Chapter 3 Host Communication

While a T100MD+ or T100MX+ PLC is running, a host computer or another T100M+ PLC (this abbreviation is used to refer to both the T100MD+ and T100MX+ in this manual) may send ASCII string commands to it to read or write to its inputs, outputs, relays, timers, counters and all the internal variables. These ASCII commands are known as the "host-link commands" and are to be serially transmitted (via RS232C or RS485 port) to and from the controller. The default serial port settings of T100M+ PLC for host-link communication are: *38400 baud, 8 data bit, 1 stop bit, no parity*. The baud rate and the communication format may be changed using the "SetBAUD" TBASIC command described in the Programmer's Reference Part II - TBASIC.

Multiple Communication Protocols

The competent T100M+ family of PLCs supports many different communication protocols to allow maximum application flexibility. In addition to its own native set of communication protocols, the T100M+ PLC also understands and speaks the following protocols:

1. ***MODBUS□** ASCII mode compatible communication protocol.
2. ***MODBUS□** RTU mode compatible communication protocol.
   (For Rev D board with Firmware revision r32 and above only)
3. ***OMRON□** Host Link Commands for the C20H PLC family.

*Note: all trademarks belong to their respective owners.

The native host link command protocol will be described in detail in this chapter as well as in Chapter 4. The MODBUS and OMRON compatible protocols will be described in Chapter 5.

Native Mode Communication Protocols

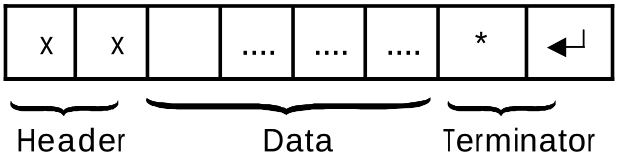
When a T100M+ PLC receives a native host-link command from COMM1 or COMM3, it will automatically send a response string corresponding to the command. This operation is totally transparent to the user and need not be handled by the user's program.

All T100M+ PLCs support both point-to-point (one-to-one) and multi-point (one-to-many) communication protocols. Each protocol has a different command structure as described below:

3.1 POINT-TO-POINT COMMUNICATION

In a point-to-point communication system, the host computer's RS232C serial port is connected to the PLC's COMM1. At any one time, only one controller may be connected to the host computer. The host-link commands do not need to specify any controller ID code and are therefore of simpler format, as shown below:

Command/Response Block Format (Point to Point)



Each command block starts with a two-byte ASCII character header, followed by a number of ASCII data and ends with a terminator which comprises an '*' character and a carriage return (ASCII value = 13_{10}). The header denotes the purpose of the command. For example, RI for Read Input, WO for Write Output, etc. The data is usually the hexadecimal representation of numeric data. Each byte of binary data is represented by two ASCII characters (00 to FF).

To begin a communication session, the host computer must first send one byte of ASCII character: Ctrl-E (=05Hex) via its serial port to the controller. This informs the controller that the host computer wishes to send a (point-to-point) host-link command to it. Thereafter, the host computer must wait to receive an echo of the Ctrl-E character from the controller. Reception of the echoed Ctrl-E character indicates that the controller is ready to respond to the command from the host computer. At this moment, the host computer must immediately send the *command block* to the controller and then wait to receive the *response block* from the controller. The entire communication session is depicted in Figure 2-1.

After the controller has received the command, it will send a response block back to the host computer and this completes the communication session. If the controller accepts the command, the response block will start with the same header as the command, followed by whatever information that has been requested by the command and the terminator.

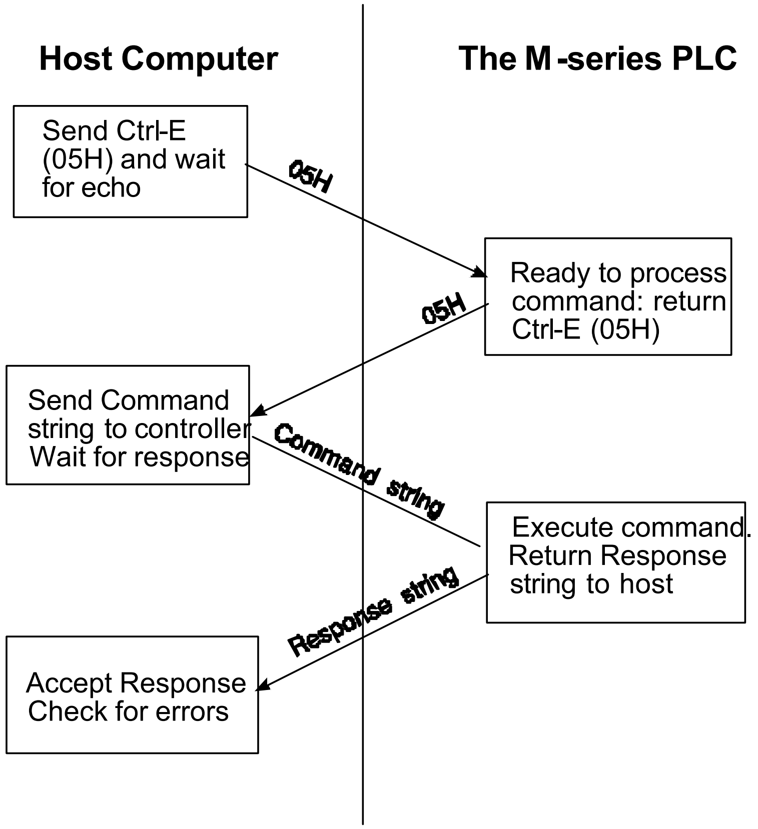

Host Computer
The M-series PLC
Send Ctrl-E (05H) and wait for echo
05H
Ready to process command: return Ctrl-E (05H)
05H
Send Command string to controller Wait for response
Command string
Execute command. Return Response string to host
Response string
Accept Response Check for errors

Figure 3.1

If an unknown command is received or if the command is illegal (such as access to an unavailable output or relay channel), the following error response will be received:

Error Response Format

E	R	*	↵

The host computer program should always check the returned response for possibilities of errors in the command and take necessary actions.

3.2 MULTI-POINT COMMUNICATION SYSTEM

In this system, one host computer may be connected to either a single T100M+ (via either RS232 or RS485) or multiple T100M+ PLCs on an RS485 network.

3.2.1 RS485 Network Interface Hardware

The built-in RS-485 interface allows the T100M+ controllers to be networked together using very low cost twisted-pair cables. Standard RS-485 allows up to 32 controllers (including the host computer node) to be networked together. When fitted with 1/8 power RS485 driver such as the 75HVD3082, up to 256 devices can be connected together. The twisted-pair cable goes from node to node in a daisy chain fashion and should be terminated by a 120 ohm resistor as shown below.

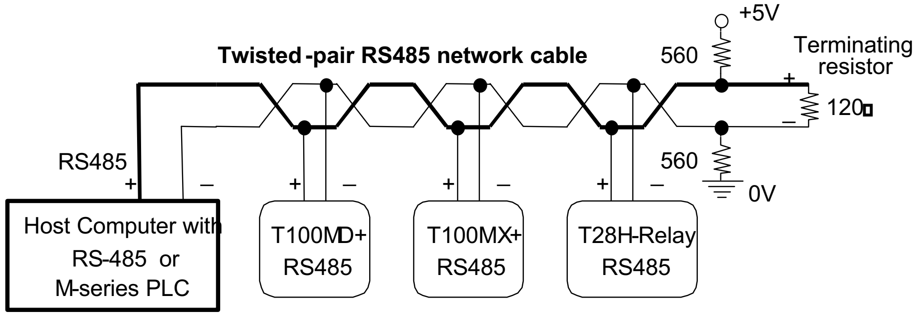


Figure 3.2

Note that the two wires are not interchangeable so they must be wired the same way to each controller. The maximum wire length should not be more than 1200 meters (4000 feet). RS-485 uses balanced or differential drivers and receivers, this means that the logic state of the transmitted signal depends on the differential voltage between the two wires and not on the voltage with respect to a common ground.

As there will be times when no transmitters are active (which leaves the wires in "floating" state), it is a good practice to ensure that the RS-485 receivers will indicate to the CPUs that there is no data to receive. In order to do this, we should hold the twisted pair in the logic '1' state by applying a differential bias to the lines using a pair of 560Ω to 1KΩ biasing resistors connected to a +9V (at least +5V) and 0V supply as shown in Figure 3-2. Otherwise, random noise on the pair could be falsely interpreted as data.

The two biasing resistors are necessary to ensure robust data communication in actual applications. Some RS485 converters may already have biasing built-in so the biasing resistors may not be needed. However, if the master is an M-series PLC then you should use the biasing resistor to fix the logic states to a known state. Although in lab environment the PLCs may be able to communicate without the biasing resistors, their use is strongly recommended for industrial applications.

3.2.2 Protection of RS485 Interface

The simple, direct multi-drop wiring shown in Figure 3-2 will work well if all the networked PLCs are in close proximity and they all share a common power supply. They will even work for long distance as long as no wiring error ever occurred. However, in an industrial environment, the PLCs are most likely far apart and they each may have their own power supply. Since processes are often modified regularly and if one day somebody by mistake shorts one of the PLC's RS485 to high voltage, **all the PLCs connected to the same RS485 wiring will be fried simultaneously.** This can result in very costly down time for the whole process, since all the PLCs connected to the network will need to be repaired.

Hence, for networking over long distances and involving more than a few PLCs, it is important to either strengthen or protect the RS485 interface, as described below:

1) You can replace the standard RS485 driver (75176) on the PLC by a fault-tolerant RS485 driver IC with part number LT1785AIN8. This 8 pin IC is made by Linear Technology and can withstand wrong voltages of up to ±60V! As an added bonus, the LT1785AIN8 is a 1/4 power RS485 driver, which means up to 128 PLCs can be connected together.

   Unfortunately this IC is much more expensive than 75176 and hence it is not provided as standard component on the T100M+ PLC. You can purchase the IC from any major electronic catalog company or contact sales@tri-plc.com for a quotation of this IC driver.

2) When using non fault-tolerant RS485 driver such as SN75176 or SN75HVD3082, we strongly recommend the following protection circuit to be added between every PLC's RS485 and the twisted pair multi-drop network cable:

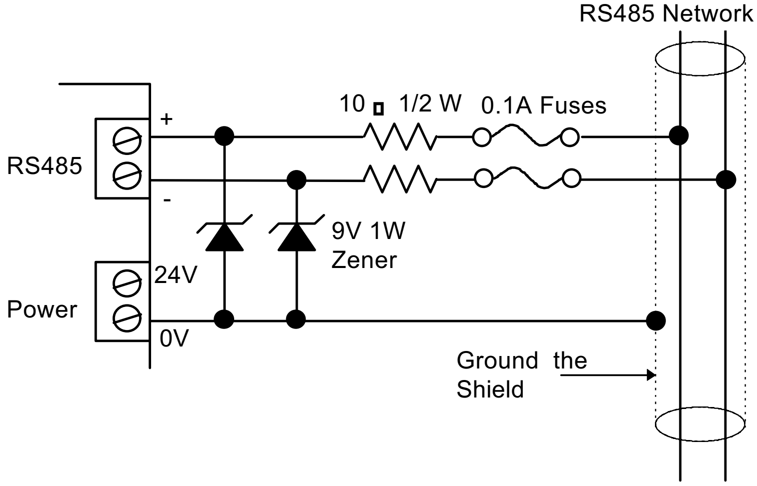


Figure 3.3

Note:

- As can be seen from the circuit, the two 9V Zener diodes clamp the signal voltage to the PLC's RS485 interface to between +9V and - 0.7V. If the high voltage persists, the 0.1A fuse will blow, effectively disconnecting the PLC from the offending network voltage.

- Even if you choose to replace the RS485 driver by LT1785AIN8 IC instead of using the zener/fuse pair wiring, you should still use shielded twisted pair cables as the multi-drop network "backbone" and connect the shield to the 0V (DC ground) power terminal of every PLC. The grounded shield then provides a common ground reference for all the different PLCs' power supplies. Even though the RS485 network may still work without a common ground reference because the signal wire pair will somehow "pull" all the RS485 to some reference point. **Failure to provide a common ground is a potential source of serious trouble** as signal wires with a floating ground easily induce large voltage differences between nodes when subjected to electromagnetic interference. Hence for reliable operation it is important to provide the common ground. A grounded shield also has the additional advantage of shielding the electrical signals from EMI.

3.2.3 Single Master RS485 Networking Fundamentals

RS485 is a half-duplex network, i.e., the same two wires are used for both transmission of the command and reception of the response. Of course, at any one time, only one transmitter may be active. The T100M+ PLCs implement master/slave network protocol. The network requires a master controller, which is typically a PC equipped with an RS485 interface. In the case of a PC, you can purchase an RS-485 adapter card or an RS232C-to-RS485 converter and connect it to the RS232C serial port. A T100M+ PLC can also be programmed to act as the master, it can communicate with other PLCs by executing the "NETCMD$" function or the "READMODBUS" or the "WRITEMODBUS" commands (the latter two are for communicating using MODBUS protocols only).

Only the master can issue commands to the slave PLCs. To transmit a command, the master controller must first enable its RS-485 transmitter and then send a multi-point command to the network of controllers. After the last stop bit has been sent, the master controller must relinquish the RS485 bus by disabling its RS485 transmitter and enabling its receiver. At this point the master will wait for a response from the slave controller that is being addressed. Since the command contains the ID of the target controller, only the controller with the correct ID would respond to the command by sending back a response string. For the network to function properly, <u>it is obvious that no two nodes can have the same ID</u>. You can use the "Setup Serial Port" command in TLServer to set the ID for each M-series PLC. You can also use the "IW" command to set the device ID. Also, all nodes must be configured to the same baud rate and communication format.

Also, care should be taken to ensure that the power supplies for all the controllers are properly isolated from the main so that no large ground potential differences exist between any controllers on the network.

3.2.4 Multi-Masters RS485 Networking Fundamentals

Since any T100MD or T100MX is capable of sending out network commands, the obvious question is whether multiple masters are allowed on the RS485 network? It is possible to have multiple masters on a single RS485 network provided the issues of collision and arbitration are taken care of. There are several means to achieve these objectives:

1) <u>Multiple Access with Collision Detection</u>

   There is nothing to stop any PLC from sending out host-link commands to other PLCs. However, If more than one PLC simultaneously enables their transmitters and send out host-link commands, then the signals will conflict and the messages will be garbled up. If the network traffic is low, then the solution may be a matter of having the master check for the correct response after sending out a command string. If there is error in the response string, the master should back off the network for a short while (use different timing for different PLCs) and then re-send the command until a correct response string is obtained. This scheme is similar to the CSMA/CD (Carrier Sensing Multiple Access/Collision-Detection) commonly used in Ethernet.

Fortunately, the “NETCMD$” function of T100M+ PLC automatically senses the RS485 lines until they are free before sending out the command string to reduce the chance of a collision. It also checks the integrity of the response string for correct FCS (Frame Check Sequence) characters before returning the string (Please refer to the Programmer’s Reference for detail description of the NETCMD$( ) function).

However, the program must still check the following items in the response string to verify that the string returned from NETCMD$( ) function indeed comes from the PLC that it had talked to and not from another PLC (which tries to send a command to someone else):

i) The ID is correct
ii) The header is identical to the command string
iii) The length of response string is correct.

Pros and Cons: This method does not incur any hardware cost, but it requires careful programming and strict checking of the response string and hence requires more effort to program. It is also the least desirable if the network traffic is moderately high as many collisions will occur and there is danger of some undetected error being allowed to pass through.

2) Token Awarding Scheme

A “token” is a software means of telling a PLC that it has been given the right to temporarily act as the master. A T100MD+ PLC or a host PC can serve as the token master. An internal relay bit or a variable of the PLC can be defined as the token. The token master will begin by giving the token (i.e., by setting the token relay bit to ‘1’ or the token variable to some fixed value) to the first PLC on the list. The PLC that has the token can then send host-link commands to other PLCs. When it has finished the job it can then send a command to the token master to relinquish its token. If it is based on a fixed timing scheme the master can assume that the PLC will complete its job after a fixed time (say 0.1 seconds) and turn off its corresponding token relay bit.

The token master then passes the token to the next PLC on the list and so on until the last PLC has relinquished its token, and the token is passed back to the first PLC on the list again. This way at any one time there will only be one active network master (the one with the token) and hence there is no danger of conflicting signals or garbled messages to handle.

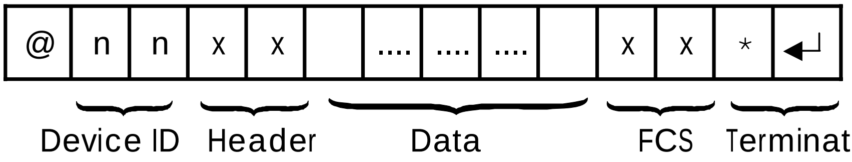
Pros and Cons: This method also does not incur any hardware cost, but it requires the programmer to draw up a plan on what internal relay or variable to use as the token and how the PLC can relinquish its token to the token master. (It could be by fixed timing or by returning a message to relinquish the token) It is a challenging job for programmers unfamiliar with networking scheme, but with some experimentation it can be achieved readily.

3) Rotating Master Signal

In this scheme we make use of the digital inputs of the T100M+ PLCs to grant the PLC the right to act as the network master. Lets call this input the “Be the Master” input. We can use a low cost H-series PLC running a sequencer to activate the “Be the Master” input line of each PLC one at a time. Each PLC is given a fixed amount of time to be the master (e.g. 0.1s each). Only when the “Be the Master” input is ON can the T100M+ PLC start sending out host-link commands to other PLCs. So at any one time there will only be one master on the network and no conflict will occur as a result.

Pros and Cons: This method is the easiest to program since there is no need to handle the token with the token master or perform extensive error check on the response string. However, this method uses one input of each PLC and as many outputs on the master-signal generator PLC as there are PLC masters. It also requires wiring the PLCs to the master-signal generator PLC and hence is the most costly method of all.

3.2.5 Command/Response Block Format (Multi-point)



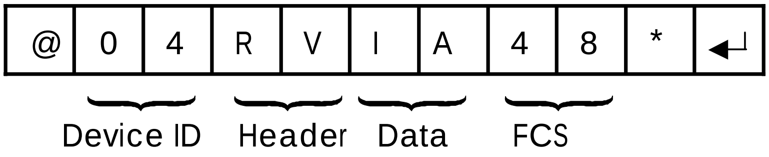
@	n	n	x	x		....	....	....		x	x	*	↵

Each command block starts with the character "@" and two-byte hexadecimal representation of the controller's ID (00 to FF), and ends with a two-byte "Frame Check Sequence" (FCS) and the terminator. FCS is provided for detecting communication errors in the serial bit-stream. If desired, the command block may omit calculating the FCS simply by putting the characters "00" in place of the FCS.

Note: we call “00” the “wildcard” FCS, which is available when the PLC is in “auto protocol” mode. This is to facilitate easy testing of multi-point protocol. However, the wildcard FCS is disabled if the PLC has executed the `SETPROTOCOL` *n*, 5 to put it’s COMM port *n* into pure native mode. In that case you will have to supply the actual FCS to your command string.

Calculation of FCS

The FCS is 8-bit data represented by two ASCII characters (00 to FF). It is a result of Exclusive OR sequentially performed on each character in the block, starting from @ in the device number to the last character in the data. An example is as follow:



@	0	4	R	V	I	A	4	8	*	↵

@	0100 0000
	XOR
0	0011 0000
	XOR
4	0011 0100
	XOR
R	0101 0010
	XOR
V	0101 0110
	XOR
I	0100 1001
	XOR
A	0100 0001

0100 1000 = 48_{16}

Value 48_{16} is then converted to ASCII characters '4' (0011 0100) and '8' (0011 1000) and placed in the FCS field.

FCS calculation program example

The following C function will compute and return the FCS for the "string" passed to it.

```
unsigned char compute_FCS(unsigned char *string){
   unsigned char result;
   result = *string++;      /*first byte of string*/
   while (*string)
      result ^= *string++; /* XOR operation */
   return (result);
}
```

A **Visual Basic** routine for FCS computation is included in the source code of a sample communication program you can download from:

http://www.tri-plc.com/applications/SerialComm.zip.

3.2.6 Communication Procedure

Unlike the point-to-point communication protocol, the host computer must NOT send the CTRL-E character before sending the command block. After the host computer has sent out the multi-point host-link command block, only the controller with the correct device ID will respond. Hence it is essential to ensure that every controller on the RS485 network assumes a different ID. Otherwise, contention may occur (i.e., two controllers simultaneously sending data on the receiver bus, resulting in garbage data being received by the host). On the other hand, if none of the controller IDs match that specified in the command block, then the host computer will receive no response at all.

The PLC automatically recognizes the type of command protocols (point-to-point or multi-point) sent by the host computer and it will respond accordingly. If a multi-point command is accepted by the controller, the response block will start with a character '@', followed by its device ID and the same header as the command. This will be followed by the data requested by the command, a response block FCS and the terminator.

Framing Errors

When the controller receives a multi-point host-link command block, it computes the FCS of the command and compares it with the FCS field received in the command block. If the two do not match, then a "framing error" has occurred. The controller will send the following Framing Error Response to the host:

Framing Error Response Block (Multi-point only)

@	x	x	F	E	x	x	*	↵
	Device ID		Header		FCS		Terminator	

Command Errors

If an unknown command is received or if the command is illegal (such as an attempt to access an unavailable channel), the following **error response** will be received:

Error Response Format

@	x	x	E	R	x	x	*	↵
	Device ID		Header		FCS		Terminator	

The host computer program should always check the returned response for possibilities of errors in the command and take necessary action.

3.3 SHOULD YOU USE POINT-TO-POINT OR MULTI-POINT PROTOCOL?

Although at first the point-to-point protocol appears simpler in format (having no ID and no FCS computation), the communication procedure is actually more complex since it involves the need to synchronize the two communicating devices by exchanging the Control-E character. The lack of error checking also makes the protocol less reliable especially in noisy environment.

In fact, the TLServer software as well as the Ethernet XServer will only accept multi-point communication protocol from the client software with the exception of the “IR*” command, which is needed to obtain the ID of a PLC with unknown ID.

Hence, if you were to write your own communication program to talk to the PLCs, we would strongly recommend using only the multi-point protocol exclusively due to its simplicity and built-in error checking capability.

3.4 TROUBLE-SHOOTING AN RS485 NETWORK

a) Single faulty device

If a single device on the RS485 network becomes inaccessible, problems can be isolated to this particular device. Check for loose or broken wiring or wrong DIP switch settings. Also double check the device ID using the host-link command "`IR*`" sent via the RS232C port of the PLC. If all attempts fail, either replace the entire PLC or the SN75176 chip that handles the RS485 interfacing and try again.

b) Multiple faulty devices

If all the PLCs are inaccessible by the host computer, it may possibly be due to a faulty RS232C-to-RS485 converter at the PC. If this is the case, disconnect the RS485 converter from the network and check it using a single PLC. Replace the converter if it is confirmed to be faulty. Next check the wire from the converter to the beginning of the network. A broken wire here can lead to the failure of the entire network.

Since an RS485 network links many PLCs together electrically and in a daisy chain fashion, problems occurring along the RS485 network sometimes affect the operation of the entire network. For example, a broken wire at the terminal of one node may mean that all the PLCs connected after this node become inaccessible by the master. If the RS485 interface of one of the PLCs has short-circuited because of component failure, then the entire network goes down with it too. This is because no other node is able to assert proper signals on the two wires that are also common to the shorted device.

Hence when trouble-shooting a faulty RS485 network, it may be necessary to isolate all the PLCs from the network. Thereafter, reconnect one PLC at a time to the network, starting from the node nearest to the host computer. Use the TRiLOGI program to check communication with each PLC until the faulty unit has been identified.

Chapter 4 Command/Response Format

This chapter describes the detail formats of the command and response blocks for all M-series PLC host link commands. Only the formats for the point-to-point communication protocol are presented, but all these commands are available to the multi-point protocol as well. To use a command for multi-point system, simply add the device ID (@nn) before the command header and the FCS at the end of the data (See Chapter 3 for detailed descriptions of multi-point communication command format).

4.1 Device ID Read

Command Format

I	R	*	↵

Response Format

I	R	16^1	16^0	*	↵

Device ID (00 to FF)

The device ID is to be used for multi-point communication protocol where the host computer can selectively communicate with any controller connected to a common RS485 bus (see Chapter 3 for details). The ID has no effect for point-to-point communication.

The device ID is stored in the PLC's EEPROM and therefore will remain with the controller until it is next changed.

4.2. Device ID Write

Command Format

I	W	16^1	16^0	*	↵

Device ID (00 to FF)

Response Format

I	W	*	↵

E.g. To set the PLC's ID to 0A, send command string "`IW0A`*" to PLC.

4.3 Read Digital Input Channels

Command Format

R	I	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	I	16^1	16^0	*	↵

8-bit Data (Hex)

Definition of Input Channels

The following table shows the input numbers as defined in TRiLOGI's Input entry table corresponding to the input channel number.

	Bit7			Input/Output Numbers				Bit0
CH00:	8	7	6	5	4	3	2	1
CH01:	16	15	14	13	12	11	10	9
CH02:	24	23	22	21	20	19	18	17
CH03:	32	31	30	29	28	27	26	25
CH04:	40	39	38	37	36	35	34	33
CH05:	48	57	56	45	44	43	42	41
CH06:	56	55	54	53	52	51	50	49
CH07:	64	63	62	61	60	59	58	57
CH08:	72	71	70	69	68	67	66	65
CH09:	80	79	78	77	76	75	74	73
CH0A$_{16}$:	88	87	86	85	84	83	82	81
CH0B$_{16}$:	96	95	94	93	92	91	90	89
CH0C$_{16}$:	104	103	102	101	100	99	98	97
CH0D$_{16}$:	112	111	110	109	108	107	106	105
CH0E$_{16}$:	120	119	118	117	116	115	114	113
CH0F$_{16}$:	128	127	126	125	124	123	122	121

The 8-bit inputs of each channel is represented by two bytes ASCII text expression of its hexadecimal value. For example: if inputs 1 to 3 are logic '0's, inputs 4 to 10 are logic '1's and all other inputs are logic '0's, then if you send command "`RI00*`", you will get response "RIF8*" (F8$_{16}$ =1111 1000$_2$).

4.4 Read Digital Output Channels

Command Format

R	O	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	O	16^1	16^0	*	↵

8-bit data (Hex)

Please refer to the Input/Output vs Channel Number table described in the section "4.3. Read Digital Input Channels" for details.

4.5 Read Internal Relay Channels

Command Format

R	R	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	R	16^1	16^0	*	↵

8-bit data (Hex)

Definition of Internal Relay Channel Numbers

All M-series PLC supports 256 internal relays, the channel definition of the first 128 internal relays is the same as the inputs and the outputs. The remaining relays and their assigned channels are shown in the following table:

	bit7			Relay numbers				bit0
CH10$_{16}$:	136	135	134	133	132	131	130	129
CH11$_{16}$:	144	143	142	141	140	139	138	137
CH12$_{16}$:	152	151	150	149	148	147	146	145
CH13$_{16}$:	160	159	158	157	156	155	154	153
CH14$_{16}$:	168	167	166	165	164	163	162	161
CH15$_{16}$:	176	175	174	173	172	171	170	169
CH16$_{16}$:	184	183	182	181	180	179	178	177
CH17$_{16}$:	192	191	190	189	188	187	186	185
CH18$_{16}$:	200	199	198	197	196	195	194	193
CH19$_{16}$:	208	207	206	205	204	203	202	201
CH1A$_{16}$:	216	215	214	213	212	211	210	209
CH1B$_{16}$:	224	223	222	221	220	219	218	217
CH1C$_{16}$:	232	231	230	229	228	227	226	225
CH1D$_{16}$:	240	239	238	237	236	235	234	233
CH1E$_{16}$:	248	247	246	245	244	243	242	241
CH1F$_{16}$:	256	255	254	253	252	251	250	249

4.6 Read Timer Contacts

Command Format

R	T	n	n	*	↵

8-bit Channel # (Hex)

Response Format

R	T	16^1	16^0	*	↵

8-bit data in Hex

Definition of Timer-Contact Channel Numbers

A timer contact is a single bit of memory and 8 timer contacts are grouped into one 8-bit channel similar to that of the inputs, outputs etc.

The following table shows the timer numbers defined in TRiLOGI's Timer entry table and their corresponding channel numbers.

CH0:	8	7	6	5	4	3	2	1
CH1:	16	15	14	13	12	11	10	9
CH2:	24	23	22	21	20	19	18	17
CH3:	32	31	30	29	28	27	26	25
CH4:	40	39	38	37	36	35	34	33
CH5:	48	57	56	45	44	43	42	41
CH6:	56	55	54	53	52	51	50	49
CH7:	64	63	62	61	60	59	58	57

4.7 Read Counter Contacts

Command Format

R	C	n	n	*	↵

8-bit channel # (Hex)

Response Format

R	C	16^1	16^0	*	↵

8-bit data in Hex

Definition of Counter-Contact Channel Numbers:

The 64 counter contacts are assigned channel # in exactly the same way as the 64 timers. Please refer to last section :"4.6. Read Timer Contacts" for details.

4.8 Read Timer Present Value (P.V.)

Command Format

R	M	N	n	*	↵

nn: Timer1=00, ... Timer16=0F.... Timer64=3F

Response Format

R	M	10^3	10^2	10^1	10^0	*	↵

Timer present value in Decimal

The present value (PV) of the specified timer is returned in decimal form as four byte ASCII text characters from 0000 to 9999.

4.9 Read Timer Set Value (S.V.)

Command Format

R	m	n	n	*	↵

nn: Timer1=00, ... Timer16=0F.... Timer64=3F

Response Format

R	m	10^3	10^2	10^1	10^0	*	↵

Timer Set Value in Decimal

The Set Value (S.V.) of the specified timer is returned in decimal form as four byte ASCII text characters from 0000 to 9999. Note that this command header contains **small letter "m"** instead of "M" in the "RM" command.

4.10 Read Counter Present Value (P.V.)

Command Format

R	U	n	n	*	↵

nn: Counter1=00, ... Counter16=0F.... Counter64=3F

Response Format

R	U	10^3	10^2	10^1	10^0	*	↵

Counter present value in Decimal

The Present Value of the specified counter is returned in decimal form as four byte ASCII text characters from 0000 to 9999.

4.11 Read Counter Set Value (S.V.)

Command Format

R	u	n	n	*	↵

nn: Counter1=00, ... Counter16=0F.... Counter64=3F

Response Format

R	u	10^3	10^2	10^1	10^0	*	↵

Counter Set Value in Decimal

The Set Value of the specified counter is returned in decimal form as four byte ASCII text characters from 0000 to 9999. Note that this header contains **small letter "u"** instead of "U" in the "RU" command.

4.12 Read Variable - Integers (A to Z)

Command Format

R	V	I	*alphabet*	*	↵

A,B.C....Z

Response Format

R	V	I	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

8 Hexadecimal Digit for 32-bit integer

E.g. To read the value of the variable “K”, send host-link command “`RVIK*`”. If variable K contains the value 123456_{10} (=$1E240_{16}$), PLC will send the response string as “`RVI0001E240*`”.

4.13 Read Variable - Strings (A$ to Z$)

Command Format

R	V	$	*alphabet*	*	↵

A,B.C....Z

Response Format

R	V	$	*a*	*a*	*a*	...	...	*a*	*a*	*a*	*	↵

ASCII characters of the string (variable length)

E.g. To read the value of the string variable “M$”, send host-link command “`RV$M`*”. If variable M$ contains the string “Hello World”, the PLC will send the response string as “`RV$Hello World`*”.

4.14 Read Variable - Data Memory (DM[1] to DM[4000])

Command Format

R	V	D	16^3	16^2	16^1	16^0	*	↵

0001 to 0FA0 (4000_{10})

Response Format

R	V	D	16^3	16^2	16^1	16^0	*	↵

4 Hexadecimal Digit for 16-bit integer

E.g. To read the value of DM[3600], send host-link command “`RVD0E10*`”. If variable DM[3600] contains the value 12345_{10} (=3039_{16}), PLC will send the response string as “`RVD3039*`”.

4.15 Read Variable - System Variables

This command allows you to read all the M-series PLC's 16-bit system variables such as the inputs[ ], outputs[ ], relays[ ], counters[ ], timers[ ], timers' P.V., counters' P.V., CLK[ ] and DATE[ ]. Although inputs, outputs etc. are also accessible via the "RI", "RO", "RR"... commands, the RVS command can access them as 16-bit words instead of as 8-bit bytes in those commands. For the 32-bit system variable HSCPV[ ], use the "RVH" command described in the next section to access it. It may be more conventional for some SCADA software driver to use a single header command "RVS" to access all the I/O, varying only the "type" number to access different I/O types.

The RVS command also can be used to access the internal variables used to store ADC, DAC and PWM values obtained during the latest execution of the ADC(), setDAC or setPWM statement. These are however not system variables in TBASIC sense. E.g. it is illegal to use ADC[2] to access the ADC channel #2 in TBASIC (you have to use the ADC(2) function instead). An 8-bit hexadecimal number is used to denote the "type" of system variable, as shown in the following table:

System Variable	**type**
input[ ]	01
output[ ]	02
relay[ ]	03
timer[ ]	04
ctr[ ]	05
timerPV[ ]	06
ctrPV[ ]	07

System Variable	**type**
clk[ ]	08
date[ ]	09
-	0A
ADC*	0B
DAC*	0C
PWM*	0D

* Not a system variable in TBASIC

Command Format

R	V	S	n	n	16^1	16^0	*	↵

type Index

type (01 to 0D) - denote the type of system variable to access,
index (01 to 1F) - index into the array, starting from 01.

Response Format

R	V	S	16^3	16^2	16^1	16^0	*	↵

4 Hexadecimal Digit for 16-bit integer

Example: To read the value of DATE[2] (which represents the month of the RTC), send command "`RVS0902*`" and if the PLC responds with "`RVS0005`" it means the month is May.

4.16 Read Variable - High Speed Counter HSCPV[ ]

Command Format

R	V	H	n	*	↵

Channel: 1 or 2

Response Format

R	V	H	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

8 Hexadecimal Digit for 32-bit integer

E.g. To read the value of HSCPV[2], send hostlink command "`RVH2*`". If variable HSCPV[2] contains the value 123456_{10} (=$1E240_{16}$), PLC will send the response string as "`RVH0001E240*`".

4.17 Write Inputs

Command Format

W	I	n	n	16^1	16^0	*	↵

Channel # (00 to 0F)   Data

Response Format

W	I	*	↵

4.18 Write Outputs

Command Format

W	O	n	n	16^1	16^0	*	↵

Channel # (00 to 0F)   Data

Response Format

W	O	*	↵

4.19 Write Relays

Command Format

W	R	n	n	16^1	16^0	*	↻

Channel # Data

Response Format

W	R	*	↻

4.20 Write Timer-contacts

Command Format

W	T	n	n	16^1	16^0	*	↻

Channel # Data
(00 to 07)

Response Format

W	T	*	↻

4.21 Write Counter-contacts

Command Format

W	C	n	n	16^1	16^0	*	↻

Channel # Data
(00 to 07)

Response Format

W	C	*	↻

4.22 Write Timer Present Value (P.V.)

Command Format

W	M	n	n	10^3	10^2	10^1	10^0	*	↻

Timer1=00, New timer PV
....
Timer64=3F (Hex)

Response Format

W	M	*	↵

Please note that the timer number starts from 00 which represent timer #1, 01 represents timer #2... and so on.

4.23 Write Timer Set Value (S.V.)

Command Format

W	m	n	n	10^3	10^2	10^1	10^0	*	↵

Timer1=00, New timer SV
....
Timer64=3F (Hex)

Response Format

W	m	*	↵

Note: the 2nd character is a lower case "m" instead of the upper case "M" of "WM" command.

4.24 Write Counter Present Value (P.V.)

Command Format

W	U	n	n	10^3	10^2	10^1	10^0	*	↵

Counter1=00, New PV
....
Counter64=3F (Hex)

Response Format

W	U	*	↵

4.25 Write Counter Set Value (S.V.)

Command Format

W	u	n	n	10^3	10^2	10^1	10^0	*	↵

Counter1=00, New Counter SV
....
Counter64=3F (Hex)

Response Format

W	u	*	↵

Note: the 2nd character is a lower case "u" instead of the upper case "U" of the "WU" command.

4.26 Write Variable - Integers (A to Z)

Command Format

W	V	I	*alphabet*	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵

A,B.C....Z 8 Hexadecimal Digit for 32-bit integer

Response Format

W	V	I	*	↵

E.g. To assign variable "K" to number 56789_{10}(=$0DD5_{16}$), send hostlink command "`WVIK00000DD5*`".

4.27 Write Variable - Strings (A$ to Z$)

Command Format

W	V	$	*alphabet*	*a*	*a*	...	...	*a*	*a*	*	↵

A,B.C....Z ASCII characters of the string (variable length)

Response Format

W	V	$	*	↵

E.g. To assign the string "T100MD+ Super PLC" to the string variable P$, send hostlink command "`WV$PT100MD+ Super PLC*`".

4.28 Write Variable - Data Memory (DM[1] to DM[4000])

Command Format

W	V	D	16^3	16^2	16^1	16^0	16^3	16^2	16^1	16^0	*	↵

16-bit Index to array 0001 to 0FA0 (4000_{10}) 16-bit Integer Data

Response Format

W	V	D	*	↵

E.g. To write the value 1234_{10} (=$4D2_{16}$)to DM[1000], send hostlink command "`WVD03E804D2*`". (1000_{10} = $3E8_{16}$)

4.29 Write Variable - System Variables

System Variable	type
input[ ]	01
output[ ]	02
relay[ ]	03
timer[ ]	04
ctr[ ]	05
timerPV[ ]	06
ctrPV[ ]	07

System Variable	type
clk[ ]	08
date[ ]	09
-	0A
ADC*	0B
DAC*	0C
PWM*	0D

* Not a system variable in TBASIC

Command Format

W	V	S	n	n	16^1	16^0	16^3	16^2	16^1	16^0	*	↵
			type		Index		16-bit Integer Data					

type (01 to 0D) - denote the type of system variable to access,
index (01 to 1F) - index into the array, starting from 01.

Response Format

W	V	S	*	↵

Example: To set clk[1] (which represents the hour of the RTC) to 14, send the command "`WVS0801000E*`" to the PLC.

4.30 Write Variable - High Speed Counter HSCPV[ ]

Command Format

W	V	H	n	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0	*	↵
			1 or 2	8 Hexadecimal Digit for 32-bit integer									

Response Format

W	V	H	*	↵

E.g. To clear the value of HSCPV[2], send hostlink command "`WVH200000000*`".

4.31 Update Real Time Clock Module

Command Format

W	r	*	↵

Response Format

W	r	*	↵

If the battery-backed MX-RTC module is installed, this command forces he PLC to write the values of the TIME[ ] and DATE[ ] variables into the RTC module. This command will be ignored by a PLC without the RTC module.

4.32 Halting the PLC

Command Format

C	2	*	↵

Response Format

C	2	*	↵

When the PLC receives this command, it temporarily halts the execution of the PLC's ladder program after the current scan. However, the PLC continues to scan the I/Os and processes host link commands sent to it and will report the current I/O data and internal variables to the host computer.

4.33 Resume PLC Operation

Command Format

C	1	*	↵

Response Format

C	1	*	↵

When the PLC receives this command, it will resume execution of the ladder program if it has been halted previously by the "C2" command. Otherwise, this command has no effect.

Important Note

The following Host Link Commands: RA, RXI, RX$, WA, WXI, WX$ and Wb are available only on newest M-series PLCs installed with CPU firmware version r47 & above. You can check your CPU firmware version by using the “Controller-> Get PLC Hardware Info” on the TRiLOGI software.

4.34 Read Analog Input *(r47 Firmware Only)*

This command forces the PLC to refresh the value of its ADC data at the analog channel before returning its value in the response string (i.e. no need for PLC to execute ADC(n) function to refresh the analog input)

Command Format

R	A	n	n	c	c	*	↵

Starting Analog Channel # (01-08h)

Channel count (01 to 08h)

Response Format

R	A	16^3	16^2	16^1	16^0	…	16^2	16^1	16^0	*	↵

Starting channel 16-bit Data (Hex) … Ending channel 16-bit Data (Hex)

E.g. To read 4 channels of Analog starting from Ch #2, Send “`RA0204*`”. The response string will contain 4 sets of data for channel 2, 3, 4 and 5.

4.35 Read EEPROM Integer Data *(r47 Firmware Only)*

Command Format

R	X	I	n	n	n	n	c	c	*	↵

EEPROM starting Address (Hex)

Word Count (01 to 20h)

Response Format

R	X	I	16^3	16^2	16^1	16^0	…	16^2	16^1	16^0	*	↵

1st EEPROM Integer 16-bit Data (Hex) … Last EEPROM 16-bit Data (Hex)

Maximum allowable word count per command is 32 (01 to 20 Hex). If "count" is > 32, only the first 32 words will be returned.

E.g. To read the 10 words of EEPROM data starting from address 100, send host-link command "`RXI00640A*`". The response string will contain 10 sets of 16-bit data (4 ASCII hex digit per set).

4.36 Read EEPROM String Data *(r47 Firmware Only)*

Command Format

R	X	$	n	n	n	n	*	↵

EEPROM String starting Address (Hex)

Response Format

R	X	$	*a*	*a*	*a*	...	...	*a*	*a*	*a*	*	↵

E.g. To read the string data stored at EEPROM address 10, send host-link command "`RX$000A*`". The response string will contain string data stored in the EEPROM (maximum 40 characters).

4.37 Write Analog Output *(r47 Firmware Only)*

Upon receiving this command, the PLC updates the value of its DAC data at the analog output channel (i.e. no need for PLC to execute SETDAC to update the analog output) .

Command Format

W	A	n	n	c	c	16^3	16^2	16^1	16^0	…	16^1	16^0	*	↵

Starting Analog channel # (01-02h)

channel count

DAC output data for 1st channel

DAC output data for subsequent ch

Response Format

W	A	c	c	*	↵

channel count (Hex)

4.38 Write EEPROM Integer Data *(r47 Firmware Only)*

Command Format

W	X	I	n	n	n	n	c	c	16^3	16^2	16^1	16^0	…

Starting EEPROM Address (0001-xxxx)
count (01-10h)
Hex data for starting EEPROM address

16^3	16^2	16^1	16^0	…	16^1	16^0	*	↻

data for subsequent EEPROM addresses

Response Format

W	X	I	*	↻

Maximum allowable word count per command is 16 (01 to 10 Hex).

4.39 WRITE EEPROM String Data *(r47 Firmware Only)*

Command Format

W	X	$	n	n	n	n	*a*	*a*	...	...	*a*	*	↻

EEPROM String Address (Hex)
ASCII characters (max. 40 characters)

Response Format

W	X	$	*	↻

E.g. To write the string data "Hello TRi" at EEPROM String address 12, send host-link command "`RX$000CHello TRi*`".

4.40 Force Set/Clear Single I/O Bit *(r47 Firmware Only)*

This new "Wbnnnnxx" command allows you to change a single I/O bit on the PLC. You can force set or clear any single input, output, relay, timer or counter bit. This has advantage over other write commands such as WI, WO, etc that affects the entire group of 8 or 16-bits organized into "channels".

Command Format

W	b	n	n	n	n	x	x	*	↵

I/O Bit address
(Hex)

00 – Clear I/O bit (OFF)
FF – SET I/O bit (ON)

Response Format

W	b	*	↵

I/O Type	Bit address nnnn (Hex)
Input #1 to #256	0000 to 00FF
Output #1 to #256	0100 to 01FF
Timer #1 to #256	0200 to 02FF
Counter #1 to #256	0300 to 03FF
Relay #1 to #256	0400 to 04FF
Relay #257 to #512	0500 to 05FF

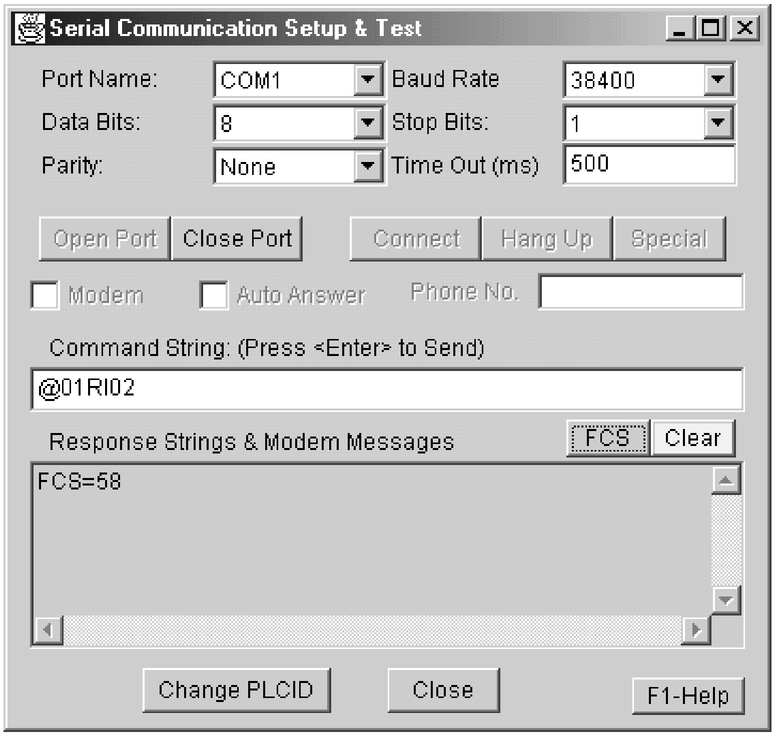
E.g. to force output 1 to ON, send "`Wb0100FF*`". To turn it OFF, send "`Wb010000*`"

4.41 Testing of Host Link Commands

You can try out all the hostlink commands using the TLServer's "Serial Communication Setup". However, the TLServer is designed to accept only multi-point protocol except the "IR*" command (which is necessary to obtain the device ID from the PLC). So you have to enter all your host link commands in multi-point format.

Since the multi-point protocol requires an FCS (frame check sequence) character to be appended to the end of the command string, you may be able to get around it by using the "wildcard" FCS "00" in place of the actual FCS. E.g. To read input channel 02 from PLC with ID = 01, you can enter the command string as "`@01RI0200*`".

For TLServer version 2.1 and above, there is an "FCS" button that let you compute the actual FCS for the string in the command string text field. You can then use the actual FCS with the command string to completely test your command. E.g. If you type in the string "`@01RI02`" in the command string (but do not press Enter), then click on the "FCS" button, the FCS for this string will be computed and shown as "FCS = 58", as shown in the following figure:



Port Name: COM1
Baud Rate 38400
Data Bits: 8
Stop Bits: 1
Parity: None
Time Out (ms) 500

Command String: (Press <Enter> to Send)
@01RI02

Response Strings & Modem Messages
```
FCS=58
```

You now can enter the complete command string as "`@01RI0258*`" and it will be accepted by TLServer. (Note: If the PLC has executed a `SETPROTOCOL` *n*,`5` to configure its serial port into pure native mode, then wildcard FCS will not be accepted and you must use the actual FCS with your command. The FCS button makes it much easier than computation by hand).

If you have changed some data using the write command, then activate On-Line Monitoring and examine the changes made using the "View Variables" window.

4.42 Visual Basic Sample Program

To help users get started writing their own Visual Basic program to communicate with the PLC, we have created a sample Visual Basic program with full source code listing. Please visit the following web page to download the visual basic sample program.

http://www.tri-plc.com/applications/VBsample.htm

4.43 Inter-PLC Networking Using NETCMD$ Command

All M-series PLCs are able to send out host link commands to other M-series or H-series PLCs using the built-in TBASIC function **`NETCMD$()`**. This function accepts host link commands in multi-point format and automatically computes the Frame Check Sequence (FCS) characters, append them to the command string and send out the whole command string together with the terminators. The function then waits for a response string and checks the integrity of the received response string for error. This function returns a string only if a proper response string has been received. Please refer to the TBASIC Reference for detailed explanation of this command.

The **`NETCMD$()`** function therefore greatly simplifies the programming tasks for handling networking between PLCs. The programmer only needs to construct the correct command string according to the formats described in this chapter, pass the formatted string to the **`NETCMD$()`** function and then check for the response string. An M-series PLC may use the NETCMD$ to map the I/O of another PLC into its internal relays and use the other PLC as its remote I/O.

There are two programming examples in your "`TRILOGI\TL4`" directory which illustrate the use of **`NETCMD$()`** to map I/Os of slave PLC to the master. Please study the two examples: "`REMOTE-H.PC4`" and "`REMOTE-M.PC4`" carefully to understand the mechanism of mapping I/Os between the PLC. The TRiLOGI program "`REMOTE-H.PC4`" will work on both H- and M-series PLCs as slaves , whereas the program "`REMOTE-M.PC4`" will only work with M-series slave PLC. This is because the M-series host link command set is a superset of H-series host link command set, and this example uses the more efficient M-series host link commands to read/write 16-bit data for networking between M-series PLC.

4.44 Inter PLC Networking Using MODBUS Protocols

The T100M+ PLCs may also pass data to each other using special MODBUS commands which are even simpler to use than NETCMD$ but are restricted to accessing variables that are mapped into MODBUS address structure. Please refer to the next chapter as well as the TBASIC Reference manual for details on using the READMODUS and WRITEMODBUS as well as the READMB2 and WRITEMB2 commands.

4.45 Using OMRON Host Link Commands

Since the T100M+ PLCs also support OMRON C20H Host Link commands, which are very similar in construct to our multi-point command/response format, you can also make use of OMRON commands to supplement the native host link commands.

We will only discuss four of the OMRON host link commands "RR", "WR", "RD" and "WD" in this section because these commands can be used by users to read/write to **multiple** I/O registers and data memory in a single command (Note: maximum length of command string should be <=80 characters).

Note: Since the M-series native protocol command set typically only supports read/write of single variable and data memory, if you want to read/write multiple memory location in a single command you can make use of these OMRON host link commands.

I. Read IR Registers

This command refers to Table 5.1 in Chapter 5 to map the PLC's I/Os to OMRON IR register space from IR0 to IR519

Command Format

@	d	d	R	R	n	n	n	n	c	c	c	c
	Device ID		Header		IR Address (Dec)				IR count (Hex)			

f	f	*	↵
FCS			

Response Format

@	d	d	R	R	s	s	16^3	16^2	16^1	16^0	…	…
	Device ID		Header		Status 00 – OK 15 - Bad		1st Data (Hex)					

16^3	16^2	16^1	16^0	f	f	*	↵
Last data				FCS			

E.g. To read Timer PV #1 to #7 using this command, send:

"`@01RR012800074D*`"

The PLC will send return a response "`@01RR00xxxxyyyyzzzz….*`"

II. WRITE IR Registers

This command refers to Table 5.1 in Chapter 5 to map the PLC's I/Os to OMRON IR register space from IR000 to IR519

Command Format

@	d	d	W	R	n	n	n	n	16^3	16^2	16^1	16^0	….
	Device ID		Header		IR Start Addr(Dec)				1st data				

16^3	16^2	16^1	16^0	f	f	*	↵
Last data				FCS			

Response Format

@	d	d	W	R	s	s	f	f	*	↵
	Device ID		Header		Status 00 – OK		FCS			

E.g. To Write to CtrPV #1 to #2 using this command, send:

`“@01WR0256xxxxyyyyff*”`

where xxxx and yyyy are the hex values to be written to CtrPV 1 & 2.

III. Read Data Memory DM[1] to DM[4000]

Command Format

@	d	d	R	D	n	n	n	n	c	c	c	c
	Device ID		Header		DM Address (Dec)				DM count (Hex)			

f	f	*	↵
FCS			

Response Format

@	d	d	R	D	s	s	16^3	16^2	16^1	16^0	…	…
	Device ID		Header		Status 00 – OK		1st Data (Hex)					

16^3	16^2	16^1	16^0	f	f	*	↵
Last data				FCS			

E.g. To read DM#112 to #130 (19 words), send:

“`@01RD0112001357*`”

The PLC will send return a response “`@01RD00xxxxyyyyzzzz…*`”

IV. WRITE Data Memory DM[1] to DM[4000]

Command Format

@	d	d	W	D	n	n	n	n	16^3	16^2	16^1	16^0	….
	Device ID		Header		DM Start Addr(Dec)				1st data				

16^3	16^2	16^1	16^0	f	f	*	↺
Last data				FCS			

Response Format

@	d	d	W	D	s	s	f	f	*	↺
	Device ID		Header		Status 00 – OK		FCS			

E.g. To Write to DM#1200 to #1201 using this command, send:

“`@01WD1200xxxxyyyyff*`”

where xxxx and yyyy are the values to be written to DM[1200] & DM[1201].
