBTCP.CONTROL.ClientServerControl.ActiveServer	0 or 1	Server active state: 0 = Disable, 1 = Enable
MBTCP.CONTROL.ClientServerControl.ActiveClient_0to15		Client 0 to 15 bit map for active status of clients
MBTCP.CONTROL.ClientServerControl.ActiveClient_16to19		Client 16 to 19 bit map for active status of clients
MBTCP.CONTROL.ClientServerControl.ActiveClientCmd[x]	0 or 1	Client 0 to 19 command active bits. One word for each client. Each bit is a command. 0 = Disable, 1 = Enable
MBTCP.CONTROL.ClientServerControl.GetStatus	0 or 1	Toggle request for status 0 = Disable, 1 = Enable
MBTCP.CONTROL.ClientServerControl.ServerStatus	0 or 1	Server active state 0 = Disabled, 1 = Enabled
MBTCP.CONTROL.ClientServerControl.Client_0to15Status		Client 0 to 15 bit map for active status of clients
MBTCP.CONTROL.ClientServerControl.Client_16to19Status		Client 16 to 19 bit map for active status of clients
MBTCP.CONTROL.ClientServerControl.ClientCmdStatus[x]	0 or 1	Clients 0 to 19 command active bits. One word for each client. Each bit is a command. 0 = Disabled, 1 = Enabled

4.3.3.7 MBTCP.CONTROL.ResetStatus

This array resets the module along with client and server status tags.

Tag Name	Range	Description
MBTCP.CONTROL.ResetStatus.Trigger	0 or 1	Toggle reset control 0 = Disable, 1 = Enable
MBTCP.CONTROL.ResetStatus.Module		Reset Module status (0 = No, else yes with any non-zero value)
MBTCP.CONTROL.ResetStatus.Server		Reset server status (0 = No, else yes with any non-zero value)
MBTCP.CONTROL.ResetStatus.Client		Reset client status (0 = No, else yes with any non-zero value)

4.3.3.8 MBTCP.CONTROL.EventSequenceCounts

This tag triggers the counting of the event sequence operation.

Tag Name	Range	Description
MBTCP.CONTROL.EventSequenceCounts	0 or 1	Triggers the counting of event sequence 0 = Disable, 1 = Enable

4.3.3.9 MBTCP.CONTROL.EventSequenceStatus

This tag triggers the request for the event sequence status.

Tag Name	Range	Description
MBTCP.CONTROL.EventSequenceStatus	0 or 1	Triggers event sequence status read 0 = Disable, 1 = Enable

4.3.3.10 MBTCP.CONTROL.GetGeneralStatus

This tag triggers the request for the general status of the module.

Tag Name	Range	Description
MBTCP.CONTROL.GetGeneralStatus	0 or 1	Triggers general status read 0 = Disable, 1 = Enable

4.3.3.11 MBTCP.CONTROL.GetEventDataStatus

This tag triggers the request of the event status.

Tag Name	Range	Description
MBTCP.CONTROL.GetEventDataStatus	0 or 1	Triggers event status read 0 = Disable, 1 = Enable

4.3.3.12 MBTCP.CONTROL. ColdBoot

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

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

4.3.3.13 MBTCP.CONTROL.WarmBoot

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

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

4.3.4 MBTCP.STATUS

This array contains the status information of the module.

4.3.4.1 MBTCP.STATUS.Block

This array contains block status.

Tag Name	Description
MBTCP.STATUS.Block.Read	Total number of read blocks transferred from the module to the processor
MBTCP.STATUS.Block.Write	Total number of write blocks transferred from the processor to the module
MBTCP.STATUS.Block.Parse	Total number of blocks successfully parsed that were received from the processor
MBTCP.STATUS.Block.Event	Total number of event command blocks received from the processor
MBTCP.STATUS.Block.Cmd	Total number of command blocks received from the processor
MBTCP.STATUS.Block.Err	Total number of block transfer errors recognized by the module

4.3.4.2 MBTCP.STATUS.ClientStatus

This array contains the status of a specific MBTCP client (0 to 19).

Tag Name	Description
MBTCP.STATUS.ClientStatus.Request	Initiates request for Client Status block from module when set to 1
MBTCP.STATUS.ClientStatus.ClientID	Specifies Client (0 to 19) to request status data from
MBTCP.STATUS.ClientStatus.CommandRequests	Total number of requests made from this port to server devices on the network
MBTCP.STATUS.ClientStatus.CommandResponses	Total number of server response messages received on the port
MBTCP.STATUS.ClientStatus.CommandErrors	Total number of command errors processed on the port. These errors could be due to a bad response or command
MBTCP.STATUS.ClientStatus.Requests	Total number of messages sent out of the port
MBTCP.STATUS.ClientStatus.Responses	Total number of messages received on the port
MBTCP.STATUS.ClientStatus.ErrorsReceived	Total number of message errors received on the port
MBTCP.STATUS.ClientStatus.ErrorsSent	Total number of message errors sent out of the port
MBTCP.STATUS.ClientStatus.CurrentError	Most recent error code recorded for the client
MBTCP.STATUS.ClientStatus.LastError	Previous most recent error code recorded for the client
MBTCP.STATUS.ClientStatus.CmdErrors[x]	Command error code for each command (0-15) on the specified client's command list

4.3.4.3 MBTCP.STATUS.EventSeqStatus

This array contains the status of the event command queue.

Tag Name	Description
MBTCP.STATUS.EventSeqStatus.ClientID	Specifies Client (0 to19) to request event status data from
MBTCP.STATUS.EventSeqStatus.MessageCount	Number of event sequence messages in block (0 to 15)
MBTCP.STATUS.EventSeqStatus.SeqNum_RetErrCode[x]	Sequence number returned error code

4.3.4.4 MBTCP.STATUS.EventSeqCounts

This array indicates the number of commands waiting in the command queue.

Tag Name	Description
MBTCP.STATUS.EventSeqCounts.ClientCmdCount_EventSeqMessage	Event command quantity waiting in queue. There are two bytes of status data per client. See below for details.

Byte 1: Number of Event sequence commands for which status has not yet been retrieved (up to 15). This corresponds to the *MNETC.STATUS.EventSeqCmdPending.Client[x]_QueueCount* controller tag.

Byte 2: Total number of commands waiting in the command queue. This includes Event Commands, Event Commands with Sequence Numbers, and Command Control messages. This corresponds to the *MBTCP.STATUS.EventSeqStatus.MessageCount* controller tag.

4.3.4.5 MBTCP.STATUS.GeneralStatus

This array contains the general status of the module including firmware revision and general communication status.

Tag Name	Description
MBTCP.STATUS.GeneralStatus.ExpectedWriteBlock	Contains the next write block ID number
MBTCP.STATUS.GeneralStatus.ProgramScanCount	Program cycle counter – increments each time a complete program cycle occurs in the module
MBTCP.STATUS.GeneralStatus.ProductCode	Product code
MBTCP.STATUS.GeneralStatus.ProductVersion	Firmware revision level number
MBTCP.STATUS.GeneralStatus.OperatingSystem	Operating level number
MBTCP.STATUS.GeneralStatus.RunNumber	Run number
MBTCP.STATUS.GeneralStatus.ReadBlockCount	Total number of read blocks transferred from the module to the processor
MBTCP.STATUS.GeneralStatus.WriteBlockCount	Total number of write blocks transferred from the processor to the module
MBTCP.STATUS.GeneralStatus.ParseBlockCount	Total number of blocks successfully parsed that were received from the processor
MBTCP.STATUS.GeneralStatus.CmdEventBlockCount	Total number of event command blocks received from the processor
MBTCP.STATUS.GeneralStatus.CmdBlockCount	Total number of command blocks received from the processor
MBTCP.STATUS.GeneralStatus.ErrorBlockCount	Total number of block transfer errors recognized by the module
MBTCP.STATUS.GeneralStatus.Client0CmdExecutionWord	Each bit in this word is used to enable/disable the commands for client 0. 0 = Disable, 1 = Enable
MBTCP.STATUS.GeneralStatus.Client1to19CmdExecutionWord	Each bit in each of the 19 words is used to enable/disable the commands for clients 1 to 19. 0 = Disable, 1 = Enable
MBTCP.STATUS.GeneralStatus.EventSeqReady	Bit mapped (1 bit per client 0 to 19) Bit = 0, no event sequence status data ready Bit = 1, event sequence status data ready
MBTCP.STATUS.GeneralStatus.MNetRequestCount	Increments each time an encapsulated Modbus TCP/IP (Service port 2000) request is received.
MBTCP.STATUS.GeneralStatus.MNetResponseCount	Increments each time an encapsulated Modbus TCP/IP (Service port 2000) response message is sent.
MBTCP.STATUS.GeneralStatus.MnetErrorSent	Increments each time an error is sent from a server on service port 2000.
MBTCP.STATUS.GeneralStatus.MNETErrorReceived	Increments each time an error is received from a server on service port 2000.
MBTCP.STATUS.GeneralStatus.MBAPRequestCount	Increments each time a MBAP (Service port 502) request is received.
MBTCP.STATUS.GeneralStatus.MBAPResponseCount	Increments each time a MBAP (Service port 502) response message is sent.
MBTCP.STATUS.GeneralStatus.MBAPErrorSent	Increments each time an error is sent from the server on service port 502.
MBTCP.STATUS.GeneralStatus.MBAPErrorReceived	Increments each time an error is received from a server on service port 502.

4.3.4.6 MBTCP.STATUS.GetEventDataStatus

This array contains the status of the event command last executed.

Tag Name	Description
MBTCP.STATUS.GetEventDataStatus .ClientRecordsCount	Number of clients contained in block (0 to 19)
MBTCP.STATUS.GetEventDataStatus .Status	Two words per client. Word 1 = Client (0 to19) Word 2 = Error code for last executed command for corresponding client

4.3.5 MBTCP.UTIL

The array is used for internal ladder processing. It should not be modified.

Tag Name	Description
MBTCP.UTIL.ReadDataSizeGet	Holds Read Data array size
MBTCP.UTIL.WriteDataSizeGet	Holds Write Data array size
MBTCP.UTIL.ReadDataBlkCount	Number of Read Data blocks – this value is the *Read Register Count* divided by the Block Transfer Size
MBTCP.UTIL.WriteDataBlkCount	Number of Write Data blocks – this value is the *Write Register Count* divided by the Block Transfer Size
MBTCP.UTIL.RBTSremainder	Remainder from the *Read Register Count* divided by the Block Transfer Size
MBTCP.UTIL.WBTSremainder	Remainder from the *Write Register Count* divided by the Block Transfer Size
MBTCP.UTIL.BlockIndex	Computed block offset for data
MBTCP.UTIL.LastRead	Latest Read Block ID received from the module
MBTCP.UTIL.LastWrite	Latest Write Block ID to be sent to the module
MBTCP.UTIL.LastWriteInit	Latest Write Block ID used during initialization
MBTCP.UTIL.ConfigFile [ ] Array	Holds variables for configuration file transfer
MBTCP.UTIL.ConfigFile. WordLength	Length of configuration data to be included in block transfer
MBTCP.UTIL.ConfigFile. BlockCount	Block transfer count for transferring the whole configuration file from PLC to the Module
MBTCP.UTIL.ConfigFile.FileOffset	Offset in configuration file to use as a starting point for copying over configuration data
MBTCP.UTIL.ConnectionInputSize	Holds size of the Connection Input array
MBTCP.UTIL.BlockTransferSize	Size of the backplane transfer blocks
MBTCP.UTIL.SlotNumber	Slot number of the module in the rack
MBTCP.UTIL.CommandControl Pending	Waiting for response from module
MBTCP.UTIL. CommandControlWriteBlockID	Block ID for Command Control
MBTCP.UTIL. EventCommandDBDataPending	Keeps an Event Command with Data message from being sent to the module before the previous Event Command with Data is completed
MBTCP.UTIL. EventCmd_DBDataBlockID	Block ID of last read block
MBTCP.UTIL.EventCmd_ DBDataWriteEventBlockID	Event response write block ID.
MBTCP.UTIL.EventCmd_ ProcessorDataPending	Event Command Processor Data Pending 0 = Yes, 1 = No
MBTCP.UTIL.EventCmd_ ProcessorDataBlockID	Event Command processor data block ID
MBTCP.UTIL. EventSeqCmdPending	Event Sequence Command Pending 0 = Yes, 1 = No
MBTCP.UTIL. EventSeqCmdBlockID	Event Sequence Command Block ID
MBTCP.UTIL. EventSeqCmdWriteEventBlockID	Event Sequence Command Write Event Block ID
MBTCP.UTIL.PassThrough. MBControlx [ ] Array	Holds variables used for processing Pass-Through messages
MBTCP.UTIL. ClientServerControlBlockID	Client and Server Control Block ID

MBTCP.UTIL.ClientStatusPending	Client Status Pending 0 = Yes, 1 = No
MBTCP.UTIL. ClientStatusWriteBlockID	Client Status Write Block ID
MBTCP.UTIL. EventSeqStatusPending	Event Sequence Status Pending 0 = Yes, 1 = No
MBTCP.UTIL. EventSeqStatusWriteBlockID	Event Sequence Status Write Block ID
MBTCP.UTIL. EventSeqCountsWriteBlockID	Event Sequence Counts Write Block ID
MBTCP.UTIL. EventSeqCountsPending	Event Sequence Counts Pending 0 = Yes, 1 = No
MBTCP.UTIL.TimeWriteBlockID	Time Write Block ID
MBTCP.UTIL. ResetStatusWriteBlockID	Reset Status Write Block ID
MBTCP.UTIL. GetEventDataStatusBlockID	Get Event Data Status Block ID

5 MVI69E-MBTCP Backplane Data Exchange

5.1 General Concepts of the MVI69E-MBTCP Data Transfer

The MVI69E-MBTCP 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 MVI69E-MBTCP module. This is the order of operations on power-up:

1. The module begins the following logical functions:
   - Initialize hardware components
   - Initialize CompactLogix backplane driver
   - 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 TCP/IP Ethernet port

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

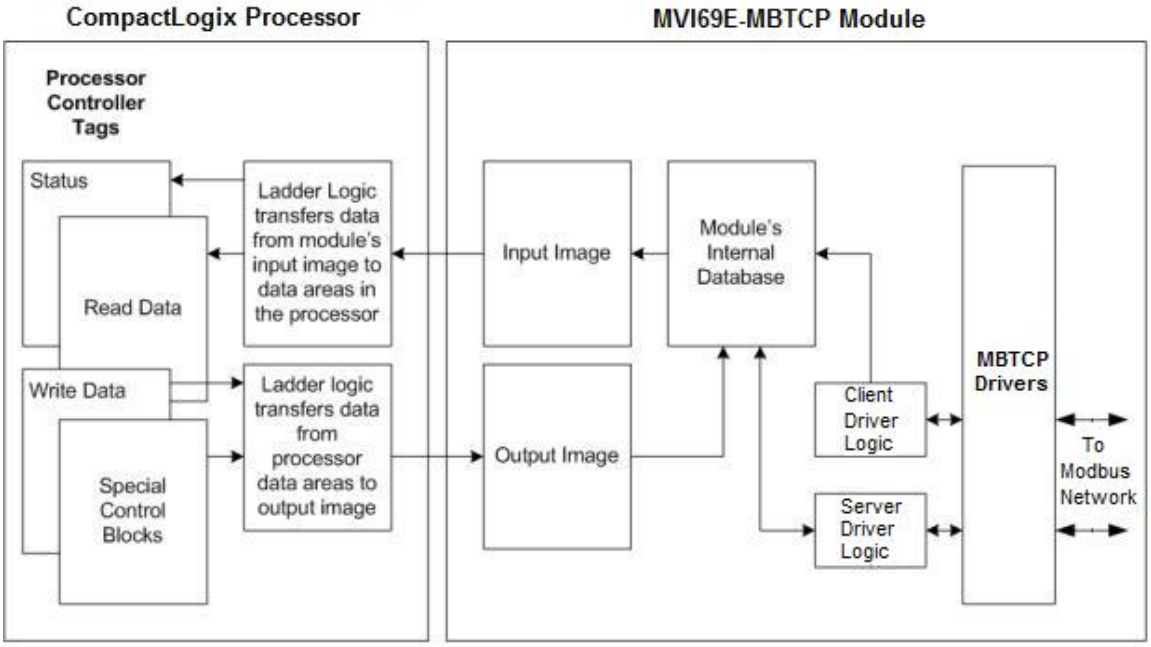
5.2 Backplane Data Transfer

The MVI69E-MBTCP 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 may be set to 62, 122, or 242 words depending on the block transfer size parameter set in the configuration file. This data area permits fast throughput of data between the module and the processor.

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

The following illustration shows the data transfer method used to move data between the CompactLogix processor, the MVI69E-MBTCP 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 MBTCP data tables with addresses from 0 to the maximum number of points for each data type.

5.3 Normal Data Transfer

Normal data transfer includes the paging of the user data found in the module's internal database (Registers 0 to 9999) and the status data. These data are transferred through read (input image) and write (output image) blocks. The following topics describe the structure and function of each block.

5.3.1 Write Block: Request from the Processor to the Module

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

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

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

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

5.3.2 Read Block: Response from the Module to the Processor

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

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

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

5.3.3 Read and Write Block Transfer Sequences

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

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

R1W1 → R2W2 → R3W1 → R1W2 → R2W1 → R3W2 → R1W1 →

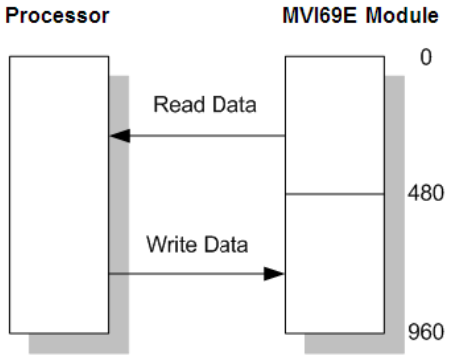
This sequence continues until interrupted by other write block numbers sent by the controller or by a command request from a node on the Modbus network or operator control through the module's Configuration/Debug port.

The following example shows a typical backplane communication application.

If the backplane parameters are configured as follows:

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

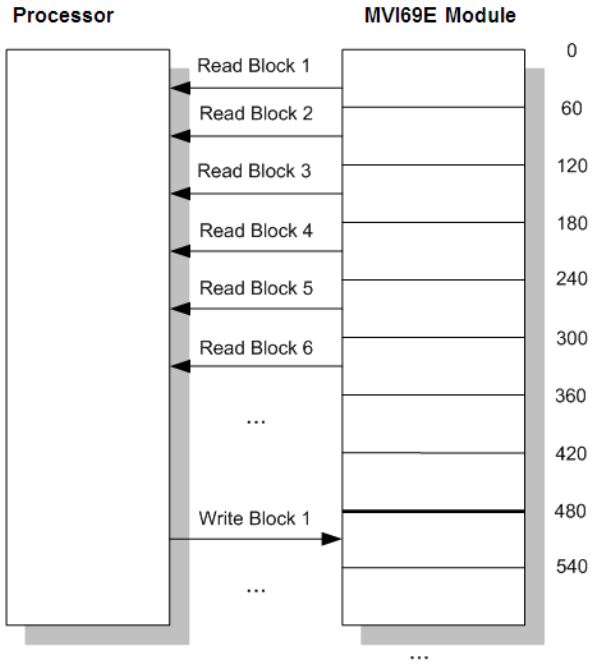
The backplane communication would be configured as follows:



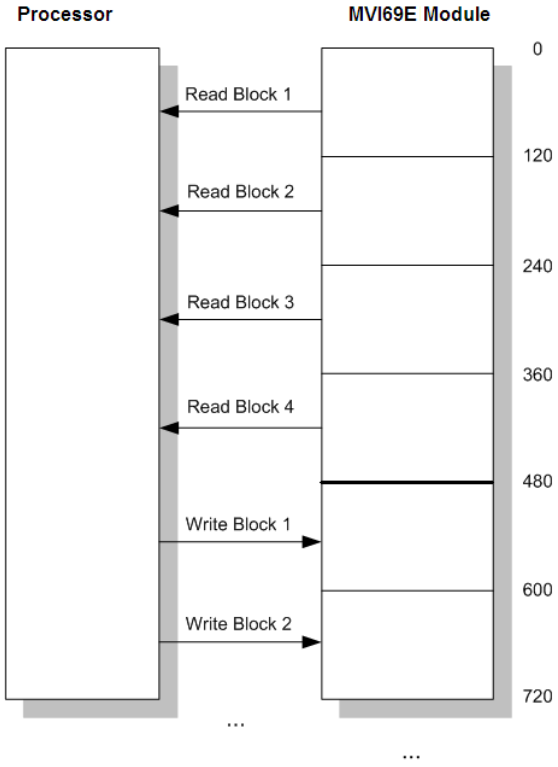
Database address 0 to 479 is continuously transferred from the module to the processor. Database address 480 to 959 is continuously transferred from the processor to the module.

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

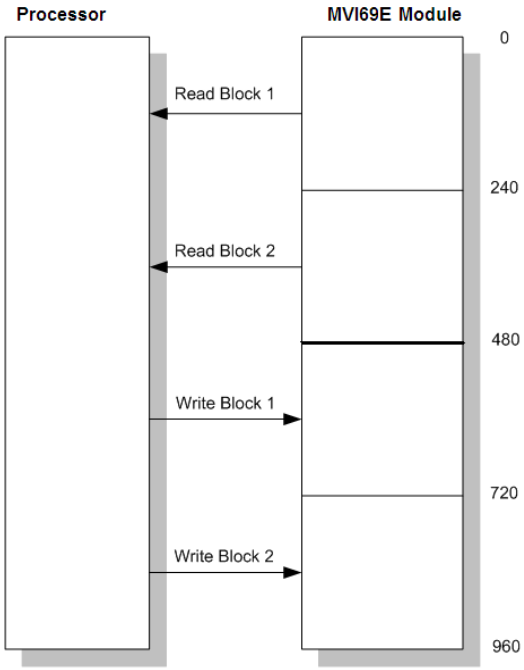
5.3.3.1 If Block Transfer Size = 60



5.3.3.2 If Block Transfer Size = 120



5.3.3.3 If Block Transfer Size = 240

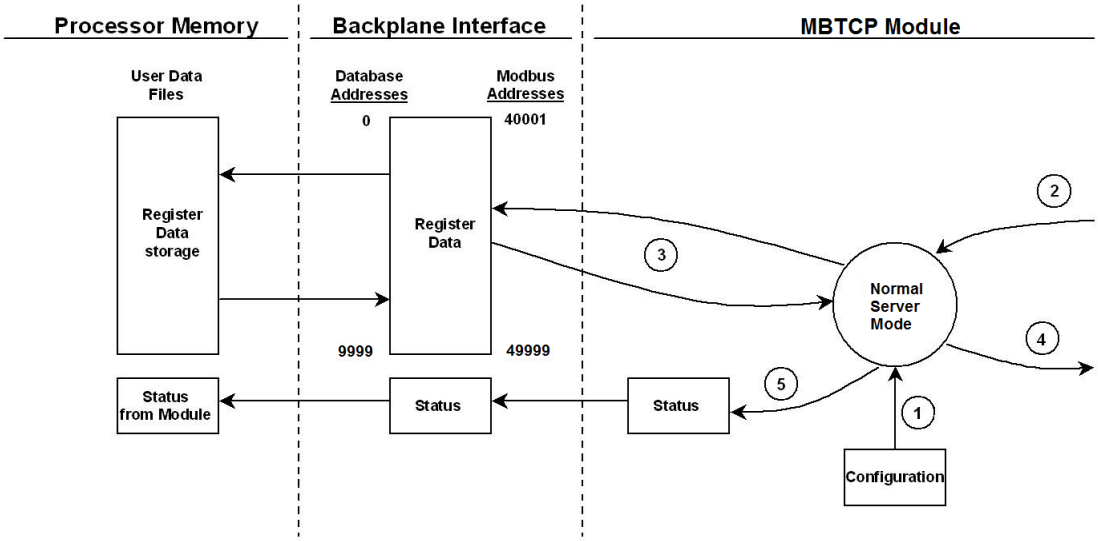


5.4 Data Flow Between the Module and Processor

The following topics describe the flow of data between the two pieces of hardware (CompactLogix processor and MVI69E-MBTCP module) and other nodes on the Modbus network. The module can act as a Modbus TCP/IP client (master), server (slave), or both simultaneously.

5.4.1 Server Mode

In Server driver mode, the MVI69E-MBTCP module responds to read and write commands issued by a client on the Modbus network. The following diagram shows the data flow for normal Server mode.

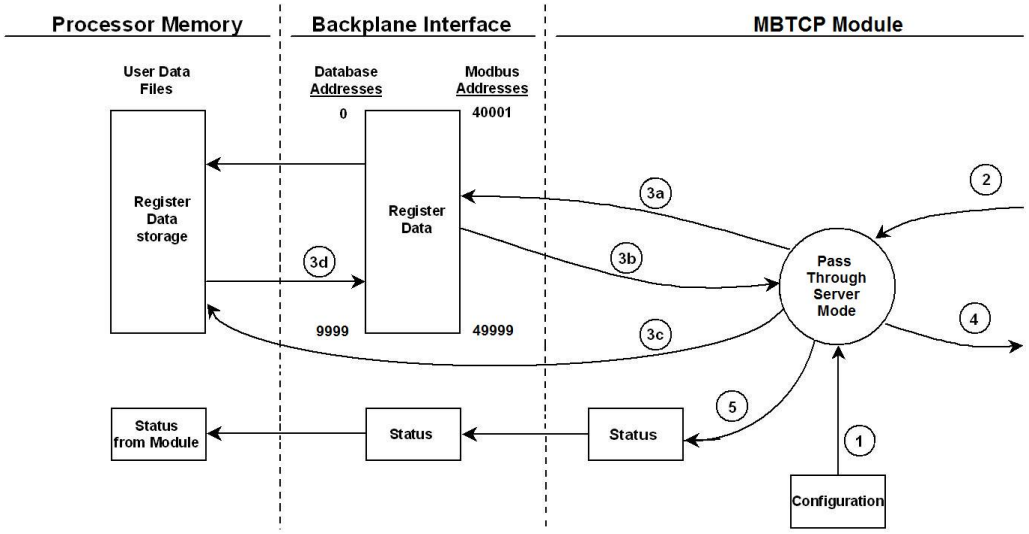


Step	Description
1	Any time the module restarts (boots or reboots), the server port driver receives configuration information from the MBTCP controller tags. This information configures the Ethernet port and defines Server driver characteristics. The configuration information may also contain instructions to offset data stored in the database to addresses different from addresses requested in the received messages.
2	A Modbus client device, such as a Modicon PLC or an HMI application, issues a read or write command to the module's IP address. The Server 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 client.
5	Counters are available in the Status Block to permit the ladder logic program to determine the level of activity of the Server driver.

In Server Pass-Through mode, write commands from the client are handled differently than they are in Normal mode. In Server 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 client 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 server port with Pass-Through enabled:



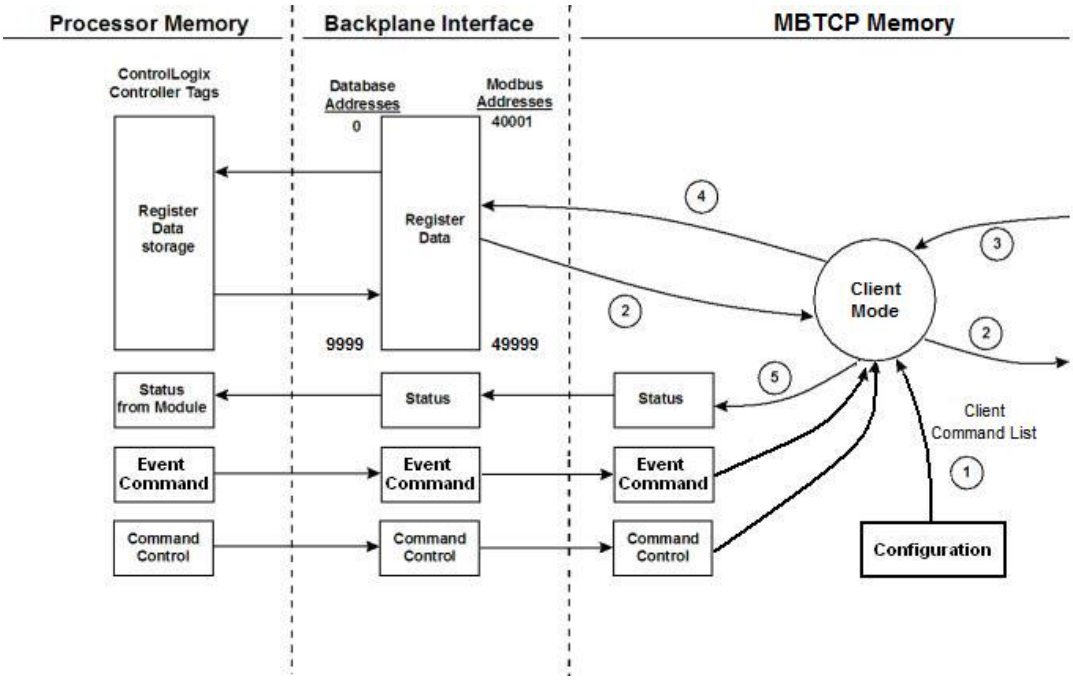
Step	Description
1	Same as normal mode.
2	Same as normal mode.
3	a. In Pass-Through mode, if the Server driver receives a read request, it looks for the data in module's internal database, just as it would in Normal mode. b. The data needed to respond to the read command is retrieved directly from the internal database and returned to the Server driver so it can build a response message. c. In Pass-Through mode, if the Server 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.

5.4.2 Master Mode

In Client mode, the MVI69E-MBTCP module issues read or write commands to server devices on the Modbus network. You configure these commands in ProSoft Configuration Builder in the Client Command List. 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 Client driver obtains configuration data from the MBTCP controller tags. The configuration data retrieved includes Ethernet configuration and the Client Command List. Special Commands can be issued directly from the CompactLogix processor using Event Commands and Command Control. The Client driver uses these command values to determine the types and order of commands to send to server on the network.
2	After configuration, the Client driver begins transmitting read and/or write commands to server nodes on the network. If the Client driver is writing data to a server, the data for the write command is retrieved from the module's internal database.
3	Once the specified server has successfully processed the command, it returns a response message to the Client driver for processing.
4	Data received from a server 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 Client 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.

5.4.2.1 Client Command List

You can define up to 10 Modbus TCP/IP client connections in the MVI69E-MBTCP. Each client connection can contain up to 16 commands each.

A valid command includes the following items:

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

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

- Server IP Address.
- Modbus Service Port of the server.
- Modbus Function Code: This is the type of command that is issued.
- Source or destination address in the server device.

5.4.2.2 Command Error Codes

As the list is read in from the processor and as the commands are processed, an error value is maintained in the module for each command. The definition for these command error codes is listed in *Communication Error Codes* (page 116). You can view the command error codes through the Ethernet diagnostics port; refer to *Diagnostics and Troubleshooting* (page 103). 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.

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. Each MBTCP client must place their own Command Error List within the Read Data area so that they do not overlap each other.

6 Legacy Mode

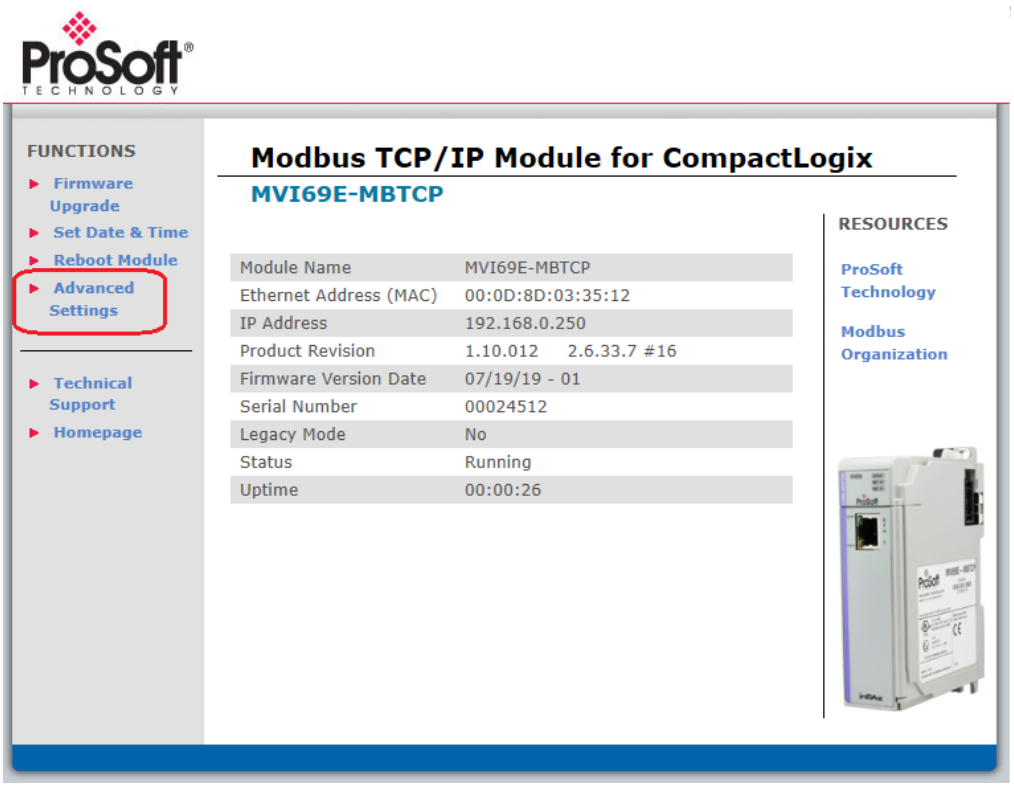
Legacy Mode allows you to replace an existing MVI69-MNET module with the MVI69E-MBTCP. This feature is only supported with MVI69E-MBTCP firmware version 1.11.001 or later.

The MVI69E-MBTCP module in Legacy Mode is backward compatible with the legacy MVI69-MNET. This means that you may replace the MVI69-MNET with the MVI69E-MBTCP module in LEGACY mode without any changes to the existing CompactLogix ladder logic application.

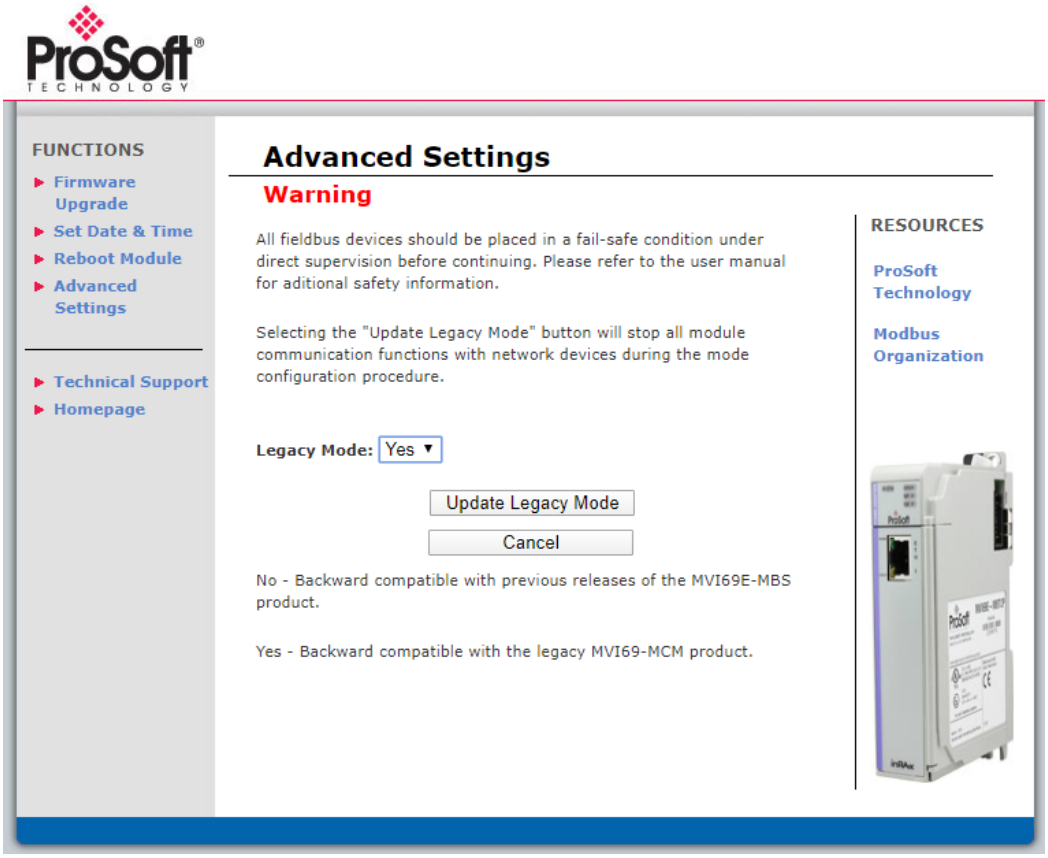
The existing user may also convert the existing MVI69-MNET PCB configuration to the MVI69E-MBTCP module in Legacy Mode. This conversion procedure is supported by PCB version 4.4.24.20.0302 or later.

6.1 Legacy Mode Configuration

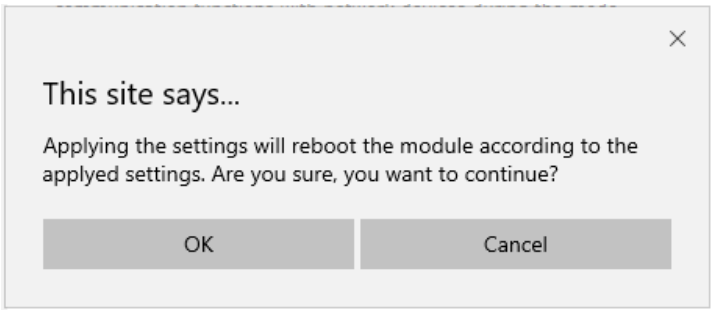
1. Open the MVI69E-MBTCP webpage. For further information, please see *Connecting to the MVI69E-MBTCP Webpage* on page 117.
2. Click on the *Advanced Settings* option.



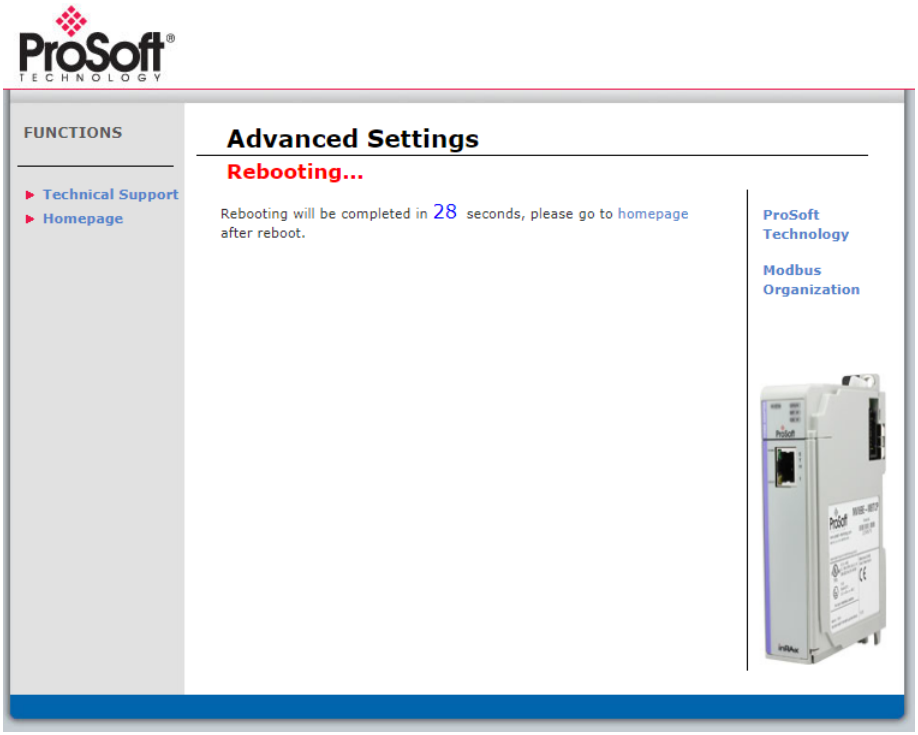
3 In the *Advanced Settings* page, change the **LEGACY MODE** field to 'Yes', then click on the **UPDATE LEGACY MODE** button.



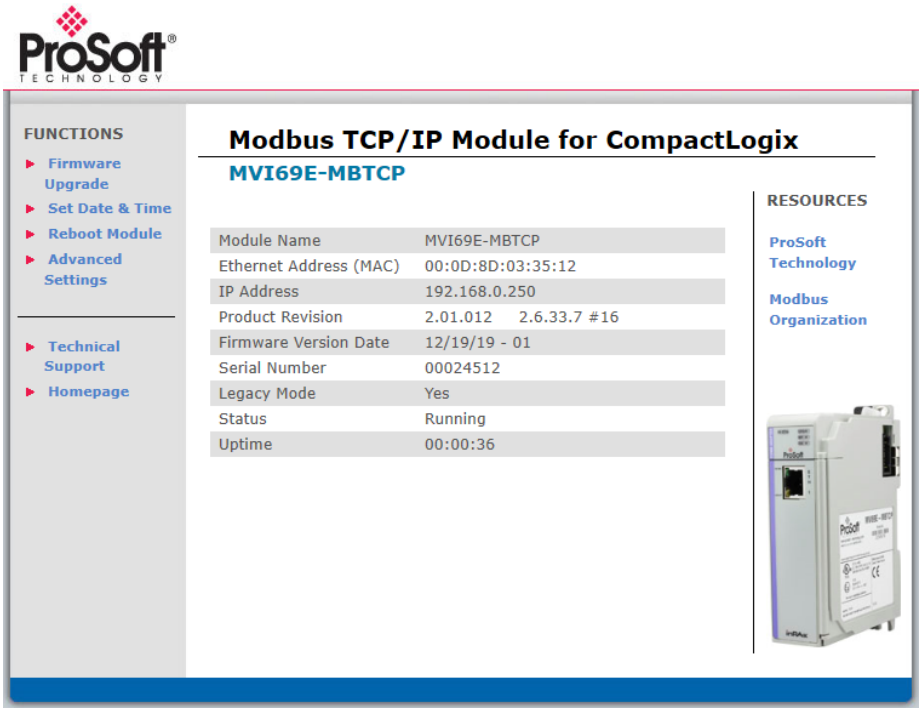
4 Confirm the update by clicking **OK**.



5 The module will reboot during the update process.



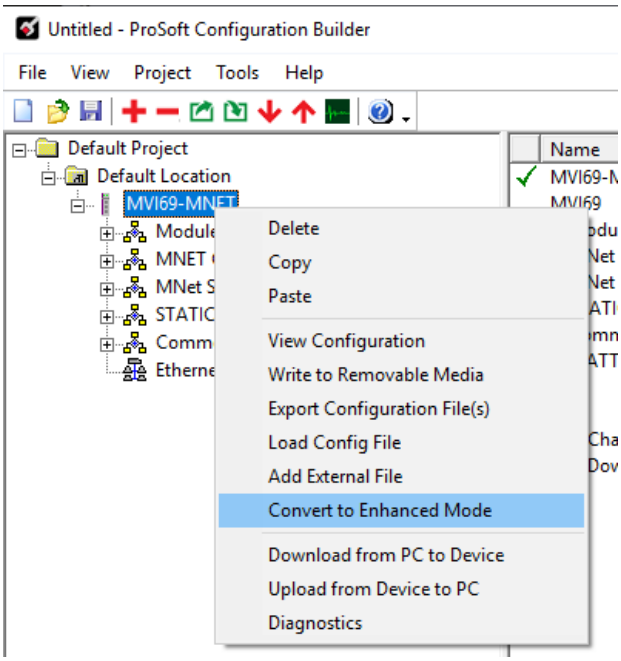
6 Once complete, the homepage displays **Legacy Mode – Yes**.



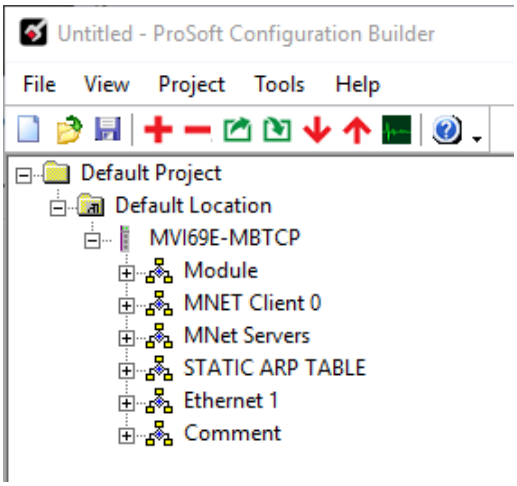
6.2 PCB Configuration

Convert the existing 'MVI69-MNET' PCB project to an 'MVI69E-MBTCP LEGACY' project.

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



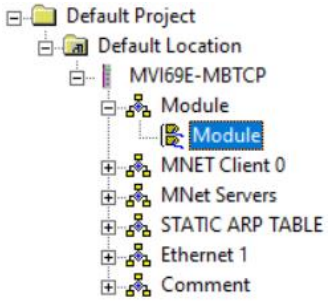
3. After the conversion, the PCB module parameters are updated.



6.2.1 Module

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

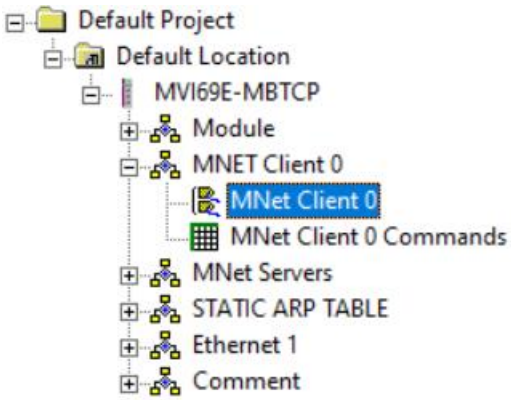
In the ProSoft Configuration Builder (PCB) tree view, double-click on the **MODULE** icon.



Parameter	Value	Description
Error/Status Pointer	-1 to 9955	The starting MVI69E-MBTCP database location to store server error/status data. If a value of -1 is entered, the error/status data will not be used. This feature returns 8 server error/status values. The descriptions of the values start at the *MBTCP.STATUS.GeneralStatus.MNETRequestCount* controller tag. Refer to the General Status description on page 67 for more information.
Read Register Start	0 to 9999	Specifies the start of the Read Data area in module memory. Data in this area is transferred from the module to the processor.
Read Register Count	0 to 10000	Specifies the size of the Read Data area.
Write Register Start	0 to 9999	Specifies the start of the Write Data area in module memory. Data in this area is transferred from the processor to the module.
Write Register Count	0 to 10000	Specifies the size of the Write Data area.
Failure Flag Count	0 to 65535	Specifies the number of consecutive backplane transfer failures that can occur before communications are halted. 0 = Ignore >0 = Failure count to disable
Block Transfer Size	60, 120 or 240	Specifies the number of words in each block transferred between the module and processor.
Initialize Output Data	Yes or No	This parameter determines if the input image data and the module's Read Register Data values are initialized with Read Register Data values from the processor. If set to **No**, the Read Register Data values in the module are set to 0 upon initialization. If set to **Yes**, the data is initialized with Read Register Data values from the processor. This option requires associated ladder logic to pass the data from the processor to the module.
Pass-Through Mode	0, 1, 2, or 3	Handling of write request messages on server ports: **0** = Store data directly in internal database. **1** = Store received message in unformatted block for processor. **2** = Store received data in formatted block for processor after swapping bytes. **3** = Store received data in formatted block for processor.
Duplex/Speed Code	0, 1, 2, 3, or 4	**0** = Auto-negotiate **1** = 10MB/half-duplex **2** = 10MB/full-duplex **3** = 100MB/half-duplex **4** = 100MB/full-duplex

6.2.2 Client 0

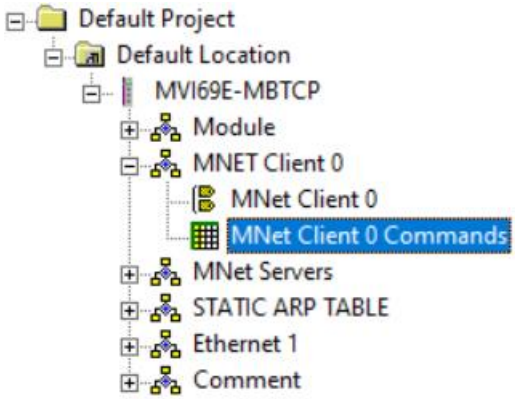
This section defines the Modbus TCP/IP Client driver. In the ProSoft Configuration Builder tree view, double-click the *MNET Client 0* icon.



Parameter	Value	Description
Error/Status Pointer	-1 to 9990	The starting MVI69E-MBTCP database location to store Client x's error/status data. If a value of -1 is entered, the error/status data will not be placed in the database. This feature returns 8 Client x error/status data values. The descriptions of these values start at the *MBTCP.STATUS.ClientStatus.CommandRequests* controller tag. Refer to the Client Status description on page 65 for more information.
Command Error Pointer	0 to 9999	Internal DB location to place command error list
Minimum Command Delay	0 to 32767 ms	Specifies the number of milliseconds to wait between commands. This helps avoid sending commands on the network faster than the servers can receive them.
Response Timeout	0 to 65535 ms	Specifies the time that the client waits for a response from the addressed server before re-transmitting the command (*Retry Count*) or skipping to the next command in the Command List.
Retry Count	0 to 10	Specifies the number of times a command is retried if it fails.
Enron-Daniels	Yes or No	Use Floating point data offset.
ARP Timeout	1 to 60 sec	Specifies the minimum time that the module will wait for an ARP response from a node. During that time, the module will not communicate with any nodes. If the module does not receive an ARP response within this period, it will proceed communicating with other nodes for 30 seconds until the next ARP request attempt.
Command Error Delay	0 to 300	Number of 0.1 second intervals to wait after command error.

Note: In Legacy Mode, the *Enabled*, *Start/Active*, *MBAP Port Override* parameters are present.

6.2.3 Client 0 Commands



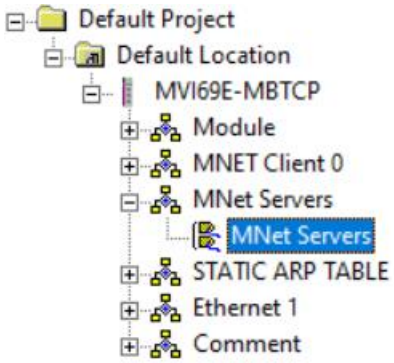
Parameter	Value	Description
Enable	0, 1, 2	This field defines whether the command is to be executed under certain conditions. **0** = The command is disabled and is not executed in the normal polling sequence. **1** = The command is executed each scan of the command list if the *Poll Interval* (see below) is set to zero. If the Poll *Interval* is set to a nonzero value, the command is executed when the interval timer expires. **2** = For write commands only. The command executes only if the internal data associated with the command changes.
Internal Address	0 to 9999 (word-level) **or** 0 to 159,999 (bit-level)	Specifies the module's internal database register to be associated with the command. For Modbus Function Codes 3, 4, 6, or 16, the allowable range is 0 to 9999. For Modbus Function Codes 1, 2, 5, or 15, the allowable range is 0 to 159,999. Note: This bit address range is available with ProSoft Configuration Builder (PCB) v4.6.0.0 or later. Previous versions have a range of 0 to 65535. 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 *MBTCP.DATA.ReadData[10]* controller tag in Studio 5000. Think of it as the 160th bit of MBTCP internal memory (MBTCP Internal register 10 * 16 bits per register = 160). Use this formula for function codes 5 or 15 for writing bits also. This controller tag is a 16bit signed integer. This means you can only enter values of -32768 to 32767 in the tag. If a value to be entered is above the 32767 (but below 65535) threshold, it displays as a negative value in the tag. Simply subtract 65536 from the value to get the 'acceptable' value to enter the tag. **Example:** You need to use an Internal bit Address of 48000, but you cannot enter '48000' into the tag because it causes an error. 48000 - 65536 = -17536 You will enter '-17536' in the Internal Address parameter for this command.
Poll Interval	0 to 65535 (1/10 second)	Specifies the minimum interval between executions of continuous commands (*Enable* code = 1). Example: The parameter is entered in 1/10th of a second. Therefore, if a value of 100 is entered, the command executes no more frequently than every 10 seconds. When the command reaches the top of the command queue and 10 seconds has not elapsed, it is skipped until the poll interval has expired.
Register Count	1 to 125 (words) or 1 to 2000 (coils)	Specifies the number of registers or digital points to be associated with the command. Modbus Function Codes 5 and 6 ignore this field as they only apply to a single data point. For Modbus Function Codes 1, 2 and 15, this parameter sets the number of single bit digital points (inputs or coils) to be associated with the command. Note: Up to 2000 coils are supported for Modbus Function Codes 1 and 2. Up to 1968 coils are supported for Modbus Function Code 15. For Modbus Function Codes 3, 4 and 16, this parameter sets the number of 16-bit registers to be associated with the command.
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. **0 = No Change**; No change is made in the byte ordering (ABCD = ABCD) **1 = Word Swap**; The words are swapped (ABCD= CDAB) **2 = Word and Byte Swap**; The words are swapped, then the bytes in each word are swapped (ABCD=DCBA) **3 = Byte Swap**; 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 are 16-bit register Ex: ABCD.
Node IP Address	1 to 255 (0 = Broadcast)	Specifies the Modbus server IP 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.

Service Port	502 (default)	Service Port of the Modbus Server to be considered. 502 or other supported ports on server command. Use a value of '502' when addressing Modbus TCP/IP servers which are compatible with the Schneider Electric MBAP specifications (this will be most devices). If a server implementation supports another service port, enter the value here.
Slave Address	1 to 255	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 the value is set to zero, the command will be a broadcast message on the network.
Modbus Function	1, 2, 3, 4, 5, 6, 15, 16	Specifies the Modbus function to be executed by the command. These function codes are defined in the Modbus protocol. 1 – Read Coil Status (0xxxx) 2 – Read Input Status (1xxxx) 3 – Read Holding Registers (4xxxx) 4 – Read Input Registers (3xxxx) 5 – Force (Write Single) Coil (0xxxx) 6 – Force (Write Single) Holding Register (4xxxx) 15 – Preset (Write) Multiple Coils (0xxxx) 16 – Preset (Write) Multiple Registers (4xxxx)
MB Address in Device	0 to 65535	Specifies the register or digital point address offset within the Modbus slave device. The MBTCP Client 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).

6.2.4 Servers

This parameter is defined in *MBTCP Servers* on page 39.

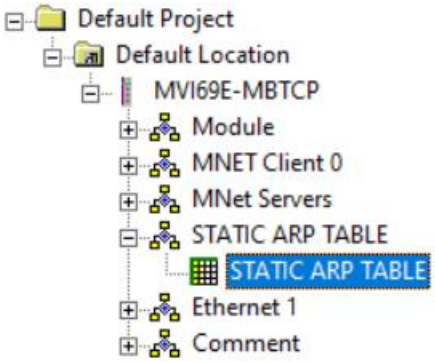


Parameter	Value	Description
Enron-Daniels	Yes or No	Use Floating point data offset.
Output Offset	0 to 9999	Specifies the offset address within module memory that is to be used with network requests for Modbus function codes 1, 5, or 15.
Bit Input Offset	0 to 9999	Specifies the offset address within module memory that is to be used with network requests for Modbus function code 2.
Holding Register Offset	0 to 9999	Specifies the offset address within module memory that is to be used with network requests for Modbus function codes 3, 6, or 16.
Word Input Offset	0 to 9999	Specifies the offset address within module memory that is to be used with network requests for Modbus function code 4.
Connection Timeout	0 to 1200 sec	Server will timeout if it does not receive any new data within the specified amount of time.

Note: In Legacy Mode, the *Start Active* and *Pass-Through Mode* parameters are present.

6.2.5 STATIC ARP TABLE

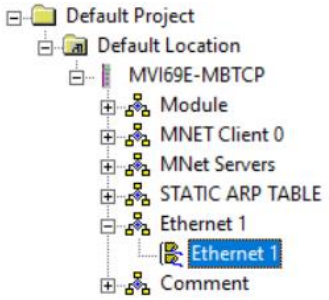
This table contains a list of static IP/MAC addresses that the module will utilized when an ARP is required. The module will accept up to 40 static IP/MAC address data sets.

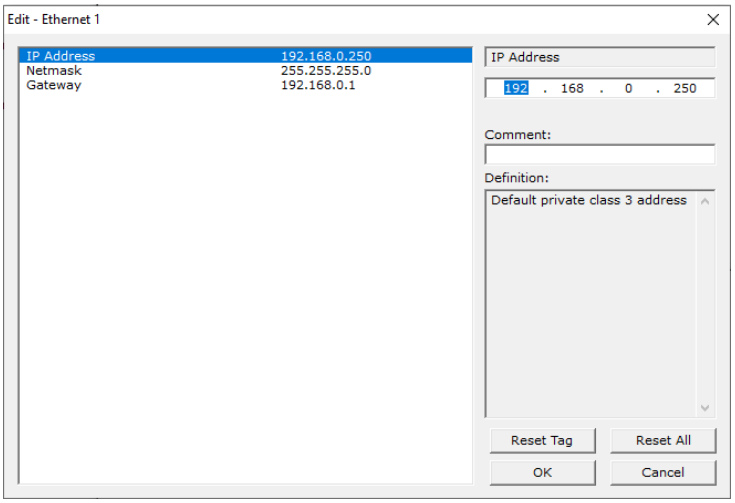


Parameter	Value	Description
IP Address	1 to 255	Specifies the Modbus server IP address on the network to be considered.
Hardware MAC Address	00 to FF	**WARNING:** If the device in the field is changed, this table must be updated to contain the new MAC address for that device and download to the module. If the MAC is not changed, no communications with the module will be provided.

6.2.6 Ethernet 1

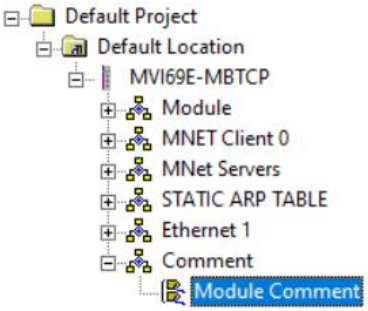
The **ETHERNET 1** option configures the module's IP Address, Subnet Mask, and Gateway.





6.2.7 Comment Parameter

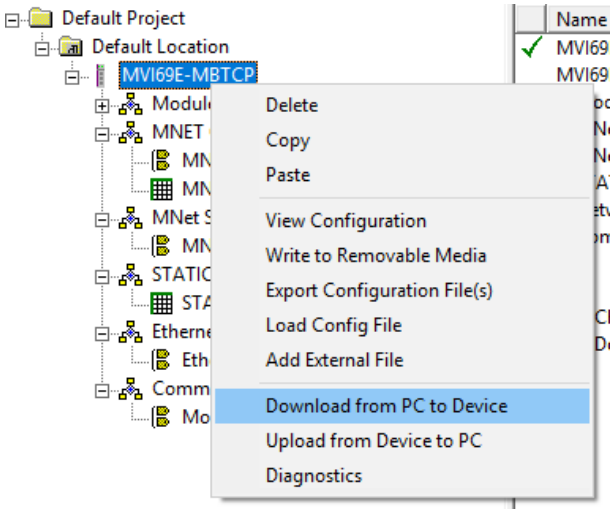
Under the **MODULE COMMENT** option, you can make a note that this configuration is a Legacy conversion.



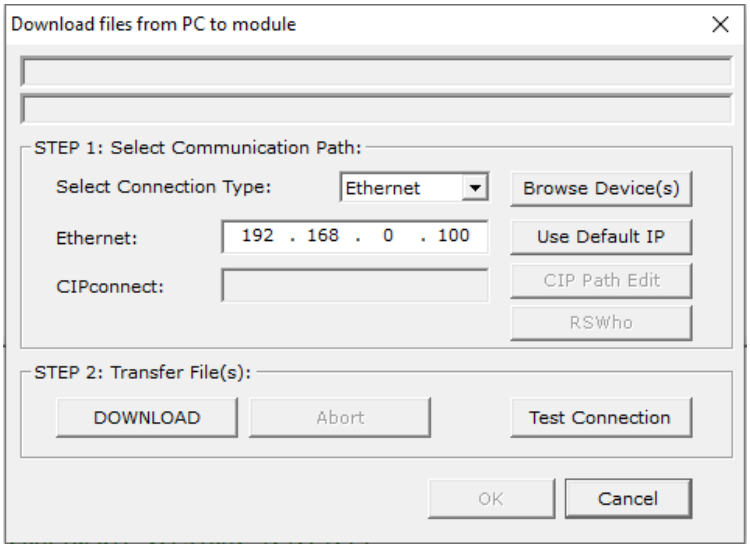
6.3 Downloading PCB Configuration to the MVI69E-MBTCP

In Legacy Mode, the configuration project is downloaded directly to the module Ethernet port as described in this section.

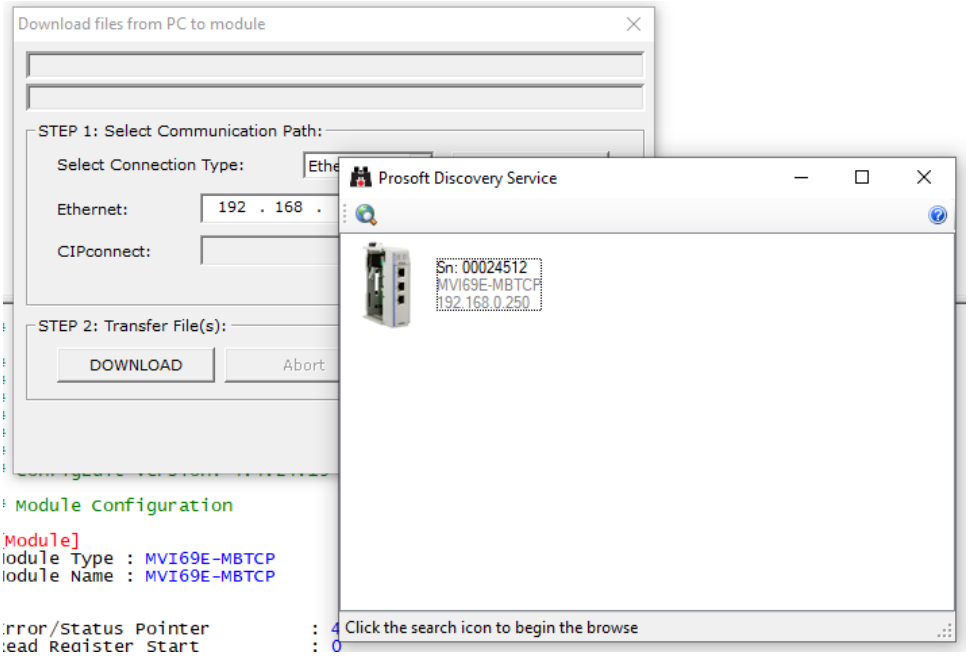
1 Right-click on the *MVI69E-MBTCP* icon and select **DOWNLOAD FROM PC TO DEVICE**.



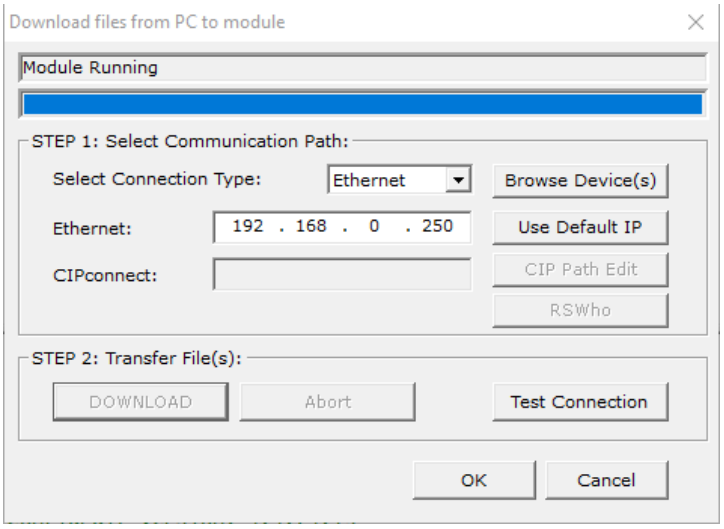
2 In the *Download files from PC to module* dialog, click on the **BROWSE DEVICE(S)** button. The ProSoft Discovery Service Utility searches for ProSoft devices on the network.



3 Double-click on the module icon.



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



Once complete, the MVI69E-MBTCP in Legacy Mode will operate similarly to the MVI69-MNET.

6.4 Optional Add-On Instruction

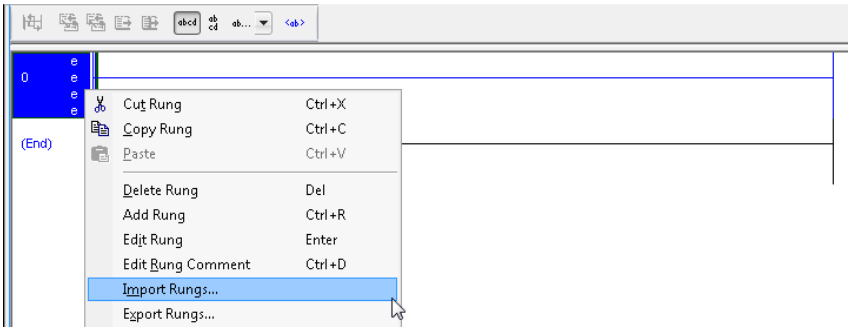
The Optional AOI supports the following optional features:

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

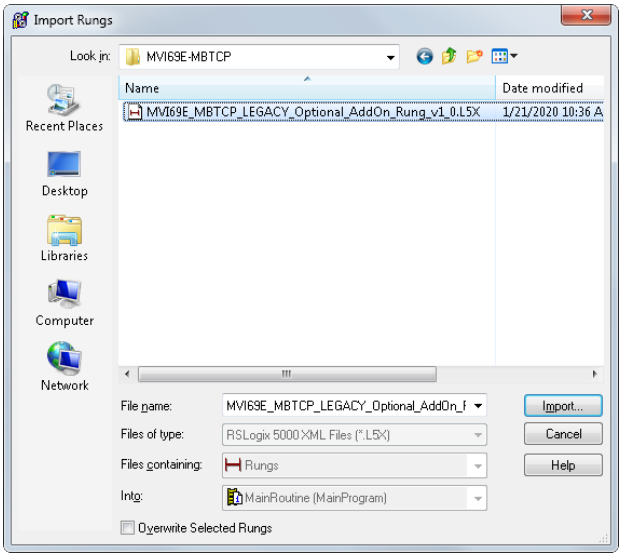
Using controller tags, the Optional AOI allows you to request and set the module's IP address, date, and time. These optional features are not supported by the MVI69E-MNET legacy module.

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

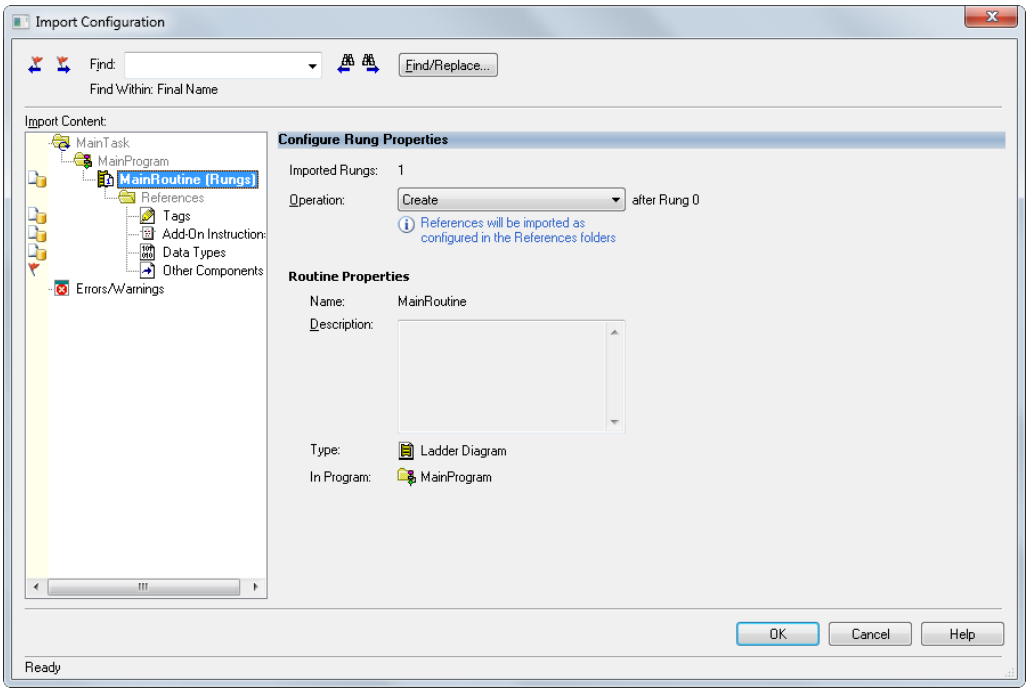
1. Add a new rung to the existing processor ladder logic. Right-click on the new rung and select *Import Rungs…*



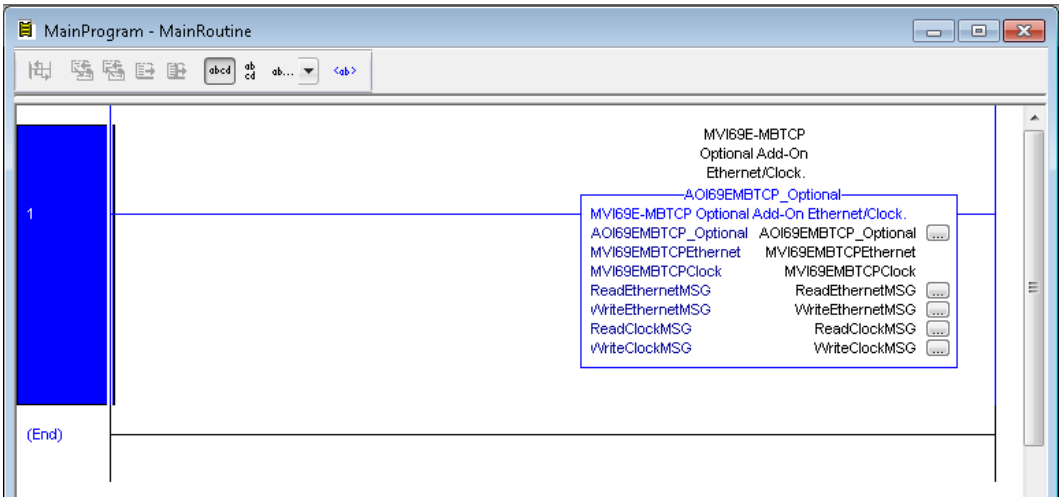
2. Select the Optional AOI file: *MVI69E_MBTCP_Optional_AddOn_Rung.L5X.*



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

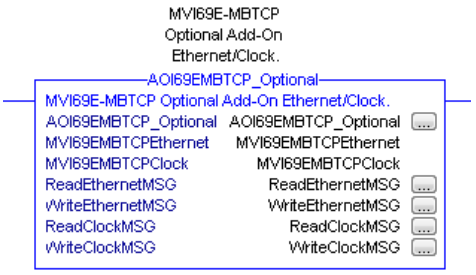


4 The imported AOI rung is now in place.

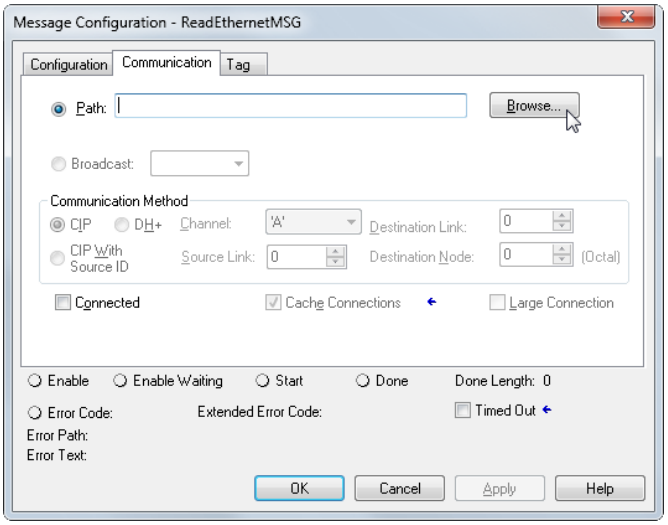


6.4.1 Setting Up the Optional AOI

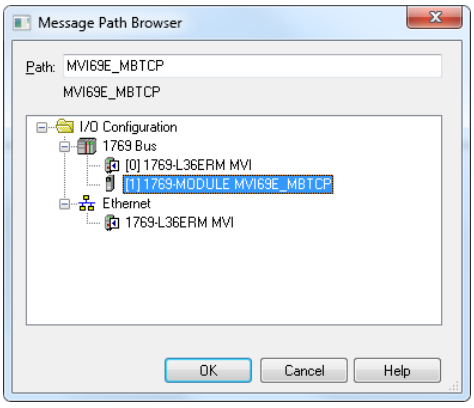
1 Click on the *ReadEthernetMSG* ... icon to configure the message route:



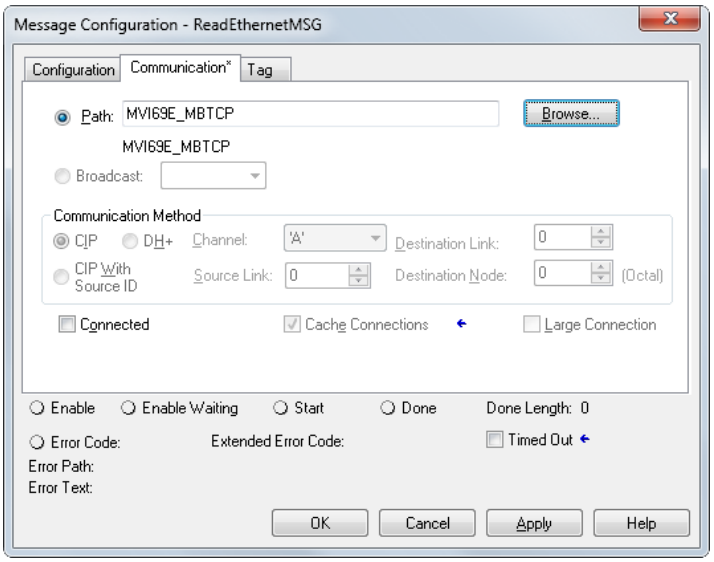
2 In the *Message Configuration* dialog, under the *Communication* tab, select the **BROWSE** button.



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



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



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

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

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

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

1 To trigger the IP settings read operation, set the *MVI69EMBTCPEthernet.Read* bit to '**1**'.

MVI69EMBTCPEthernet	{...}
MVI69EMBTCPEthernet.Read	1

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

MVI69EMBTCPEthernet	{...}
MVI69EMBTCPEthernet.Read	0

3 The data is stored in the *MVI69EMBTCPEthernet.Config* tags (IP, Netmask, Gateway) as follows:

MVI69EMBTCPEthernet.Config	{...}
MVI69EMBTCPEthernet.Config.IP	{...}
MVI69EMBTCPEthernet.Config.IP[0]	192
MVI69EMBTCPEthernet.Config.IP[1]	168
MVI69EMBTCPEthernet.Config.IP[2]	0
MVI69EMBTCPEthernet.Config.IP[3]	250
MVI69EMBTCPEthernet.Config.Netmask	{...}
MVI69EMBTCPEthernet.Config.Netmask[0]	255
MVI69EMBTCPEthernet.Config.Netmask[1]	255
MVI69EMBTCPEthernet.Config.Netmask[2]	255
MVI69EMBTCPEthernet.Config.Netmask[3]	0
MVI69EMBTCPEthernet.Config.Gateway	{...}
MVI69EMBTCPEthernet.Config.Gateway[0]	192
MVI69EMBTCPEthernet.Config.Gateway[1]	168
MVI69EMBTCPEthernet.Config.Gateway[2]	0
MVI69EMBTCPEthernet.Config.Gateway[3]	1

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

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

1 Populate the IP settings in the *MVI69EMBTCPEthernet.Config* tag:

MVI69EMBTCPEthernet.Config	{...}
MVI69EMBTCPEthernet.Config.IP	{...}
MVI69EMBTCPEthernet.Config.IP[0]	192
MVI69EMBTCPEthernet.Config.IP[1]	168
MVI69EMBTCPEthernet.Config.IP[2]	0
MVI69EMBTCPEthernet.Config.IP[3]	250
MVI69EMBTCPEthernet.Config.Netmask	{...}
MVI69EMBTCPEthernet.Config.Netmask[0]	255
MVI69EMBTCPEthernet.Config.Netmask[1]	255
MVI69EMBTCPEthernet.Config.Netmask[2]	255
MVI69EMBTCPEthernet.Config.Netmask[3]	0
MVI69EMBTCPEthernet.Config.Gateway	{...}
MVI69EMBTCPEthernet.Config.Gateway[0]	192
MVI69EMBTCPEthernet.Config.Gateway[1]	168
MVI69EMBTCPEthernet.Config.Gateway[2]	0
MVI69EMBTCPEthernet.Config.Gateway[3]	1

2 Set the *MVI69EMBTCPEthernet.Write* bit to '**1**' to trigger the IP settings write operation.

MVI69EMBTCPEthernet	{...}
MVI69EMBTCPEthernet.Read	0
MVI69EMBTCPEthernet.Write	1

3 The *MVI69EMBTCPEthernet.Write* bit will automatically reset to '**0**' once the operation is concluded.

MVI69EMBTCPEthernet	{...}
MVI69EMBTCPEthernet.Read	0
MVI69EMBTCPEthernet.Write	0

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

1 Toggle the *MVI69EMBTCPClock.Read* bit to '**1**' to toggle the date/time read operation.

MVI69EMBTCPClock	{...}
MVI69EMBTCPClock.Read	1
MVI69EMBTCPClock.Write	0

2 The *MVI69EMBTCPClock.Read* bit will automatically reset to '**0**' once the operation is concluded.

MVI69EMBTCPClock	{...}
MVI69EMBTCPClock.Read	0
MVI69EMBTCPClock.Write	0

3 The date and time read from the MVI69E-MBTCP is stored at the *MVI69EMBTCPClock.Config* tag.

MVI69EMBTCPClock	{...}
MVI69EMBTCPClock.Read	0
MVI69EMBTCPClock.Write	0
MVI69EMBTCPClock.Config	{...}
MVI69EMBTCPClock.Config.Year	2020
MVI69EMBTCPClock.Config.Month	2
MVI69EMBTCPClock.Config.Day	6
MVI69EMBTCPClock.Config.Hour	8
MVI69EMBTCPClock.Config.Minute	33
MVI69EMBTCPClock.Config.Seconds	10

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

1 Populate date and time values in the *MVI69EMBTCPClock.Config* tag.

MVI69EMBTCPClock.Config	{...}
MVI69EMBTCPClock.Config.Year	2020
MVI69EMBTCPClock.Config.Month	2
MVI69EMBTCPClock.Config.Day	6
MVI69EMBTCPClock.Config.Hour	8
MVI69EMBTCPClock.Config.Minute	33
MVI69EMBTCPClock.Config.Seconds	10

2 Toggle the *MVI69EMBTCPClock.Write* bit to '**1**' to trigger the write date/time operation.

MVI69EMBTCPClock	{...}
MVI69EMBTCPClock.Read	0
MVI69EMBTCPClock.Write	1

3 The *MVI69EMBTCPClock.Write* tag will be automatically reset to '**0**' once the write date/time operation is concluded.

MVI69EMBTCPClock	{...}
MVI69EMBTCPClock.Read	0
MVI69EMBTCPClock.Write	0

7 Diagnostics and Troubleshooting

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.

7.1 LED Status Indicators

The LEDs indicate the module's operating status.

ETH	CFG
CLT	BP
SRV	OK

LED	Color	Description
ETH	Green	Application is running and Ethernet is ready
	Off	Possible causes: • Network communication has not started yet • Physical ethernet connection is down • Ethernet communication is disabled as the system shuts down.
CLT	Red	Exception response received from the server; bad address, command, etc
	Off	Possible causes: • No client-side communication has started yet • The database, MBTCP client, and outputs have been initialized • MBTCP client operations are disabled as the system shuts down.
	Amber	The Modbus TCP client is initializing
SRV	Off	Possible causes: • During initialization, server functions have not been started • Server communications are disabled as the system shuts down.
	Amber	The module is initializing its server-side services
CFG	Red	Possible causes: • Configuration failure during start-up or module boot • Module detected a bad or unreadable configuration file; problem opening the configuration file or the file is corrupted
	Green	Configuration is valid
	Amber	Possible causes: • During initialization, the configuration process has started and is being validated • Configuration file has been received and is being processed

LED	Color	Description
	Off	Application is not running, or backplane has failed. Possible causes: • Communication with the backplane has failed • Backplane process is not up and running; no link to the controller so configuration status cannot be confirmed • Nothing to configure; module is waiting or idle. Cannot open file for writing or if no configuration file is received from the processor • Backplane data failure counter (*Failure Flag Count* parameter in the Module configuration) exceeded the limit • Module is in shutdown function; configuration is no longer active once the system is shutting down.
BP	Red	Possible causes: • Backplane disruption or communication error (PLC may not be in RUN mode) • Initialization failed; backplane communication took too long. Block timeout occurred during module initialization • Backplane data failure counter (*Failure Flag Count* parameter in the Module configuration) has exceeded the limit.
	Green	Backplane transfers are successful; communications with PLC are operational
	Amber	Initialization state: Module is in the process of establishing communication with the controller.
	Off	Possible causes: • During power-up, no communication with the backplane has started yet • Module is in shutdown function; backplane link is no longer active once the system is shutting down.
OK	Red	Possible causes: • Module is powering up • Module has encountered a fatal error • Module is in shutdown function
	Green	The module has properly initialized and is running

During module configuration, the OK LED is red and the BP LED is on. If the 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.

7.2 Ethernet LED Indicators

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

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

7.3 Clearing a Fault Condition

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

To clear the condition, follow these steps:

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

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

7.4 Troubleshooting

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

7.4.1 Processor Errors

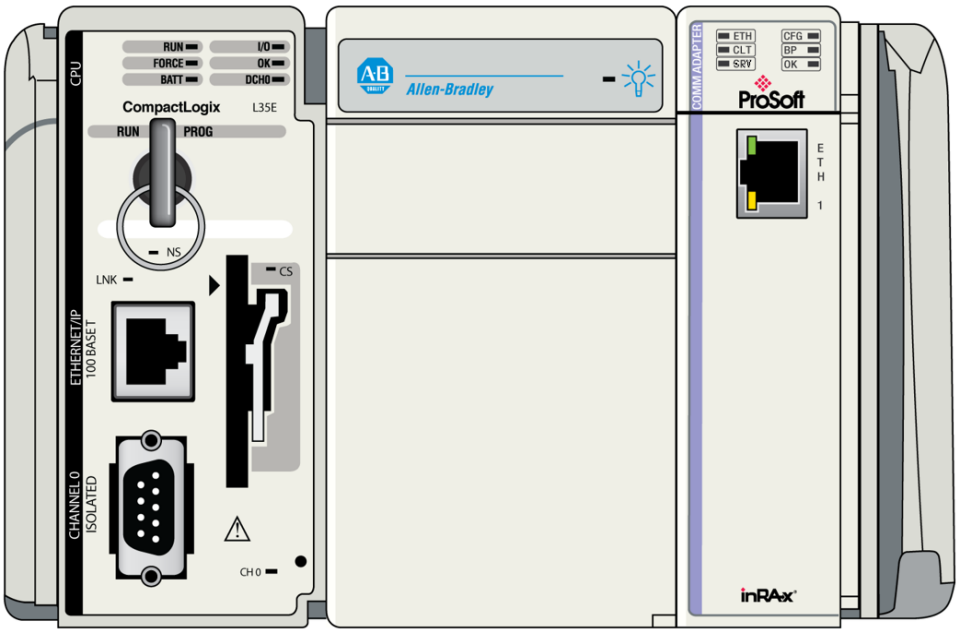
Problem Description	Steps to take
Processor fault	Verify that the module is securely plugged into the slot that has been configured for the module in the I/O Configuration in RSLogix. Verify that the slot location in the rack has been configured correctly in the ladder logic.
Processor I/O LED flashes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

7.4.2 Module Errors

Problem Description	Steps to take
BP LED remains OFF or blinks slowly 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 and re-insert it, and then restore power to the rack.

7.5 Connecting the PC to the Module's Ethernet Port

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

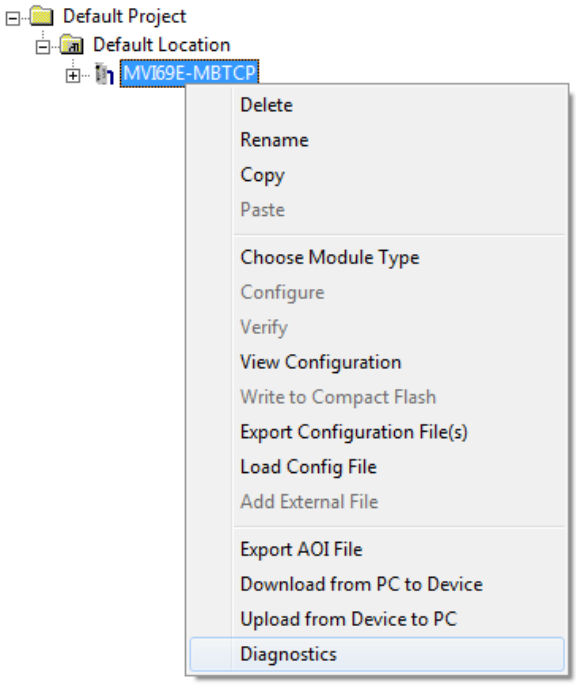


7.5.1 Setting Up a Temporary IP Address

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

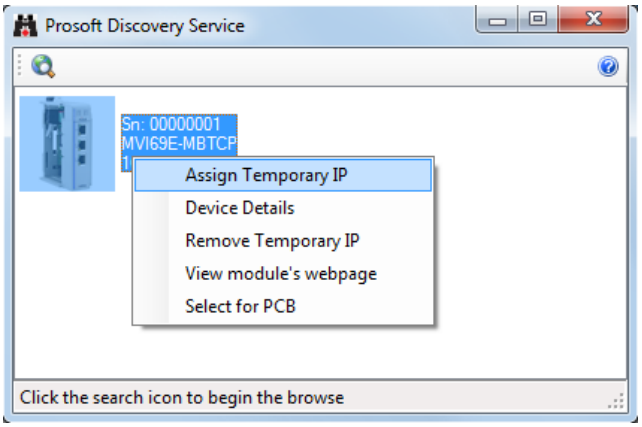
To use ProSoft Configuration Builder, arrange the Ethernet connection so that there is no router/ layer 3 switch between the computer and the module, OR reconfigure the router/ layer 3 switch to allow routing of the UDP broadcast messages.

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

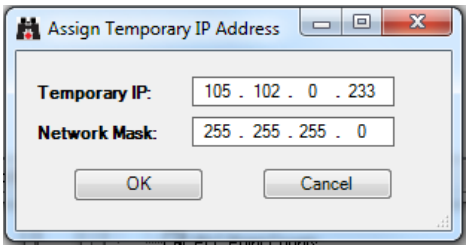


3. In the *Diagnostics* window, click the **SET UP CONNECTION** button.

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



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



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

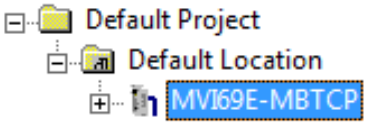
6 Close the ProSoft Discovery Service window. Enter the temporary IP address in the Ethernet address field of the *Connection Setup* dialog box, then click **TEST CONNECTION** to verify that the module is accessible with the current settings.

7 If the *Test Connection* is successful, click **CONNECT**. The *Diagnostics* window is now accessible. See *Using the Diagnostics Menu in ProSoft Configuration Builder* (page 110) for more information.

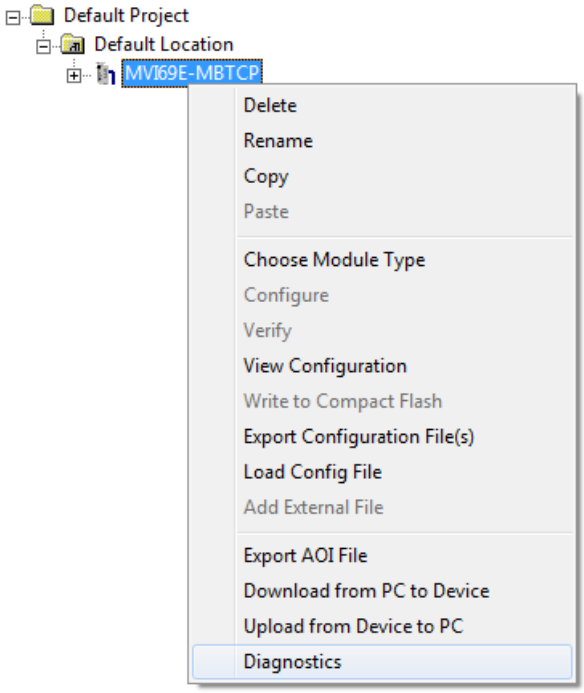
7.6 Using the Diagnostics Menu in ProSoft Configuration Builder

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

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

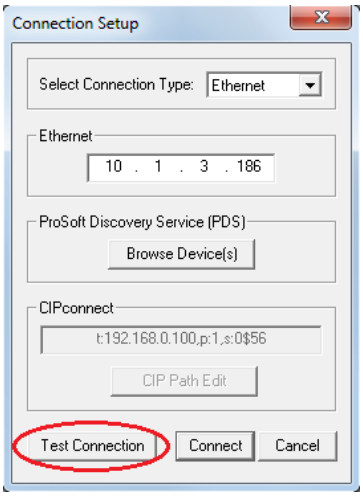


2 Right-click the module and choose **DIAGNOSTICS.**

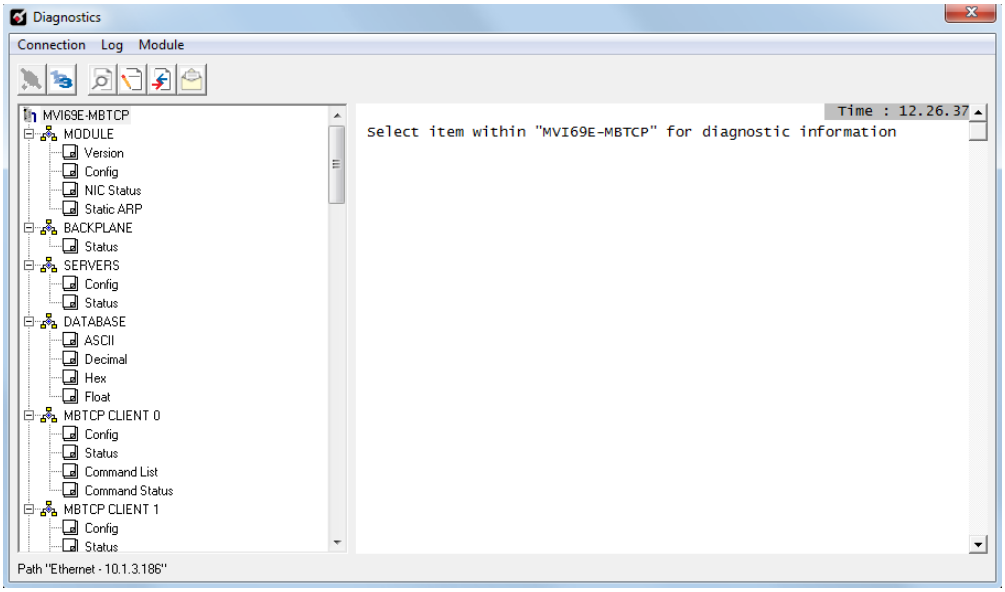


3 After the *Diagnostics* window opens, click the **SETUP CONNECTION** button to browse for the module's IP address.

4 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.

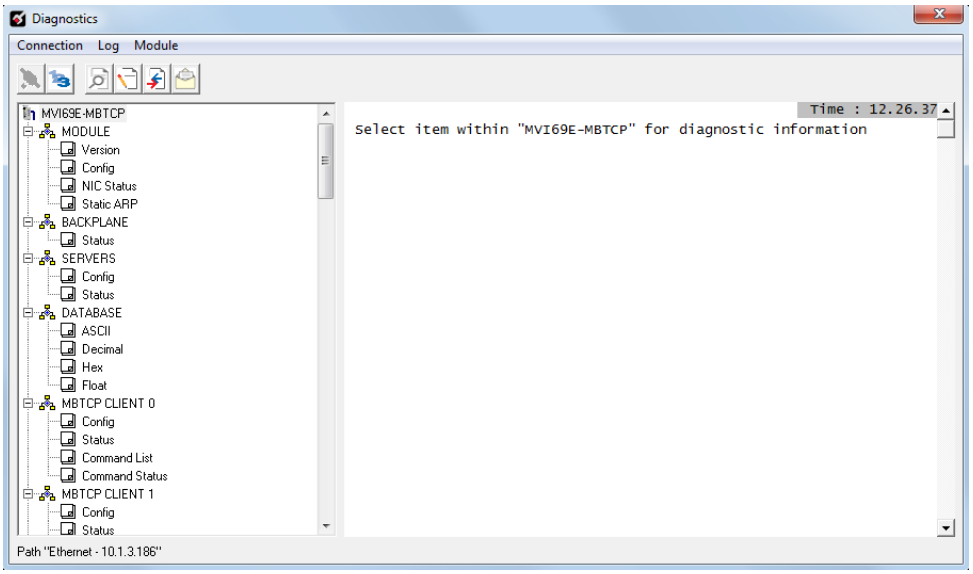


5 If the **TEST CONNECTION** is successful, click **CONNECT**. The *Diagnostics* Window is now accessible.



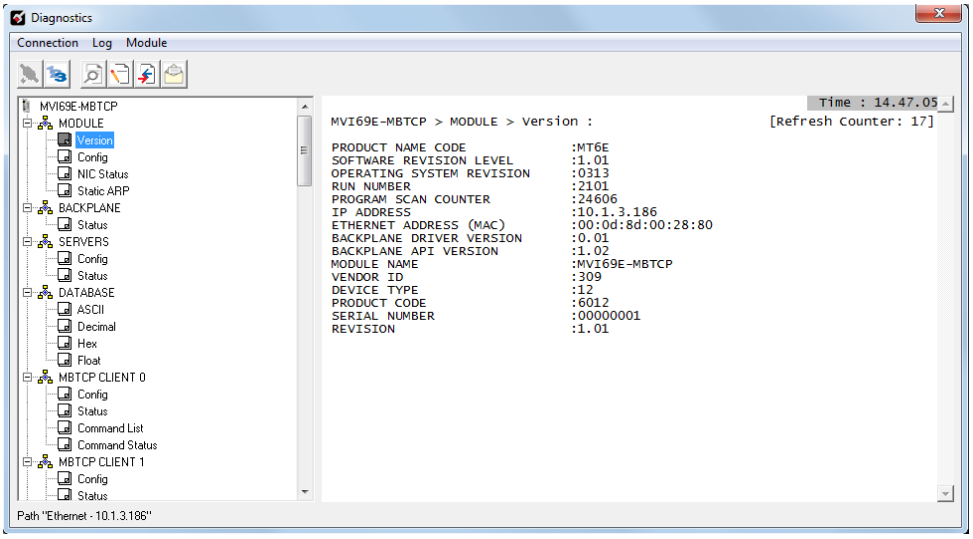
7.6.1 Diagnostics Menu

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



7.6.2 Monitoring General Information

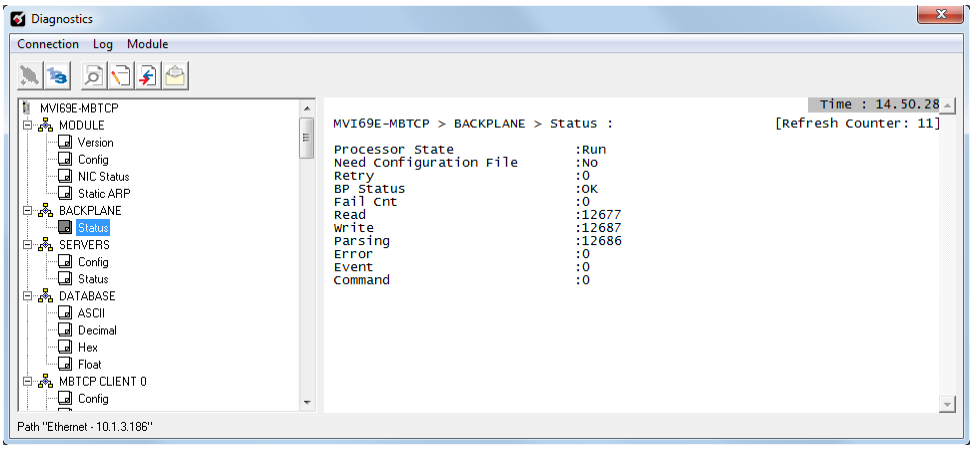
In the *Diagnostics* window in ProSoft Configuration Builder, click **MODULE** and then click **VERSION** to view module version information.



7.6.3 Monitoring Backplane Information

In the *Diagnostics* window in ProSoft Configuration Builder, click **BACKPLANE** to view the backplane information:

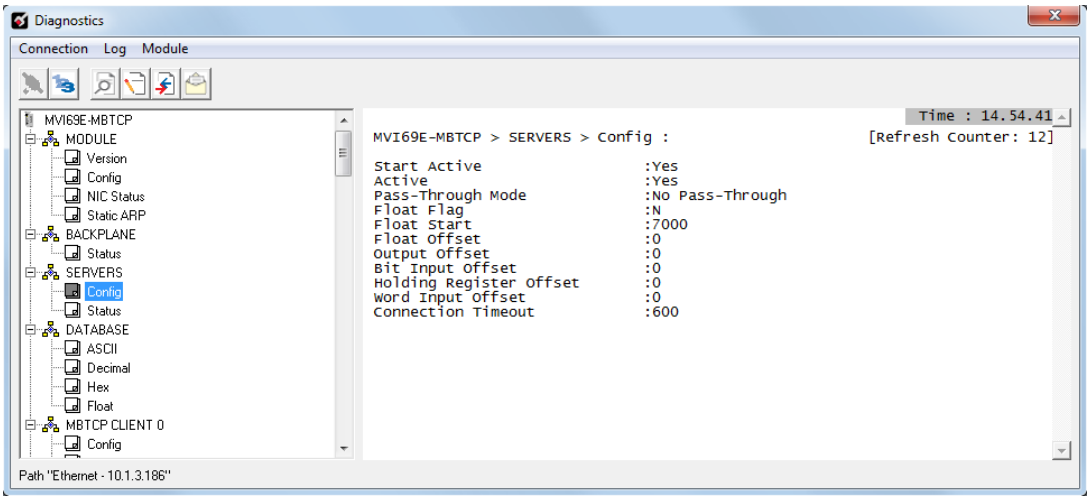
- **STATUS**

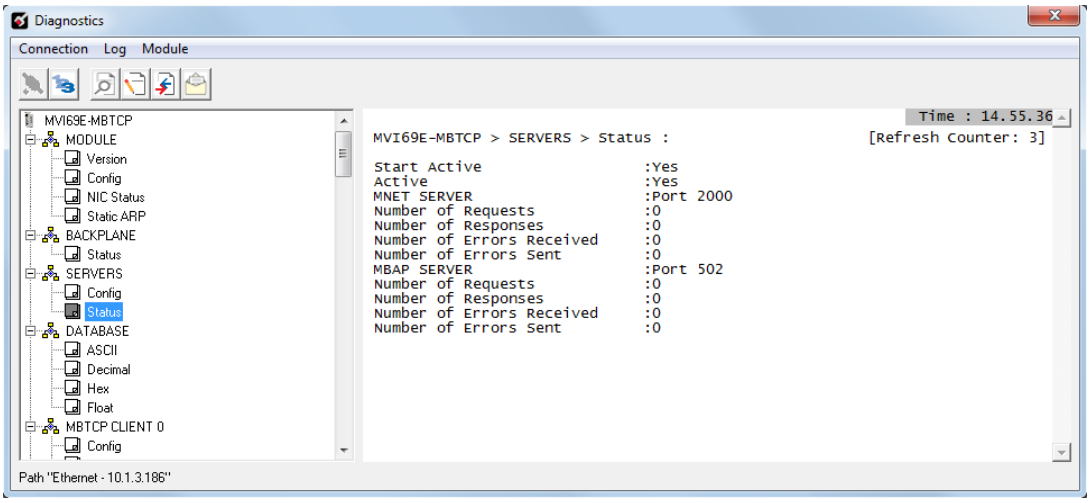


7.6.4 Modbus Server Driver Information

In the *Diagnostics* window in ProSoft Configuration Builder, click **SERVERS** to view the server information. The menu has two sub-menus:

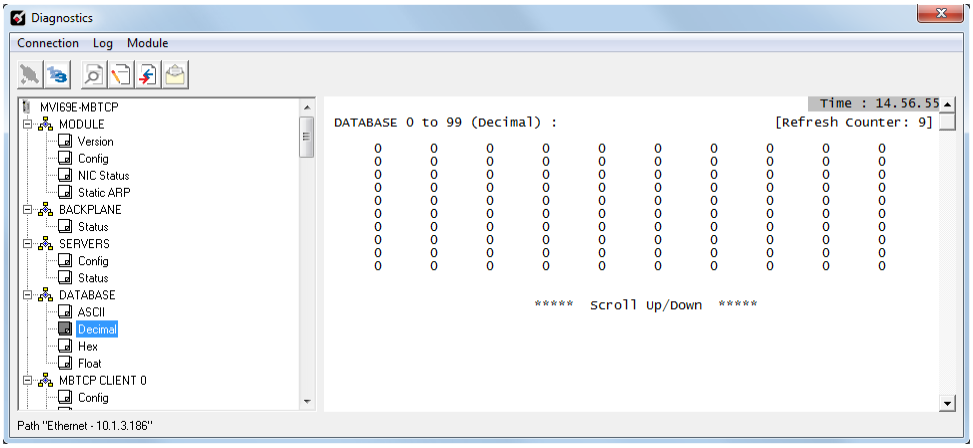
- **CONFIGURATION**
- **STATUS**





7.6.5 Monitoring Data Values in the Module's Database

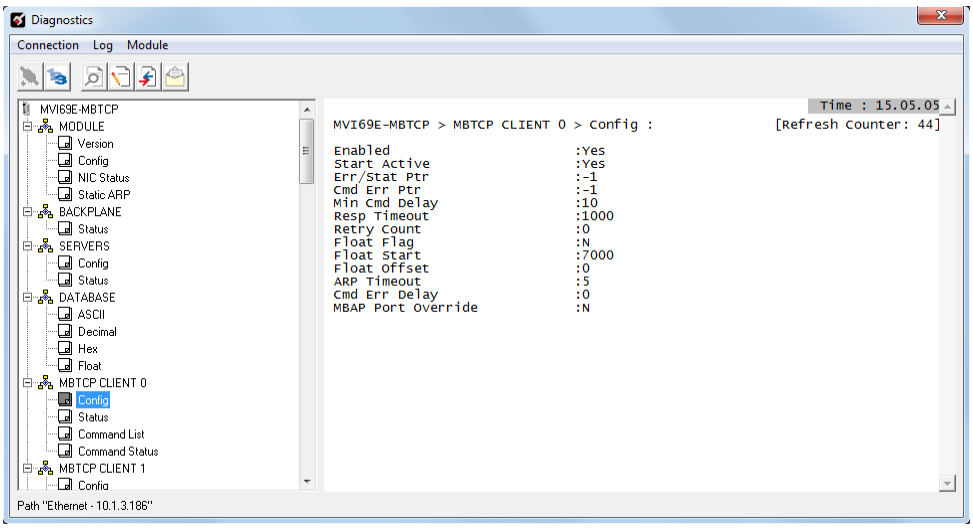
In the *Diagnostics* window in ProSoft Configuration Builder, click **DATABASE** and then click **DECIMAL** to view the contents of the MVI69E-MBTCP internal database. You can view data values in ASCII, Hexadecimal, and Float format.



7.6.6 Modbus Client Driver Information

In the *Diagnostics* window in ProSoft Configuration Builder, click **MBTCP CLIENT X** to view Modbus Client driver information, where **X** is the number of the Modbus Client. The Modbus Client Driver menus have four submenus:

- **CLIENT X CONFIGURATION**
- **CLIENT X STATUS**
- **CLIENT X COMMAND LIST**
- **CLIENT X COMMAND STATUS**



7.7 Communication Error Codes

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

7.7.1 Standard Modbus Protocol Exception Code Errors

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

7.7.2 Module Communication Error Codes

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

7.7.3 Command List Entry Errors

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

7.7.4 MBTCP Client-Specific Errors

Code	Description
-33	Failed to connect to server specified in command
-36	MBTCP command response timeout
-37	TCP/IP connection ended before session finished

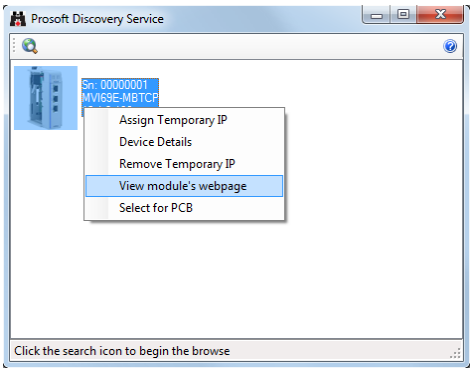
Note: If an error code is reported that is not listed above, check with the documentation of the end device. Device-specific error codes can be produced by the end device.

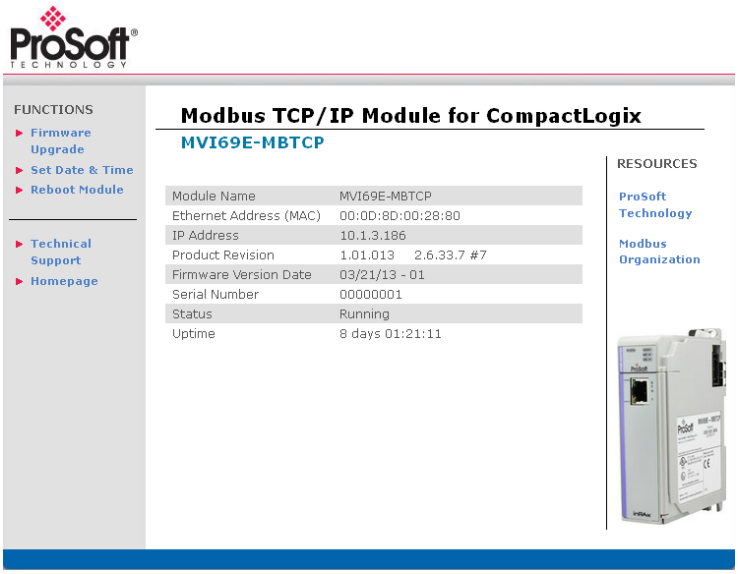
7.8 Connecting to the MVI69E-MBTCP Webpage

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

1. In the *Diagnostics* window in ProSoft Configuration Builder, click the **SET UP CONNECTION** button.

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





8 Reference

8.1 Product Specifications

The MVI69E-MBTCP allows Rockwell Automation® CompactLogix® processors to interface easily with other Modbus TCP/IP compatible devices.

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

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

8.1.1 General Specifications - Modbus Client/Server

Specification	Description	
Communication Parameters	Supports Modbus MBAP and encapsulated (Server) messaging 10/100 Base-T Ethernet-compatible interface	
Modbus Modes	Client driver supports up to twenty connections for active reading and writing of data with Modbus TCP/IP compatible devices Server driver supports connections to up to five Modbus TCP/IP clients using Service Port 502 with standard MBAP messaging, and up to five clients using Modbus RTU/ASCII on Service Port 2000 (and others)	
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 (Server Only, responds to Subfunction 00)	15: Force (Write) Multiple Coils 16: Preset (Write) Multiple Holding Registers 17: Report Slave ID (Server Only) 22: Mask Write Holding Register (Server Only) 23: Read/Write Holding Registers (Server Only)

8.1.2 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
Application/Diagnostics Port (ETH 1)	Diagnostics over Ethernet connection RJ45 Port

8.2 About the Modbus TCP/IP 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 client and server devices on a network. Later enhancements to the protocol allow communication over Ethernet networks using TCP/IP as a "wrapper" for the Modbus protocol. This protocol is known as Modbus TCP/IP.

Modbus TCP/IP is a client/server protocol. The client establishes a connection to the remote server. When the connection is established, the client sends the Modbus TCP/IP commands to the server. The MVI69E-MBTCP module simulates up to 30 clients, and works both as a client and a server.

Aside from the benefits of Ethernet versus serial communications (including performance, distance, and flexibility) for industrial networks, the Modbus TCP/IP protocol allows for remote administration and control of devices over an Internet connection. It is important to note that not all Internet protocols are implemented in the module; for example, HTTP and SMTP protocols are not available. Nevertheless, the efficiency, scalability, and low cost of a Modbus TCP/IP network make this an ideal solution for industrial applications.

The MVI69E-MBTCP module acts as an input/output module between devices on a Modbus TCP/IP network and the Rockwell Automation backplane and processor. The module uses an internal database to pass data and commands between the processor and the client and server devices on the Modbus TCP/IP network.

8.2.1 Modbus Client

The MVI69E-MBTCP Modbus client actively issues Modbus commands to Modbus servers on the Modbus TCP/IP network, supporting up to 16 commands for each client. The clients have an optimized polling characteristic that polls servers with communication problems less frequently.

Parameter	Description
Command List	Up to 16 commands per client, each fully configurable for function, server IP 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 server status list is maintained per active Modbus client.

8.2.2 Modbus Server

The MVI69E-MBTCP Modbus Server driver permits a remote client to interact with all data contained in the module. This data can be derived from other Modbus server devices on the network, through a client port, or from the CompactLogix processor.

Parameter	Description
Service Port	MBAP messaging on Service Port 502 Encapsulated messaging on Service Port 2000
Status Data	Error codes, counters and port status available

8.2.3 Function Codes Supported by the Module

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

Function Code	Definition	Supported as Client	Supported as Server
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 Server 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 server device.

8.2.4 Read Coil Status (Function Code 01)

8.2.4.1 Query

This function allows you to obtain the ON/OFF status of logic coils (Modbus 0x range) used to control discrete outputs from the addressed server only. Broadcast mode is not supported with this function code. In addition to the server 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 server 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 server device number 11.

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

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

8.2.4.2 Response

An example response to Read Coil Status is as shown in the table below. The data is packed one bit for each coil. The response includes the server 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 server interface device is serviced at the end of a controller's scan, data reflects coil status at the end of the scan. Some servers 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.

8.2.5 Read Input Status (Function Code 02)

8.2.5.1 Query

This function allows you to obtain the ON/OFF status of discrete inputs (Modbus 1x range) in the addressed server. PC Broadcast mode is not supported with this function code. In addition to the server 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 server 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 server number 11.

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

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

8.2.5.2 Response

An example response to Read Input Status is as shown in the table below. The data is packed one bit for each input. The response includes the server 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 server interface device is serviced at the end of a controller's scan, the data reflect input status at the end of the scan. Some servers 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.

8.2.6 Read Holding Registers (Function Code 03)

8.2.6.1 Query

This function allows you to retrieve the contents of holding registers 4xxxx (Modbus 4x range) in the addressed server. 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 server 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 server number 11.

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

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

8.2.6.2 Response

The addressed server 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 server 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 servers limit the quantity of register content provided each scan; thus for large register quantities, multiple transmissions are made using register content from sequential scans.

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

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

8.2.7 Read Input Registers (Function Code 04)

8.2.7.1 Query

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

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

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

8.2.7.2 Response

The addressed server 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 server 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	E9

8.2.8 Force Single Coil (Function Code 05)

8.2.8.1 Query

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

The use of server address 00 (Broadcast Mode) forces all attached servers 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 server number 11 to turn ON coil 0173.

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

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

8.2.8.2 Response

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

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

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

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

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

8.2.9 Preset Single Register (Function Code 06)

8.2.9.1 Query

This Function Code allows you to modify the contents of a Modbus 4x range in the server. 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 server address zero (Broadcast mode), all server 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 server 11.

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

8.2.9.2 Response

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

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

8.2.10 Diagnostics (Function Code 08)

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

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

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

A server device can, however, be forced into 'Listen Only Mode' in which it monitors the messages on the communications system but not respond to them. This can affect the outcome of your application program if it depends upon any further exchange of data with the remote device. Generally, the mode is forced to remove a malfunctioning remote device from the communications system.

8.2.10.1 Sub-function Codes Supported

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

8.2.10.1.1 Return Query Data 00

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

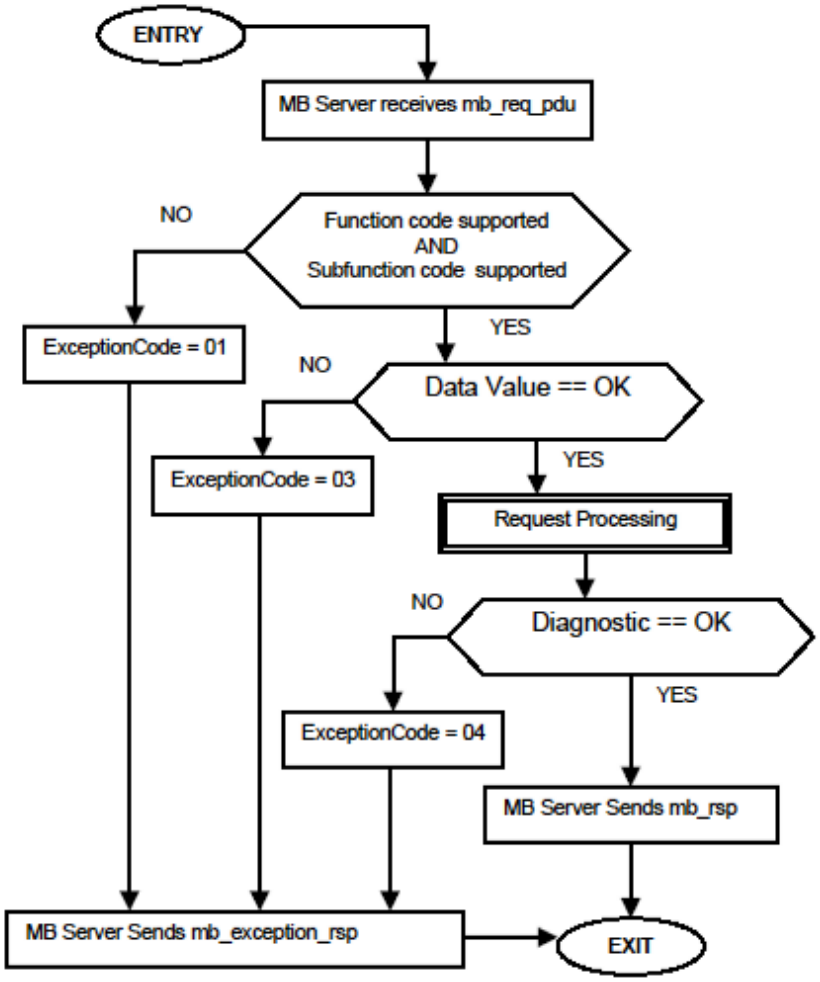
Sub-function	Data Field (Request)	Data Field (Response)
00 00	Any	Echo Request Data

8.2.10.2 Example and State Diagram

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

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

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



8.2.11 Force Multiple Coils (Function Code 15)

8.2.11.1 Query

This function forces each coil (Modbus 0x range) in a consecutive block of coils to a desired ON or OFF state. Any coil that exists within the controller can be forced to either state (ON or OFF). However, because the controller is actively scanning, unless the coils are disabled, the controller can also alter the state of the coil. Coils are numbered from zero (coil 00001 = zero, coil 00002 = one, and so on). The desired status of each coil is packed in the data field, one bit for each coil (1= ON, 0= OFF). The use of server address 0 (Broadcast Mode) forces all attached servers 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	Function Code	Coil Address High	Coil Address Low	Number of Coils High	Number of Coils Low	Byte Count	Force Data High 20 to 27	Force Data Low 28 to 29	Error Check Field (2 bytes)
0B	0F	00	13	00	0A	02	CD	01	CRC

8.2.11.2 Response

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

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

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

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

8.2.12 Preset Multiple Registers (Function Code 16)

8.2.12.1 Query

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

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

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

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

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

8.2.12.2 Response

The normal response to a function 16 query is to echo the address, function code, starting address and number of registers to be loaded.

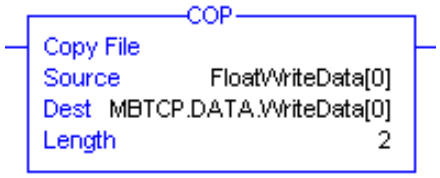
Node Address	Function Code	Data Start Address High	Data Start Address Low	Number of Points High	Number of Points Low	Error Check Field (2 bytes)
0B	10	00	01	00	02	CRC

8.3 Floating-Point Support

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

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

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



This instruction moves one floating-point value in two 16-bit integer images to *MBTCP.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 *MBTCP.DATA.ReadData[0]*, which is an integer tag, to a floating-point tag (something you would do to receive floating-point values from the module) is shown below.

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

8.3.1 ENRON Floating Point Support

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

8.3.2 Configuring the Floating Point Data Transfer

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

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

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

8.3.2.1 Example #1

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

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

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

8.3.2.2 Example #2

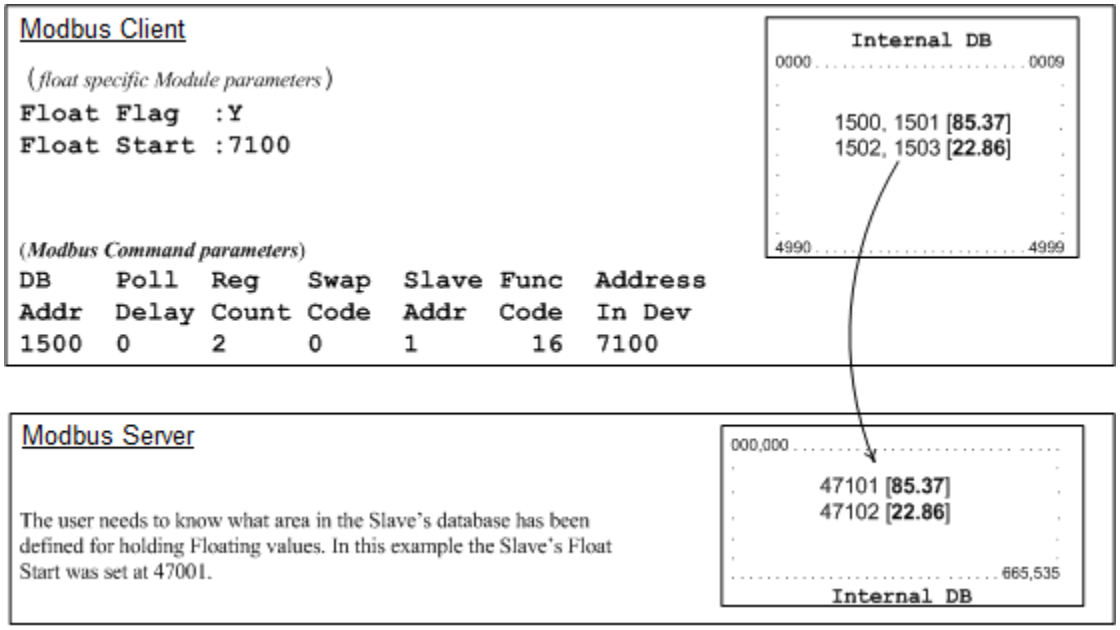
Modbus Address	Data Type	Parameter
47101	32 bit REAL	TEMP Pump #1
47103	32 bit REAL	Pressure Pump #1
47105	32 bit REAL	TEMP Pump #2
47107	32 bit REAL	Pressure Pump #2

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

8.3.3 Examples

8.3.3.1 Example #1

Client is issuing Modbus command with FC 16 (with Float Flag: Yes) to transfer Float data to server.



8.3.3.1.1 (Float specific module parameters)

Parameter	Description
Float Flag: **Y**	The client will consider the data values that need to be sent to the server as floating point data where each data value is composed of 2 words (4 bytes or 32 bits).
Float Start	Client determines that if this address number is less than or equal to 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 server. Otherwise the client ignores the *Float Flag: Y* and treat data as composed of 1 word, 2 bytes.

8.3.3.1.2 (Modbus Command parameters)

Parameter	Description
DB Addr	The starting database address to obtain and write out to the server device.
Reg Count	The number of data points to send to the server. 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	The orientation of the Byte and Word structure of the data value. This is device dependent. See *MBTCP Client x Commands* (page 43).
Func Code	Modbus function code to write the float values to the server. FC16.
Addr in Dev	Database address in the server to write the data.

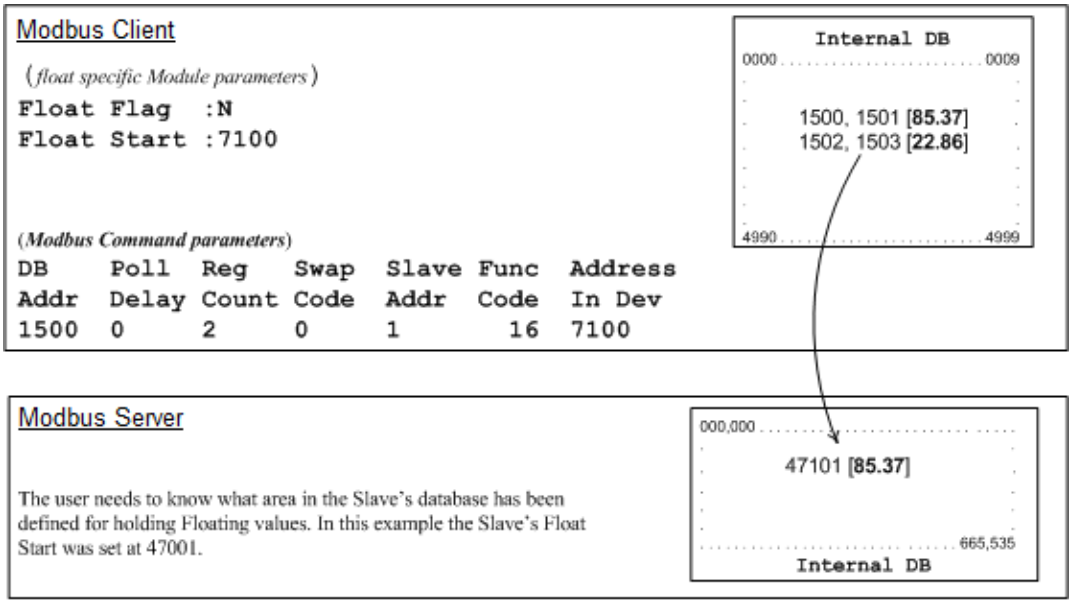
In the example above, the client's Modbus command to transmit inside the Modbus packet is as follows:

	Server 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 client'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 servers look for the byte count in the data packet to know the length of the data to read from the wire. Other servers know at which byte the data begins and read from the wire the remaining bytes in the packet as the data the client is sending.

8.3.3.2 Example #2

Client is issuing Modbus command with FC 16 (with Float Flag: No) to transfer Float data.



Parameter	Description
Float Flag: **N**	Client will ignore the floating values and treat each register data as a data point composed of 1 word, 2 bytes or 16 bits
Float Start	N/A
DB Addr	The starting database address to obtain and write out to the server device.
Reg Count	The number of data points to send to the server.
Swap Code	The orientation of the Byte and Word structure of the data value. This is device dependent. See *MBTCP Client x Commands* (page 43).
Func Code	Modbus function code to write the float values to the server. FC16.
Addr in Dev	Database address in the server to write the data.

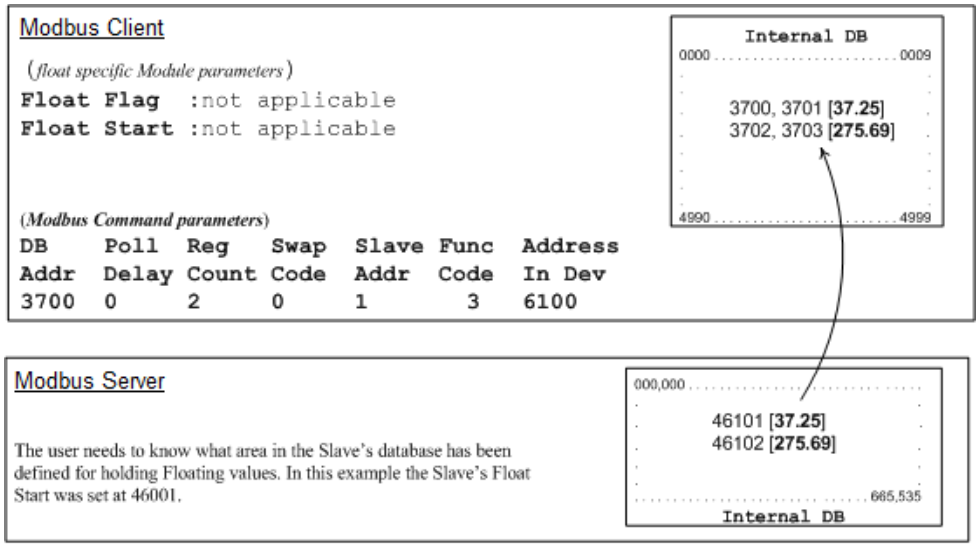
In the above example, the client's Modbus command to transmit inside the Modbus packet is as follows:

	Server 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 client'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 server 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 servers read only half the data from the client's transmission. Other servers read all 8 bytes in this example because they know where in the packet the data starts and ignore the byte count parameter inside the Modbus packet.

8.3.3.3 Example #3

Client is issuing Modbus command with FC 3 to transfer Float data from server.



Parameter	Description
Float Flag	N/A with Modbus Function Code 3
Float Start	N/A with Modbus Function Code 3
DB Addr	The starting database address to store the data obtained from the server.
Reg Count	The number of data points to request from the server.
Swap Code	The orientation of the Byte and Word structure of the data value. This is device dependent. See *MBTCP Client x Commands* (page 43).
Func Code	Modbus function code to read values from the server. FC3.
Addr in Dev	Database address in the server to obtain the data.

The client's Modbus command to transmit inside the Modbus packet is as follows:

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

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

	Server 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

A Non-Enron/Daniel supported server's Modbus command that is transmitted inside the Modbus packet is as follows:

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

8.4 Function Blocks

Data contained in this database is paged through the input and output images by coordination of the CompactLogix ladder logic and the MVI69E-MBTCP 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 blocks
2000 to 2019	Event Command blocks
3000 to 3019	Client status request/response blocks
4000 to 4019	Event Sequence Command blocks
4100 to 4119	Event Sequence Command Error Status blocks
4200	Get queue and event sequence block counts
5001 to 5016	Command Control blocks
8000 to 8019	Add Event with data for a client
8100	Get Event with data status
9250	Get general module status data
9500	Set driver and command active bits
9501	Get driver 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

8.4.1 Event Command Blocks (2000 to 2019)

Event Command blocks send Modbus commands directly from the ladder logic to the specified *MBTCP Client x*. The Event Command is added to the high-priority queue and interrupts normal polling so this special command can be sent as soon as possible.

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

8.4.1.1 Blocks 2000 to 2019: Request from Processor to Module

Offset	Description
0	Block ID 2000 to 2019 indicates this block contains a command to execute by the Client Driver. The last two digits indicate which client to use. **Example:** '2015' utilizes client 15
1 to 4	IP address for the intended server for the message. Each digit (0 to 255) of the IP address is placed in one of the four registers
5	TCP service port to use with the message
6	Modbus node address to use with the message
7	Internal Modbus address in the module to use
8	Count parameter that determines the number of digital points or registers to associate with the command
9	Swap type for integer data only
10	Modbus function code
11	Modbus address in the slave device associated with the command
12 to (n-1)	Spare

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

8.4.1.2 Blocks 2000 to 2019: Response from Module to Processor

Offset	Description
0	Block ID 2000 to 2019 requested by the processor
1	The next read request block identification code
2	Result of the event request. 1 = The command was placed in the command queue. 0 = No room was found in the command queue. -1 = The client is not enabled and active.
3	Number of commands in queue
4 to (n-1)	Spare

8.4.2 Client Status Request/Response Blocks (3000 to 3019)

These blocks request the status of a specific MVI69E-MBTCP client.

8.4.2.1 Block 3000 or 3019: Request from Processor to Module

Offset	Description
0	Block ID 3000 to 3019 identification code indicates this block requests the status from a specific MVI69E-MBTCP client. The last two digits indicate which client to use. **Example:** '3015' uses client 15
1 to (n-1)	Spare

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

8.4.2.2 Block 3000 to 3019: Response from Module to Processor

Offset	Description
0	Block ID 3000 to 3019 requested by the processor
1	Write Block ID
2 to 11	Client status data
12 to 27	Command error list data for client
28 to (n-1)	Spare

8.4.3 Event Sequence Request Blocks (4000 to 4019)

These blocks send Modbus TCP/IP commands directly from controller tags by ladder logic to the *Client Command Priority* queue on the module. Event Commands are not placed in the module's internal database and are not part of the *MNET Client x Command List* in ProSoft Configuration Builder.

8.4.3.1 Block 4000 to 4019: Request from Processor to Module

Offset	Description
0	Block ID 4000 to 4019 indicates this block triggers the event sequence of the MVI69E-MBTCP client. The last two digits indicate which client to use. **Example:** '4015' uses client 15
1 to 4	IP address for the intended server for the message. Each digit (0 to 255) of the IP address is placed in one of the four registers
5	TCP service port for message
6	Modbus node address for the message
7	Internal Modbus address in the module
8	Count parameter that determines the number of digital points or registers to associate with the command
9	Swap type for integer data only
10	Modbus function code
11	Modbus address in the slave device for the command
12	Sequence number
13 to (n-1)	Spare

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

8.4.3.2 Block 4000 to 4019: Response from Module to Processor

Offset	Description
0	Block ID 4000 to 4019 requested by the processor
1	Write Block ID
2	0 = Fail 1 = Success -1 = Client is not enabled and active
3	Number of commands in queue
4 to (n-1)	Spare

8.4.4 Event Sequence Command Error Status Blocks (4100 to 4119)

This block displays the result of each command sent to the client. The request includes the client identification and the command sequence number. The response is the event count and error code for each event. A value of '0' in the error code means there was no error detected.

8.4.4.1 Block 4100 to 4119: Request from Processor to Module

Offset	Description
0	Block ID 4100 to 4119 indicates this block triggers the event sequence command error status request of a specific MVI69E-MBTCP client. The last two digits indicate which Client client to use. **Example: '**4115' uses client 15
1 to (n-1)	Spare

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

8.4.4.2 Block 4100 to 4119: Response from Module to Processor

Offset	Description
0	Block ID 4100 to 4119 requested by the processor
1	Write Block ID
2	Number of Event Sequence Messages in block (0 to 15)
3	Sequence Number
4	Return Error Code
5	Sequence Number
6	Return Error Code
7	Sequence Number
8	Return Error Code
…	…
31	Sequence Number
32	Return Error Code
33 to (n-1)	Spare

8.4.5 Get Queue and Event Sequence Block Counts Block (4200)

This block requests the command queue count and the number of pending event sequence commands for all module clients.

8.4.5.1 Block 4200: Request from Processor to Module

Offset	Description
0	Block ID 4200
1 to (n-1)	Spare

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

8.4.5.2 Block 4200: Response from Module to Processor

Offset	Description
0	Block ID 4200
1	Write Block ID
2	Client 0 command queue count (MSB Most Significant Byte) and event sequence messages waiting (LSB Least Significant Bit)
3	Client 1 command queue count (MSB Most Significant Byte) and event sequence messages waiting (LSB Least Significant Bit)
4	Client 2 command queue count (MSB Most Significant Byte) and event sequence messages waiting (LSB Least Significant Bit)
…	…
20	Client 18 command queue count (MSB Most Significant Byte) and event sequence messages waiting (LSB Least Significant Bit)
21	Client 19 command queue count (MSB Most Significant Byte) and event sequence messages waiting (LSB Least Significant Bit)
22 to (n-1)	Spare

8.4.6 Command Control Blocks (5001 to 5016)

Command Control blocks place commands into the module's command priority queue. Unlike Event Command blocks, which contain all the values needed for one command, Command Control is used with commands already defined in the *MNET Client x Command List* in ProSoft Configuration Builder.

8.4.6.1 Block 5001 to 5016: Request from Processor to Module

Offset	Description
0	Command queue block identification code of 5001 to 5016
1	Client index (0 to 19) to be used
2	Command Index in the command list for the first command to be entered into the command queue
3 to 17	Command indexes of the next commands to be placed in the command queue
18 to (n-1)	Spare

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

8.4.6.2 Block 5001 to 5016: Response from Module to Processor

Offset	Description
0	Command queue block identification code of 5001 to 5016
1	The next write block ID
2	Client index (0 to 19) to be used
3	Number of commands in the block placed in the command queue. Return values: -2 = Client index is not valid. -1 = Client is not enabled and active.
4	Number of commands in queue
5 to (n-1)	Spare

8.4.7 Add Event with Data for Client Blocks (8000)

The 8000-series blocks are similar to the 2000-series Event Command blocks. The 8000-series blocks get the command data from the processor, instead of from the module's database. These blocks use "write" Modbus Function Codes (5, 6, 15, 16) only.

8.4.7.1 Block 8000: Request from Processor to Module

Offset	Description
0	Block ID 8000 indicates this block adds an event with data of a specific MVI69E-MBTCP client. The last two digits indicate which client to use. **Example:** '8015' uses client 15
1 to 4	IP address for the server for the message. Each digit (0 to 255) of the IP address is placed in one of the four registers.
5	TCP service port to use with the message
6	Modbus node address to use with the message
7	Modbus Function Code: 5, 6, 15 or 16 only
8	Modbus address in the slave device to associate with the command
9	Count value for operation: bit count for function 15 (1 to 2000 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.
10 to 59	Data values to be used by the command
60 to (n-1)	Spare

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

8.4.7.2 Block 8000: Response from Module to Processor

Offset	Description
0	Block ID 8000 for event command with data request
1	The next read request block identification code
2	Error Code for request: 0 = No error -1 = Client is not enabled -3 = Client is not active -4 = Client busy with previous event command -5 = Invalid Modbus command -6 = Invalid point count for command
3 to (n-1)	Spare

8.4.8 Get Event with Data Status Block (8100)

This block requests status data for Event with Data Commands.

8.4.8.1 Block 8100: Request from Processor to Module

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

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

8.4.8.2 Block 8100: Response from Module to Processor

Offset	Description
0	Block ID 8100 status data for Event with Data Commands
1	The next read request block identification code
2	Number of client records contained in block (0 to 19)
3	Client Index (0 to 19)
4	Error code for last command executed for client
5	Client Index (0 to 19)
6	Error code for last command executed for client
7 to 42	Data for other clients being reported
43 to (n-1)	Spare

8.4.9 Get General Module Status Data Block (9250)

This block requests general module status.

8.4.9.1 Block 9250: Request from Processor to Module

Offset	Description
0	Block ID 9250 to request the general module status response block

8.4.9.2 Block 9250: Response from Module to Processor

Offset	Description
0	Block ID 9250 requested by processor
1	The next read request block identification code
2	Program Scan Count: this value increments on every complete program cycle in the module.
3 to 4	Product Code: The two registers contain the product code of "MB6E" for the MVI69E-MBTCP.
5 to 6	Product Version: these two registers contain the product version for the current running software
7 to 8	Operating System: these two registers contain the month and year values for the program operating system.
9 to 10	Run Number: these two registers contain the run number value for the current software.
11	Read Block Count: total number of read blocks transferred from the module to the processor.
12	Write Block Count: total number of write blocks transferred from the processor to the module.
13	Parse Block Count: total number of blocks successfully parsed that were received from the processor.
14	Event Command Block Count: total number of Event Command blocks received from the processor.
15	Command Block Count: total number of command blocks received from the processor.
16	Error Block Count: Total number of block errors recognized by the module.
17	Client 0 command execution word: each bit in this word enables/disables the commands for client 0. If the bit is set, the command executes. If the bit is clear, the command is disabled
18 to 36	Client 1 to client 19 command execution words
37 to 38	Event Sequence Ready: bit mapped -1 bit for each client 0 to 19 Bit = 0: No event sequence status data ready Bit = 1: Event seq. status data ready
39	Encapsulated Modbus TCP/IP request count: this counter increments each time the module receives an Encapsulated Modbus TCP/IP (Service Port 2000) request from a remote Modbus TCP/IP client.
40	Encapsulated Modbus TCP/IP response count: this counter increments each time an Encapsulated Modbus TCP/IP (Service Port 2000) response is sent back to a remote Modbus TCP/IP client command.
41	Encapsulated Modbus TCP/IP error sent: this counter increments each time the server sends an error to the remote Modbus TCP/IP client.
42	Encapsulated Modbus TCP/IP error received: this counter increments each time an error is received from a remote Modbus TCP/IP client.
43	Modbus MBAP request count: this counter increments each time an MBAP (Service Port 502) request is received from a remote Modbus TCP/IP client.
44	Modbus MBAP response count: this counter increments each time an MBAP (Service Port 502) response is sent back to a remote Modbus TCP/IP client command.
45	Modbus MBAP error sent: this counter increments each time the server sends an error to the remote MBAP Modbus TCP/IP client.
46	Modbus MBAP error received: this counter increments each time an error is received from a remote MBAP Modbus TCP/IP client.
47 to (n-1)	Spare

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

8.4.10 Set Driver and Command Active Bits Block (9500)

This block enables and disables the Modbus TCP/IP clients and servers of the module.

8.4.10.1 Block 9500: Request from Processor to Module

Offset	Description
0	Block ID 9500 to set server and client enable/disable state
1	Server active state 0 = Disabled 1 = Enabled
2	Client 0 to client 15-bit map for active status of clients
3	Client 16 to client 19-bit map for active status of clients
4 to 23	Client 0 to client 19 command active bits. One word for each client with each bit used to turn on and off the commands for the client. 0 = Disabled 1 = Enabled
24 to (n-1)	Spare

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

8.4.10.2 Block 9500: Response from Module to Processor

Offset	Description
0	Block ID 9500 requested by processor
1	The next write block ID
2 to (n-1)	Spare

8.4.11 Get Driver and Command Active Bits Block (9501)

This block requests the active state of MBTCP Driver and Client commands.

8.4.11.1 Block 9501: Request from Processor to Module

Offset	Description
0	Block ID 9501 to get MBTCP Driver and command active status
1 to (n-1)	Spare

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

8.4.11.2 Block 9501: Response from Module to Processor

Offset	Description
0	Block ID 9501 requests the active state of MBTCP Driver and Client commands
1	The next write block ID
2	Server active state 0 = Disabled 1 = Enabled
3	Client 0 to 15 bit map for active status of clients
4	Client 16 to 19 bit map for active status of clients
5 to 24	Client 0 to client 19 command active bits. One word for each client with each bit used to turn on and off the commands for the client. 0 = Disabled 1 = Enabled
25 to (n-1)	Spare

8.4.12 Pass-Through Formatted Word Data Block for Functions 6 & 16 (9956)

If the server 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 MBTCP Add-On Instruction handles the receipt of all Modbus write functions and responds as expected to commands issued by the remote Modbus client device.

Note: Mutual exclusion on Pass-Through Block IDs 9956, 9957, 9958, and 9959 from all server connections. When multiple server connections are active and they receive write commands with the same Function Code, the same block identifier from the above list is needed. The module processes the command from the server which first received a command.

The module returns an Exception Code error code 6 (Node is busy - retry command later error) from the other server that received the command last. The client retries the command on the busy port after a short delay. This prevents Pass-Through blocks on multiple servers from overwriting each other.

8.4.12.1 Block 9956: Request from Module to Processor

Offset	Description
0	Read Block ID 9956
1	Write Block ID 9956
2	Number of word registers in the Modbus data set
3	Starting address for the Modbus data set
4 to 53	Modbus Data
54 to (n-1)	Spare

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

The ladder logic is responsible for parsing and copying the received message and performing the proper control operation as expected by the client device. The processor must then respond to the Pass-Through control block with an output image write block with the following format. This informs the module that the command has been processed and can be cleared from the Pass-Through queue.

8.4.12.2 Block 9956: Response from Processor to Module

Offset	Description
0	Write Block ID 9956
1 to (n-1)	Spare

8.4.13 Pass-Through Formatted Float Data Block for Functions 6 & 16 (9957)

8.4.13.1 Block 9957: Request from Module to Processor

Offset	Description
0	Read Block ID 9957
1	Write Block ID 9957
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to 53	Modbus Data
54 to (n-1)	Spare

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

The ladder logic is responsible for parsing and copying the received message and performing the proper control operation as expected by the client device. The processor must then respond to the Pass-Through block with a write block with the following format. This informs the module that the command has been processed and can be cleared from the Pass-Through queue.

8.4.13.2 Block 9957: Response from Processor to Module

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

8.4.14 Pass-Through Formatted Block for Function 5 (9958)

8.4.14.1.1 Block 9958: Request from Module to Processor

Offset	Description
0	Read Block ID 9958
1	Write Block ID: 9958
2	Number of word registers in the Modbus data set
3	Starting address for the Modbus data set
4 to 53	Modbus Data
54 to (n-1)	Spare

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

The ladder logic is responsible for parsing/copying the received message and performing the proper control operation as expected by the client device. The processor must respond to the Pass-Through control block with an output image write block with the following format. This informs the module that the command has been processed and can be cleared from the Pass-Through queue.

8.4.14.1.2 Block 9958: Response from Processor to Module

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

8.4.15 Pass-Through Formatted Block for Function 15 (9959)

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

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

8.4.15.1 Block 9959: Request from Module to Processor

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

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

The ladder logic is responsible for parsing and copying the received message and performing the proper control operation as expected by the client device. The processor must then respond to the Pass-Through control block with a write block with the following format. This informs the module that the command has been processed and can be cleared from the Pass-Through queue.

8.4.15.2 Block 9959: Response from Processor to Module

Offset	Description
0	Write Block ID 9959
1 to n	Spare

8.4.16 Pass-Through Formatted Block for Function 23 (9961)

8.4.16.1 Block 9961: Request from Module to Processor

Offset	Description
0	Read Block ID 9961
1	Write Block ID 9961
2	Number of word registers in Modbus data set
3	Starting address for Modbus data set
4 to 53	Modbus Data
54 to (n-1)	Spare

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

The ladder logic is responsible for parsing/copying the received message and performing the proper control operation as expected by the client device. The processor must respond to the Pass-Through control block with an output image write block with the following format. This informs the module that the command has been processed and can be cleared from the Pass-Through queue.

8.4.16.2 Block 9961: Response from Processor to Module

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

8.4.17 Pass-Through Block for Function 99 (9970)

8.4.17.1 Block 9970: Request from Module to Processor

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

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

The ladder logic is responsible for parsing/copying the received message and performing the proper control operation as expected by the client device. The processor must respond to the Pass-Through control block with an output image write block with the following format. This informs the module that the command has been processed and can be cleared from the Pass-Through queue.

8.4.17.2 Block 9970: Response from Processor to Module

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

8.4.18 Set Module Time Using Received Time Block (9972)

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

8.4.18.1 Block 9972: Request from Processor to Module

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

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

8.4.18.2 Block 9972: Response from Module to Processor

Offset	Description
0	Block ID 9972
1	Write Block ID
2	Return code: 0 = OK -1 = Error
3-n	Spare

8.4.19 Pass Module Time to Processor Block (9973)

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

8.4.19.1.1 Block 9973: Request from Processor to Module

Offset	Description
0	Block ID 9973
1-n	Spare

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

8.4.19.1.2 Block 9973: Response from Module to Processor

Offset	Description
0	Block ID 9973
1	Write Block ID
2	Year (0 to 9999)
3	Month (1 to 12)
4	Day (1 to 31)
5	Hour (0 to 23)
6	Minutes (0 to 59)
7	Seconds (0 to 59)
8	Milliseconds (0 to 999)
9 to (n-1)	Spare

8.4.20 Reset Status Block (9997)

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

8.4.20.1 Block 9997: Request from Processor to Module

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

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

8.4.20.2 Block 9997: Response from Module to Processor

Offset	Description
0	Block ID 9997
1	Write Block ID
2-n	Spare

8.4.21 Warm-boot Control Block (9998)

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

8.4.21.1 Block 9998: Request from Processor to Module

Offset	Description
0	Block ID 9998
1 to (n-1)	Spare

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

8.4.22 Cold-boot Control Block (9999)

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 MBTCP memory, error and status data.

8.4.22.1 Block 9999: Request from Processor to Module

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

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

8.5 Ethernet Port Connection

8.5.1 Ethernet Cable Specifications

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

The Ethernet port or ports on the module are Auto-Sensing. 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.

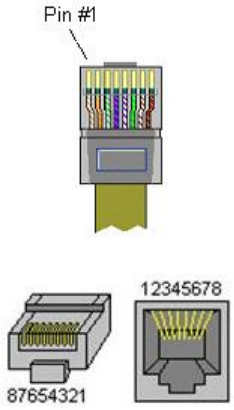
Some hubs have one input that can accept either a straight-through or crossover cable, depending on a switch position. In this case, you must ensure that the switch position and cable type agree.

Refer to Ethernet Cable Configuration (page 157) for a diagram of how to configure Ethernet cable.

8.5.1.1 Ethernet Cable Configuration

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

Crossover Cable	
RJ-45 PIN	**RJ-45 PIN**
1 Rx+	3 Tx+
2 Rx-	6 Tx-
3 Tx+	1 Rx+
6 Tx-	2 Rx-



Straight-through Cable	
RJ-45 PIN	**RJ-45 PIN**
1 Rx+	1 Tx+
2 Rx-	2 Tx-
3 Tx+	3 Rx+
6 Tx-	6 Rx-

8.5.1.2 Ethernet Performance

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

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

9 Support, Service, and Warranty

9.1 Contacting Technical Support

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

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

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

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

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

For additional ProSoft Technology contacts in your area, please see:
www.prosoft-technology.com/About-Us/Contact-Us

9.2 Warranty Information

For details regarding ProSoft Technology's legal terms and conditions, please see:
www.prosoft-technology.com/ProSoft-Technology-Legal-Terms-and-Conditions

For Return Material Authorization information, please see:
www.prosoft-technology.com/Services-Support/Return-Material-Instructions