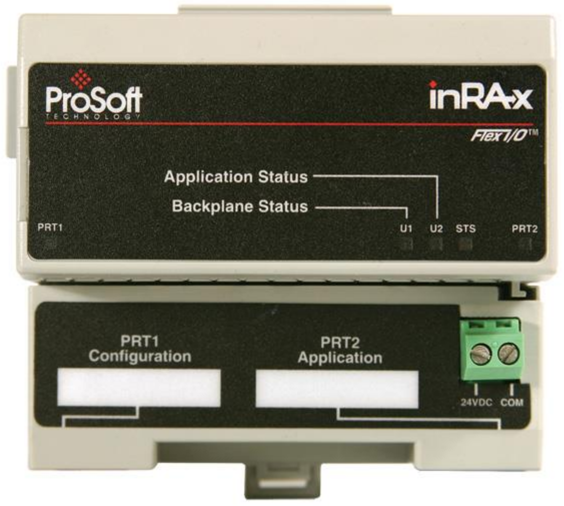
Where Automation Connects.



MVI94-MCM

Flex I/O Platform

Serial Communications Modbus Communication Module

December 5, 2016

USER MANUAL

Your Feedback Please

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

How to Contact Us

ProSoft Technology, Inc.
9201 Camino Media, Suite 200
Bakersfield, CA 93311
+1 (661) 716-5100
+1 (661) 716-5101 (Fax)
www.prosoft-technology.com

support@prosoft-technology.com

MVI94-MCM User Manual

December 5, 2016

ProSoft Technology® Product Documentation

In an effort to conserve paper, ProSoft Technology no longer includes printed manuals with our product shipments. User Manuals, Datasheets, Sample Ladder Files, and Configuration Files are provided on our website: **www.prosoft-technology.com**

Important Installation Instructions

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

WARNING - EXPLOSION HAZARD **-** SUBSTITUTION OF COMPONENTS MAY IMPAIR SUITABILITY FOR CLASS I, DIV. 2;

WARNING - EXPLOSION HAZARD **-** WHEN IN HAZARDOUS LOCATIONS, TURN OFF POWER BEFORE REPLACING OR WIRING MODULES

WARNING - EXPLOSION HAZARD **-** DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.

THIS DEVICE SHALL BE POWERED BY CLASS 2 OUTPUTS ONLY.

MVI (Multi Vendor Interface) Modules

WARNING - EXPLOSION HAZARD - DO NOT DISCONNECT EQUIPMENT UNLESS POWER HAS BEEN SWITCHED OFF OR THE AREA IS KNOWN TO BE NON-HAZARDOUS.

AVERTISSEMENT - RISQUE D'EXPLOSION - AVANT DE DÉCONNECTER L'ÉQUIPEMENT, COUPER LE COURANT OU S'ASSURER QUE L'EMPLACEMENT EST DÉSIGNÉ NON DANGEREUX.

Warnings

North America Warnings

A Warning - Explosion Hazard - Substitution of components may impair suitability for Class I, Division 2.
B Warning - Explosion Hazard - When in Hazardous Locations, turn off power before replacing or rewiring modules.
Warning - Explosion Hazard - Do not disconnect equipment unless power has been switched off or the area is known to be nonhazardous.
C Suitable for use in Class I, division 2 Groups A, B, C and D Hazardous Locations or Non-Hazardous Locations.

MVI94 Markings

Electrical Ratings

- Backplane Current Load: 800 mA @ 5 Vdc
- Operating Temperature: 0°C to 60°C (32°F to 140°F)
- Storage Temperature: -40°C to 85°C (-40°F to 185°F)
- Shock: 30g Operational; 50g non-operational; Vibration: 5 g from 10 Hz to 150 Hz
- Relative Humidity 5% to 95% without condensation)
- All phase conductor sizes must be at least 1.3 mm^2 and all earth ground conductors must be at least 4mm^2.

Label Markings

Agency Approvals and Certifications

cUL	C22.2 No. 213-1987

CE c UL us

Contents

1 Quick Start

This section describes the procedure to be followed for installing and configuring the module for communications. These steps should be followed for successful implementation of a module in a user application.

1 Define the communication characteristics of the Modbus port.
2 If configured as a Modbus master, define the command lists to be used.
3 Fill in the blank configuration form for application using the data sets defined in steps one and two.
4 Edit the configuration text file **94MCMM.CFG or 94MCMS.CFG** (using Notepad or some other text editor) to reflect the desired data from the configuration form and save the file under a different name. The example 94MCMM.CFG file is shipped in the module's memory, and is available from the www.prosoft-technology.com website.
5 Connect the module to a 24 Vdc power source.
6 Select the directory containing the correct configuration file on the computer.
7 Start the terminal emulation program on the computer.
8 Press the **[?]** key on the terminal to verify that the module is communicating with the computer and that the main menu mode is current.
9 Press the **[R]** key on the terminal emulator to select the receive option. Immediately press the [Y] key.
10 Press the **[ALT-F3]** key on the terminal emulator and enter name of the configuration file to load into the module **94MCMM.CFG or 94MCMS.CFG** if using one of the example files. The configuration will be downloaded, and the module will restart using the new configuration.
11 Connect the module's Modbus port to the Modbus network. If everything is configured correctly and the cable connections are correct, communications should be present on the port.
12 Monitor the communication statistics for the port to verify that everything is working correctly.
13 View the virtual Modbus database in the module using the terminal emulator.
14 Create the ladder logic program for your system. An example ladder program is available on the www.prosoft-technology.com web site. This logic is responsible for transferring the data between the module and processor.
15 Connect the module to the ControlNet network or Flex I/O communication adapter. If all is configured correctly, the data in the module should be visible in the processor.
16 Use the Configuration/Debug port to view the backplane transfer statistics.

2 Hardware Installation

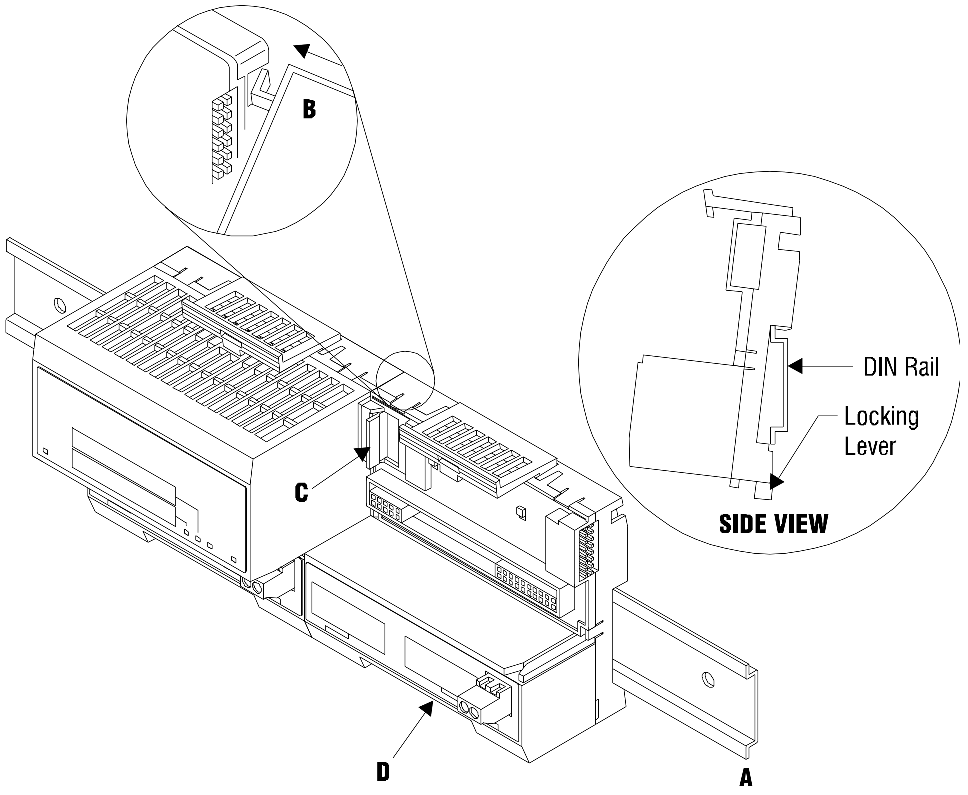
In This Chapter

2.1 Verify Package Contents

Make sure that you verify the contents of the product before you discard the packing material. The following components should be included with the product:

1 A MVI94 Flex I/O Base
2 A MVI94 Module with 3 jumpers installed
3 One Serial Adapter Cable

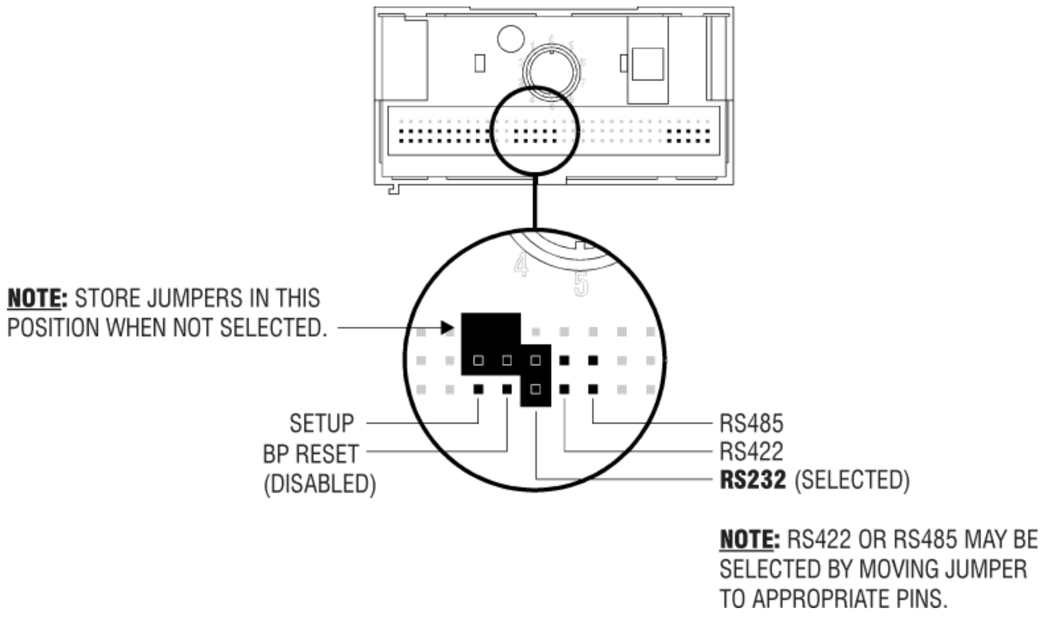
2.2 Mounting the MVI 94 Flex I/O Base



1 Remove the cover plug (if used) in the male connector of the unit to which you are connecting this Base.
2 Check to make sure that the 16 pins in the male connector on the adjacent device are straight and in line so that the mating female connector on this Base will mate correctly.
3 Make certain that the female flexbus connector **C** is **fully retracted** into the Base.
4 Position the Base on a 35 x 7.5mm DIN-rail **A** at a slight angle with the hook **B** on the left side of the Base hooked into the right side of the unit on the left.
5 Rotate the Base onto the DIN-rail with the top of the rail hooked under the lip on the rear of the Base. Use caution to make sure that the female flexbus connector does not strike any of the pins in the mating male connector.
6 Press the terminal base down onto the DIN-rail until flush. The locking tab **D** snaps into position and locks the terminal base to the DIN-rail.
7 If the Base does not lock in place, use a screwdriver or similar device to move the locking tab down, press the Base flush with the DIN-rail and release the locking tab to lock the base in place.
8 **Gently** push the female flexbus **C** connector into the adjacent base or adapter male connector to complete the flexbus connections.

2.3 Setting Jumpers

Before installing the MVI94 module onto its base, the module's configuration can be set using the jumpers on the bottom of the module as shown in this figure.


NOTE: STORE JUMPERS IN THIS POSITION WHEN NOT SELECTED.
SETUP
BP RESET (DISABLED)
RS485
RS422
RS232 (SELECTED)
NOTE: RS422 OR RS485 MAY BE SELECTED BY MOVING JUMPER TO APPROPRIATE PINS.

Port 2 RS-232/422/485: Select with jumper (shipped in 232).

BP Reset: If the MVI94 module is to be reset when the Flex Bus is reset, install the BP RESET jumper in the Enabled position.

ATTENTION: Do not remove or replace a base unit when power is applied. Interruption of the flexbus can result in unintended operation or machine motion.

SETUP: To place the module in SETUP mode, install the jumper in the Selected position (DOS default). To prevent the module from being in Setup mode, leave the jumper in the disabled position.

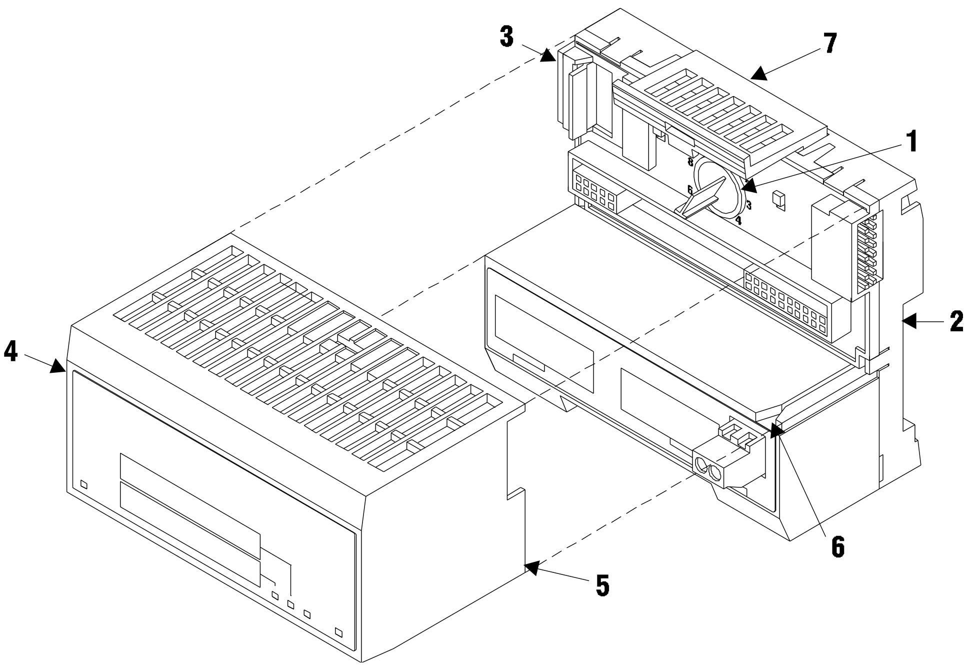
2.4 Installing the Module onto the Base

1 Rotate the keyswitch **1** on the Base clockwise to position #1.

2 Make certain the flexbus connector **3** on the Base is pushed all the way to the left to connect with the neighboring base or adapter. **The Module cannot be installed unless the flexbus connector is fully extended.**

3 Make sure that the pins on the bottom of the Module are straight so they will align properly with the connector socket on the Base.

4 Position the Module with its alignment bar 5 aligned with the groove 6 on the Base.

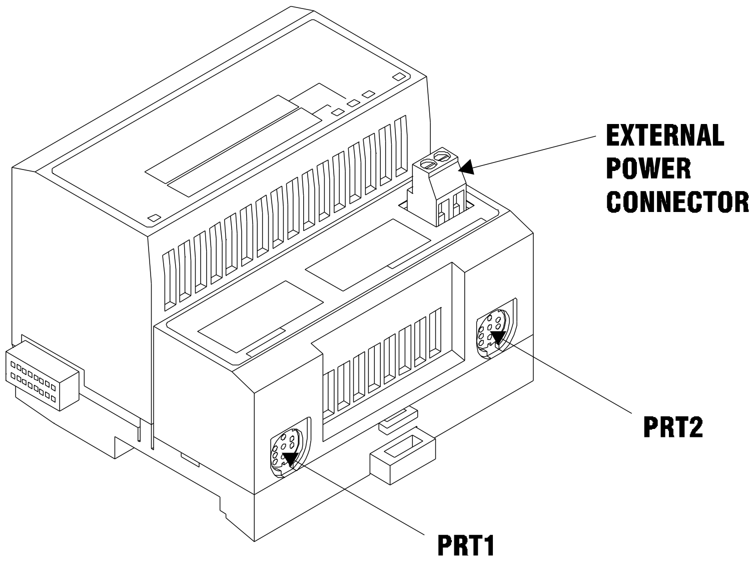


5 Press firmly and evenly to seat the Module in the Base. The Module is seated when the latch **7** on the Base is locked into the Module.

2.5 Installing the Serial Adapter Cables

Two identical serial adapter cables are supplied. Each cable has a locking-type 8 pin Mini-DIN plug on one end and a DB-9 male connector on the other end. The Mini-DIN connector on each cable is inserted into the Mini-DIN receptacles marked **PRT1** and **PRT2** on the Base.

To install the locking-type Mini-DIN connector, slide the spring-loaded sleeve back while inserting the plug into the receptacle on the Base, and then release the sleeve when fully seated. The locking mechanism prevents the cable from being removed during normal operation. To remove the cable, slide the sleeve back and remove the plug.



2.6 Wiring the Power Connections

External power is supplied to the Base on the 2 pin screw terminal block. The power supply can be either 24Vdc or 12Vdc, and should be located in close proximity of the base.

- Connect dc common to the **COM** terminal
- Connect +24V dc or +12V dc to the **24VDC** terminal

3 Configuration

In This Chapter

3.1 [Module]

[Section]/Item	Value	Range	Description
[Module]			Configuration header for Module.
Module Name:		Up to 80 chars	Name of the module to use on reports. Use this parameter to identify your module in your system.
Maximum Register:		1 to 3996	This parameter defines the maximum register in the virtual Modbus database. You should size the database for your application, leaving room for expansion in the future. Requests for registers outside of the range selected are returned with an error message.
Error/Status Block Pointer:		-1 to 3995	This value represents the relative starting position in the module's internal Modbus database where the Error/Status data is stored. The table can be placed anywhere in the module's data space. The content of the Error/Status table is updated at the frequency defined in the following parameter. If a value of -1 is set for the parameter, the data is not placed in the database.
Error/Status Frequency:		0 to 65535	This parameter specifies the number of program cycles between each update of the Error/Status block data in the module. If the parameter is set to a value of 0, the data is never updated.
BT Read Start Register:		0 to 3995	This parameter specifies the starting register in the internal Modbus database to write over the backplane.
BT Read Register Count:		12 to 3996	This parameter specifies the number of registers in the internal Modbus database to write over the backplane. This parameter computes the number of blocks to transfer from the module to the backplane. The number of blocks must be >=2 for proper backplane data transfer.
BT Write Start Register:		0 to 3995	This parameter specifies the starting register in the internal Modbus database to fill with data read over the backplane.
BT Write Register Count:		12 to 3996	This parameter specifies the number of registers in the internal Modbus database to consider from the read operations over the backplane. This parameter computes the number of blocks to transfer from the backplane to the module. The number of blocks must be >=2 for proper backplane data transfer.
Backplane Fail Count:		0 to 65535	This parameter specifies the number of consecutive backplane transfer failure before communications is disabled. If the parameter is set to a value of 0, communications is not disabled.

3.2 [Port]

The [PORT] section of the configuration file sets the Modbus port communication parameters. The following table lists the parameters defined in this section.

[Section]/Item	Value	Range	Description
[PORT]			Configuration header for communication port.
Type:		0, 1, 3, 4	This parameter specifies if the port will emulate a Modbus master device (0), a Modbus slave device without pass-through (1), a Modbus slave device with pass-through and data swapping (3), or a Modbus slave device with pass-through (4).
Float Flag		Yes or No	This flag specifies if the floating point data access functionality is to be implemented. If the float flag is set to Yes, Modbus functions 3, 6, and 16 will interpret floating point values for registers as specified by the following two parameters.
Float Start		0 to 32767	This parameter defines the first register of floating point data. All requests with register values greater than or equal to this value will be considered floating point data requests. This parameter is only used in the Float Flag is enabled. For example, if a value of 7000 is entered, all requests for registers 7000 and above are considered as floating point data.
Float Offset		0 to 3995	This parameter defines the start register for floating point data in the internal database. This parameter is only used if the Float Flag is enabled. For example, if the Float Offset value is set to 3000 and the Float Start parameter is set to 7000, data requests for register 7000 will use the internal Modbus register 3000.
Protocol		0 or 1	This parameter specifies the Modbus protocol to be used on the port. Valid protocols are 0 = Modbus RTU and 1 = Modbus ASCII.
Baud Rate:		110 to 115K	This is the baud rate to be used on the port. Enter the baud rate as a value. For example, to select 19K baud, enter 19200. Valid entry for this field include: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, and 115.
Parity:		None, Odd, or Even	This is the parity setting for the port.
Data Bits:		5 or 8	This parameter sets the number of data bits for each word used by the protocol.
Stop Bits:		1 or 2	This parameter sets the number of stop bits for each data value sent.
RTS On:		0 to 65535	This parameter sets the number of milliseconds to delay after RTS is asserted before the data is transmitted.
RTS Off:		0 to 65535	This parameter sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal will be set low.

[Section]/Item	Value	Range	Description
Minimum Response Delay:		0 to 65535	This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices.
Use CTS Line:		Yes or No	This parameter specifies if the CTS modem control line is to be used. If the parameter is set to No, the CTS line will not be monitored. If the parameter is set to Yes, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire).

3.3 [Modbus Master]

The [Modbus Master] section of the configuration file defines the command list specifications for the master port. The following table describes the parameters defined in this section:

[Modbus Master]		**Configuration header for Modbus Master.**
Command Count:	0 to 100	This parameter specifies the number of commands to be processed by the Modbus Master port.
Minimum Command Delay:	0 to 65535	This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries.
Command Error Pointer:	-1 to 3995	This parameter sets the address in the internal Modbus database where the command error data is placed. If the value is set to -1, the data is not transferred to the database.
Response Timeout:	0 to 65535	This parameter represents the message response timeout period in 1 mSec increments. This is the time that a port configured as a master will wait before re-transmitting a command if no response is received from the addressed slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network.
Retry Count	0 to 10	This parameter specifies the number of times a command will be retried if it fails.
Error Delay Count	0 to 65535	This parameter specifies the number of polls to skip on the slave before trying to re-establish communications. After the slave fails to respond, the master will skip commands to be sent to the slave the number of times entered in this parameter.

3.4 [Modbus Slave]

The [Modbus Slave] section of the configuration file defines the slave address and the data storage offsets for the slave port. The following table lists the parameters defined in this section.

[Modbus Slave]		Configuration header for Modbus Slave
Internal Slave ID	0 to 255	This parameter defines the virtual Modbus slave address for the internal database. Any requests received by the port with this address will be processed by the module. Each device must have a unique address on a network.
Bit Input Offset	0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 2 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
Word Input Offset	0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 4 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
Output Offset	0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 1, 5, or 15 commands. For example, if the value is set to 100, an address request of 0 will return the value at register 100 in the database.
Holding Register Offset	0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus functions 3, 6, or 16 commands. For example, if the value is set to 50, and address request of 0 will return the value at register 50 in the database.
Use Guard Band Timer	Y or N	This parameter specifies if the packet Guard Band is to be used. This is used for multi-drop only.
Guard Band Timeout	0 to 65535	This parameter specifies the Guard Band time between packets in multi-drop slave mode. A value of 0 uses the default time. A value of 1 to 65535 sets the time in milliseconds.

3.5 Floating-Point Data Handling (Modbus Master)

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

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

The MVI94-MCM module can be used with many different processors. The following describes using the module with a ControlLogix processor.

3.5.1 Floating-Point Data Handling (Modbus Master)

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

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

Read Floating-Point Data

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

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

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

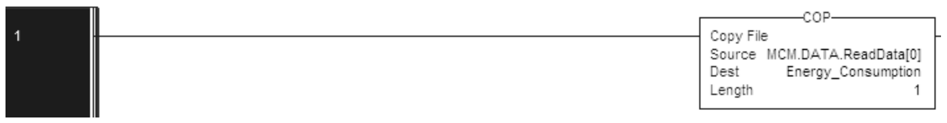
Parameter	Value	Description
Enable	1	Sends the command every time through the command list.
IntAddress	1000	Places data at address 1000 of the module memory. Based on the configuration in ModDef this will put the data at the tag **MCM.DATA.READDATA[0]**.
PollInt	0	No delay for this command.
Count	2	Reads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

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

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

[+]	Energy_Consumption	REAL[1]	Float

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



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

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

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

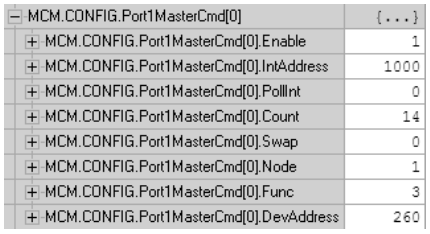
Note: Do not use a MOV statement. A MOV will convert the data from the Source register to the destination register data type. This would create a data casting statement and will result in the loss or corruption of the original data.

Read Multiple Floating-Point Registers

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

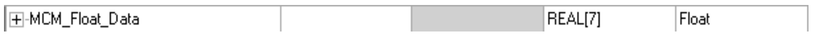
Value		Description	Type
40261	KW	Demand (power)	Float. upper 16 bits
40263	VAR	Reactive Power	Float. upper 16 bits
40265	VA	Apparent Power	Float. upper 16 bits
40267		Power Factor	Float. upper 16 bits
40269	VOLTS	Voltage, line to line	Float. upper 16 bits
40271	VOLTS	Voltage, line to neutral	Float. upper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

Configure the command to read these 7 floats as follows.



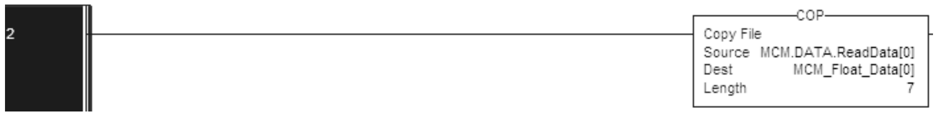
MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	1000
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	14
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func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

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



MCM_Float_Data			REAL[7]	Float

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



```
2
                COP
Copy File
Source  MCM.DATA.ReadData[0]
Dest      MCM_Float_Data[0]
Length                    7
```

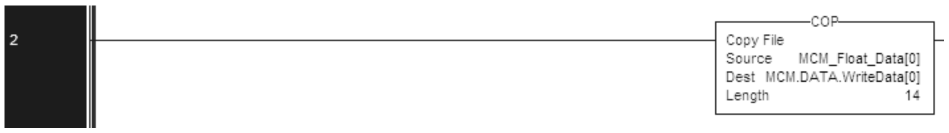
The "Length" parameter is set to the number of Floating-Point values that must be copied from the **MCM.DATA.READDATA** array.

Write Floats to Slave Device

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

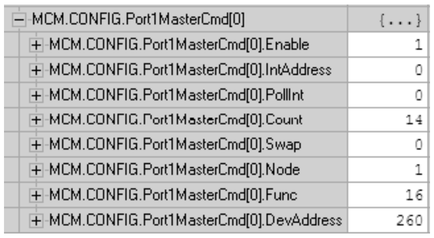
Value		Description	Type
40261	KW	Demand (power)	Float. upper 16 bits
40263	VAR	Reactive Power	Float. upper 16 bits
40265	VA	Apparent Power	Float. upper 16 bits
40267		Power Factor	Float. upper 16 bits
40269	VOLTS	Voltage, line to line	Float. upper 16 bits
40271	VOLTS	Voltage, line to neutral	Float. upper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

You must use a **COP** statement to copy the data from floating-point data tags within the ControlLogix processor, into the **MCM.DATA.WRITEDATA** array used by the MVI94-MCM module. Below is an example.



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

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



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

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

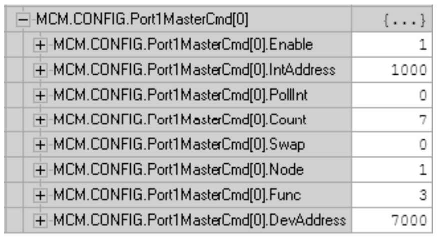
Read Floats with Single Modbus Register Address (Enron/Daniel Float)

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

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

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

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

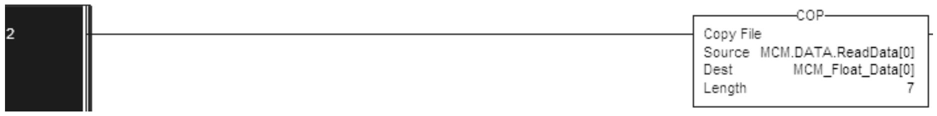


MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	1000
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	7
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func	3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

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

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

The **COP** statement for this type of data is the same as shown in Read Multiple Floating-Point Registers (page 28).



```
2                                         COP
                              Copy File
                              Source  MCM.DATA.ReadData[0]
                              Dest         MCM_Float_Data[0]
                              Length                     7
```

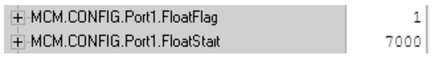
Write to Enron/Daniel Floats

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

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

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

Configure the Float Start and Float Flag parameters as shown.

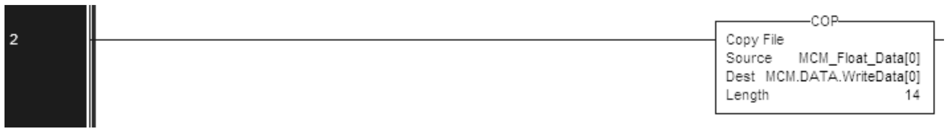


MCM.CONFIG.Port1.FloatFlag	1
MCM.CONFIG.Port1.FloatStart	7000

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

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

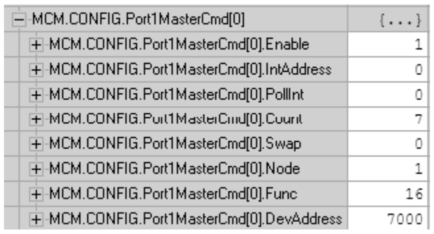
1 First, copy the floating-point data from the ControlLogix processor into the **MCM.DATA.WRITEDATA** array used by the MVI94-MCM module. Below is an example.



Copy File
Source MCM_Float_Data[0]
Dest MCM.DATA.WriteData[0]
Length 14

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

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



MCM.CONFIG.Port1MasterCmd[0]	{...}
MCM.CONFIG.Port1MasterCmd[0].Enable	1
MCM.CONFIG.Port1MasterCmd[0].IntAddress	0
MCM.CONFIG.Port1MasterCmd[0].PollInt	0
MCM.CONFIG.Port1MasterCmd[0].Count	7
MCM.CONFIG.Port1MasterCmd[0].Swap	0
MCM.CONFIG.Port1MasterCmd[0].Node	1
MCM.CONFIG.Port1MasterCmd[0].Func	16
MCM.CONFIG.Port1MasterCmd[0].DevAddress	7000

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

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

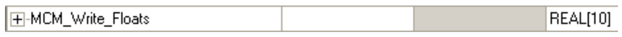
3.5.2 Floating-Point Data Handling (Modbus Slave)

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

1 Copy the data to and from the MVI94-MCM module with a tag configured as a data type REAL in the ControlLogix processor.

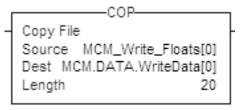
Each floating-point value will occupy 2 registers on the Modbus network. Some Master devices use Enron or Daniel Float data. These types of floats require one Modbus register for each float in the module memory. If your Master requires this addressing, refer to the following section.
For standard floating-point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

2 First, configure a tag within the FLEX processor.



⊞-MCM_Write_Floats			REAL[10]

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

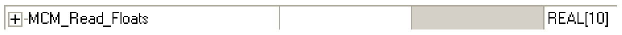


```
COP
Copy File
Source   MCM_Write_Floats[0]
Dest  MCM.DATA.WriteData[0]
Length                    20
```

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

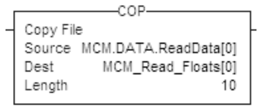
To copy data from the MVI94-MCM module to a floating-point tag within the FLEX processor

1 Configure a tag within the FLEX processor as shown.



MCM_Read_Floats REAL[10]

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



COP
Copy File
Source MCM.DATA.ReadData[0]
Dest MCM_Read_Floats[0]
Length 10

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

Enron/Daniel Float Configuration

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

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

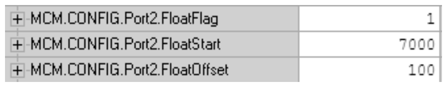
For example, an Enron Float device is required to access address 47001 for floating-point data, and each Modbus register would emulate a single float value (does not require 2 Modbus addresses for 1 float value).

A Master device requiring this type of addressing, would require that for every count of 1, the MVI94-MCM module responds to the request message with 4 bytes (one 32-bit REAL) value.

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

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

Here is a sample configuration for the module.



MCM.CONFIG.Port2.FloatFlag	1
MCM.CONFIG.Port2.FloatStart	7000
MCM.CONFIG.Port2.FloatOffset	100

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

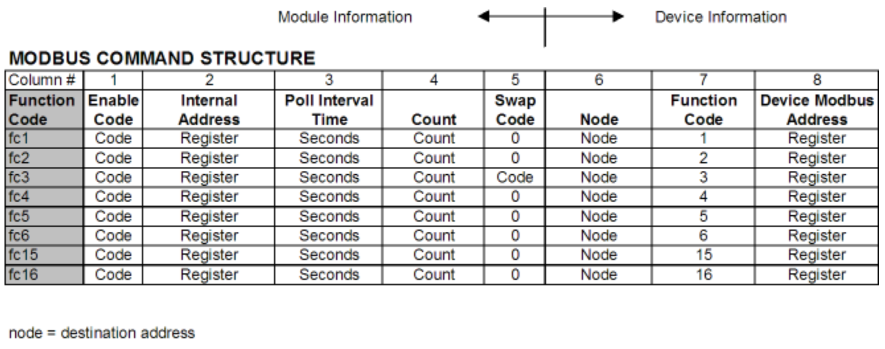
Module Address	**Modbus Address**	**Tag Address**
100	47001	MCM.DATA.WriteData[100]
102	47002	MCM.DATA.WriteData[102]
104	47003	MCM.DATA.WriteData[104]
110	47006	MCM.DATA.WriteData[110]
120	47011	MCM.DATA.WriteData[120]
200	47051	MCM.DATA.WriteData[200]
300	47101	MCM.DATA.WriteData[300]
500	47201	MCM.DATA.WriteData[500]

3.6 [Commands]

The [COMMANDS] section of the configuration file sets the Modbus master port command list. This list polls Modbus slave devices attached to the Modbus master port. The module supports numerous commands. This permits the module to interface with a wide variety of Modbus protocol devices.

The command list is formatted differently than the other sections of the configuration file. Commands are present in a block between the labels **START** and **END**. These labels inform the program where the list resides. The module's program will parse all commands after the **START** label until it reaches the **END** label or until the command count entered for the port is reached.

The function codes used for each command are those specified in the Modbus protocol. Each command list record has the same format. The first part of the record contains the information relating to the MVI94-MCM, communication module and the second part contains information required to interface to the Modbus slave device. The command structure is displayed in the following table for all functions supported:



Module Information ← | → Device Information

MODBUS COMMAND STRUCTURE

Column #	1	2	3	4	5	6	7	8
Function Code	**Enable Code**	**Internal Address**	**Poll Interval Time**	**Count**	**Swap Code**	**Node**	**Function Code**	**Device Modbus Address**
fc1	Code	Register	Seconds	Count	0	Node	1	Register
fc2	Code	Register	Seconds	Count	0	Node	2	Register
fc3	Code	Register	Seconds	Count	Code	Node	3	Register
fc4	Code	Register	Seconds	Count	0	Node	4	Register
fc5	Code	Register	Seconds	Count	0	Node	5	Register
fc6	Code	Register	Seconds	Count	0	Node	6	Register
fc15	Code	Register	Seconds	Count	0	Node	15	Register
fc16	Code	Register	Seconds	Count	0	Node	16	Register

node = destination address

Each parameter is discussed in the following topics.

3.6.1 Enable Code

This field defines whether the command is to be executed and under what conditions. If the parameter is set to 0, the command is disabled and will not be executed in the normal polling sequence. The command can be executed under the control of the processor using a Command Control block. Setting the parameter to a value of 1 for the command causes the command to be executed each scan of the command list if the Poll Interval Time is set to zero. If the Poll Interval time is set, the command will be executed, when the interval timer expires. If the parameter is set to 2, the command will execute only if the internal data associated with the command changes. This value is valid only for write commands. If the parameter is set to 3, it will function in the same manner as an enable code of 1except that the floating-point is disabled for this command. A parameter value of 4 is the same as a parameter value of 2 but with floating-point handling disabled for the command.

3.6.2 Internal Address

This field specifies the virtual Modbus database register to be associated with the command. If the command is a read function, the data read from the slave device will be placed starting at the register value entered in this field. If the command is a write function, the data written to the slave device will be sourced from the address specified. Register addresses specified for commands must reside in the range specified by the Maximum Register parameter under the [MODBUS MASTER] section.

3.6.3 Poll Interval Time

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

3.6.4 Count

This parameter specifies the number of registers or digital points to be associated with the command. Functions 5 and 6 ignore this field as they only apply to a single data point. For functions 1, 2 and 15, this parameter sets the number of digital points (inputs or coils) to be associated with the command. For functions 3, 4 and 16, this parameter sets the number of registers to be associated with the command.

3.6.5 Swap Code

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

Swap Code	Description
0	None - No Change is made in the byte ordering (1234 = 1234)
1	Words - The words are swapped (1234=3412)
2	Words & Bytes - The words are swapped then the bytes in each word are swapped (1234=4321)
3	Bytes - The bytes in each word are swapped (1234=2143)

3.6.6 Node

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

3.6.7 Function Code

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

Func	Description
1	Read Coil Status
2	Read Input Status
3	Read Holding Registers
4	Read Input Registers
5	Single Coil Write
6	Single Register Write
15	Multiple Coil Write
16	Multiple Register Write

3.6.8 Device MODBUS Address

This parameter specifies the starting Modbus register or digital point address to be considered by the command in the Modbus slave device. Refer to the documentation of each Modbus slave device on the network for their register and digital point address assignments.

The FC determines the addresses range and that this value will be the register or bit OFFSET into a given data range. For example, if the command is to be a bit command (FC 1, 2, 5, or 15) to Read/Write a Coil 0X address 00001, then the value to enter here would be 0. For Coil address 00110, the value here would be 109. For register Read/Write commands (FC 3, 4, 6, or 16) in the 3X (FC4) or 4X (FC3), say 30001 or 40001, the value here would, again be 0. For 31101 or 41101, the value to enter for this parameter would be 1100.

3.7 Uploading and Downloading the Configuration File

ProSoft modules are shipped with a pre-loaded configuration file. In order to edit this file, you may transfer the file from the module to your PC or locate the file at www.prosoft-technology.com. After editing, you must transfer the file back to the module for your changes to take effect.

This section describes these procedures.

Important: The illustrations of configuration/debug menus in this section are intended as a general guide and may not exactly match the configuration/debug menus in your own module. For specific information about the configuration/debug menus in your module, refer to The Configuration/Debug Menu (page 45).

3.7.1 Required Hardware

You can connect directly from your PC's serial port to the serial port on the module to view configuration information, perform maintenance, and send or receive configuration files.

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

- 80486 based processor (Pentium preferred)
- 1 megabyte of memory
- At least one UART hardware-based serial communications port available. USB-based virtual UART systems (USB to serial port adapters) often do not function reliably, especially during binary file transfers, such as when uploading/downloading configuration files or module firmware upgrades.

3.7.2 Required Software

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

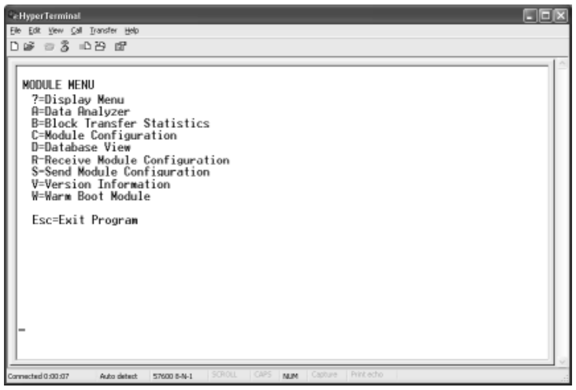
A simple communication program called HyperTerminal is pre-installed with recent versions of Microsoft Windows operating systems. If you are connecting from a machine running DOS, you must obtain and install a compatible communication program. The following table lists communication programs that have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

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

The module uses the Ymodem file transfer protocol to send and receive configuration files from your module. If you use a communication program that is not on the list above, please be sure that it supports Ymodem file transfers.

3.7.3 Transferring the Configuration File to Your PC

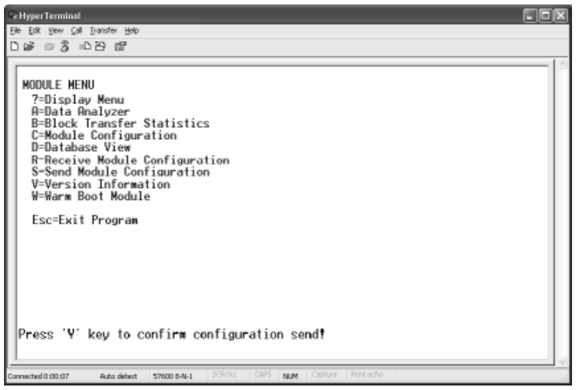
1 Connect your PC to the Configuration/Debug port of the module using a terminal program such as HyperTerminal. Press **[?]** to display the main menu.


HyperTerminal
File Edit View Call Transfer Help

```
MODULE MENU
  ?=Display Menu
  A=Data Analyzer
  B=Block Transfer Statistics
  C=Module Configuration
  D=Database View
  R=Receive Module Configuration
  S=Send Module Configuration
  V=Version Information
  W=Warm Boot Module

  Esc=Exit Program
```

Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

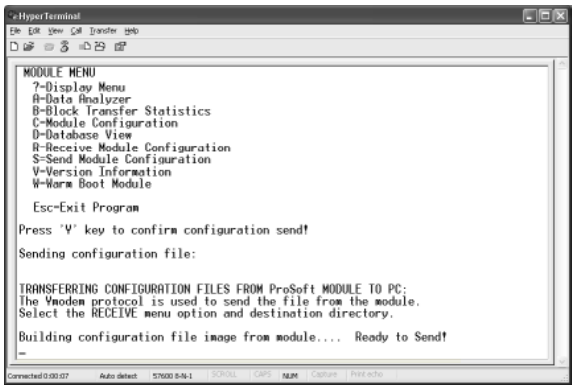
2 Press **[S]** (Send Module Configuration). The message "Press Y key to confirm configuration send!" is displayed at the bottom of the screen.


HyperTerminal
File Edit View Call Transfer Help

```
MODULE MENU
  ?=Display Menu
  A=Data Analyzer
  B=Block Transfer Statistics
  C=Module Configuration
  D=Database View
  R=Receive Module Configuration
  S=Send Module Configuration
  V=Version Information
  W=Warm Boot Module

  Esc=Exit Program



Press 'Y' key to confirm configuration send!
```

Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

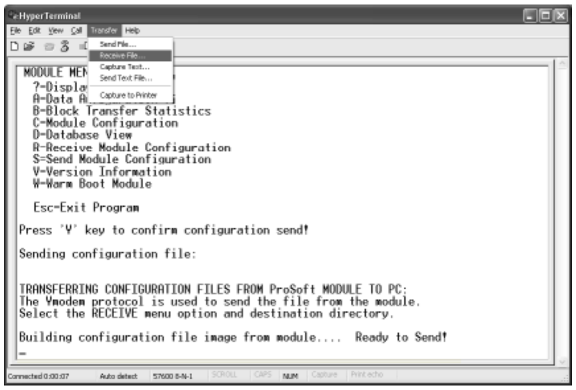
3 Press **[Y]**. The screen now indicates that the module is ready to send.



```
MODULE MENU
  ?=Display Menu
  A=Data Analyzer
  B=Block Transfer Statistics
  C=Module Configuration
  D=Database View
  R=Receive Module Configuration
  S=Send Module Configuration
  V=Version Information
  W=Warm Boot Module

  Esc=Exit Program

Press 'Y' key to confirm configuration send!

Sending configuration file:


TRANSFERRING CONFIGURATION FILES FROM ProSoft MODULE TO PC:
The Ymodem protocol is used to send the file from the module.
Select the RECEIVE menu option and destination directory.

Building configuration file image from module....  Ready to Send!
_
```

Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

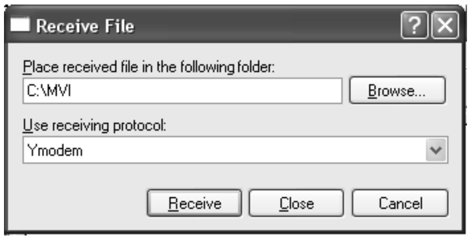
4 From the **Transfer** menu in HyperTerminal, select **Receive File**. This action opens the *Receive File* dialog box.



```
MODULE MEN
  ?-Displa
  A-Data A
  B-Block Transfer Statistics
  C-Module Configuration
  D-Database View
  R-Receive Module Configuration
  S=Send Module Configuration
  V-Version Information
  W-Warm Boot Module

  Esc-Exit Program

Press 'Y' key to confirm configuration send!

Sending configuration file:


TRANSFERRING CONFIGURATION FILES FROM ProSoft MODULE TO PC:
The Ymodem protocol is used to send the file from the module.
Select the RECEIVE menu option and destination directory.

Building configuration file image from module....  Ready to Send!
_
```

Connected 0:00:07 | Auto detect | 57600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

5 Use the Browse button to choose a folder on your computer to save the file.

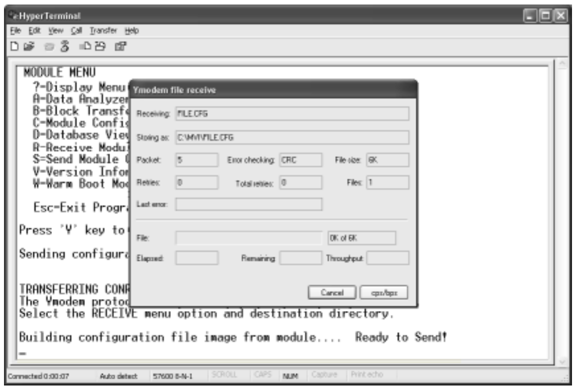


Place received file in the following folder: C:\MVI

Use receiving protocol: Ymodem

- **Note:** ProSoft Technology suggests that you upload the configuration file pre-loaded on your module. However, configuration files are also available at **www.prosoft-technology.com**.

6 Select Ymodem as the receiving protocol.
7 Click the Receive button. This action opens the *Ymodem File Receive* dialog box, showing the progress of your file transfer.

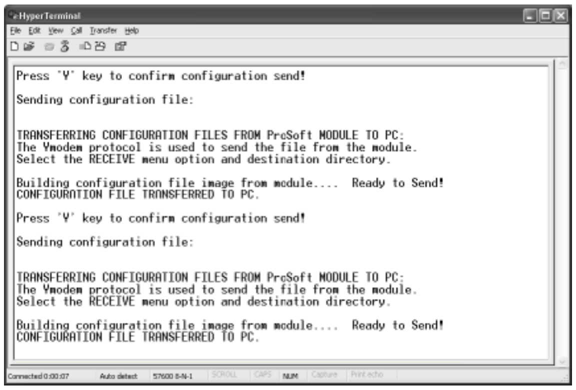


```
MODULE MENU
  ?-Display Menu
  A-Data Analyzer
  B-Block Transf
  C-Module Confi
  D-Database Vie
  R-Receive Modu
  S=Send Module
  V-Version Info
  W-Warm Boot Mo

  Esc-Exit Progr

Press 'Y' key to

Sending configur


TRANSFERRING CON
The Ymodem protoc
Select the RECEIVE menu option and destination directory.

Building configuration file image from module....  Ready to Send!
_
```

Receiving: FILE.CFG
Storing as: C:\MVI\FILE.CFG
Packet: 5 Error checking: CRC File size: 6K
Retries: 0 Total retries: 0 Files: 1
Last error:
File: 0K of 6K
Elapsed: Remaining: Throughput:

Connected 0:00:07 | Auto detect | 57600 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

When the configuration file has been transferred to your PC, the dialog box will indicate that the transfer is complete.



```
Press 'Y' key to confirm configuration send!

Sending configuration file:


TRANSFERRING CONFIGURATION FILES FROM ProSoft MODULE TO PC:
The Ymodem protocol is used to send the file from the module.
Select the RECEIVE menu option and destination directory.

Building configuration file image from module....  Ready to Send!
CONFIGURATION FILE TRANSFERRED TO PC.

Press 'Y' key to confirm configuration send!

Sending configuration file:


TRANSFERRING CONFIGURATION FILES FROM ProSoft MODULE TO PC:
The Ymodem protocol is used to send the file from the module.
Select the RECEIVE menu option and destination directory.

Building configuration file image from module....  Ready to Send!
CONFIGURATION FILE TRANSFERRED TO PC.
```

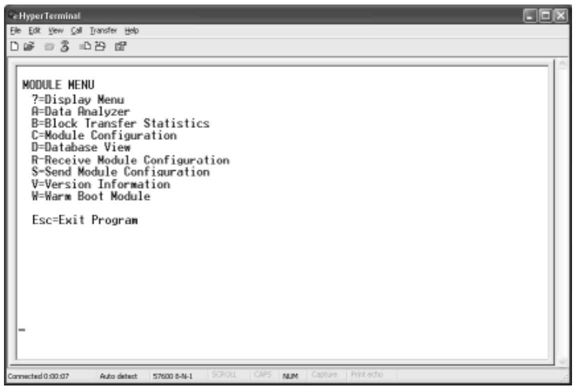
The configuration file is now on your PC at the location you specified.

8 You can now open and edit the file in a text editor such as Notepad. When you have finished editing the file, save it and close Notepad.

3.7.4 Transferring the Configuration File to the Module

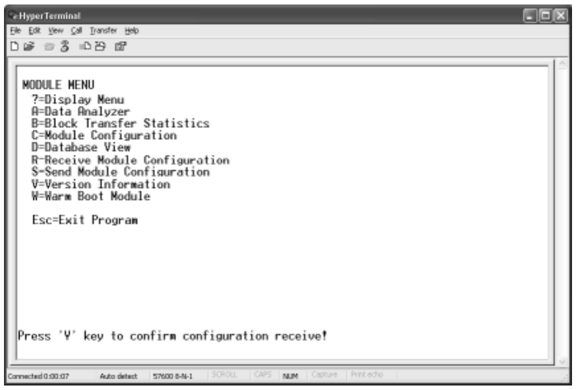
Perform the following steps to transfer a configuration file from your PC to the module.

1 Connect your PC to the Configuration/Debug port of the module using a terminal program such as HyperTerminal. Press **[?]** to display the main menu.



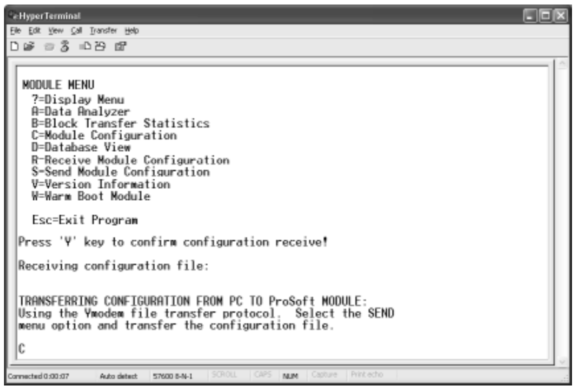
```
MODULE MENU
  ?=Display Menu
  A=Data Analyzer
  B=Block Transfer Statistics
  C=Module Configuration
  D=Database View
  R=Receive Module Configuration
  S=Send Module Configuration
  V=Version Information
  W=Warm Boot Module

  Esc=Exit Program
```

2 Press **[R]** (Receive Module Configuration). The message "Press Y key to confirm configuration receive!" is displayed at the bottom of the screen.


HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

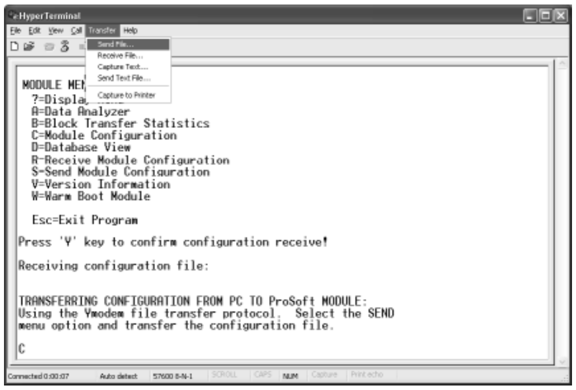
```
MODULE MENU
 ?=Display Menu
 A=Data Analyzer
 B=Block Transfer Statistics
 C=Module Configuration
 D=Database View
 R=Receive Module Configuration
 S=Send Module Configuration
 V=Version Information
 W=Warm Boot Module

 Esc=Exit Program

Press 'Y' key to confirm configuration receive!
```

3 Press **[Y]**. The screen now indicates that the PC is ready to send.


HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

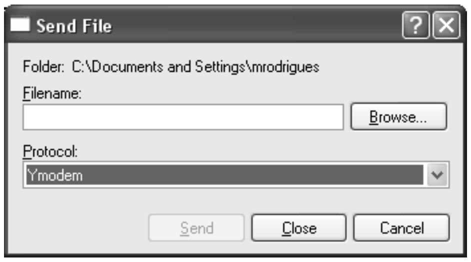
```
MODULE MENU
 ?=Display Menu
 A=Data Analyzer
 B=Block Transfer Statistics
 C=Module Configuration
 D=Database View
 R=Receive Module Configuration
 S=Send Module Configuration
 V=Version Information
 W=Warm Boot Module

 Esc=Exit Program
Press 'Y' key to confirm configuration receive!

Receiving configuration file:


TRANSFERRING CONFIGURATION FROM PC TO ProSoft MODULE:
Using the Ymodem file transfer protocol.  Select the SEND
menu option and transfer the configuration file.

C
```

4 From the **Transfer** menu in HyperTerminal, select **Send File**.


HyperTerminal
File Edit View Call Transfer Help
Send File...
Receive File...
Capture Text...
Send Text File...
Capture to Printer
Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

```
MODULE MEN
 ?=Displa
 A=Data Analyzer
 B=Block Transfer Statistics
 C=Module Configuration
 D=Database View
 R=Receive Module Configuration
 S=Send Module Configuration
 V=Version Information
 W=Warm Boot Module

 Esc=Exit Program
Press 'Y' key to confirm configuration receive!

Receiving configuration file:


TRANSFERRING CONFIGURATION FROM PC TO ProSoft MODULE:
Using the Ymodem file transfer protocol.  Select the SEND
menu option and transfer the configuration file.

C
```

The Send File dialog appears.

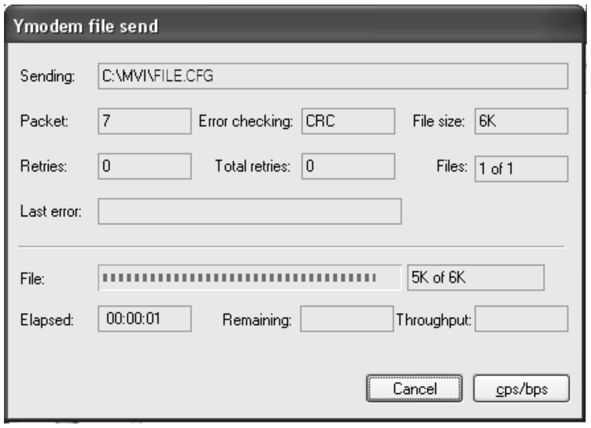


Folder: C:\Documents and Settings\mrodrigues
Filename:
Protocol: Ymodem

5 Use the Browse button to locate the configuration file your computer.

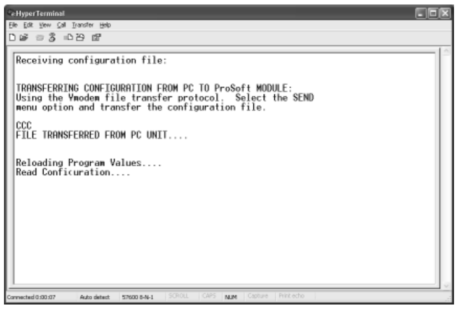
Note: This procedure assumes that you are uploading a newly edited configuration file from your PC to the module. However, configuration files are also available at **www.prosoft-technology.com**.

6 Select **YMODEM** as the protocol.
7 Click the Send button. This action opens the *Ymodem File Send* dialog box.



Sending: C:\MVI\FILE.CFG
Packet: 7 Error checking: CRC File size: 6K
Retries: 0 Total retries: 0 Files: 1 of 1
Last error:
File: 5K of 6K
Elapsed: 00:00:01 Remaining: Throughput:

When the file transfer is complete, the module's configuration/debug screen indicates that the module has reloaded program values, and displays information about the module.


HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:07 Auto detect 57600 8-N-1 SCROLL CAPS NUM Capture Print echo

```
Receiving configuration file:


TRANSFERRING CONFIGURATION FROM PC TO ProSoft MODULE:
Using the Ymodem file transfer protocol.  Select the SEND
menu option and transfer the configuration file.

CCC
FILE TRANSFERRED FROM PC UNIT....


Reloading Program Values....
Read Conficuration....
```

8 Your module now contains the new configuration.

4 Diagnostics and Troubleshooting

In This Chapter

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

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

4.1 The Configuration/Debug Menu

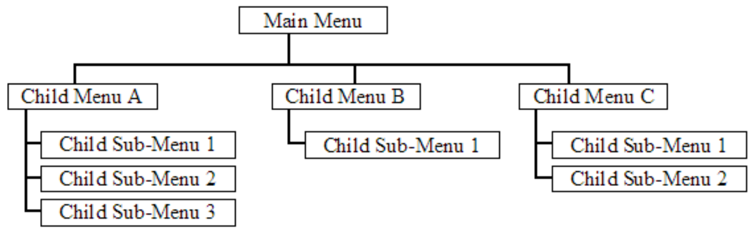
The Configuration and Debug menu for this module is arranged as a tree structure, with the Main Menu at the top of the tree, and one or more sub-menus for each menu command. The first menu you see when you connect to the module is the Main menu.

Because this is a text-based menu system, you enter commands by typing the command letter from your computer keyboard Microsoft Windows HyperTerminal. The module does not respond to mouse movements or clicks. The command executes as soon as you press the command letter — you do not need to press **[Enter]**. When you type a command letter, a new screen will be displayed in the Microsoft Windows HyperTerminal application.

4.1.1 Navigation

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

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



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

4.1.2 Keystrokes

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

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

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

4.2 Using the Configuration/Debug Port

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

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

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

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

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

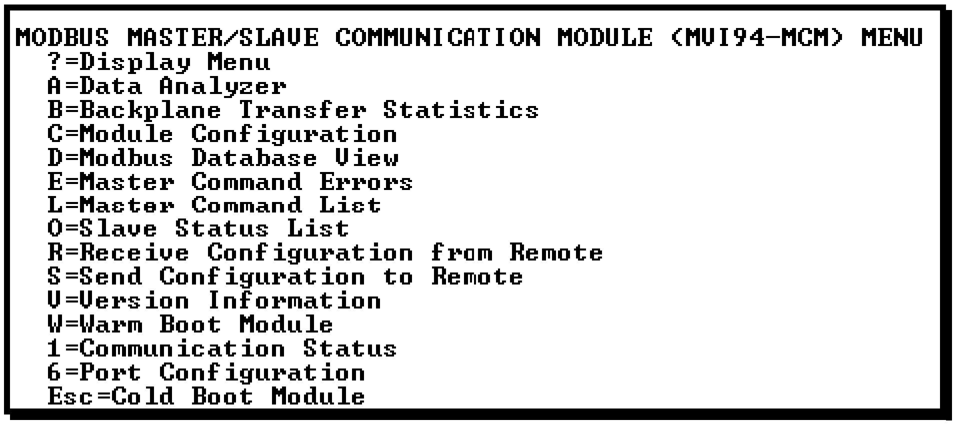
1 Verify that the null modem cable is connected properly between your computer's serial port and the module. A regular serial cable will not work.
2 Verify that RSLinx is not controlling the COM port. Refer to Disabling the RSLinx Driver for the Com Port on the PC.
3 Verify that your communication software is using the correct settings for baud rate, parity and handshaking.
4 On computers with more than one serial port, verify that your communication program is connected to the same port that is connected to the module.

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

4.3 Module Configuration

4.3.1 Main Menu

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



```
MODBUS MASTER/SLAVE COMMUNICATION MODULE (MVI94-MCM) MENU
  ?=Display Menu
  A=Data Analyzer
  B=Backplane Transfer Statistics
  C=Module Configuration
  D=Modbus Database View
  E=Master Command Errors
  L=Master Command List
  O=Slave Status List
  R=Receive Configuration from Remote
  S=Send Configuration to Remote
  V=Version Information
  W=Warm Boot Module
  1=Communication Status
  6=Port Configuration
  Esc=Cold Boot Module
```

Caution: Some of the commands available to you from this menu are designed for advanced debugging and system testing only, and can cause the module to stop communicating with the processor or with other devices, resulting in potential data loss or other communication failures. Use these commands only if you fully understand their potential effects, or if you are specifically directed to do so by ProSoft Technology Technical Support Engineers.

There may be some special command keys that are not listed on the menu but that may activate additional diagnostic or debugging features. If you need these functions, you will be advised how to use them by Technical Support. Please be careful when pressing keys so that you do not accidentally execute an unwanted command.

Redisplaying the Menu

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

Opening the Data Analyzer Menu

Press **[A]** to open the Data Analyzer Menu. Use this command to view all bytes of data transferred on each port. Both the transmitted and received data bytes are displayed. Refer to Data Analyzer (page 54) for more information about this menu.

Important: When in analyzer mode, program execution will slow down. Only use this tool during a troubleshooting session. Before disconnecting from the Config/Debug port, please press **[S]** to stop the data analyzer, and then press **[M]** to return to the main menu. This action will allow the module to resume its normal high speed operating mode.

Viewing Block Transfer Statistics

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

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

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

Viewing Modbus Configuration

Press **[C]** to view the module name and the configuration of the internal Modbus database.

Opening the Database View Menu

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

Use this menu command to view the current contents of the module's database. For more information about this submenu, see Database View Menu (page 58).

Opening the Client Command Error List Menu

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

Opening the Command List Menu

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

Viewing the Slave Status List

Press **[O]** to view the 256 slave status values associated with the port. The slave status values are defined as follows:

0 = slave is not used,

1 = slave being actively polled,

2 = slave suspended and

3 = slave disabled.

Transferring the Configuration File from The Module to the PC

On the Diagnostics Menu this is referred to as *Send Module Configuration*.

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

Press **[Y]** to confirm the file transfer, and then follow the instructions on the terminal screen to complete the file transfer process.

After the file has been successfully uploaded, you can open and edit the file to change the module's configuration.

Transferring the Configuration File from the PC to the Module

On the Diagnostics Menu this is referred to as *Receive Module Configuration*.

Press **[R]** to receive (download) the configuration file from your PC to the module and store the file on the module's Compact Flash Card (Personality Module) or Flash RAM.

Press **[Y]** to confirm the file transfer, and then follow the instructions on the terminal screen to complete the file transfer process.

After the file has been successfully downloaded, the module will restart the program and load the new configuration information. Review the new configuration using menu commands **[6]** and **[0]** to verify that the module is configured correctly.

Viewing Version Information

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

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

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

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

Warm Booting the Module

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

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

Viewing Port Communication Status

Press **[1]** from the Main Menu to view the port communication status for the application port.

Use this command to view communication status and statistics for the selected port. This information can be informative when trouble-shooting communication problems.

Viewing Port Configuration

Press **[6]** from the Main Menu to view configuration information for the application port.

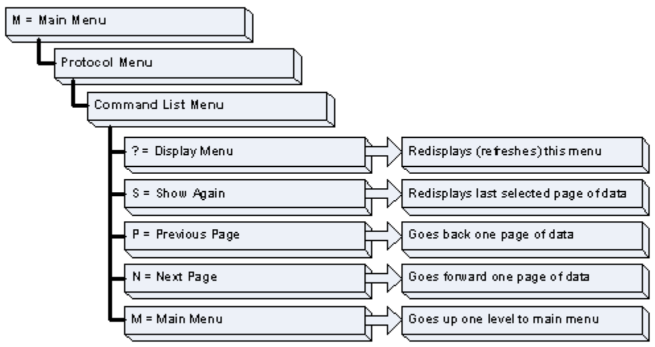
Use this command to display detailed configuration information for the port.

Cold Booting the Module

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

4.3.2 Master Command Error List Menu

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


M = Main Menu
Protocol Menu
Command List Menu
? = Display Menu
Redisplays (refreshes) this menu
S = Show Again
Redisplays last selected page of data
P = Previous Page
Goes back one page of data
N = Next Page
Goes forward one page of data
M = Main Menu
Goes up one level to main menu

Redisplaying the Current Page

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

Moving Back Through 5 Pages of Commands

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

Viewing the Previous Page of Commands

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

Moving Forward (Skipping) Through 5 Pages of Commands

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

Viewing the Next Page of Commands

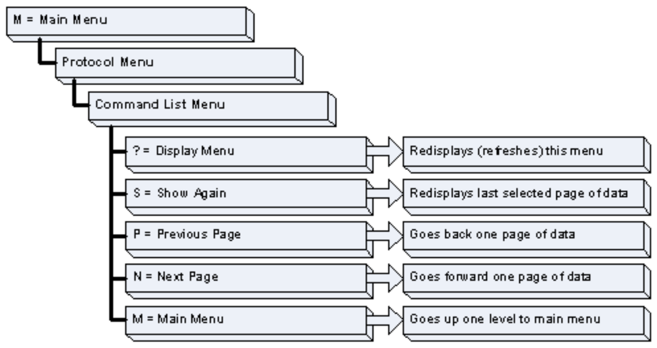
Press **[N]** to display the next page of commands.

Returning to the Main Menu

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

4.3.3 Master Command List Menu

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


M = Main Menu
Protocol Menu
Command List Menu
? = Display Menu
Redisplays (refreshes) this menu
S = Show Again
Redisplays last selected page of data
P = Previous Page
Goes back one page of data
N = Next Page
Goes forward one page of data
M = Main Menu
Goes up one level to main menu

Redisplaying the Current Page

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

Viewing the Previous 50 Commands

Press **[-]** to view the previous 50 commands.

Viewing the Previous Page of Commands

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

Viewing the Next 50 Commands

Press **[+]** to view the next 50 commands from the Master command list.

Viewing the Next Page of Commands

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

Returning to the Main Menu

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

4.3.4 Data Analyzer

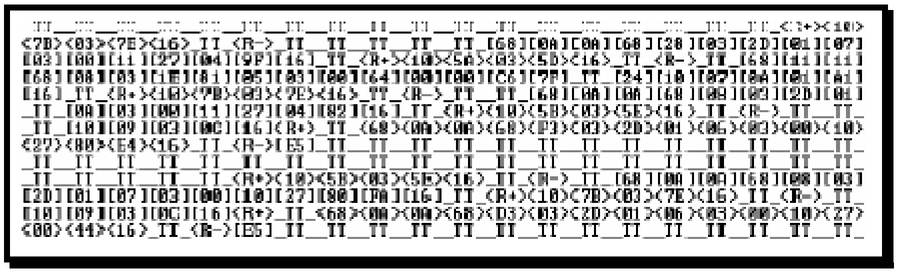
The data analyzer mode allows you to view all bytes of data transferred on each port. Both the transmitted and received data bytes are displayed. Use of this feature is limited without a thorough understanding of the protocol.

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

Important: When in analyzer mode, program execution will slow down. Only use this tool during a troubleshooting session. Before disconnecting from the Config/Debug port, please press **[S]** to stop the data analyzer, and then press **[M]** to return to the main menu. This action will allow the module to resume its normal high speed operating mode.

Analyzing Data for the first application port

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



```
_TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT_<R+><10>
<7B><03><7E><16>_TT_<R->_TT__TT__TT__TT__TT_[68][0A][0A][68][28][03][2D][01][07]
[03][00][11][27][04][9F][16]_TT_<R+><10><5A><03><5D><16>_TT_<R->_TT_[68][11][11]
[68][08][03][1E][81][05][03][00][64][00][00][C6][7F]_TT_[24][10][07][0A][01][A1]
[16]_TT_<R+><10><7B><03><7E><16>_TT_<R->_TT__TT_[68][0A][0A][68][08][03][2D][01]
_TT_[0A][03][00][11][27][04][82][16]_TT_<R+><10><5B><03><5E><16>_TT_<R->_TT__TT_
_TT_[10][09][03][0C][16]<R+>_TT_<68><0A><0A><68><F3><03><2D><01><06><03><00><10>
<27><00><E4><16>_TT_<R->[E5]_TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT_
_TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT_
_TT__TT__TT__TT__TT_<R+><10><5B><03><5E><16>_TT_<R->_TT_[68][0A][0A][68][08][03]
[2D][01][07][03][00][10][27][00][FA][16]_TT_<R+><10><7B><03><7E><16>_TT_<R->_TT_
[10][09][03][0C][16]<R+>_TT_<68><0A><0A><68><D3><03><2D><01><06><03><00><10><27>
<00><44><16>_TT_<R->[E5]_TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT__TT_
```

Analyzing Data for the second application port

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

Displaying Timing Marks in the Data Analyzer

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

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

Removing Timing Marks in the Data Analyzer

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

Viewing Data in Hexadecimal Format

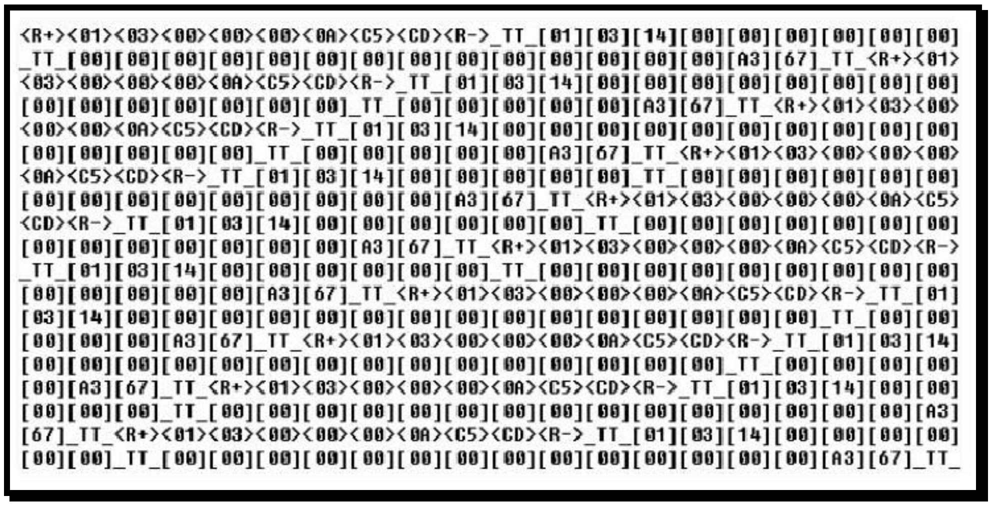
Press **[H]** from the *Database View* menu to display the data on the current page in hexadecimal format.

Viewing Data in ASCII (Text) Format

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

Starting the Data Analyzer

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



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

The Data Analyzer displays the following special characters:

Character	Definition
[ ]	Data enclosed in these characters represent data received on the port.
< >	Data enclosed in these characters represent data transmitted on the port.
<R+>	These characters are inserted when the RTS line is driven high on the port.
<R->	These characters are inserted when the RTS line is dropped low on the port.
<CS>	These characters are displayed when the CTS line is recognized high.
_TT_	These characters are displayed when the timing mark interval has been reached. This parameter is user defined.

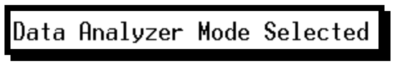
Stopping the Data Analyzer

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

Important: When in analyzer mode, program execution will slow down. Only use this tool during a troubleshooting session. Before disconnecting from the Config/Debug port, please press **[S]** to stop the data analyzer, and then press **[M]** to return to the main menu. This action will allow the module to resume its normal high speed operating mode.

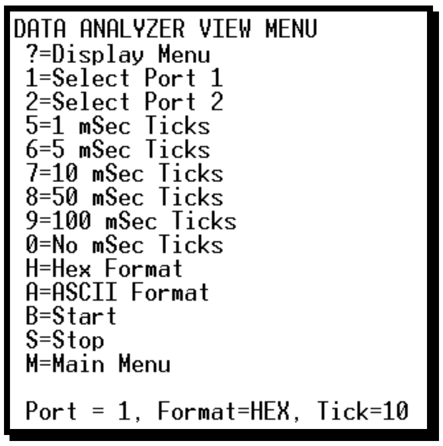
Data Analyzer Tips

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



```
Data Analyzer Mode Selected
```

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



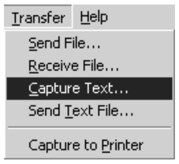
```
DATA ANALYZER VIEW MENU
 ?=Display Menu
 1=Select Port 1
 2=Select Port 2
 5=1 mSec Ticks
 6=5 mSec Ticks
 7=10 mSec Ticks
 8=50 mSec Ticks
 9=100 mSec Ticks
 0=No mSec Ticks
 H=Hex Format
 A=ASCII Format
 B=Start
 S=Stop
 M=Main Menu

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

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

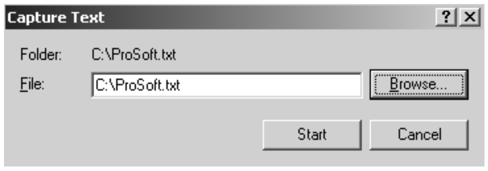
For most applications, HEX is the best format to view the data, and this does include ASCII based messages (because some characters will not display on HyperTerminal and by capturing the data in HEX, we can figure out what the corresponding ASCII characters are supposed to be).

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

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



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

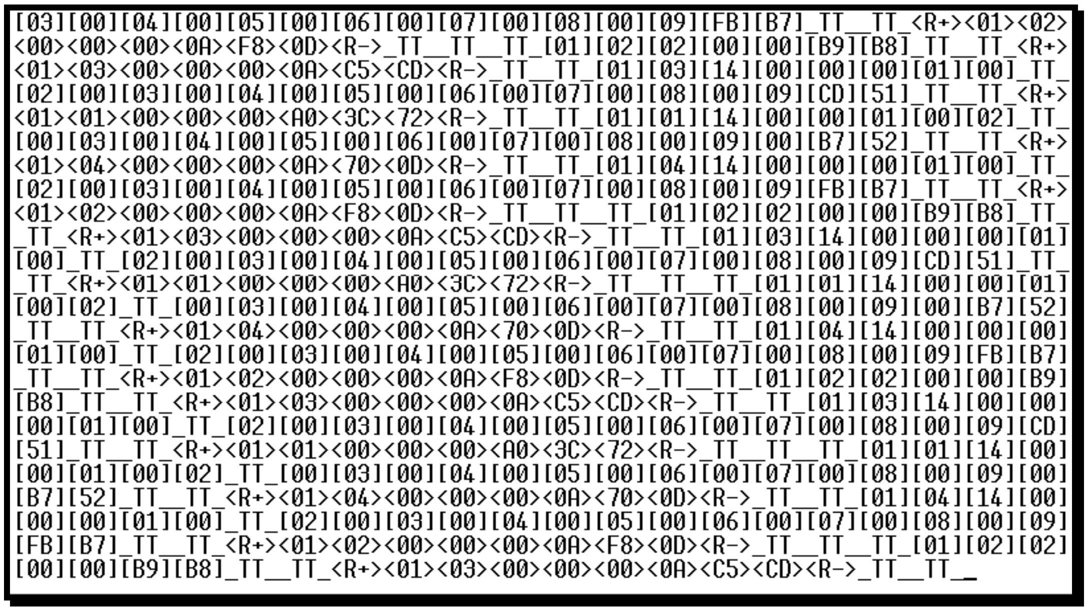


Folder: C:\ProSoft.txt
File: C:\ProSoft.txt

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

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

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



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

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

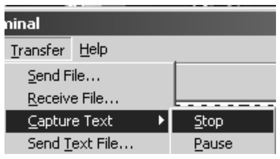
All characters shown in <> brackets are characters being sent out by the module.

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

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

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

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



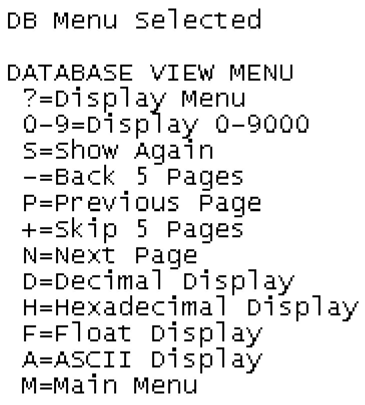
You have now captured, and saved the file to your PC. This file can now be used in analyzing the communications traffic on the line, and assist in determining communication errors.

Returning to the Main Menu

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

4.3.5 Database View Menu

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



```
DB Menu Selected

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

Viewing Register Pages

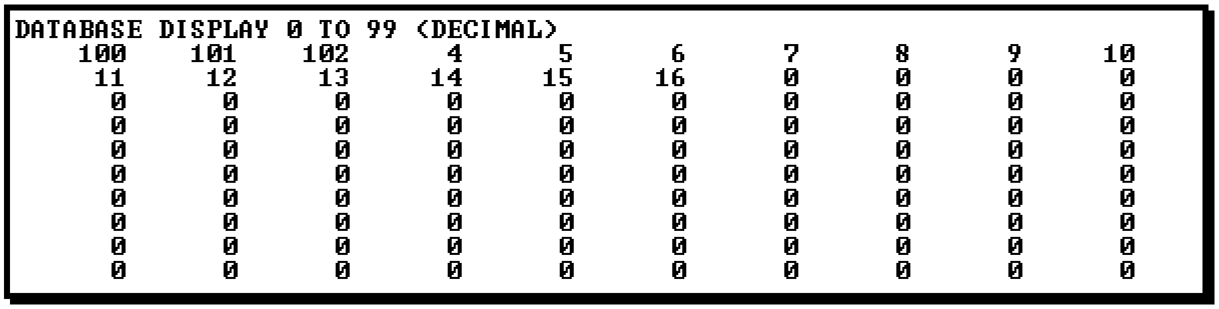
To view sets of register pages, use the keys described below:

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

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

Displaying the Current Page of Registers Again

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



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

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

Moving Back Through 5 Pages of Registers

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

Moving Forward (Skipping) Through 5 Pages of Registers

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

Viewing the Previous Page of Registers

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

Viewing the Next Page of Registers

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

Viewing Data in Decimal Format

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

Viewing Data in Hexadecimal Format

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

Viewing Data in Floating-Point Format

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

Viewing Data in ASCII (Text) Format

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

Returning to the Main Menu

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

4.4 Error/Status Data

The module error/status data areas are discussed in this section. The module contains three areas related to this data. The user defines the location of two of these data sets in the virtual Modbus database of the module. The error/status data contains module data, the command error list data set contains the errors associated with the command list and the slave status list contains the current communication status of each slave on the master port.

4.4.1 Error and Status Data Table

The error/status data table is located at the virtual Modbus address assigned by the user. If the address is set to -1 or the frequency parameter is set to 0, the data will not be placed in the database. It will only be available through the Configuration/Debug Port. If valid address and frequency values are assigned, the module will update the Modbus data area.

The data area will be initialized with zeros whenever the processor is initialized. This occurs during a cold-start (power-on), reset (reset push-button pressed) or a warm-boot operation (commanded or loading of new configuration).

The data area is a 22-word register block. The structure of the block is shown in the following table.

Word	**Description**
System Information	
0	Program Cycle Counter
1	Product Name (ASCII) = MCM
2	
3	Revision (ASCII)
4	
5	Operating System Rev (ASCII)
6	
7	Production Run Number (ASCII)
8	
Modbus Port	
9	Number of Command Requests
10	Number of Command Responses
11	Number of Command Errors
12	Number of Requests
13	Number of Responses
14	Number of Errors Received
15	Number of Errors Sent
Block Transfer Statistics	
16	Total Number of Read Block Operations
17	Total Number of Write Block Operations
18	Total Number of Blocks Parsed
19	Total Number of Event Blocks Received
20	Total Number of Command Blocks Received
21	Total Number of Block Transfer Errors

Counter Values are initialized to 0 at power up.

4.4.2 Command Error List

Each command in the command list has a reserved word value for a status/error code. This error data list can be read using the Configuration/Debug Port. Additionally, the data can be placed in the virtual Modbus database of the module. The configuration parameter "Command Error Pointer" defines the register address in the virtual Modbus database where the data will be placed.

The first word in the register location defined contains the status/error code for the first command in the port's command list. Each successive word in the command error list is associated with the next command in the list. Therefore, the size of the data area is dependent upon the number of commands defined.

Refer to the following Error Codes section to interpret the status/error codes present in the data area.

4.4.3 Slave Status List

The slave status list views the communication status of each slave device on a master port. Slaves attached to the master port can have one of the following states.

0	The slave is inactive and not defined in the command list for the master port.
1	The slave is actively being polled or controlled by the master port and communication is successful.
2	The master port has failed to communicate with the slave device. Communication with the slave is suspended for a user defined period based on the scanning of the command list.

Slaves are defined to the system when the module initializes the master command list. Each slave defined will be set to a state of 1 in this initial step. If the master port fails to communicate with a slave device (retry count expired on a command), the master will set the state of the slave to a value of 2 in the status table. This suspends communication with the slave device for a user specified scan count (**Error Delay Count** value in the configuration). Each time a command in the list is scanned that has the address of a suspended slave, the delay counter value will be decremented. When the value reaches zero, the slave state will be set to one. This will enable polling of the slave.

The slave status list can only be viewed using the module’s configuration/debug port. The 'O' option on the main menu displays the status of all 256 slave units.

4.4.4 Error Codes

The module error codes are listed in this section. Error codes are separated into Modbus error codes and module error codes.

Modbus Error Codes

These error codes are returned to the module from Modbus slave devices attached to the master port. These codes are the standard Modbus errors. The error codes are listed in the following table.

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

Module Error Codes

Module error codes are generated by the module's program. These errors can result due to configuration or communication problems. These errors are stored in the Modbus master, command list error table. A word is allocated for each command in the memory area.

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

The following table lists the errors returned by the module for errors found when parsing the command list.

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	All parameters set to 0
-47	All parameters set to -1

Use the error codes returned for each command in the list to determine the success or failure of the command. If the command fails, use the error code to determine the cause of failure.

4.5 LED Definition

This section defines the indications provided on the MVI94-MCM module through LEDs. A description of each LED is provided in the following topics.

4.5.1 PRT1

This LED indicates data transmit and receive activity on the configuration port. When the TXD or RXD pin is active on the port, the LED will illuminate green. When the port is not active, the LED remains in the off state.

4.5.2 U1

This LED indicates backplane data transfer operation. When the module is successfully writing data to the FLEX I/O backplane, the LED will be in the off state. When the module is reading a new block of data from the FLEX I/O backplane, the LED will be in the on state (amber). During normal operation of the module, this LED should turn on and off at a vary rapid rate. If the LED never turns on, check your ladder logic to verify that the data transfer is set up correctly.

4.5.3 U2

This LED indicates communication errors on the Modbus master port. The LED is illuminated (amber) when no error exists on the port. If a communication error is recognized on the port, the LED will turn off. If the LED is off, check for errors in the command list to determine the error condition recognized by the module.

4.5.4 STS

This LED indicates the "health" of the module. When power is applied to the module, the LED is illuminated. If the LED is green, the program is working correctly and the user configuration is being used. If the LED is red, the program is halted. Try restarting the module by cycling power.

4.5.5 PRT2

This LED indicates data transmit and receive activity on the Modbus master port. When the TXD or RXD pin is active on the port, the LED will be green. When the port is not active, the LED will be off.

5 Reference

In This Chapter

5.1 Product Specifications

The MVI94 Modbus Master/Slave Communication Module allows Rockwell Automation® FLEX® processors to interface easily with other Modbus protocol compatible devices.

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

5.1.1 General Specifications

Some of the general specifications include:

- Operation via simple ladder logic
- Complete setup and monitoring of module through Debug port and user configuration file
- Flex backplane interface via I/O access

5.1.2 FLEX I/O Interfaces

Specification	Description
Form Factor	Single Slot 1794 Backplane compatible Locate in any slot of Backplane
Backplane current load	20 mA @ 5 V
External power supply	12 Vdc to 24 Vdc 340 mA to 170 mA
Operating temperature	0°C to 55°C (32°F to 131°F)
Storage temperature	-40°C to 85°C (-40°F to 185°F)
Shock	30 g operational 50 g non-operational 5 g from 10150 Hz
Relative humidity	5% to 95% (without condensation)
LED indicators	Module status Backplane transfer status Application status Serial activity and error LED status
Configuration Serial port (PRT1)	Mini-DIN RS-232 Hardware handshaking
Application serial Port (PRT2)	Mini-DIN RS-232/422/485 jumper selectable 500V optical isolation from backplane
Dimensions (with Module installed in Base)	3.7H x 3.7W x 2.7D inches 94H x 94W x 69D mm

5.1.3 General Specifications - Modbus Master/Slave

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

5.1.4 Functional Specifications

Modbus Master

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

Command List	Up to 100 commands per Master port, each fully configurable for function, slave address, register to/from addressing and word/bit count.
Polling of Command List	Configurable polling of command list, including continuous and on change of data, and dynamically user or automatic enabled.
Status Data	Error codes available on an individual command basis. In addition, a slave status list is maintained per active Modbus Master port.

Modbus Slave

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

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

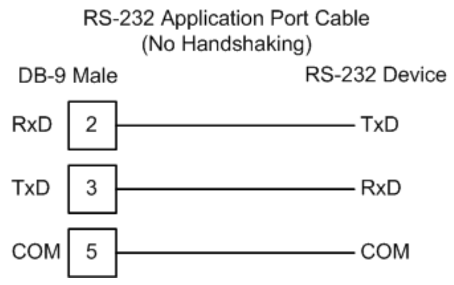
5.2 Cable Connections

The application ports on the MVI94-MCM module support RS-232, RS-422, and RS-485 interfaces. Please ensure that the jumpers are set correctly for the type of interface you are using.

Note for modules with RS-232 connection to a radio or modem: Some radios or modems require hardware handshaking (control and monitoring of modem signal lines) on the RTS and CTS lines of an RS-232 connection. Enable this by setting the *UseCTS* parameter in the module configuration to **1**.

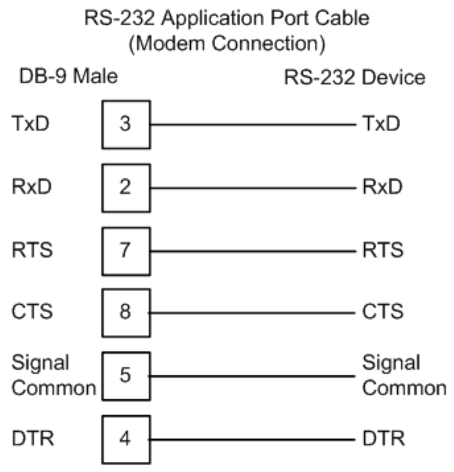
5.2.1 RS-232 Application Port(s)

When the RS-232 interface is selected, the use of hardware handshaking (control and monitoring of modem signal lines) is user definable. If no hardware handshaking will be used, here are the cable pinouts to connect to the port.



RS-232: Modem Connection (Hardware Handshaking Required)

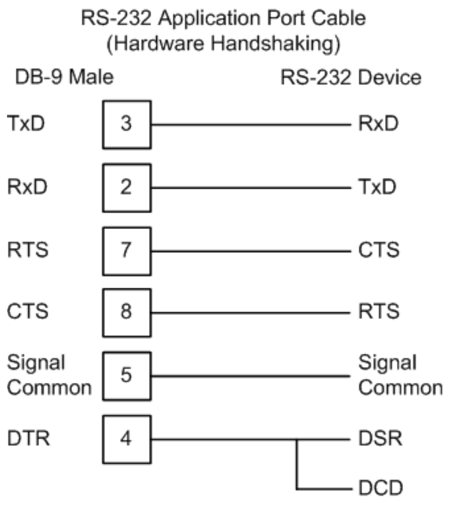
This type of connection is required between the module and a modem or other communication device.



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

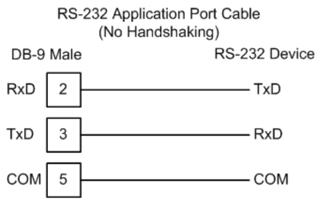
RS-232: Null Modem Connection (Hardware Handshaking)

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

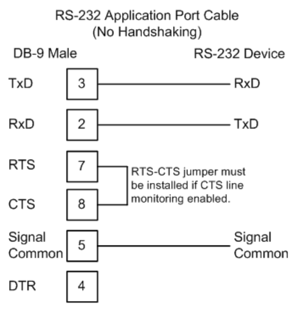


RS-232: Null Modem Connection (No Hardware Handshaking)

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

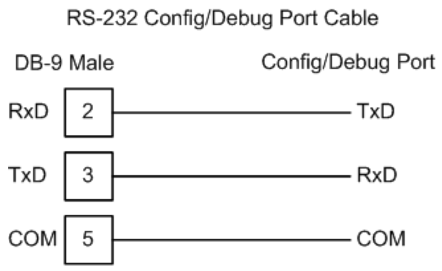


Note: For most null modem connections where hardware handshaking is not required, the *Use CTS Line* parameter should be set to **N** and no jumper will be required between Pins 7 (RTS) and 8 (CTS) on the connector. If the port is configured with the *Use CTS Line* set to **Y**, then a jumper is required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


RS-232 Application Port Cable
(No Handshaking)
DB-9 Male
RS-232 Device
TxD
3
RxD
RxD
2
TxD
RTS
7
CTS
8
RTS-CTS jumper must be installed if CTS line monitoring enabled.
Signal Common
5
Signal Common
DTR
4

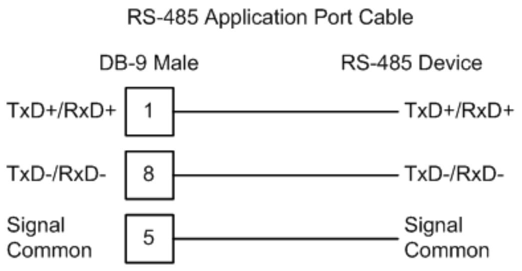
5.2.2 RS-232 Configuration/Debug Port

This port is physically an eight-pin, Mini-DIN8F connection. A Mini-DIN8M to DB9M adapter cable is included with the module. This port permits a PC-based terminal emulation program to view configuration and status data in the module and to control the module. Here are the cable pinouts for RS-232 communication on this port.



5.2.3 RS-485 Application Port(s)

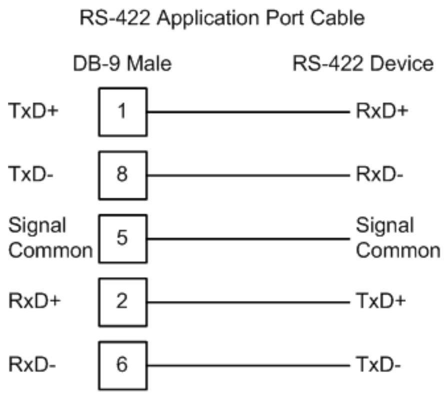
The RS-485 interface requires a single two or three wire cable. The Common connection is optional, depending on the RS-485 network devices used. The cable required for this interface is shown below:



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

5.2.4 RS-422

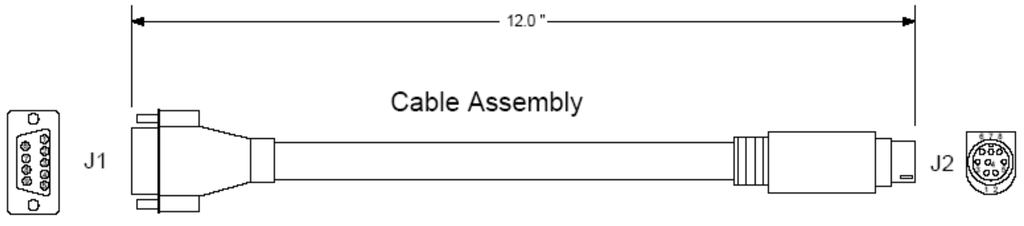
The RS-422 interface requires a single four or five wire cable. The Common connection is optional, depending on the RS-422 network devices used. The cable required for this interface is shown below:

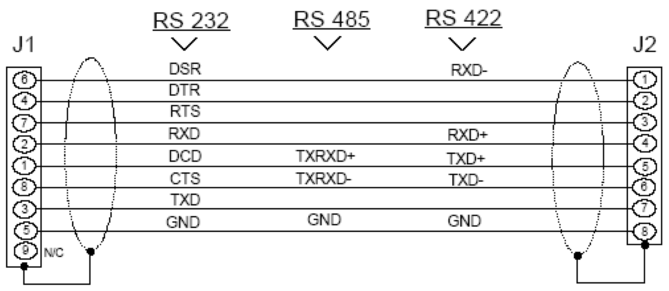


RS-485 and RS-422 Tip

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

5.2.5 DB9 to Mini-DIN Adaptor (Cable 09)





Wiring Diagram

5.3 Functional Overview

The MVI94-MCM communication module interfaces Modbus master and slave devices with the Flex I/O system. The module contains a virtual Modbus database that is defined by the user. When configured as a master, this database is used for the request and command messages sent from the Modbus master port to Modbus slave devices. As a slave this database serves data to the master device and passes control data to the processor via the backplane. Data areas in the virtual Modbus database can be reserved for status and error information generated by the module under user control.

The virtual Modbus database also interfaces with the Flex I/O system using the Flex I/O bus (backplane). Data is made available to the processor on a ControlNet network using this backplane interface. Input and output image tables in the module present the data in the virtual Modbus database to the backplane.

A Modbus master port is present on the communication module to continuously poll Modbus slave devices. Up to 100 user-defined commands can be defined for the port. Data read from Modbus slave devices are placed in the virtual Modbus database. Any write requests for the Modbus slave devices are sourced with data from the virtual Modbus database.

The module can be configured to place slave devices that are not responding to commands at a lower priority. If the module recognizes a slave device has failed to respond to a message after the user defined retry count, it will mark the slave as "in communication failure" and set the error delay counter to the specified value. Each time the module encounters this slave in the command list, the counter will be decremented. When the value reaches zero, the slave will be placed in an active status. This facility can improve communication throughput on the Modbus network.

Commands can be activated in the module under processor control. This feature permits the processor to issue a command in the command list under program control. When a command is activated, it will be placed in the command queue for immediate execution. Normal command polling will begin after the command queue is completely processed.

Additionally, the processor can send a command directly to any slave attached to the Modbus master port. This feature can be used for commands that are not issued on a regular basis and are not in the command list. Commands submitted as events are placed in the command queue for immediate execution. They will preempt normal operation of the poll list. When the command queue is completely empty, normal command polling will resume.

Data in the virtual Modbus database is accessible through a configured Modbus slave port. Remote Modbus master devices can monitor and control data in this database through this port.

The module provides a Configuration/Debug port for use with an external computer executing a terminal emulation program. The terminal emulation program provided with the module permits uploading and downloading of the configuration information required by the module. Additionally, the Configuration/Debug port provides a view into the virtual Modbus database, communication statistics and the configuration.

5.3.1 About the MODBUS Protocol

MODBUS is a widely-used protocol originally developed by Modicon in 1978. Since that time, the protocol has been adopted as a standard throughout the automation industry.

The original MODBUS specification uses a serial connection to communicate commands and data between Master and Slave devices on a network. Later enhancements to the protocol allow communication over other types of networks.

MODBUS is a Master/Slave protocol. The Master establishes a connection to the remote Slave. When the connection is established, the Master sends the MODBUS commands to the Slave. The MVI94-MCM module can work as a Master and as a Slave.

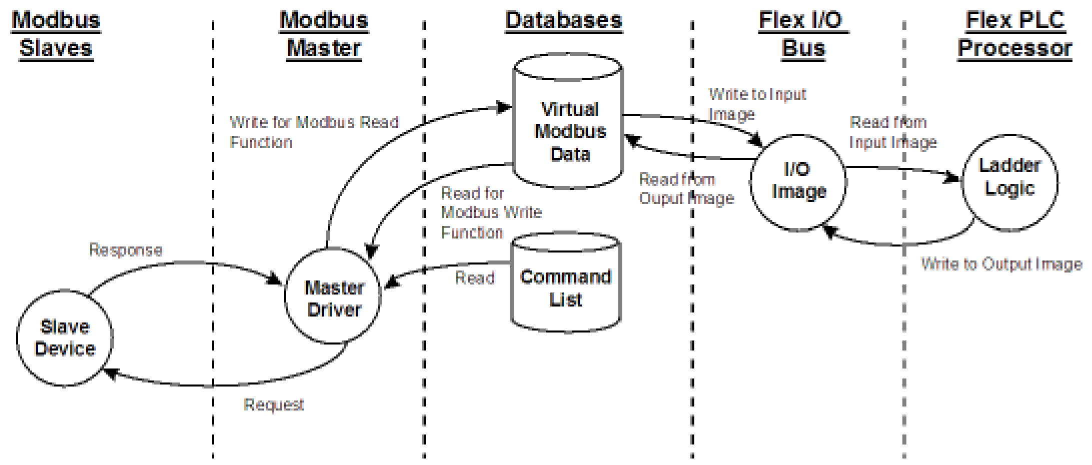
The MVI94-MCM module also works as an input/output module between itself and the Rockwell Automation backplane and processor. The module uses an internal database to pass data and commands between the processor and Master and Slave devices on MODBUS networks.

5.3.2 Virtual Modbus Database Concepts

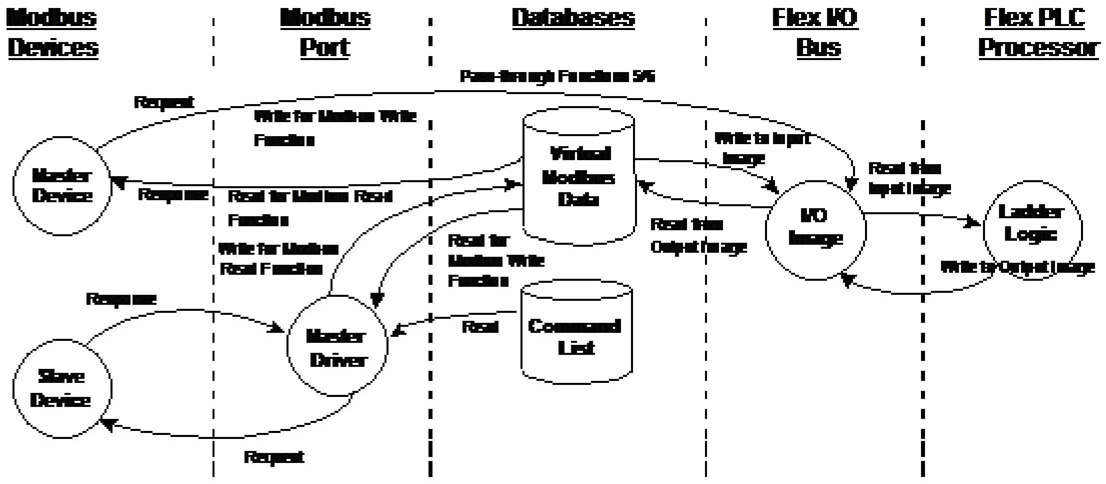
Central to the functionality of the module is the virtual Modbus database. This database is used as the interface between remote Modbus master and slave devices and the Flex I/O bus. The size, content and structure of the database are completely user defined.

The Flex I/O bus reads data from and write data to the database using the backplane interface. The module interfaces data contained in remote Modbus slave devices to the virtual Modbus database when using the Modbus master port. User commands are issued out the master port from a command list. These commands gather or control data in the Modbus slave devices. When configured as a slave, control information from the Modbus master and data from the processor are exchanged over the backplane. Pass-through mode allows the passing of single bit or register data directly to the processor, bypassing the Modbus database. The following diagrams illustrate the relationships discussed above.

No Pass-Through

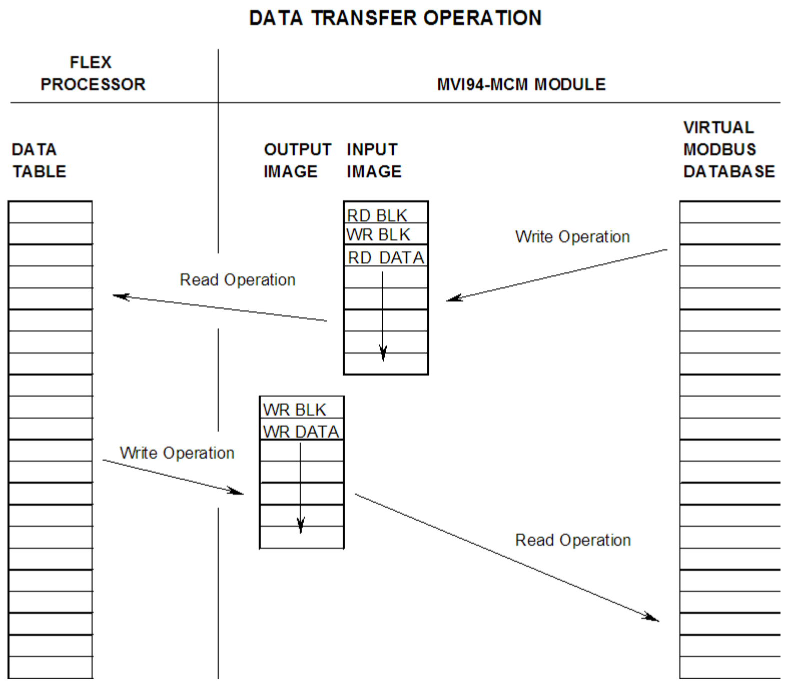


Pass-Through



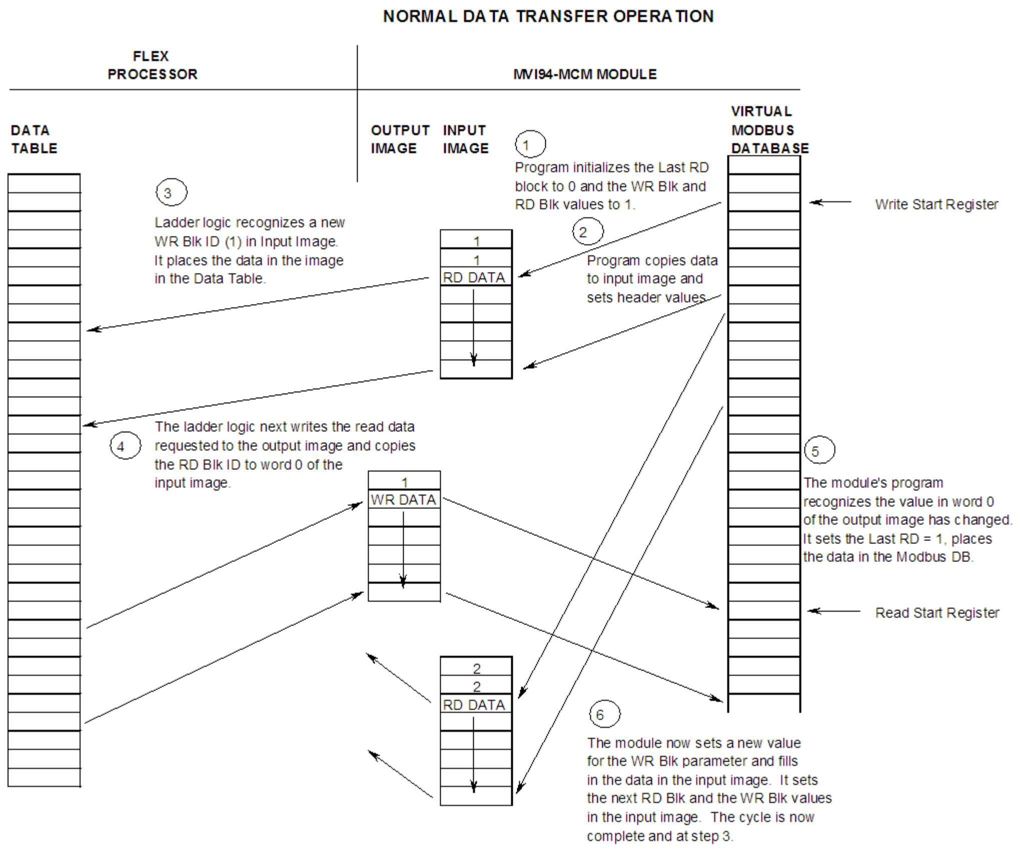
5.3.3 Data Transfer

Data is transferred over the backplane using the module's input and output images. The module is configured with an eight-word input image and a seven-word output image. The module and the processor use these images to page data and commands. The input image is set (written) by the module and is read by the processor. The output image is set (written) by the Flex processor and read by the module. The following diagram shows this relationship.



The module's program is responsible for setting the block identification code used to identify the data block written and the block identification code of the block it wants to read from the processor. User configuration information determines the read (BT Read Start Register) and write (BT Write Start Register) locations in the virtual Modbus database and the amount of data transferred (BT Read Register Count and BT Write Register Count). Each read and write operation transfers a six-word data area. The read operation (from the processor to the module) contains a two-word header that defines the block identification code of the read data and the block identification code of the next write block requested. These identification codes are in the range of 0 to 666. A value of zero indicates that the block contains no data and should be ignored. The first valid block identification code is one and refers to the first block of six words to be read or written.

The module and the processor constantly monitor input and output images. How does either one know when a new block of data is available? Recognizing a change in the header information of the image (word 0) solves the problem. For example, when the module recognizes a different value in the first word of the output image, new data is available from the processor. When the processor recognizes a new value in the first word of the input image, new data is available from the module. This technique requires the storage of the previously processed data block identification code. The following illustration shows the normal sequence of events for data transfer:


NORMAL DATA TRANSFER OPERATION
FLEX PROCESSOR
MVI94-MCM MODULE
DATA TABLE
OUTPUT IMAGE
INPUT IMAGE
VIRTUAL MODBUS DATABASE
1
Program initializes the Last RD block to 0 and the WR Blk and RD Blk values to 1.
Write Start Register
3
Ladder logic recognizes a new WR Blk ID (1) in Input Image. It places the data in the image in the Data Table.
2
Program copies data to input image and sets header values.
1
1
RD DATA
4
The ladder logic next writes the read data requested to the output image and copies the RD Blk ID to word 0 of the input image.
1
WR DATA
5
The module's program recognizes the value in word 0 of the output image has changed. It sets the Last RD = 1, places the data in the Modbus DB.
Read Start Register
2
2
RD DATA
6
The module now sets a new value for the WR Blk parameter and fills in the data in the input image. It sets the next RD Blk and the WR Blk values in the input image. The cycle is now complete and at step 3.

The steps outlined in the diagram above are discussed below:

1 During program initialization, the write and read block identification codes are set to one. The last block read variable is set to zero.

2 The program copies the first six-word block of the virtual Modbus database starting at the user defined BT Read Start Register to the input image (words 2 to 7). It then sets the current read block code in word 1 of the input image. To "trigger" the write operation, the program places the current write block code into word 0 of the input image.

3 The Flex processor recognizes a new value in word 0 of the input image (based on the last_read_block_code not equal to read_block_code) in its ladder logic. The ladder logic computes the offset into the file based on the following formula:

```
read_file_offset = (read_block_code - 1) * 6
```

4 The new data contained in the input image (words 2 to 7) is copied to the offset in the processor's user data file. The last_read_block_code storage register in the processor is updated with the new read_block_code.

Note: If the data area transferred from the module exceeds the size of a single user file in the Flex processor, logic will be required to handle multiple files.

5 The ladder logic next examines the value of the write_block_code and computes the offset into the read data file as follows:

```
write_file_offset = (read_block_code - 1) * 6
```

6 The required 6-word, read data is copied to the module's output image (words 1 to 6). To "trigger" the transfer operation, the ladder logic moves the write_block_code into word 0 of the output image.
7 The module's program recognizes the new write_block_code. It transfers the data to the correct offset in the virtual Modbus database using the following function:

```
Modbus_offset = BT write_Start_Register + (read_block_code - 1) * 6
```

8 The module sets the last_write_block_code to the value of write_block_code.
9 The module now selects the next read and write blocks. The data for the write operation is placed in the input image and the write_block_code is set. The module "triggers" the transfer operation by setting the new read_block_code in word 0 of the input image. The sequence continues at step 3.

The previous discussion is for normal data transfer operation. The following table lists the block identification codes used by the module.

Type	Number	Description
R/W	1 to 666	Data blocks used to transfer data from the module to the backplane and from the backplane to the module. The module's input/output images are used for the data transfers.
R	1000 to 1255	These blocks transfer event messages (Modbus commands) from a device on the backplane to the module. The block number contains the slave module for the command. For example, to use slave device 5 attached to the master port, the block number 1005 should be used.
R	2001 to 2006	These blocks transfer a list of commands to execute from a device on the backplane to the module. The number of commands to be considered is coded in the block number. For example, to add three commands to the command queue, use block 2003.
W	9956	This block transfers function code 6 single register data to the processor in pass-through mode.
W	9957	This block transfers function code 6 single floating-point register data to the processor in pass-through mode.
W	9958	This block transfers function code 5 single-bit data to the processor in pass-through mode.
W	9960	This block transfers function code 22 single masked write data to the processor in pass-through mode.
R	9998	Warm boot the module. When the module receives this block, it will reset all program values using the configuration data.

Type	Number	Description
R	9999	Cold boot the module. When the module receives this block, it will perform a hardware reset.

Data is transferred between the processor and the module using the block identification codes of 1 to 666 and 9956 to 9958 and 9960. The other block codes control the module from the processors ladder logic. They are implemented when the ladder logic needs to control the module. In order to use one of the blocks, the ladder logic inserts the data and code in the output image of the module. The data should be set before the code is placed in the block. This operation should be performed after the receipt of a new write block from the module. A discussion of each set of codes is given in the following topics.

Event Blocks (1000 to 1255)

These control blocks are sent from the processor to the module to execute a Modbus command out the Modbus master port. It should be used for commands that are not found in the command list. Use blocks 1000 to 1255 to execute commands in the list under processor control. The format for this block is shown in the following table.

Block Request from Processor to Module

Word Offset	Description
0	This word contains the slave device address on the Modbus network to be considered with the command. This value is added (0 to 255) to the value of 1000. This generates a block identification code of 1000 to 1255.
1	This word contains the internal Modbus address in the module for the command.
2	This word contains the count parameter that determines the number of digital points or registers to associate with the command.
3	The parameter specifies the swap type for the data. This function is only valid for function code 3, 6, and 16.
4	This word contains the Modbus function code for the command.
5	This word contains the Modbus address in the slave device to be associated with the command.
6	Reserved

The data contained in the block is used by the module to construct a valid Modbus command and defines what data in the virtual Modbus database to use with the command. These parameters are the same as defined for the command list. Refer to the Commands section of the documentation for more information.

When the module receives a block 1000 to 1255, it will check the command queue for an available position. If there is space in the command queue, the module will construct a command and place it in the queue. The module will respond with the following block in the input image after processing request.

Block Response from Module to Processor

Word Offset	Description
0	This word contains the block 1000 to 1255 requested by the processor.
1	This word contains the next read request block identification code.
2	This word contains the result of the event request. If a value of one is present, the command was issued. If a value of zero is present, no room was found in the command queue.
3	Not used.
4	Not used.
5	Not used.
6	Not used.

The ladder logic can examine word 2 of the input image to determine if the module was able to execute the command. If invalid parameters are set in the event request block, the command may still be placed in the queue and there will be no error indication.

Command Blocks (2001 to 2006)

These control blocks are sent from the processor to the module to execute one or more commands in the module's command list out the Modbus master port. Commands selected for execution need not have the Enable Code set (1 or 2) but must be valid commands. The format for this block is shown in the following table.

Block Request from Processor to Module

Word Offset	Description
0	Command queue block identification code of 2001 to 2006.
1	This word contains the index in the command list for the first command to be entered into the command queue.
2	This word contains the index in the command list for the second command to be entered into the command queue.
3	This word contains the index in the command list for the third command to be entered into the command queue.
4	This word contains the index in the command list for the fourth command to be entered into the command queue.
5	This word contains the index in the command list for the fifth command to be entered into the command queue.
6	This word contains the index in the command list for the sixth command to be entered into the command queue.

When the module receives one of these blocks, it examines word 0 of the output image. This word defines the number of commands contained in the block. The command count is determined by subtracting 2000 from the word value. This permits the controller to set from one to six commands into the command queue. The indexes submitted in the block should be valid for the command list. After the module determines the number of commands to consider, it inserts each command in the command queue. The response message sent from the module to the processor is as follows:

Block Response from Module to Processor

Word Offset	Description
0	This word contains the block 2001 to 2006 requested by the processor.
1	This word contains the next read request block identification code.
2	This word contains the number of commands in the block placed in the command queue.
3	Not used.
4	Not used.
5	Not used.
6	Not used.

The ladder logic can examine word 2 of the input image to determine the number of commands placed in the command queue. Any errors associated with the command can be viewed in the command list error table.

Transferring the Command Error List to the Processor

You can transfer the command error list to the processor from the module database. To place the table in the database, set the Command Error Pointer parameter to the database location desired.

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

Pass-Through Blocks (9956 to 9958, 9960)

When the port is configured as a slave with pass-through, Modbus write commands 5, 6, and 22 that are addressed to the slave will be passed across the backplane for processing by the ladder logic. Due to their length, Modbus write commands 15, 16, and 23 will have their data written to the database in the same manner as no pass-through mode. **The module must complete the block transfer before it can accept another pass-through write command.** The first word contains the block identification number indicating the type of command received. The second contains the length of the data in words being written across the backplane. The third word contains the destination address from the write command. The fourth contains the data. If floating-point is enabled and the data comes from a floating-point register, the fourth word will contain the first word of the data and the fifth word will contain the second word of data.

Pass-Through Block 9956, 9957, 9958, or 9960 from Module to Processor

Word	Description
0	This word contains the block 9956 to 9958 representing the type of pass-through data being transferred.
1	This word contains the number of data registers being transferred.
2	This word contains the destination address for the data.
3	This word contains the register data. - For 9960: AND Mask
4	This word contains the second register data when floating-point data is being transferred. - For 9960: OR Mask
5	Reserved
6	Reserved

Warm Boot Block (9998)

This block does not contain any data. When the processor places a value of 9998 in word 0 of the output image, the module will perform a warm-start. This involves clearing the configuration and all program status data. Finally, the program will load in the configuration information from the Flash ROM and begin running. There is no positive response to this message other than the status data being set to zero and the block polling starting over.

Cold Boot Block (9999)

This block does not contain any data. When the processor places a value of 9999 in word 0 of the output image, the module will perform a hardware restart. This will cause the module to reboot and reload the program. There is no positive response to this message other than the status data being set to zero and the block polling starting over.

Master Command List

The MVI94-MCM communication module's primary services are data concentration and communication gateway. The Modbus master port polls Modbus slave devices based on user defined commands and places the data in the virtual Modbus database. The Flex I/O bus interfaces with this database to a Flex processor. The user is responsible for defining the structure and content of the virtual Modbus database.

In order to interface the virtual Modbus database with Modbus slave devices, you must construct a command list. The commands in the list specify the Modbus slave device to be utilized, the function to be performed (read or write), the data area in the device to interface with and the position in the virtual Modbus database to be associated with the device data. Up to 100 commands can be entered for this purpose. The list is processed from top (command #0) to bottom. A poll interval parameter is associated with each command to specify a minimum delay time in seconds between the issuance of a command. If the user specifies a value of 10 for the parameter, the command will be executed no more frequently than every 10 seconds. Additionally, a user specified time delay could be inserted between the issuance of each command. This is useful for slow responding slave devices.

Write commands have a special feature, as they can be set to execute only if the data in the write command changes. If the data in the command has not changed since the command was last issued, the command will not be executed. If the data in the command has changed since the command was last issued, the command will be executed. Use of this feature can lighten the load on the Modbus network. In order to implement this feature; set the enable code for the command to a value of 2.

5.4 Modbus Protocol Specification

The following pages give additional reference information regarding the Modbus protocol commands supported by the MVI94-MCM.

5.4.1 Commands Supported by the Module

The format of each command in the list depends on the MODBUS Function Code being executed.

The following table lists the functions supported by the module.

Function Code	Definition	Supported in Master	Supported in Slave
1	Read Coil Status	X	X
2	Read Input Status	X	X
3	Read Holding Registers	X	X
4	Read Input Registers	X	X
5	Set Single Coil	X	X
6	Single Register Write	X	X
8	Diagnostics		X
15	Multiple Coil Write	X	X
16	Multiple Register Write	X	X
17	Report Slave ID		X
22	Mask Write 4X		X
23	Read/Write		X

Each command list record has the same general format. The first part of the record contains the information relating to the communication module and the second part contains information required to interface to the MODBUS slave device.

5.4.2 Read Coil Status (Function Code 01)

Query

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

The addressing allows up to 2000 coils to be obtained at each request; however, the specific Slave device may have restrictions that lower the maximum quantity. The coils are numbered from zero; (coil number 1 = zero, coil number 2 = one, coil number 3 = two, and so on).

The following table is a sample read output status request to read coils 0020 to 0056 from Slave device number 11.

Adr	Func	Data Start Pt Hi	Data Start Pt Lo	Data # Of Pts Ho	Data # Of Pts Lo	Error Check Field
11	01	00	13	00	25	CRC

Response

An example response to Read Coil Status is as shown in Figure C2. The data is packed one bit for each coil. The response includes the Slave address, function code, quantity of data characters, the data characters, and error checking. Data will be packed with one bit for each coil (1 = ON, 0 = OFF). The low order bit of the first character contains the addressed coil, and the remainder follow. For coil quantities that are not even multiples of eight, the last characters will be filled in with zeros at high order end. The quantity of data characters is always specified as quantity of RTU characters, that is, the number is the same whether RTU or ASCII is used.

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

Adr	Func	Byte Count	Data Coil Status 20 to 27	Data Coil Status 28 to 35	Data Coil Status 36 to 43	Data Coil Status 44 to 51	Data Coil Status 52 to 56	Error Check Field
11	01	05	CD	6B	B2	OE	1B	CRC

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

5.4.3 Read Input Status (Function Code 02)

Query

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

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

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

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

Response

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

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

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

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

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

5.4.4 Read Holding Registers (Function Code 03)

Query

Read Holding Registers (03) allows the user to obtain the binary contents of holding registers 4xxxx in the addressed Slave. The registers can store the numerical values of associated timers and counters which can be driven to external devices. The addressing allows up to 125 registers to obtained at each request; however, the specific Slave device may have restriction that lower this maximum quantity. The registers are numbered form zero (40001 = zero, 40002 = one, and so on). The broadcast mode is not allowed.

The example below reads registers 40108 through 40110 from Slave 584 number 11.

Adr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
11	03	00	6B	00	03	CRC

Response

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

Because the Slave interface device is normally serviced at the end of the controller's scan, the data will reflect the register content at the end of the scan. Some Slaves will limit the quantity of register content provided each scan; thus for large register quantities, multiple transmissions will be made using register content from sequential scans.

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

Adr	Func	ByteCnt	Hi Data	Lo Data	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
11	03	06	02	2B	00	00	00	64	CRC

5.4.5 Read Input Registers (Function Code 04)

Query

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

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

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

Response

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

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

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

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

5.4.6 Force Single Coil (Function Code 05)

Query

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

The use of Slave address 00 (Broadcast Mode) will force all attached Slaves to modify the desired coil.

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

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

Adr	Func	Data Coil # Hi	Data Coil # Lo	Data On/off Ind	Data	Error Check Field
11	05	00	AC	FF	00	CRC

Response

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

Adr	Func	Data Coil # Hi	Data Coil # Lo	Data On/ Off	Data	Error Check Field
11	05	00	AC	FF	00	CRC

The forcing of a coil via MODBUS function 5 will be accomplished regardless of whether the addressed coil is disabled or not (*In ProSoft products,* the coil *is only affected if the necessary ladder logic is implemented).*

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

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

5.4.7 Preset Single Register (Function Code 06)

Query

Function (06) allows the user to modify the contents of a holding register. Any holding register that exists within the controller can have its contents changed by this message. However, because the controller is actively scanning, it also can alter the content of any holding register at any time. The values are provided in binary up to the maximum capacity of the controller unused high order bits must be set to zero. When used with Slave address zero (Broadcast mode) all Slave controllers will load the specified register with the contents specified.

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

Adr	Func	Data Start Reg Hi	Data Start Reg Lo	Data #of Regs Hi	Data #of Regs Lo	Error Check Field
11	06	00	01	00	03	CRC

Response

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

Adr	Func	Data Reg Hi	Data Reg Lo	Data Input Reg Hi	Data Input Reg Lo	Error Check Field
11	06	00	01	00	03	CRC

5.4.8 Force Multiple Coils (Function Code 15)

Query

This message forces each coil in a consecutive block of coils to a desired ON or OFF state. Any coil that exists within the controller can be forced to either state (ON or OFF). However, because the controller is actively scanning, unless the coils are disabled, the controller can also alter the state of the coil. Coils are numbered from zero (coil 00001 = zero, coil 00002 = one, and so on). The desired status of each coil is packed in the data field, one bit for each coil (1= ON, 0= OFF). The use of Slave address 0 (Broadcast Mode) will force all attached Slaves to modify the desired coils.

Note: Functions 5, 6, 15, and 16 are the only messages (other than Loopback Diagnostic Test) that will be recognized as valid for broadcast.

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

Adr	Func	Hi Add	Lo Add	Quantity	Byte Cnt	Data Coil Status 20 to 27	Data Coil Status 28 to 29	Error Check Field	
11	0F	00	13	00	0A	02	CD	00	CRC

Response

The normal response will be an echo of the Slave address, function code, starting address, and quantity of coils forced.

Adr	Func	Hi Addr	Lo Addr	Quantity	Error Check Field	
11	0F	00	13	00	0A	CRC

The writing of coils via Modbus function 15 will be accomplished regardless of whether the addressed coils are disabled or not.

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

5.4.9 Preset Multiple Registers (Function Code 16)

Query

Holding registers existing within the controller can have their contents changed by this message (a maximum of 60 registers). However, because the controller is actively scanning, it also can alter the content of any holding register at any time. The values are provided in binary up to the maximum capacity of the controller (16-bit for the 184/384 and 584); unused high order bits must be set to zero.

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

Adr	Func	Hi Add	Lo Add	Quantity		Byte Cnt	Hi Data	Lo Data	Hi Data	Lo Data	Error Check Field
11	10	00	87	00	02	04	00	0A	01	02	CRC

Response

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

Adr	Func	Hi Addr	Lo Addr	Quantity		Error Check Field
11	10	00	87	00	02	56

5.4.10 Modbus Exception Responses

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

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

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

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

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

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

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

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

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

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

Modbus Exception Codes

Code	Name	Meaning
01	Illegal Function	The function code received in the query is not an allowable action for the Slave. This may be because the function code is only applicable to newer devices, and was not implemented in the unit selected. It could also indicate that the Slave is in the wrong state to process a request of this type, for example because it is unconfigured and is being asked to return register values.
02	Illegal Data Address	The data address received in the query is not an allowable address for the Slave. More specifically, the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 would succeed; a request with offset 96 and length 5 will generate exception 02.
03	Illegal Data Value	A value contained in the query data field is not an allowable value for Slave. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does not mean that a data item submitted for storage in a register has a value outside the expectation of the application program, because the Modbus protocol is unaware of the significance of any particular value of any particular register.
04	Slave Device Failure	An unrecoverable error occurred while the Slave was attempting to perform the requested action.
05	Acknowledge	Specialized use in conjunction with programming commands. The Slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the Master. The Master can next issue a poll program complete message to determine if processing is completed.
06	Slave Device Busy	Specialized use in conjunction with programming commands. The Slave is engaged in processing a long-duration program command. The Master should retransmit the message later when the Slave is free.
08	Memory Parity Error	Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The Slave attempted to read record file, but detected a parity error in the memory. The Master can retry the request, but service may be required on the Slave device.
0a	Gateway Path Unavailable	Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
0b	Gateway Target Device Failed To Respond	Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

5.5 Configuration Form

This section contains configuration forms that will aid in the configuration of the module. If you design your system before trying to directly implement it, you will have a greater chance of success. Fill in the configuration form for your application, then edit the configuration text file.

[Section]/Item	Value	Range	Description
[Module]			Configuration header for Module.
Module Name:		Up to 80 chars	Name of the module for use on reports. Use this parameter to identify your module in your system.
Maximum Register:		1 to 3996	This parameter defines the maximum register in the virtual Modbus Database. You should size the database for your application leaving room for expansion in the future. Requests for registers outside of the range selected will be returned with an error message.
Error/Status Block Pointer:		-1 to 3995	This value represents the relative starting position in the module's internal Modbus database where the Error/Status data will be stored. The table can be placed anywhere in the module's data space. The content of the Error/Status table is updated at the frequency defined in the Error/Status Frequency. If a value of -1 is set for the parameter, the data will not be placed in the database.
Error/Status Frequency:		0 to 65535	This parameter specifies the number of program cycles between each update of the Error/Status Block data in the module. If the parameter is set to a value of 0, the data is never updated.
BT Read Start Register:		0 to 3995	This parameter specifies the starting register in the internal Modbus database to write over the backplane.
BT Read Register Count:		12 to 3996	This parameter specifies the number registers in the internal Modbus database to write over the backplane. This parameter computes the number of blocks to transfer from the module to the backplane. The number of blocks must be >= 2 for proper backplane data transfer.
BT Write Start Register:		0 to 3995	This parameter specifies the starting register in the internal Modbus database to fill with data read over the backplane.
BT Write Register Count:		12 to 3996	This parameter specifies the number registers in the internal Modbus database to consider from the read operations over the backplane. This parameter computes the number of blocks to transfer from the backplane to the module. The number of blocks must be >= 2 for proper backplane data transfer.
Backplane Fail Count:		0 to 65535	This parameter specifies the number of consecutive backplane transfer failures before communications is disabled. If the parameter is set to a value of 0, communications is not disabled.
Type:		0, 1, 3, 4	This parameter specifies if the port will emulate a Modbus master device (0), a Modbus slave device without pass-through (1), a Modbus slave device with pass-through and data swapping (3), or a Modbus slave device with pass-through (4).

[Section]/Item	Value	Range	Description
Float Flag:		Y or N	This flag specifies if the floating-point data access functionality is to be implemented. If the float flag is set to Y, Modbus functions 3, 6 and 16 will interpret floating point values for registers as specified by the two following parameters.
Float Start:		0 to 32767	This parameter defines the first register of floating-point data. All requests with register values greater-than or equal to this value will be considered floating-point data requests. This parameter is only used if the Float Flag is enabled. For example, if a value of 7000 is entered, all requests for registers 7000 and above will be considered as floating-point data.
Float Offset:		0 to 3995	This parameter defines the start register for floating-point data in the internal database. This parameter is only used if the Float Flag is enabled. For example, if the Float Offset value is set to 3000 and the float start parameter is set to 7000, data requests for register 7000 will use the internal Modbus register 3000.
Protocol:		0 or 1	This parameter specifies the Modbus protocol to be used on the port. Valid Protocols are 0=Modbus RTU and 1=Modbus ASCII.
Baud Rate:		110 to 115K	This is the baud rate to be used on the port. Enter the baud rate as a value. For example, to select 19K baud, enter 19200. Valid entry for this field include: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600 and 115.
Parity:		0 to 4	This is the Parity code to be used for the port. The coded values are as follows: 0=None, 1=Odd, 2=Even, 3=Mark and 4=Space.
Data Bits:		5 to 8	This parameter sets the number of data bits for each word used by the protocol.
Stop Bits:		1 or 2	This parameter sets the number of stop bits to be used with each data value sent.
RTS On:		0 to 65535	This parameter sets the number of milliseconds to delay after RTS is asserted before the data will be transmitted.
RTS Off:		0 to 65535	This parameter sets the number of milliseconds to delay after the last byte of data is sent before the RTS modem signal will be set low.
Minimum Response Delay		0 to 65535	This parameter specifies the minimum number of milliseconds to delay before responding to a request message. This pre-send delay is applied before the RTS on time. This may be required when communicating with slow devices.
Use CTS Line:		Y or N	This parameter specifies if the CTS modem control line is to be used. If the parameter is set to N, the CTS line will not be monitored. If the parameter is set to Y, the CTS line will be monitored and must be high before the module will send data. This parameter is normally only required when half-duplex modems are used for communication (2-wire).
[Modbus Master]			Configuration header for Modbus master.
Command Count:		0 to 100	This parameter specifies the number of commands to be processed by the Modbus Master port.

[Section]/Item	Value	Range	Description
Minimum Command Delay:		0 to 65535	This parameter specifies the number of milliseconds to wait between issuing each command. This delay value is not applied to retries.
Command Error Pointer:		-1 to 3995	This parameter sets the address in the internal Modbus database where the command error data will be placed. If the value is set to -1, the data will not be transferred to the database.
Response Timeout:		0 to 65535	This parameter represents the message response timeout period in 1-ms increments. This is the time that a port configured as a master will wait before re-transmitting a command if no response is received from the addressed slave. The value is set depending upon the communication network used and the expected response time of the slowest device on the network.
Retry Count:		0 to 10	This parameter specifies the number of times a command will be retried if it fails.
Error Delay Count:		0 to 65535	This parameter specifies the number of polls to skip on the slave before trying to re-establish communications. After the slave fails to respond, the master will skip commands to be sent to the slave the number of times entered in this parameter.
[Modbus Slave]			Configuration header for Modbus slave.
Internal Slave ID:		0 to 255	This parameter defines the virtual Modbus slave address for the internal database. Any requests received by the port with this address will be processed by the module. Verify that each device has a unique address on a network.
Bit Input Offset:		0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 2 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
Word Input Offset:		0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 4 commands. For example, if the value is set to 150, an address request of 0 will return the value at register 150 in the database.
Output Offset:		0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 1, 5, or 15 commands. For example, if the value is set to 100, an address request of 0 will return the value at register 100 in the database.
Holding Register Offset:		0 to 3995	This parameter specifies the offset address in the internal Modbus database for network requests for Modbus function 3, 6, or 16 commands. For example, if the value is set to 50, an address request of 0 will return the value at register 50 in the database.
Use Guard Band Timer		Y or N	This parameter specifies if the packet Guard Band is to be used. This is used for multi-drop only.
Guard Band Timeout		0 to 65535	This parameter specifies the Guard Band time between packets in multi-drop slave mode. A value of 0 uses the default time. A value of 1 to 65535 sets the time in milliseconds.

Module Information ← | → Device Information

MODBUS COMMAND ENTRY FORM

Column #	1	2	3	4	5	6	7	8
	Enable Code	Internal Address	Poll Interval Time	Count	Swap Code	Node	Function Code	Device Modbus Address

6 Support, Service & Warranty

In This Chapter

6.1 Contacting Technical Support

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

1 Product version number
2 System architecture
3 Network details

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

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

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

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

6.2 Warranty Information

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

Documentation is subject to change without notice.

Index