Introduction:

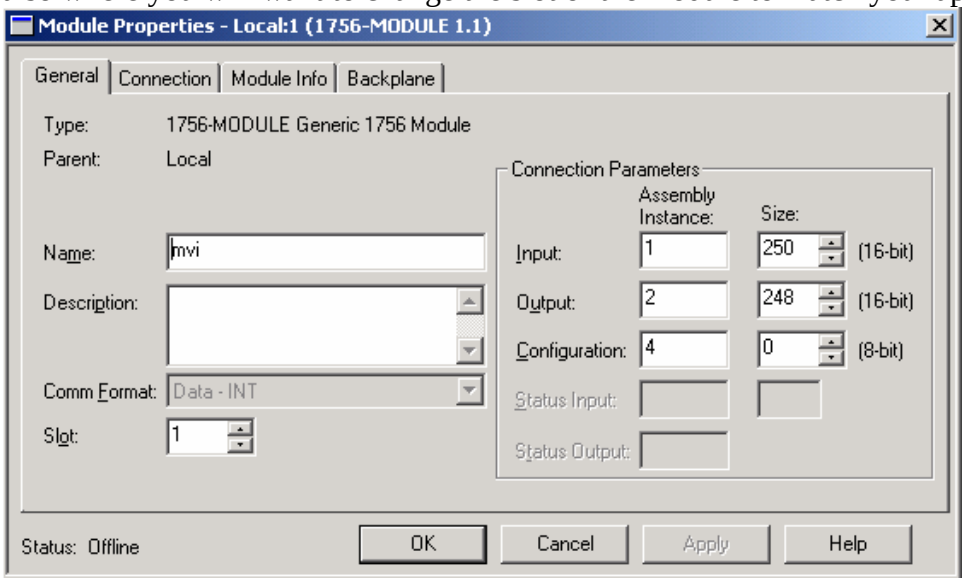
Implementation of the MVI56-MCM module for modbus communications:

Initial configuration of the module should be done using the sample ladder file for the mvi56mcm module. This can be obtained from our ftp site at the following address: ftp://ftp.prosoft-technology.com/pub/Ladder/MVI56_MCM/

You will want to download this file into a new control logix application, or copy it into an existing application.

I. Module Setup and Overview

First of all, you will want to configure the module Input and Output image size (This will already be configured if using the sample ladder file, but since proper operation of the module depends on this, it's best to verify this configuration). The diagram below shows the proper configuration for the input and output image size of the module. This field is also where you will want to change the slot of the module to match your application.



Make sure that the Comm Format is set for Data-INT. This will set the size for 16-bit integer values. If this is configured inaccurately, then you will not get the proper size of the data transfer, or you may get an error for the module that reads: "failed to modify properties, invalid input size".

Now that the setup of the module has been verified, you will want to move on to the configuration of the module for communications. For this you will want to go under the **controller tags** location on the project tree. All of the configuration for the module will be done under the tag **MCM**. In order for any of the new values entered into these fields to be used by the module, you will need to either WarmBoot, ColdBoot, cycle power to the module, or re-download the program to the processor. The diagram below shows all of the various locations for the module configuration.

⊟-MCM	{...}	{...}		MCMModuleDef
⊞-MCM.ModDef	{...}	{...}		MCMModule
⊞-MCM.CFG_Port	{...}	{...}		MCM_CfgPort
⊞-MCM.Port1	{...}	{...}		MCMPort1
⊞-MCM.Port2	{...}	{...}		MCMPort2
⊞-MCM.P1Cmd	{...}	{...}		MCMCmd[100]
⊞-MCM.P2Cmd	{...}	{...}		MCMCmd[100]
⊞-MCM.InStat	{...}	{...}		MCMInStat
⊞-MCM.ReadData	{...}	{...}	Decimal	INT[600]
⊞-MCM.WriteData	{...}	{...}	Decimal	INT[600]
⊞-MCM.BP	{...}	{...}		MCMBackplane
⊞-MCM.Port1Aux	{...}	{...}		MCMPort1Aux
⊞-MCM.Port2Aux	{...}	{...}		MCMPort2Aux

The tab **MCM.ModDef** will be used to setup the database of the module. The module has a 5000 register database that can be used for read and write data for the module. This area sets up what registers are to be used for read data, and what registers will be used for write data. The tags label **MCM.ReadData** and **MCM.WriteData** will be stored in the module based on the locations defined under this tag.

MCM.Port1 and MCM.Port2 are used to configure the serial ports configuration for the modbus communications. This area is used to setup the port as a master or slave, baud rate, protocol, and various other configurations.

The next two tags **MCM.P1.Cmd** and **MCM.P2.Cmd** will be used to setup the modbus master commands for the module. This area will be ignored when the port is configured as a slave device. Each port can be configured for up to 100 modbus master commands. The number of commands that will be processed by a master device is based on the tag **MCM.Port?.CmdCount**. Only the number of commands specified in this tag location will be processed by the master.

The tags **MCM.InStat** and **MCM.BP** provide status information for the modbus communications on each of the ports, and the backplane communications of the module and the Control Logix processor .

Data read from either a master device writing to the port as a slave, or data read from a slave device will be located in the tag labeled **MCM.ReadData**. This location is data read from the MVI56-MCM module back into the control logix processor.
Data placed in the tag name **MCM.WriteData** is to be used for information that is in the control logix processor and needs to be written out to the module's internal database. This will include information that needs to be written out to a slave device (when setup as a master), or information that has been written to the module when the port is configured as a slave.

II. Module Configuration

This section will cover the configuration of the module. Further information on this can be found in the MVI56-MCM Setup Guide and the MVI56-MCM User Manual. Both of these documents can be found at the following website:
ftp://ftp.prosoft-technology.com/Manuals/MVI56_MCM/

The first step in the configuration of the module is in the tag labeled **MCM.ModDef**. This will setup the read and write data locations.

MCM.ModDef	{...}
MCM.ModDef.WriteStartReg	0
MCM.ModDef.WriteRegCnt	600
MCM.ModDef.ReadStartReg	600
MCM.ModDef.ReadRegCnt	600
MCM.ModDef.BPFail	0
MCM.ModDef.ErrStatPtr	-1

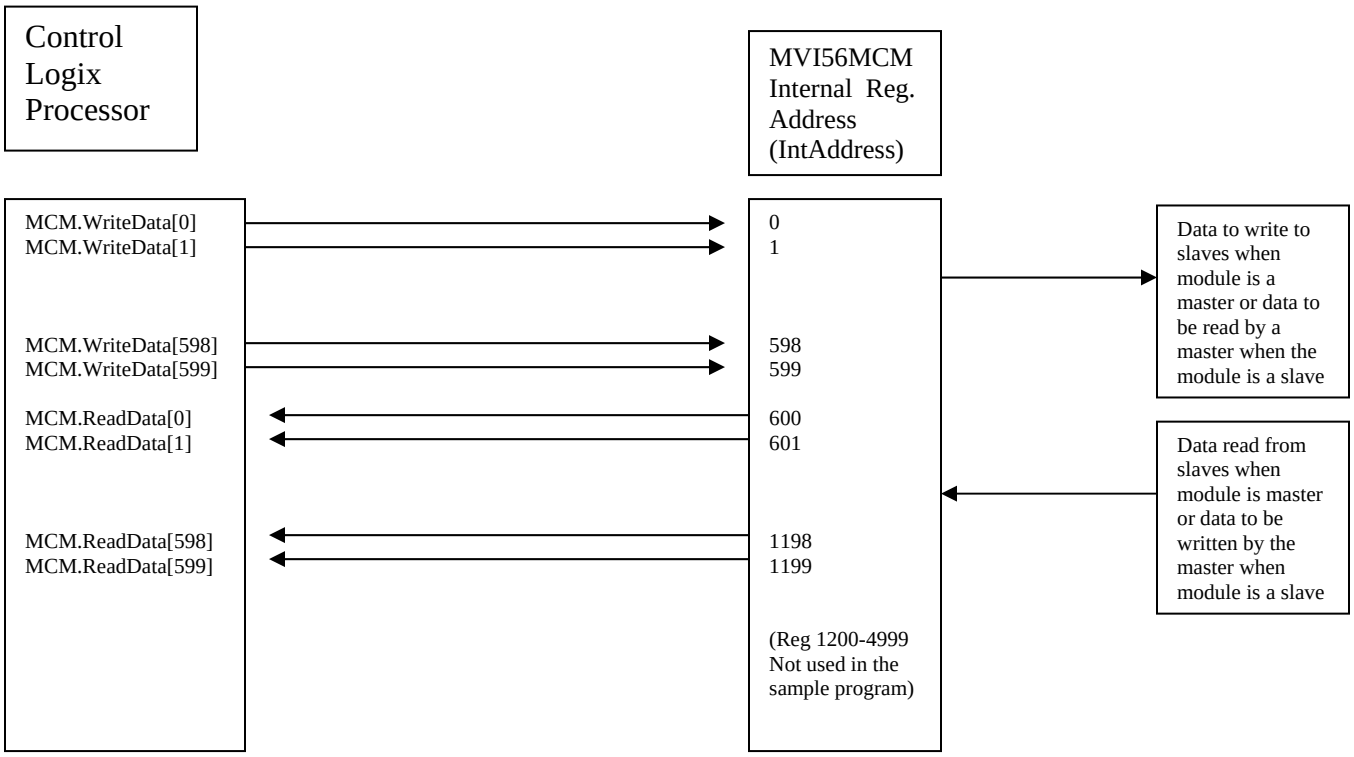
The tags **MCM.ModDef.WriteStartReg and MCM.ModDef.WriteRegCnt** will be used to determine which of the module's 5000 register will be used for data to be written from the control logix processor out to the MVI56MCM module. The **WriteStartReg** will be used to determine the starting register location for **WriteData [0-599]** and the **WriteRegCnt** will be used to determine how many of the 5000 registers will be used for information to be written out to the module. The sample ladder file will setup 600 registers for write data, labeled **MCM.WriteData[0-599].** For additional registers, you will need to change the size of the array from 600 to the value you need (Changing the array size will zero out all tags in the MCM tag location). Since the module pages data in blocks of 200 registers at a time, you will want to keep this as a number divisible by 200 and also change rung 10 in the ladder file WriteData. The second LEQ statement will need the Source B value changed to reflect these new registers. The value in this statement will need to be equal to the number of registers for write data/ 200. Example 2000 registers will need a value of 10 (10 = 2000/200).
The tags **MCM.ModDef.ReadStartReg** and **MCM.ReadRegCnt** will configure in the same manner, the only difference is that this is for registers to be read from the module into the control logix processor. You will want to setup this as unique registers, and make sure that it will not overlap the **WriteData** registers. To add additional registers for **ReadData** you will want to change the array size and modify rung 2 of the ReadData ladder routine.
The tag **MCM.ModDef.BPFail** will determine the number of consecutive times that backplane communications will fail before the module will stop responding on the modbus ports. A value of zero will have the module respond on the modbus ports even if the backplane communications has been lost.

Finally the tag **MCM.ModDef.ErrStatPtr** determines where in the module database you will place status information. A value of –1 disables this parameter for the sample application. All of this information is also available in the MCM.InStat fields.

The following configuration sets up the module database for **WriteData[0-599]** to be stored in the module memory at register 0-599, and **ReadData[0-599]** to be stored in the module memory at registers 600-1199 like shown below:

-MCM.ModDef	{...}
-MCM.ModDef.WriteStartReg	0
-MCM.ModDef.WriteRegCnt	600
-MCM.ModDef.ReadStartReg	600
-MCM.ModDef.ReadRegCnt	600
-MCM.ModDef.BPFail	0
-MCM.ModDef.ErrStatPtr	-1

With the above configuration, the following is the layout of the tags and addressing:



III. Port Setup

The following section will setup the configuration of the port for communications with the various modbus devices. These values will need to be configured for Port1 and Port2.

MCM.Port1	{...}	{...}		MCMPort1
MCM.Port1.Enabled	1		Decimal	INT
MCM.Port1.Type	0		Decimal	INT
MCM.Port1.FloatFlag	0		Decimal	INT
MCM.Port1.FloatStart	0		Decimal	INT
MCM.Port1.FloatOffset	0		Decimal	INT
MCM.Port1.Protocol	0		Decimal	INT
MCM.Port1.Baudrate	19200		Decimal	INT
MCM.Port1.Parity	0		Decimal	INT
MCM.Port1.DataBits	8		Decimal	INT
MCM.Port1.StopBits	1		Decimal	INT
MCM.Port1.RTSOn	0		Decimal	INT
MCM.Port1.RTSOff	0		Decimal	INT
MCM.Port1.MinResp	0		Decimal	INT
MCM.Port1.UseCTS	0		Decimal	INT
MCM.Port1.SlaveID	0		Decimal	INT
MCM.Port1.BitInOffset	0		Decimal	INT
MCM.Port1.WordInOffset	0		Decimal	INT
MCM.Port1.OutOffset	0		Decimal	INT
MCM.Port1.HoldOffset	0		Decimal	INT
MCM.Port1.CmdCount	100		Decimal	INT
MCM.Port1.MinCmdDelay	0		Decimal	INT
MCM.Port1.CmdErrPtr	1100		Decimal	INT
MCM.Port1.RespTO	1000		Decimal	INT
MCM.Port1.Retry_Count	2		Decimal	INT
MCM.Port1.ErrorDelayCntr	0		Decimal	INT

The tag **MCM.Port1.Enabled** will either enable or disable the port (note: even a port that is disabled will require valid data values for baud rate, data bits, and stop bits). Valid values for this location are 0 for disabled, 1 for enabled.
MCM.Port1.Type will determine if the port is to be used as a modbus master port or a modbus slave port. Valid values for this field are as follows:

0- Modbus master
1- Modbus slave port
2- Modbus slave port with pass thru. This will allow for a write message to this slave to be passed through the MVI56-MCM module database, and go directly into the ladder logic. This module will set the MCM.BP.LastRead value to 9996 and the modbus write command will be handled by rung 8 in the ReadData ladder file in the expanded ladder logic. This will allow for an unparsed modbus message to be moved into the tag location **MBMsg[0-499].** Here you will need to

parse out the data value and move it into the appropriate registers using the ladder logic.

3- Modbus slave port with formatted pass thru and data values swapped. This mode will allow for the same register to be read and written by a modbus master device, and will also swap the bytes within the data value (some devices may require the byte swapping, but with most device you will want to use a value of 4).
4- Modbus slave port with formatted pass thru (no swapping). This mode will also allow for the same register location to be read and written by the master device. Rungs 9, 10, and 11 in the ReadData ladder file will handle this information. Special consideration must be taken into effect when implementing this mode. Please see the MVI56_MCM_User_Manual for more details on this mode of operation.

Note: Port type 2 through 4 require the expanded ladder logic samples.

The tags **MCM.Port1.FloatFlag, MCM.Port1.FloatStart, and MCM.Port1.FloatOffset** can be used to setup floating point data. Please see the Set Up Guide for more information on these features.

MCM.Port1.Protocol will define whether the module will operate using Modbus RTU (value of 0), or Modbus ASCII (value of 1). When this is an option, you will want to use Modbus RTU mode since data transfer in this mode is more reliable and efficient.

MCM.Port1.Baudrate is to be used to setup the baud rate of traffic on the serial port. Valid values for this field are 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 28800, 384 or 3840 (for 38,400), 576 or 5760 (for 57,600), and 115, 1152, or 11520 (for 115,200).

The tag **MCM.Port1.Parity** will define the type of parity to be used by that port. Valid values for this field are 0= None, 1= Odd, 2= Even, 3= Mark, and 4= Space.

The tags **DataBits**, and **StopBits** setup how many data bits and how many stop bits will be used in the Modbus message. Valid values for **DataBits** are 7 or 8, and valid values for **StopBits** are 1 or 2.

Note: Modbus RTU mode will always need to use 8 data bits

The tags **MCM.Port1.RTSOn** and **MCM.Port1.RTSOff** will configure RTS on and off delays for the port. These values are helpful when the port is using RS-422 or RS-485 wiring. Values in this field are represented in milliseconds. A value of 10 will configure the delay for 10 milliseconds. Generally a value of 0 for the **RTSOff** is recommended as too large of a value in this field can cause you to miss the beginning part of the return response.

The **MinResp** tag is used to set the minimum number of milliseconds the port, configured as a slave, will wait before sending a response back to the master.

UseCTS will enable the use of the cts line. A value of 0 will not enable the CTS line, while a value of 1 will enable the use of the CTS line.

The tag **MCM.Port1.SlaveID** will be used to set the slave ID number of the module when the port is configured as a slave.

The next set of values, **BitInOffset, WordInOffset, OutOffset,** and **HoldOffset** will configure the internal database offset values for that port. This will only be used when the port is configured as a slave, and when you need to setup specific modbus addressing. Please refer to the MVI56_MCM_Set_Up_Guide for more details on the use of these features.

The value **CmdCount** will configure the number of modbus master commands that the module will issue to the various slave devices on that port. The value in this field will determine the number of **MCM.P1** or **P2.Cmd** that the port will scan.
Example: A value of 10 in this field will allow the master to look at **P1.Cmd[0] – P1.Cmd[9]**.

The **MinCmdDelay** will setup the minimum number of milliseconds the master will wait before sending a query out to the slave device. This is a delay time between commands

The **CmdErrPtr** will determine where in the module database the command errors will be stored. This will need to be place in a location that is in the **ReadData** tag location for this information to be seen. A value of 1100 will place this information in the location **ReadData[500]** and will extend for the number of commands that are configured in the **CmdCount** field. Each Modbus master command will have an error code associated with it. A value of 0 indicates that the command is operating properly, for a complete list of error codes refer to the MVI56_MCM_User_Manual.

MCM.Port1.RespTO will configure the minimum number of milliseconds the port configured as a master will wait for a response from the slave device. A value of 0 will default to 1000 milliseconds or 1 second.

RetryCount will set the number of times the master will retry a command to a slave device before moving on to the next command. A value of 2 will setup 2 retries. This means that a message will be sent out to the slave device, if there is no response, then the master will retry that command 2 more times before moving onto the next command in the scan list.

MCM.Port1.ErrorDelayCntr can be used to setup the number commands that will be skipped in the **P1.Cmd** field when a command has gone into error. For most applications a value of 0 is best for this field, since when configured inaccurately this can cause communications with the slave devices to cease.

IV. Master Command Configuration

The following section will detail the configuration of the Modbus master commands. This section determines the communications with the master port and the slave devices that are connected to that port.

MCM.P1Cmd	{...}	{...}		MCMCmd[100]
MCM.P1Cmd[0]	{...}	{...}		MCMCmd
MCM.P1Cmd[0].Enable	1		Decimal	INT
MCM.P1Cmd[0].IntAddress	600		Decimal	INT
MCM.P1Cmd[0].PollInt	0		Decimal	INT
MCM.P1Cmd[0].Count	10		Decimal	INT
MCM.P1Cmd[0].Swap	0		Decimal	INT
MCM.P1Cmd[0].Device	1		Decimal	INT
MCM.P1Cmd[0].Func	3		Decimal	INT
MCM.P1Cmd[0].DevAddress	0		Decimal	INT
MCM.P1Cmd[1]	{...}	{...}		MCMCmd
MCM.P1Cmd[1].Enable	1		Decimal	INT
MCM.P1Cmd[1].IntAddress	9760		Decimal	INT
MCM.P1Cmd[1].PollInt	0		Decimal	INT
MCM.P1Cmd[1].Count	160		Decimal	INT
MCM.P1Cmd[1].Swap	0		Decimal	INT
MCM.P1Cmd[1].Device	1		Decimal	INT
MCM.P1Cmd[1].Func	1		Decimal	INT
MCM.P1Cmd[1].DevAddress	0		Decimal	INT

The tag **MCM.P1.Cmd[0].Enable** will need to be configured for the command to be executed. A value of 0 in this field will disable the command and it will not be sent out the port. A value of 1 will enable the command to be transmitted by the master every time it comes to that command in the scan list. If a value of 2 is entered in this field, the command will only be issued when the data to be written out to the slave device has changed. This can only be used on write commands to be sent out to the slaves.

IntAddress will determine where in the module's 5000 register database the data will be stored to or written from. On a read command this will determine once the information has been read from a slave, where it will be placed in the module database. On read commands you will want to configure this for a location that is setup for **ReadData**. The internal database location of the **ReadData** and **WriteData** tags is determined by the configuration setup in the **MCM.ModDef** tag location.
For write data the **IntAddress** will determine where to obtain the information to be written out to the slave device. This will need to be a location that is setup as **WriteData**.

NOTE: When using a bit level command you will want to define this field at the bit level. For instance, when using a function code 1,2 for a read command you will need to have a value of 9600 to place the data in **MCM.ReadData[0]** (register 600 * 16 bits per register = 9600). In the above configuration for **MCM.P1Cmd[1]** a value of 9760 will place the data in address 610 of the module memory and this will then be stored in **MCM.ReadData[10]**.

The tag **PollInt** can be used to setup an interval for the data to be polled at. Values in this field will represent the number of seconds that a master device will wait before issuing this command. This is helpful for non-critical information. For example a value of 60 will setup this command to be issued every 60 seconds or 1 minute.

MCM.P1.Cmd[0].Count will setup how many words (when using a word range read or write command) or bits will be read from the slave device.

Swap will be used to setup the swapping of the data values received from the slave device. This is useful when reading floating point data. Valid values for this field are as follows:

0- Module performs no data swapping (mode that will be used for most applications).
1- Swap word values from each word pair received.
2- Swap word and byte values for each word pair received.
3- Swap bit values.

Device will determine the modbus slave address that the command will be issued to. Valid values for a Modbus slave are 1-255. A **Device** of 0 will be used to issue write commands to all slaves on this port(cannot be used for read commands).

MCM.P1.Cmd[0].Func will determine the modbus function code that will be issued in the command to the slave device. Valid values for this field are as follows:

1- Read Coil Status. This will read modbus addresses 0001-9999. These are bit values used to indicate coil status, and can also be written to using a function code of 5 or 15.
2- Read Input Coils. This will read modbus addresses 10001-29999. These are also bit values, but are read only data values.
3- Read Holding Registers. This is to be used for Modbus addresses 40001-47999. This is a 16 bit word value, and can be written to using the function codes of 6 and 16.
4- Read Input Registers. Will read modbus addresses 30001-39999. These are also 16 bit word values, but are read only data, and cannot be written to by the master.
5- Write Single Coil Status. This will write to modbus addresses 0001-9999. This command will write to only one coil. If you want to write to multiple coils you will need to use a function code of 15.
6- Write Single Register. For modbus addresses 40001-47999. This will write one single register value out to a slave device. For multiple register writes you will need to use a function code of 16.

15- Multiple Coil Write. This function code will write multiple coil values to the slave addresses 0001-9999.
16- Multiple Register Write. Will write multiple register values to the slave device at addresses 40001-47999.

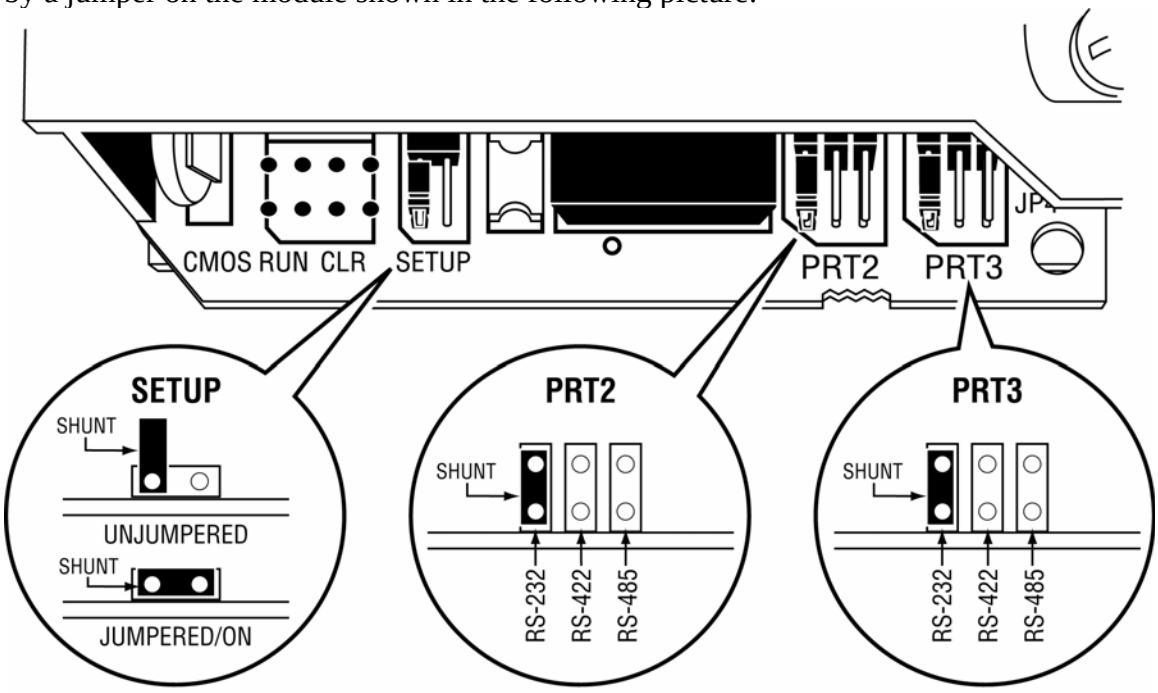
DevAddress will be used to indicate the modbus slave address for the register or registers associated with that command. This is the offset address for the modbus slave device. With modbus, to read an address of 40001, what will actually be transmitted out port is a function code of 03 (one byte) with an address of 00 00 (two bytes). This means that to read an address of 40501, you would want to put a **Func** of 3 with a **DevAddress** of 500. This applies to modbus addresses 10001-47999. This is why you will always need to make sure of the function code that is to be used to read or write information from the slave.
Below is a definition that will help with your DevAddress setup:

FC 1,5, or 15 **DevAddress** = Modbus address in device – 0001
Example: Modbus address 0001 = DevAddress 0
Modbus address 1378 = DevAddress 1377
FC 2 **DevAddress =** Modbus address in device – 10001
Example: Modbus address 10001 = DevAddress 0
Modbus address 10345 = DevAddress 344
FC 3,6, or 16 **DevAddress** = Modbus address in device – 40001
Example: Modbus address 40001 = DevAddress 0
Modbus address 40591 = DevAddress 590
FC 4 **DevAddress** = Modbus address in device – 30001
Example: Modbus address 30001 = DevAddress 0
Modbus address 34290 = DevAddress 4289

Note: Some devices show their addressing already as an offset address (the address that actually goes out on the modbus communication line). If your device manufacturer tells you to use a function code 3 with an address of 100, then that is what you want to use in the DevAddress field. If they give you the modbus address like shown above, use the offset formula.

V. Jumper and wiring settings

The MVI56MCM module will support RS232, 422, and 485 wiring. This selection is set by a jumper on the module shown in the following picture:



The setup jumper should always be in the UNJUMPERED position, unless instructed by the factory to install this jumper. Each of the PRT jumpers is clearly labeled. PRT2 is the label for P1, and PRT3 is the jumper setting for P2 (the bottom port).

The following is the wiring diagram for each type of cable:

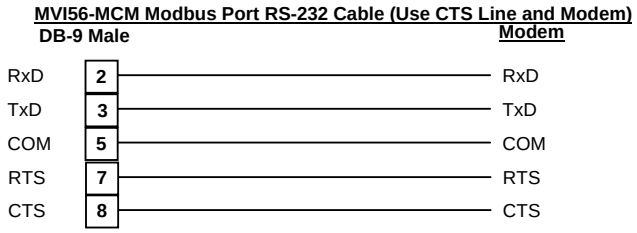
RS-232

When the RS-232 interface is selected, the use of the modem control lines is user definable. If no modem control lines will be used, the cable to connect to the port is as shown below:

MVI56-MCM Modbus Port RS-232 Cable (No Handshaking)

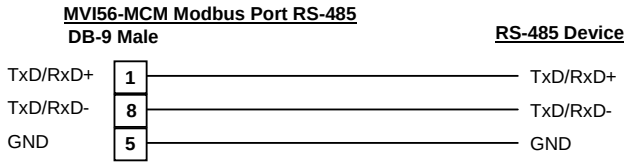
DB-9 Male		RS-232 Host
RxD	2	TxD
TxD	3	RxD
COM	5	COM

The RTS line is controlled by the RTS on and off parameters set for the port. If the CTS line is used (usually only required for half-duplex modems), the RTS and CTS lines must either be connected together or connected to the modem. The diagram below displays the cable required when connecting the port to a modem.



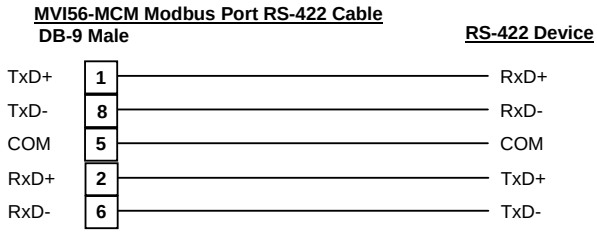
RS-485

When the RS-485 interface is used, a single two or three wire cable is required. The use of the ground is optional and dependent on the RS-485 network. The cable required for this interface is shown below:



RS-422

When the RS-422 interface is used, a four or five wire cable is required. The use of the ground is optional and dependent on the RS-422 network. The cable required for this interface is shown below:



The above wiring diagrams show the pins used for each of the different wiring types. The module ships with a DB9 to RJ45 connector to ease in the wiring to the module. If you wish to make your own RJ45 connection to go directly into the module then you will use the same pins as specified above. The wiring of the RJ45 to DB9 adapter cable is just a straight through cable (pin 1 on RJ45 to pin 1 on DB9, 2-2, 3-3, etc…)

When using RS485 wiring, if you do not have communications at first, try switching the + and – lines. Different device manufacturers have different interpretations of + and -.

The module ships with 2 DB9 to screw terminal adapters (1454-9F). These are designed to make an easy connection to the module when using RS422 or RS485 (not to be used for RS232 mode).

This manual was designed to ease implementation of ProSoft Technology's MVI56-MCM module. Not all of the features and specifications of the module were described in this manual. For complete functionality of the module you will need to reference the following documents:

MVI56_MCM_User_Manual

These documents can be found at on our ftp site at the following location:
ftp://ftp.prosoft-technology.com/pub/Manuals/MVI56_MCM