CONTROL TECHNOLOGY CORPORATION

5200 Communications Guide

5200 Communications Guide

Blank

WARNING: Use of CTC Controllers and software is to be done only by experienced and qualified personnel who are responsible for the application and use of control equipment like the CTC controllers. These individuals must satisfy themselves that all necessary steps have been taken to assure that each application and use meets all performance and safety requirements, including any applicable laws, regulations, codes and/or standards. The information in this document is given as a general guide and all examples are for illustrative purposes only and are not intended for use in the actual application of CTC product. CTC products are not designed, sold, or marketed for use in any particular application or installation; this responsibility resides solely with the user. CTC does not assume any responsibility or liability, intellectual or otherwise for the use of CTC products.

The information in this document is current as of the following Hardware and Firmware revision levels. Some features may not be supported in earlier revisions. See www.ctc-control.com for the availability of firmware updates or contact CTC Technical Support.

Model Number	Hardware Revision	Firmware Revision
5200	All Revisions	>= 5.00.36

TABLE OF CONTENTS

CHAPTER

1

Communications Summary

With the release of the 5200 firmware revision 5.00 and above, numerous new features are available. Many of these features are in the area of communications, while a number of significant ones allow for greater programming flexibility. This manual's focus is on those features relevant to the area of communications, some of which are listed below:

- (2) Serial ports capable of the CTNet Binary protocol, CTC ASCII Protocol, User Defined, Modbus RTU/ASCII Master and Slave protocols.
- COM1 and COM2 are independently configurable; including baud rates to 115Kb, stop bits, data bits, parity, and communication protocols.
- Serial communications settings saved and restored at power up.
- Telnet Server for remote administration interface
- FTP Client and Server, reference *Model 5200 Logging and FTP Client Applications Guide, 951-520015 and Remote Administration Guide, 951-520001.*
- HTTP 1.0 Web server for WebMON *(WebMON 2.0 User's Guide, 951-520012)* diagnostics.
- Modbus/TCP RTU Master and Slave
- UDP Peer to Peer
- TCP client/server raw socket interface, bidirectional
- CTNet Binary protocol
- SMTP support for sending emails.
- POP3 inbox support for receiving emails and processing embedded script messages.
- Up to 7 serial ports, including 2 local and 5 virtual TCP to terminal servers or host applications
- Configurable connection throttling to enhance overall system performance
- String formatted output messages with embedded register values from within Quickstep (`printf` format).
- SNTP Time Server synchronization for real time clock.
- DHCP support
- DNS name registration via DHCP
- 'C' Programming for custom protocols along with support for UDP Datagrams.

- Configuration of most parameters via the Java WebMON Administration Interface applet.

CHAPTER

2

Serial Communications

The controller contains two RS-232 serial ports. Optionally, COM2 can be ordered with RS-485. RS-485 operation is transparent to software, with automatic line turnaround and timing controlled by hardware. These ports support numerous communications protocols, many of which are detailed elsewhere within this document. This section is meant as a general overview.

Port Settings via Registers

Serial port parameters may be modified directly via registers, such as when programming via Quickstep. The factory default communication settings for the two serial ports are:

Baud Rate - 19200
Data Bits - 8
Parity - None
Stop Bits - 1

All parameters may be changed using available registers. Use register 12000 to select either port by storing a 1 or 2. Set the following registers based on the configuration desired:

Set register 12301 to select the baud rate as follows:

2 - 1,200
3 - 2,400
4 - 4,800
5 - 9,600
6 - 19,200 (default)
7 - 38,400
8 – 57,600
9 – 115,400

Set register 12308 to select the parity as follows:

0 - None (default)
1 - Odd
2 - Even

Set register 12309 to select the stop bits as follows:
1 - Stop bit on transmit (default)
2 - Stop bits on transmit

Set register 12310 to select the data bits as follows (not including parity):
7 - Data bits
8 - Data bits

For example, the following Quickstep instructions will change the baud rate on port 1 to 9600 Baud:

```
store 1 to Reg_12000
store 5 to Reg_12301
```

Serial port settings are non-volatile and may be saved to serial E^2 memory. Saving these and other parameters is done by writing a 1 to register 20096.

In summary the following are relevant serial port control registers:

12000	Select Controller Communications Port: W access, 1 = COM1, 2 = COM2, 3 – 7 = TCP raw virtual socket connections.
12000	Message Transmission Status for Controllers: R access, 0 = not busy, 1 = busy.
12001	Transmit Message from Data Table: W only, Store row number to transmit.
12001-12255	Controller Receive Buffer Access, R only, 1 character per location.
12300	Protocol Variation: R/W, Controls RS-232 terminal protocol modes. 0 = computer, 1 = terminal (default)
12301	Serial Baud Rate Selection: R/W, 2 = 1200, 3=2400, 4 = 4800, 5 = 9600, 6 = 19.2K (default), 7 = 38.4K, 8 = 57.6K, 9 = 115.2K.
12302	Serial Input Buffer Counter: (R) number of characters available. (W) any value to clear buffer and zero count.
12303	Disable Automatic Parsing: R/W, 0 = inhibits response, 1 = resumes normal response to incoming messages. Disable parsing to process own protocols.
12304	Extract Number from RS-232 Receive Buffer: R only, Automatically assembles ASCII strings into a numeric value. The result is a signed 32-bit number. Automatically assembles strings of ASCII characters containing numeric information into a numeric value. Number multiplied by 10,000, allowing decimal points to 4 places.
12305	Communications Priority: R/W, when running multiple tasks. 0 = normal, 1 = priority.
12308	Serial Parity: R/W, 0=None (default), 1=Odd, 2= Even
12309	Serial Stop Bits: R/W, 1 (default) or 2
12310	Serial Data Bits: R/W, 7 or 8 (default)
12316	Message String Transfer Register: R/W, write records number of `message.ini` file to send out serial port selected in 12000 register, read returns status with 0 = success. See the Model 5200 Script Configuration Guide.

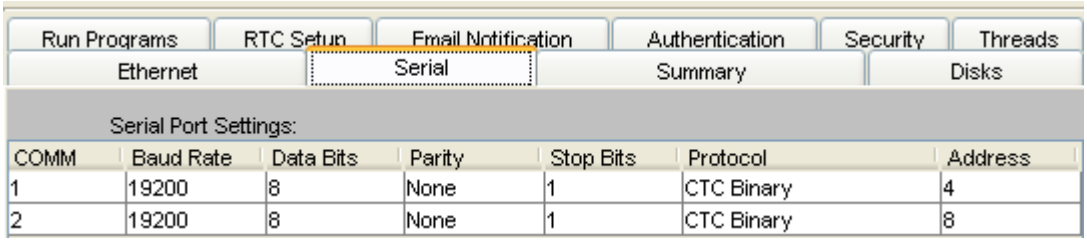
12320	Serial Active Protocol Selection: R/W; by default the protocol is set to CTC (0). Write to this port last after setting up any relevant parameters since this register enables the selected protocol immediately. CTC Binary & ASCII – 0 Modbus Master RTU – 1 (max of 120 16 bit Modbus Registers/block read; do not set manually, as it will be set when configuring the Modbus Master Register Control Block. Up to 256 may be read using automatic de-blocking feature of the Control Block) Modbus Master ASCII – 2 (max of 56 16 bit Modbus Registers/block read; do not set manually, as it will be set when configuring the Modbus Master Register Control Block. Up to 256 may be read using automatic de-blocking feature of the Control Block) Modbus Slave RTU – 3 (max of 120 16 bit Modbus Registers or 60 32 bit 5200 registers) Modbus Slave ASCII – 4 (max of 56 16 bit Modbus Registers or 28 32 bit 5200 registers)
12321	Serial Active Address: R/W, address to be used by the controller, based upon the enabled protocol. By default the Global Serial Address is used unless overridden by writing a different one for the enabled port (12000 register) to this register. Currently on Modbus Slave protocols use this address. Modbus Master uses the Modbus Master Register Control Block, 21000 – 21299.
12322	Global Serial Address: R/W, Address to be used as the 5200 power up default for Modbus Slave Serial Protocols unless overridden by a write to register 12321. To save this value permanently a 1 must be written to register 20096.

Only baud rate, stop bits, data bits, parity, protocol, and port specific address are saved to non-volatile memory.

Port Settings via WebMON

Alternatively to directly modifying registers, serial port parameters may be modified using the WebMON utility. Reference the document for details, as a review, the "Serial" tab allows immediate configuration of the local COMM1 and COMM2 serial ports, within the controller. All changes take effect immediately and are placed in permanent storage, thereby surviving power cycling. Once parameters are updated an immediate read is done of all parameters, providing visual verification of your changes.

The COMM configuration provides a table of two rows, one for each serial port. It consists of a number of data entry fields, each with their own special functionality:



- COMM
- Baud Rate
- Data Bits
- Parity
- Stop Bits

- Protocol
- Address

COMM

This is not an editable field. It is used to reference either COMM1 (row 1) or COMM2 (row 2).

Baud Rate

A pull down list box is available to select the desired baud rate. Baud rates from 1200 to 115,200 are available. Note that using baud rates above 19,200 can cause system degradation, depending upon the protocol and data flow of the system.

Data Bits

A pull down list box is available to select either "7" or "8" data bits.

Parity

A pull down list box is available to select "None", "Odd", or "Even" parity.

Stop Bits

A pull down list box is available to select either "1" or "2" stop bits.

Protocol

A pull down list box is available to select the individual protocols to be active on each port. Details for each are provided in the 5200 Communications Guide (950-520000). Available selections are:

- CTC Binary (Default, compatible with CTCMON and ctccom32.dll)
- Modbus Master RTU – controller polls the device.
- Modbus Master ASCII – controller polls the device.
- Modbus Slave RTU – controller polled by external device
- Modbus Slave ASCII – controller polled by external device

Address

This is the address to be use when Modbus protocols are selected. When in Master mode only a single device may be polled. To poll multiple devices the Address register must be changed by the Quickstep program, dynamically. An address from 1 to 255 is valid.

CHAPTER

3

Networking Communications

The 5200 series controllers can be configured to communicate over Ethernet using one of several transport protocols: CTNet, UDP, and TCP. This section discusses the how to setup and configure the controller for network communications.

CTNet

CTNet is a proprietary, non-routable protocol typically used for legacy communications to the 2700 controller products. It tends to be faster than UDP or TCP/IP due to the lack of processing overhead, but like UDP, it lacks acknowledgement of each packet.

Note that the Binary Message subset of the CTNet protocol can optionally be sent using UDP and TCP via IP Encapsulation. Reference that section for further details.

UDP

User Datagram Protocol is used to send packets across an IP Network in an unreliable manner, with no packet acknowledgement. The protocol is fully routable across the network, unlike CTNet. It is the preferred interface for many products when performance is required and the application itself can perform error recovery. The 5200 supports UDP packet transport for peer to peer communications, CTCMon, and CTServer products.

TCP

Transmission Control Protocol is used to establish connection-oriented, sequenced, and error free sessions over an IP Network. The protocol is fully routable across the network, unlike CTNet, and each data packet is acknowledged when received correctly by the receiver. Retransmission of lost packets is built into the protocol. Typical retry timers of 250 milliseconds limit the uses of TCP in a real-time controller. The 5200 supports TCP packet transport for FTP, Telnet, Modbus TCP Master/Slave, RAW client/server connections, CTCMon, and CTServer products.

When using any of these protocols it is important to note that whenever the 5200 is placed on a network, it should be connected to a switch, not a hub. A switch will isolate traffic to broadcasts that are specific to the controller whereas a hub will cause the 5200 to receive all traffic on its link.

Configuring a CTNet Node using Registers

Details of the CTNet protocol can be found within the *"Guide to CTC Serial Data Communications"* manual and *"CTC Monitor User Guide"* manual. Both of which are available for download from the Control Technology website, www.ctc-control.com. To use CTNet, a valid CTNet node number between 1 and 32767 must be set. To use UDP protocol, the controller must be set up with a TCP/IP address, subnet mask, and optional gateway.

The CTNet node number of the controller is stored in register 20000. Simply write the node number to register 20000, write a 1 to register 20096, and then cycle power on the controller for the change to be accepted.

```
Store 21 to Reg_20000
Store 1 to Reg_20096
```

Configuring IP Addresses using Registers

If you are not using DHCP to automatically obtain your IP address, then the TCP/IP address is configured statically as follows:

Sample IP Address - 168.254.132.34 (random example)
Sample Subnet Mask - 255.255.255.0 (typical)
Sample Gateway - 168.254.132.88 (random example)

The actual values to use will depend on the network that the controller is connected to. Contact your IT department to determine acceptable addresses for your network.

Registers 20048 to 20051 are the 4 parts of the IP address:

```
store 168 to Reg_20048
store 254 to Reg_20049
store 132 to Reg_20050
store 34 to Reg_20051
```

Registers 20064 to 20067 are the 4 parts of the Subnet Mask:

```
store 255 to Reg_20064
store 255 to Reg_20065
store 255 to Reg_20066
store 0 to Reg_20067
```

Registers 20080 to 20083 are the 4 parts of the Gateway Address (optional):

```
store 168 to Reg_20080
```

```
store 254 to Reg_20081
store 132 to Reg_20082
store 88 to Reg_20083
```

A gateway is only required if the controller needs to communicate over a Wide-Area Network (WAN). If not using a gateway, then set these registers to 0 (default). The controller can talk to devices on a Local Area Network without using a gateway, but not over the Internet or outside its subnet. The following command saves the IP address and all other modified IP address parameters to non-volatile memory:

```
store 1 to Reg_20096
```

Finally, cycle power to the controller to activate the new IP information active.

The IP address can be set up through a Quickstep program or with CTC Monitor. Note that if you set the IP address registers to 0, then write 1 to Reg_20096 and cycle power, the controller will use DHCP to obtain its network information automatically. You will be aware that the controller is attempting to connect to a DHCP server when the S3 LED is flashing repeatedly, at a high rate (100ms/second). The S3 LED will stop flashing once the 5200 has obtained an IP address from a DHCP server. While searching for a valid DHCP address, serial port CTC Monitor access will be available to a limited number of registers, typically 20000 and above, but Quickstep and Ethernet communications will be disabled. Once an IP address is available the 5200 will continue to boot, initializing the network and starting Quickstep application software.

Configuring the IP address automatically with DHCP

The controller is capable of retrieving its IP information automatically, from a DHCP server, RFC 2131. The Dynamic Host Configuration Protocol (DHCP) is a communication protocol that lets network administrators automate assigning of IP addresses within a network.

All devices (computers, controllers, etc.) that reside on a TCP/IP network must have an IP address assigned. Without DHCP, the IP address must be entered manually at each device, such as detailed in the previous section. If devices move to another location in another part of the network, a new IP address must be entered. DHCP allows a network administrator to supervise and distribute IP addresses from a central point and automatically assigns a new IP address when a computer is plugged into a different location on the network. DHCP also provides other services beyond assigning IP addresses. It provides features including Domain Name Service (DNS) server addresses, gateway information, and Simple Network Time Protocol (SNTP, section 6.0) servers, thus allowing for fully automatic configuration of the controller IP parameters.

DHCP uses the concept of a "lease" or amount of time that a given IP address will be valid for a computer. The lease time can vary depending upon how long a user is likely to require the network connection at a particular location. DHCP also supports static addresses for devices that need a permanent IP address.

DHCP is enabled by default in the controller. At power up, the controller will request to use whatever IP address is set in the 20048 block (except 0.0.0.0, which enables DHCP), and the DHCP server will either allow it or supply a new IP address. This final address will be temporarily written to the 20048 block, but not permanently. Although not stored permanently, it is still the active IP address for the system. Only the user or Quickstep can make this IP address permanent, by storing a 1 to register 20096. If you do not want to use DHCP, it can only be disabled by setting an actual IP address and subnet mask.

Setting the Controller's DNS Name via Telnet

When the controller communicates with a DHCP server, it also requires a unique system name that is typically used for DNS resolution (assuming the server is using dynamic DNS). Presently this name is derived from the controller's serial number, placing "CTC_BF_" before the number. For example, if the serial number is 100-52801, then the DNS name entry for the controller is CTC_BF_10052801. User-definable names are also possible and may be set using the "*set systemname <name>*" command within the Telnet administration screen, followed by writing a 1 to register 20096 (to save the change), and rebooting the controller.

Note that many software packages and other devices with CTC communications drivers can identify controllers only by IP address and not by name.. Depending on how your network is configured, DHCP may change the IP address of the controller without warning, causing devices and software to lose connection or connect to the wrong controller. In this case, it is better to manually assign a static IP address to the controller. The network administrator should be contacted prior to assigning any IP address, to avoid conflicts.

Communicating to the Controller Using CTNet

CTNet is a lightweight non-routable Ethernet protocol used by legacy CTC controllers. It is recommended that UDP be used, instead, whenever possible, since it is routable.

In order to communicate with the controller from a PC using CTNet protocol, the WinPCap driver must be installed on the PC and an updated `ctccom32v2.dll` file must be installed in the Windows system32 directory.

The latest version of the WinPCap driver may be downloaded from the customer care section of CTC's website www.ctc-control.com. Compatibility information will be included with the download. Currently Windows 95, 98, ME, NT4, 2000, and XP are supported.

To install the driver:

1. First, uninstall any previously installed CTNet drivers, including CTC Transport and CTC Packet Driver. If you have not previously installed these drivers, this step can be skipped. DO NOT INSTALL WinPCap OVER AN EXISTING CTNet DRIVER.
2. Double click the WinPCap.exe file and run through the installation program.

3. In your Windows system32 directory (typically Windows\system for Windows 95, 98, and ME and WINNT\system32 for Windows NT/2000/XP) replace the existing *ctccom32v2.dll* file with the file included with the WinPCap download.
4. Restart the PC.

Once the driver is installed, CTC Monitor 2.8 or later can be used to communicate to the controller. Every controller on the network must have a unique node number, and each PC based connection must use a unique Host node number.

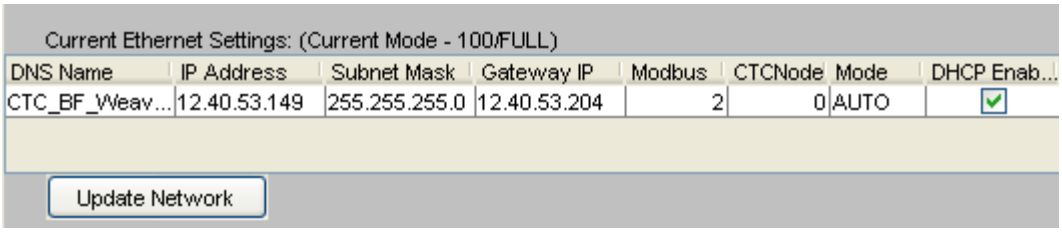
Note that WinPCap only needs to be installed when using the non-routable binary protocol version of CTNet, that used in legacy 2700 products using the 2217 Ethernet Controller. Operating CTNet over UDP and TCP can be done using IP Encapsulation and does not required WinPCap. The 2700 does require the 2717 controller for backward compatibility.

Network Configuration via WebMON

Alternatively to directly modifying registers, network parameters may be modified using the WebMON utility. Reference the document for details, as a review, the "Ethernet" tab is used to set various network parameters. Settable parameters include general network IP information, SNTP Time server interface and POP3 email. SNTP, SMTP, and POP3 network configuration can be found in their respective sections.

Ethernet Settings

The Ethernet Settings consists of a number of data entry fields, each with their own special functionality:



- DNS Name
- IP Address
- Subnet Mask
- Gateway IP
- Modbus
- CTC Node
- Mode
- DHCP Enabled

DHCP Enabled (check box to enable)

The controller is capable of retrieving its IP information automatically (IP Address, Subnet Mask, and Gateway IP), from a DHCP server, RFC 2131. The Dynamic Host Configuration Protocol (DHCP) is a communication protocol that lets network administrators automate assigning of IP addresses within a network.

All devices (computers, controllers, etc.), which reside on a TCP/IP network, must have an IP address assigned. Without DHCP, the IP address must be entered manually at each device. If devices move to another location in another part of the network, a new IP address must be entered. DHCP allows a network administrator to supervise and distribute IP addresses from a central point and automatically assigns a new IP address when a computer is plugged into a different location on the network. DHCP also provides other services beyond that of just an IP address. It provides Domain Name Service (DNS) server addresses, gateway information, Simple Network Time Protocol servers, etc., thus allowing for fully automatic configuration of the controller IP parameters.

DHCP uses the concept of a "lease" or amount of time that a given IP address will be valid for a computer. The lease time can vary depending upon how long a user is likely to require the network connection at a particular location. DHCP also supports static addresses for devices that need a permanent IP address.

Checking the check box on the Setup Screen enables DHCP. At power up, the controller will request to use whatever IP address is currently set (except 0.0.0.0 which enables DHCP), the DHCP server will either allow it or supply a new IP address. This final address will temporarily be written to the 20048 register block of the controller, but not permanently, and will appear in the “IP Address” data entry field. Once complete with all changes, simply press the “Update Network” button to notify the controller of changes. Values are immediately read back from the controller allowing for visual confirmation.

DNS Name

When the controller communicates with a DHCP server it also requires a unique system name that is typically used for DNS resolution (assuming the server is using dynamic DNS). Presently this name is derived from the controller’s serial number, placing “CTC_BF_“, before the number. For example if the serial number was 100-52801 then the DNS name entry for the controller would become CTC_BF_10052801. User settable names are also possible by simply double-clicking the data entry field and entering a unique name. **Up to 20 characters are allowed** in the Controllers DNS Name. When the “Update Network” button is selected the controller will immediately notify the DHCP server of a name change, if DHCP is enabled. If dynamic DNS is enabled, on your host, the name change will become available immediately on your network.

 Many software packages, and other devices with CTC communications drivers, do not have the capability to identify controllers by name, only by IP Address. Depending on how your network is configured, DHCP may change the IP

address of the controller without warning, causing devices and software to lose connection or connect to the wrong controller. In this case, it is better to manually assign a static IP address to the controller. The network administrator should be contacted prior to assigning any IP address, to avoid conflicts.

IP Address

If you are not using DHCP to automatically obtain your IP Address information then the TCP/IP IP address is configured statically. It must be entered using a 'dot' notation as follows:

Example IP Address 168.254.132.34 (example)

The actual values to use will depend on the network that the controller is connected to. Contact your IT department to determine acceptable addresses for your network.

Subnet Mask

If you are not using DHCP to automatically obtain your IP Address Information then the TCP/IP subnet mask address is configured statically. It must be entered using a 'dot' notation as follows:

Example Subnet Mask: 255.255.255.0 (typical)

The actual values to use will depend on the network that the controller is connected to. Contact your IT department to determine acceptable addresses for your network.

Gateway IP

If you are not using DHCP to automatically obtain your IP Address then the TCP/IP Gateway address is configured statically. It must be entered using a 'dot' notation as follows:

Example Gateway 168.254.132.88 (example)

The actual values to use will depend on the network that the controller is connected to. Contact your IT department to determine acceptable addresses for your network. A value of 0.0.0.0 will disable the use of a gateway. A Gateway is the address to which requests will be forwarded if they are outside the range of you IP domain, as tested against the assigned subnet mask. Typically a gateway is used to forward requests to another network and/or the internet.

Modbus

The Modbus address is used to set the address which will be used by the Modbus/TCP communications protocol. It is typically referred to as the Device ID. It may be set from 1 to 255.

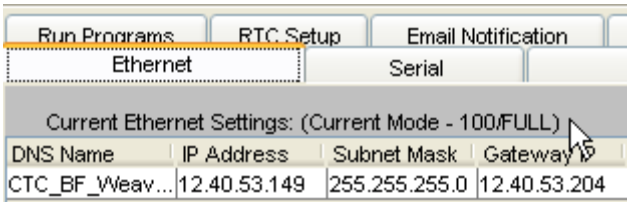
CTC Node

The CTC Node number is used by the CTNet protocol. This is a lightweight non-routable Ethernet protocol used by legacy CTC controllers. It is recommended that UDP be used, instead, whenever possible, since it is routable. Setting this node number to 0 disables its use in the controller. Be careful setting this node number since no two controllers can have the same address. Valid numbers are from 1 to 32767. Some very old CTC controllers only communicate on nodes 1 to 254.

Mode

Mode is used to set the Ethernet connection method, speed and duplex, and typically is not used. By default it is set to Auto. Auto means, auto-negotiate, or let the controller and external router/switch negotiate connection speed and duplex. The fastest possible will generally be negotiated, 100 Megabits/Full Duplex. Sometimes, where old wiring may exist or noisy environments, it is best to reduce the speed of the Ethernet interface. Also if Ethernet speed is not important, the slower speed will reduce the load on the controller and generally allow increased performance by other aspects of the controller during peak Ethernet traffic.

A pull-down box is provided to override the default. Available are 100 full/half duplex, 10 full/half duplex, and auto. Note that the current negotiated speed is shown in the text area above the data entry fields. Below shows the current speed is negotiated to 100 full duplex:



CHAPTER

4

ASCII Computer/Terminal Protocol

The 5200 supports a number of serial port communication protocols, the default, along with the CTC Binary Protocol, is a simple ASCII protocol. Both run at the same time and are automatically detected based on the serial data stream. The ASCII Protocol is a simple way to send commands to the controller. The commands are in the form of simple ASCII messages. Most computer languages provide a method for sending ASCII messages to a serial communications port.

ASCII Computer Protocol

Controllers are initialized to the CTC ASCII terminal protocol upon power-up. To change the terminal protocol, you must send a command to the controller's serial port establishing a new protocol. In the following example, the P sets the protocol and C establishes the CTC ASCII computer protocol. All commands are followed by a carriage return <CR>, ASCII 13, which signals the controller that the command is complete. Most versions of BASIC automatically add the required carriage return at the end of the transmission.

To set the CTC ASCII computer protocol:

1. Enter the following command:
   P C <CR>
2. To acknowledge the change to the computer protocol, the controller responds with:
   P C Ø <CR>

Ending the response with a carriage return is consistent with the computer protocol.

Once you have opened the serial port and set the computer protocol, you can begin sending commands to the controller. The following example forces the number 1200 into register 10, the command is "R=1200". The command must end with the code for a carriage return command, ASCII 13. The following statement, in BASIC, accomplishes this transmission:

```
PRINT #1, "R10=1200"
```

Computers and versions of BASIC vary. Refer to manufacturer's published data. By sending this command, we assume that the serial port 1 is already opened and defined as port No. 1. Most versions of BASIC automatically add the required carriage return at the end of the transmission. Check with your version of BASIC to see if it automatically adds the carriage return command.

When operating in the CTC ASCII computer protocol, the controller responds with a carriage return command, acknowledging message reception. Your BASIC program should receive and test this message. If a transmission error occurs, the controller instead responds with an error message. You can program the message test as follows:

```
LINE INPUT #1, R$
IF R$<>"" THEN GOTO 100
```

The statement, LINE INPUT #1, R$, tells the computer to receive the controller's response and to assign the response to character string R$. In most versions of BASIC, a response consisting of only a carriage return is received as a null string or an empty message. The statement, IF R$<>"" THEN GOTO 100, has the computer test the response. If the controller's response in not equal to a null string, a transmission error occurred. At this point, the program jumps to line 100.

The controller's response must be taken in by the computer. If it is not, the response remains in the computer's communication buffer, and affects the computer's ability to receive future messages.

ASCII Terminal Protocol

At times you may want to use a dumb terminal or a computer running a terminal emulation program to communicate with a controller. You can use a lap top computer configured as a dumb terminal for diagnostic or debugging purposes, forcing outputs on or off, reading register values, or forcing a value to be stored into a register. The CTC ASCII computer protocol is not suited to this task, since it has been optimized for use in communicating with a running computer program. It addition, you must terminate each response with a carriage return, signaling the completion of the message.

When you use a dumb terminal to directly view the response of the controller, the carriage return places the terminal's cursor to the beginning of the same line, and the next message overwrites the previous message and responses. The CTC ASCII terminal protocol solves this problem by responding to commands from a terminal or computer with an instantaneous line feed, <LF> ASCII 10, moving the terminal to the next line on its screen. The controller transmits its response, if any, with a carriage return and a line feed. Any messages sent to or from the controller are recorded on successive lines. Except for the use of line feeds, the terminal protocol is identical to the computer protocol.

Controllers are initialized to the CTC ASCII terminal protocol upon power-up. If you have changed it, you must reset the protocol. In the following example, the P sets the protocol and T establishes the CTC ASCII terminal protocol. All commands are followed by a carriage return. To set the CTC ASCII terminal protocol:

1. Enter the following command:
   P T <CR>
2. To acknowledge the change to the computer protocol, the controller responds with:
   <LF>
   P T <CR>
   <LF>

The controller immediately responded with a line feed and the response ended with both a carriage return and a line feed. This creates a readable display on the terminal. This response is also consistent with the terminal protocol.

ASCII Protocol Commands

Using either the computer or terminal protocols you can access any of the controller's registers. The example commands use **<CR>** to stand for a carriage return (ASCII 13) and <LF> for a line feed (ASCII 10):

Initiate computer mode:

Send - PC<CR>
Response - PC0<CR>

Initiate terminal mode:

Send - PT<CR>
Response - <LF>PT<CR><LF>

Read a counter/register:

Send - R<counter/register number><CR>
Response:
Computer mode - < counter/register number ><CR>
Terminal mode - <LF>< counter/register number ><CR><LF>
Note: Register read/write commands can be chained together using a ';' as a separator. Each command will be responded to uniquely.
Example: R1000=5;R1005;R1006<CR>

Write a counter/register:

Send - R<counter/register number>=<new value><CR>
Response:

Computer mode - <CR>
Terminal mode - <LF>

Note: Register read/write commands can be chained together using a ';' as a separator, each command will be responded to uniquely.

Example: `R1000=5;R1005;R1006<CR>`

Returned Error Messages

Number too small – If a register is specified as zero, then the controller sends the following error message:

Computer mode - `<less than sign,< > <bell, 07H><CR>`
Terminal mode - `<LF><less than sign,< ><bell, 07H><CR><LF>`

Number too large – If a register is specified that is greater than the number supported, then the controller sends the following error message:

Computer mode - `<greater than sign,> > <bell, 07H><CR>`
Terminal mode - `<LF><greater than sign,> > <bell, 07H><CR><LF>`

Protocol error – If a "P" command (protocol) is not in the correct format then the controller will send the following error message:

Computer mode - `P<bell, 07H><CR>`
Terminal mode - `<LF>P<bell, 07H><CR><LF>`

Syntax error – If the controller can not make any sense of the command, then it sends the following message:
Computer mode - ?<bell, 07H><CR>
Terminal mode - <LF>?<bell, 07H><CR><LF>

CHAPTER

5

TCP/IP Raw Sockets

Up to 5 TCP Client/Server RAW Socket sessions are supported by the 5200 controller. These socket sessions provide a virtual pipe, with no formatting of data. To the controller they merely appear as another serial port, even though the connected device can reside virtually anywhere on a network connection. This interface is extremely useful for connection to external programs, such as Visual Basic or Ethernet based terminal servers such as the Newport or Lantronix devices. Lantronix is described within this section, Newport is similiar.

TCP Client

A TCP Client RAW Socket session is when the host computer runs a TCP Server and the controller connects to it. Typically a well-known IP address and public TCP port number is available for this connection. Once the connection is made, any data sent to the actively selected serial port (12000 register) is sent to the host and anything sent by the host to the controller is placed in it's receive buffer, exactly like an actual serial port. To initiate a connection, a number of registers must be configured.

The RAW Socket session register blocks begin at a base of 22000 and extend to 22049, one repeating block pattern (10 registers locations per block) for each serial port supported. The actual block used has nothing to do with the serial port itself when referenced from Quickstep since the serial port assignment is a configurable parameter. Blue Fusion Controllers have 2 physical serial ports (COM1=1, COM2=2, 0 not used) within the controller. They can also access virtual serial ports 3 to 7, which may be assigned as desired. Remember that server connections will use the next available port when allowing connections from a host client. Therefore, it is important to reserve your port first prior to enabling a Server register block.

Registers are defined based on their offset from their base, repeating after each 10. Therefore, beginning at register 22000:

REGISTER	DESCRIPTION	USE
22XX0	Controller Serial port ID register	Enter 3 to 7 for virtual port identifier.

22XX1	Client/Server register	To initiate a connection, set this register to 0..If the controller is a server, set to 1. In the case of this section, we are a client, so this would be 0 because we initiate the connection.
22XX2	IPA register	Most significant octet of IP address to connect to.
22XX3	IPB register	
22XX4	IPC register	
22XX5	IPD register	Least significant octet of IP address to connect to.
22XX6	TCP Port Connection register	TCP Port to connect to (client) or listen on (server).
22XX7	Connection Status register	On read, -1 = not initialized, 0 = offline, 1 = online. Write a 1 to initiate connection or start server thread.
22XX8	Index register	Provides access to special purpose registers and general data access. Recommend using serial port buffer for data, not this interface but available to mimic the peer to peer interface.
22XX9	Data Array register	R/W access to register selected by the Index register.

XX represents a multiplier of 10, which is the size of a block (00, 01, 02…).

An example for a script program to initialize a connection to a host at IP address 12.40.53.185 and TCP port 3001 is shown below. Note the controller Serial Port ID Register, number 22000, must be set up first:

```
22000 = 3       # set up this client connection as controller port 3
22001 = 0       # set that we are the client, initiating connection
22002 = 12      # most significant octet of IP address 12.40.53.185
22003 = 40
22004 = 53
22005 = 185     # least significant octet of IP address 12.40.53.185
22006 =  3001   # TCP port to attempt connection to
22007 = 1       # To initiate a connection write a 1 to the status register then read
                # it until it is a 1
                # which means connected.  0 is offline, -1 is not initialized.
```

Once register 22007 is read as a 1, then port 3 will appear as a standard serial port to a Quickstep application. As with any serial port, the port must be selected first by writing the port number to register 12000 prior to transferring data or initiating commands. The port is available for reading and writing upon connection to the host, i.e., when register 22007 = 1. Should a connection ever be lost, 22007 will contain a 0 and a read of 12000 (Message status register) will return a 1, indicating transmitter busy, or in this case, offline. With TCP the transmitter will never be busy unless offline. The controller will periodically retry the client connection.

TCP Server

A TCP Server RAW Socket session is when the host computer is the client, connecting to the controller on a public TCP port number. Once the connection is made, any data sent to the actively selected serial port is sent to the host and anything sent by the host is placed in the receive buffer, exactly like a controller serial port. In order to allow a server to be active the same registers as detailed in Client must be configured, except a 1 is placed in register 22XX1 and our port number to listen on is stored in 22XX6:
Registers are defined based on their offset from their base, repeating after each 10. Therefore beginning at register 22000:

REGISTER	DESCRIPTION	USE
22XX0	Controller Serial port ID register	Enter 3 to 7 for virtual port identifier.
22XX1	Client/Server register	To initiate a connection, set this register to 0. If the controller is a server, set to 1. In the case of this section, we are a server, so this would be 1 because we listen for the connection.
22XX2	IPA register	Most significant octet of IP address to connect to.
22XX3	IPB register	
22XX4	IPC register	
22XX5	IPD register	Least significant octet of IP address to connect to.
22XX6	TCP Port Connection register	TCP Port to connect to (client) or listen on (server).
22XX7	Connection Status register	On read, -1 = not initialized, 0 = offline, 1 = online. Write a 1 to initiate connection or start server thread.
22XX8	Index register	Provides access to special purpose registers and general data access. Recommend using serial port buffer for data, not this interface but available to mimic the peer to peer interface.

22XX9	Data Array register	R/W access to register selected by the Index register.

A server thread will be launched as soon as a 1 is written to the status register. Note that only one connection is allowed at a time since all information is directed to and from a controller virtual serial port. If more than one connection attempt is made to the same port number defined in the configuration block, it will be initially accepted and then rejected.

Lantronix CoBox/Xpress interface Example

The Lantronix CoBox-DR1-IAP or Xpress-DR-IAP (Device Server (www.lantronix.com) is one of several serial to Ethernet converter devices which will work with the controller using the TCP RAW Client socket protocol. To the controller, this device is communicated to over TCP port 3001 and becomes a simple virtual serial port to Quickstep. It operates exactly as a resident local port, supporting the same communication protocols. Communications is tunneled over the network to the device. Even a serial port version of CTMon or a CTC 4010 User Interface can be connected and run over this interface, allowing for easy port expansion. Modbus is also supported.

By encapsulating serial data and transporting it over Ethernet, devices such as these allow virtual serial links to be established over Ethernet and distributed virtually anywhere within a plant or global enterprise.

Lantronix CoBox Serial to Ethernet Converters

CHAPTER

6

UDP Peer to Peer Protocol Overview

Peer to Peer communications allows a controller to monitor another controller's registers, from across a network. In essence, the designated registers become public and a copy of their contents is periodically transmitted across the network, to the requesting controller, thereby making them appear as though they are local. The update scan time is configurable and the registers may be read from or written to in a manner similar to normal registers.

Peer-to-Peer Protocol Registers

The controller can only perform peer-to-peer operations with other 5100/5200 modules. Model 5200 controllers can also communicate with Model 2700 controllers via the 2717 communications module, but not via the 2217 module. The 5200's peer-to-peer registers let it communicate directly with other 5200 modules without requiring a dedicated server. It can also gather register information locally for different network protocols.

Registers 21000-21299 are read/write registers that are reserved for peer-to-peer networks. Each block of 10 sequential registers is assigned to a designated peer node and defines the peer environment for that connection. You can retrieve data from and automatically update up to 100 sequential registers with a single request. Also note that this register block can be used for many other functions, besides peer to peer, such as Modbus, interfacing in a similar manner. Reference that section for further details.

Registers 21000-21299

21XX0	**First Octet IP Address Register (Most Significant) - R/W** This is the first octet of the IP address (XXX.000.000.000) that is used to make peer requests.
21XX1	**Second Octet IP Address Register - R/W** This is the second octet of the IP address (000.XXX.000.000) that is used to make peer requests.
21XX2	**Third Octet IP Address Register - R/W** This is the third octet of the IP address (000.000.XXX.000) that is used to

	make peer requests.
21XX3	**Fourth Octet IP Address Register (Least Significant) - R/W** This is the fourth octet of the IP address (000.000.000.XXX) that is used to make peer requests. Once a peer connection is attempted, you cannot change the IP octet register settings.
21XX4	**Start Register - R/W** This register stores the starting register address in the controller for peer-to-peer communications. You can change this register number after a peer connection is attempted, but the number of sequential registers must stay the same (see *Register 21XX5* for more information).
21XX5	**Sequential Number Register - R/W** This register stores the number of sequential registers (starting with Register 21XX4) you want to read during a peer-to-peer session. The value 1 represents a single register and the maximum number of registers allowed is 100. Configure this register before setting up any other registers. Do not change this value during a peer-to-peer transaction or all data will be lost and new values will have to be entered. If you modify this register, it lets you reset the peer connection.
21XX6	**Poll Timer Register - R/W** Set this register to 0 for a single read request. Specify a value (in units of ms/count) if this register is going to receive periodic updates from the server controller (the controller sending information to the register). The minimum value allowed is 50 ms. For example, the value 500 would refresh the data registers with new peer data every ½ second (500 ms). You can write to this register at any time. Writing a 0 to this register while actively conducting a peer-to-peer session cancels the periodic update and causes a new single read transaction to occur. A time-out (Status Flag Register 21XX7 = 0) occurs if the server has not refreshed peer data in a time equal to 2-½ multiplied by the poll timer value. You can access this register at any time once you have initialized the Sequential Number Register (Register 21XX5). Data registers are mentioned in numerous places throughout the listings below. These registers are represented by Register 21XX9, which is a phantom register. For more information, refer to the 21XX9 listing in this section.
21XX7	**Status Flag Register - Read-Only** This register reflects the current status of the data registers. Its value is based on any requested operations. Typically, you initiate an operation and then wait for a status of 1. Possible values are:

STATUS	DESCRIPTION
0	Offline; no connection is present.
1	Last request is successful and completed. Data is available in the data registers if requested.
-1	Requested operation has failed.
-2	Busy; connecting to the desired host.
-3	Busy; reading data.
-4	Busy; writing data.
-10	Aborted operation; out of local memory or resources.

21XX8 **Index Offset Register - R/W**

This register lets you access each of the requested sequential data registers. It works in conjunction with Register 21XX9 and acts as its pointer. You can store the number of a general or special purpose register in 21XX8 and 21XX9 can then access the resource contained in the pointer. The first register (with an index of 0) is the Start Register (Register 21XX4). 1 is the next register, and so forth. Once Register 21XX5 (the Sequential Number Register) is initialized, you can change this register's value at any time. For more information on how pointer registers function, refer to the *Register Reference Guide*.

The index register also has a few special features when you set it to 1000 or above. Modifications are made by writing to the data register and setting the index register appropriately as described below:

1000 - Peer Request Time-Out Register - The timer starts when a peer node request is initiated and stops (times out) if no response is received within the time specified by this register. Retries only occur if automatic updates are active (Register 21XX6 is set to a value other than 0). Defaults are 500 ms for single register reads and time-out value * 2.5 for automatically updated register read transactions.

1001 - Peer Request Failed Index Register - This register indicates when a peer transaction fails and an error occurs. The Status Flag Register (21XX7) is set to a value other than 1. Any data that was read or written when the error occurred has an offset value that is stored in 1001. If you read the data register, it returns the offset failure value. Data written before this offset value is valid. For example, if your

process continuously updates 50 registers and the register returns a value of 25, it means the process failed while trying to write the 25th element of data. All data written before this element was written correctly.

1002 - Peer Request Retry Counter Index Register - This debugging register points the data register to the retry counter. Quickstep can set this register to any value. The register is incremented by 1 when a time-out occurs because of waiting for data from a peer node.

1003 - Protocol Index Register - This register tells the data register what protocol to use for setting the peer block registers. You must set this register before setting the Start Register (21XX4). Default mode is 0 for UDP Peer-to-Peer protocol. 2 is used for Modbus TCP Master mode, and 3 for Modbus Serial Master.

1004 - Peer Request TCP Client Port Index Register - This register points the data register to the destination TCP Port address for your connection. You must set this register before setting the Start Register (21XX4). 1004 is currently used for Modbus TCP Master mode with a default port number of 502 (the industry standard).

1005 - Modbus Master Unit ID Index Register - This register points the data register to the Unit ID field value used in the Modbus Master request packet. The default ID is 00 but you can set it to any desired value. This ID affects all subsequent transmissions and allows multiplexed nodes to be addressed in a Modbus environment.

1006 - Modbus Master Exception Index Register - This register tells the data register where the last Modbus Exception error code is stored from a previously received message. Referencing this register helps to interpret failure types.

1007 – Register Remapping Start Index Register – This option allows remote registers to be mapped into the 23000 to 24999 consecutive memory space. Previously an index register at 21XX8 needed to be set and then data read from 21XX9. This can result in slow operation if a lot of data needs to be transferred. Setting 21XX8 to 1007 and then writing the register value from 23000 to 24999 will allow all data to be remapped to that register block area, consecutively, based upon the block size (21XX5). A write to the remapped area will result in a remote write. By default re-mapping is not active.

1998 - Write Enable Index Register - This register is used to control the updating of writes to the peer. When enabled (default, 1), any write to a data register (21XX9 or remapped area) will cause a single write to the remote host. Setting this register to a 0 inhibits write operations.

	This allows the programmer to update the register block, as required, then set the Write Enable Register back to a '1' to update the entire block, on the remote host, by sending a single packet. The Write Enable Register will not return a '1' on a read operation of '1998' until an acknowledge from the remote host has been received, verifying the write occurred. The transition of the Write Enable Register from 0 to 1 causes the block write. However, writing a 1 when a 1 already exists has no effect.
21XX9	**Data Registers/Peer Request Time-Out Register - R/W** This phantom register contains peer data that is read or written in a peer transaction. It is a "window" into a register array in the controller. The array size is set by Register 21XX5 and the offset is specified by Register 21XX8. Data integrity is indicated in Register 21XX7. For more information on how phantom registers function, refer to the *5200 Register Reference Guide*.

Initiating a Peer to Peer Session

In general, the initializing of the peer-to-peer mechanism works as follows:

- Write the desired number of registers to 21XX5 register.
- Write the slave's IP address to 21XX0 - 21XX3 register
- Write the register to begin reading from the slave device to 21XX4
- Write a 1007 to register 21XX8 to select the re-mapping area.
- Write where in the 23000-24999 register range you want it to appear to register 21XX9.
- Write a 0 to the index register 21XX8 to default it back to viewing the first data item.
- Write the scan time, typically 100ms to register 21XX6, to initiate the connection and begin peer to peer.

Monitor status register 21XX7 for a 1 prior to reading/writing to either the 21XX9 data area or the re-mapped area in the 23000-24999 block.

Blank

CHAPTER 7

Modbus

The Modbus Protocol is a messaging structure developed by Modicon in 1979. It is used for master-slave/client-server communications between intelligent devices and has become an industry standard. Details of the protocol may be found at the web site www.modbus.org. This protocol allows a master to periodically poll the controller to collect the desired information. Modbus supports two major flavors of data representation: RTU and ASCII. RTU is a more compact protocol, consisting of binary characters, while ASCII represents each binary nibble as a separate character, hence doubling the length of transmissions. RTU is also more secure in that it includes a CRC-16 at the end of the message while ASCII only has a single LRC. **The CTC 5200 controllers support Modbus Master/Slave TCP RTU, Modbus Master Serial RTU/ASCII, and Modbus Slave Serial RTU/ASCII.**

Tools used to test the protocol are available from a number of sources. The 5200 controller was tested using those available from www.win-tech.com, namely their ModScan32 for RTU/ASCII Slave testing and ModSim32 for Master.

Modbus Slave RTU TCP & RTU/ASCII Serial

A polling master can drive a slave controller using the Modbus protocol. The 5200 controller supports slave mode both over an Ethernet TCP connection and/or a serial connection. Modbus allows for interfaces to such things as coils, analog, register, etc. Since the 5200 controller is able to access all of its resources via its register interface, typically only the Holding Register commands are used: Write Single Register (function code 0x06), Write Multiple Registers (function code 0x10), and Read Holding Registers (0x03). Alternatively, the "Read Input Discrete" (maps to digital in modules), "Read coils", and "Write Single Coil" (maps to digital output modules) are supported.

	Function codes		
	Code	Sub code	(hex)
Read input discrete	02		02
Read coils	01		01
Write single coil	05		05

Write multiple coils	15		0F
Read input register	04		04
Read multiple registers	03		03
Write Single register	06		06
Write multiple registers	16		10
Read/write multiple registers	23		17
Mask write register	22		16
Read file record	20	6	14
Write file record	21	6	15
Read device identification	43	14	2B

Figure 7.1: Modbus Function codes from Modbus.org (highlighted yellow are supported by 5200)

You should also note that Modbus Holding registers are 16 bits in width and those of the 5200 controller are 32 bits, since Modbus is Big Endean. This means when reading register 1 in the 5200 controller, the high 16 bits equates to Modbus register 1 and the low 16 bits to Modbus register 2. Modbus register 3 would be the high 16 bits of register #2, and so on. The number of registers that can be read by a polling master at one time is limited:

Modbus RTU TCP – 120 Modbus 16 bit registers (60 5200 registers).
Modbus RTU Serial - 120 Modbus 16 bit registers (60 5200 registers).
Modbus ASCII Serial - 56 Modbus 16 bit registers (28 5200 registers).

This maximum is a limitation imposed by the Modbus TCP specification (which limits receive buffers to 255 bytes), not by the controller.

Modbus TCP Slave is always enabled and available for requests on TCP port 502 (standard). Either a Quickstep program or other means must manually enable Modbus RTU/ASCII Serial. This is done simply by writing a 3 to the "Serial Active Protocol Selection" Register, 12320, for RTU, or a 4 for ASCII. Prior to enabling it is recommended that the 5200 controller Modbus Unit/Device address also be set, using register 12321. Should a non-volatile controller wide default Modbus address be desired, set register 12322 with the address followed by a write to register 20096. Differences between TCP and serial implementations are detailed in section 3.1.1.

As a demonstration of the functionality of the Modbus RTU TCP/Slave interface, this section details the interface of Win-Tech's ModScan32 software and how it applies with regards to the 5200 controller. As mentioned before, CTC only supports the Holding Register interface. Upon installation of ModScan32, a screen such as Figure 3.3 will appear. Note that the Address field is set to 1, but the display screen starts at 40001. This is Modbus nomenclature. Address of 1 is the same as the upper 16 bits of the

controller register 1. Note Length is set to 50 (120 max), and Device ID is ignored since TCP is point to point (Device ID is ignored only when in TCP slave mode, not when the controller operates as a Master or serial slave.).

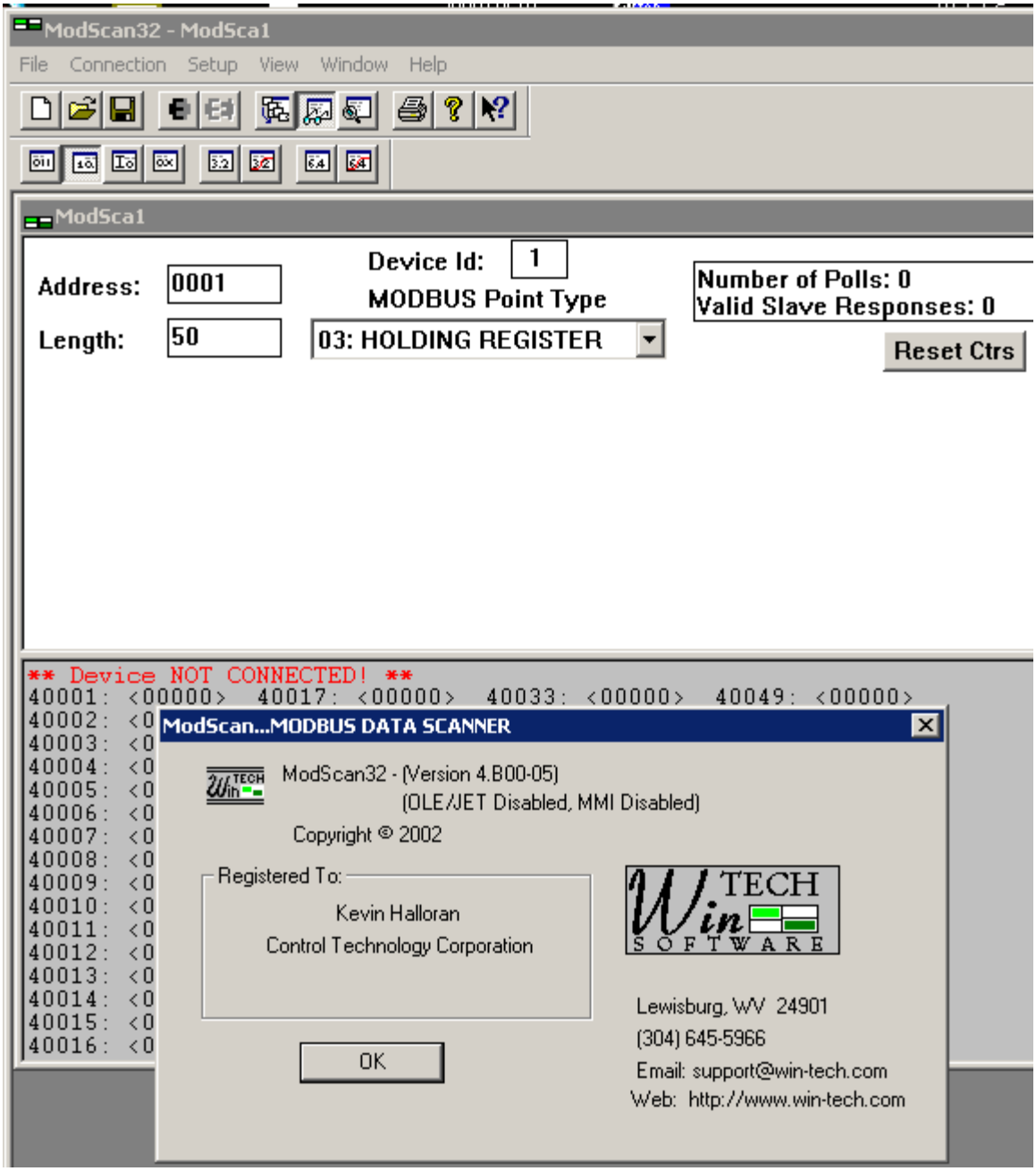


Figure 7.3: ModScan32 Master Scanning Program (only Holding Register supported)

Figure 7.4 shows the setup for an interface to a controller with a TCP address of 12.40.53.199 and the Modbus Slave running a server on the standard port of 502:

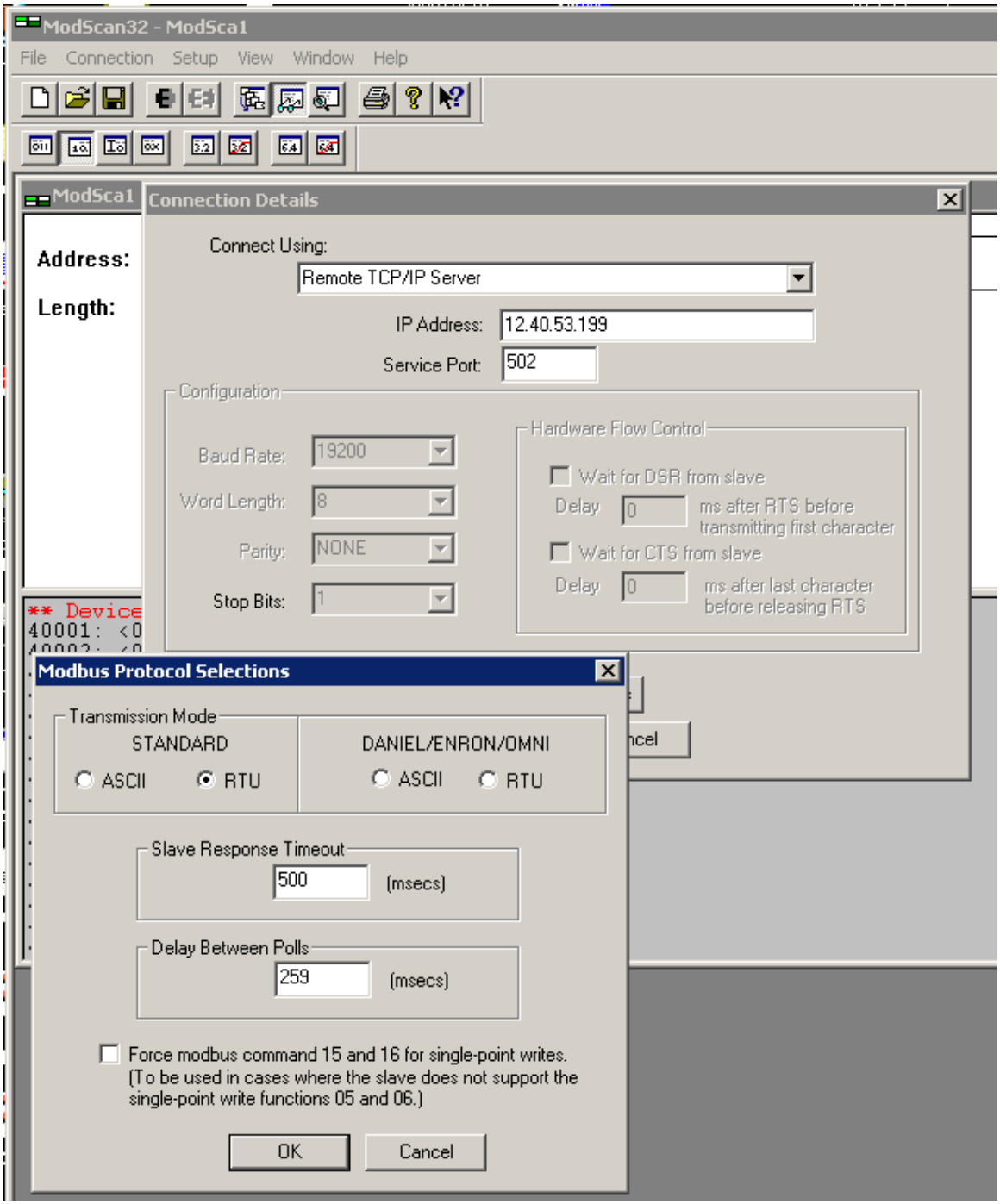


Figure 7.4: ModScan32 Master Scanning Program TCP Connection Setup

In order to do a single register write to a Modbus 16-bit register, double click that register. Figure 7.5 shows changing Modbus register 40002 (Address 2) to a value of 5, which would translate to the lower 16 bits of Quickstep register 1. Remember Modbus Address 1 is the upper 16 bits.

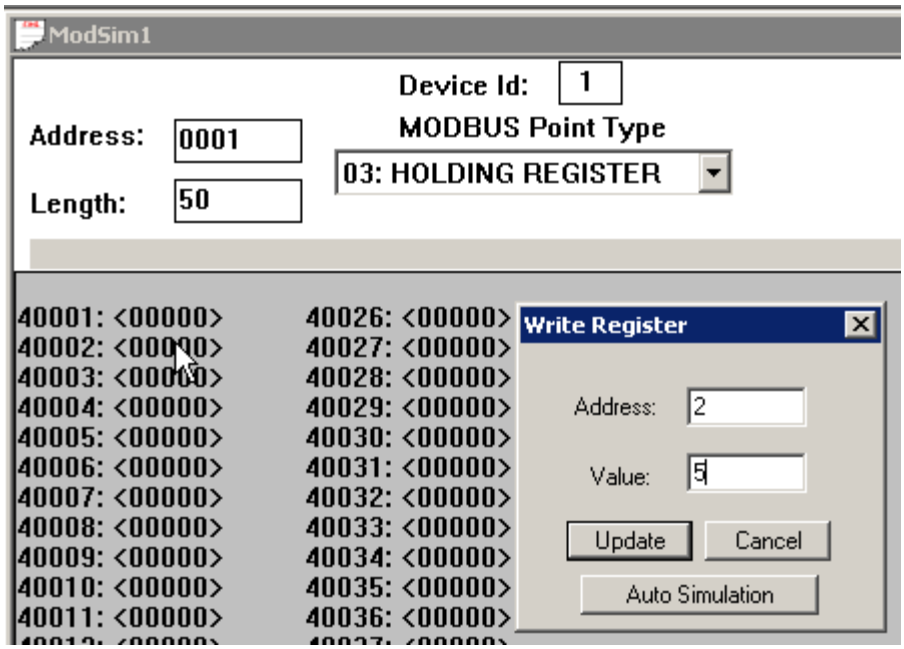


Figure 7.5: Single register write, value 5 to 40002

Changing a number of registers all at once is known as a Write Multiple Register access. This can be done using the Extended Access option:

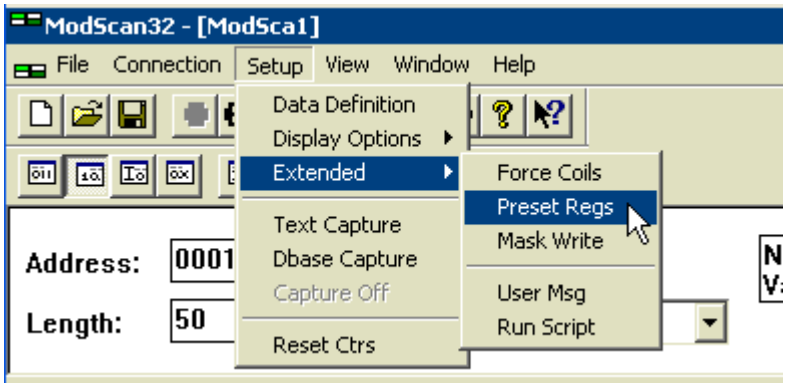


Figure 7.6: Write Multiple register (Preset Regs) selection

The Preset Multiple Registers pop-up will appear. Note that in TCP, the 5200 controller ignores any slave or node identifiers since it is a single device and not acting as a gateway. Set the Modbus register you wish to start changes with and the number of registers to change, up to a maximum of the number that you are viewing:

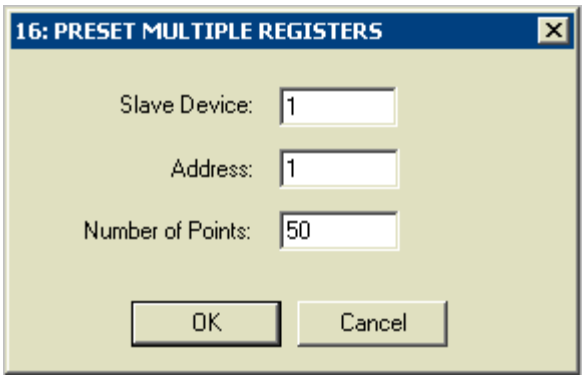


Figure 7.7: Preset Multiple register dialog

In this case we will change Addresses 1 to 10 to sequential numbers 1 to 10:

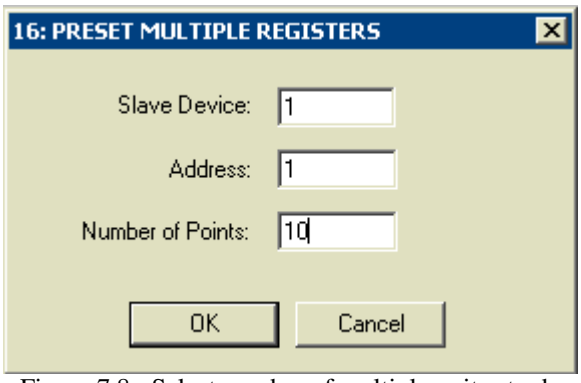


Figure 7.8: Select number of multiple writes to do

As shown below, the current register values are displayed in the dialog box.

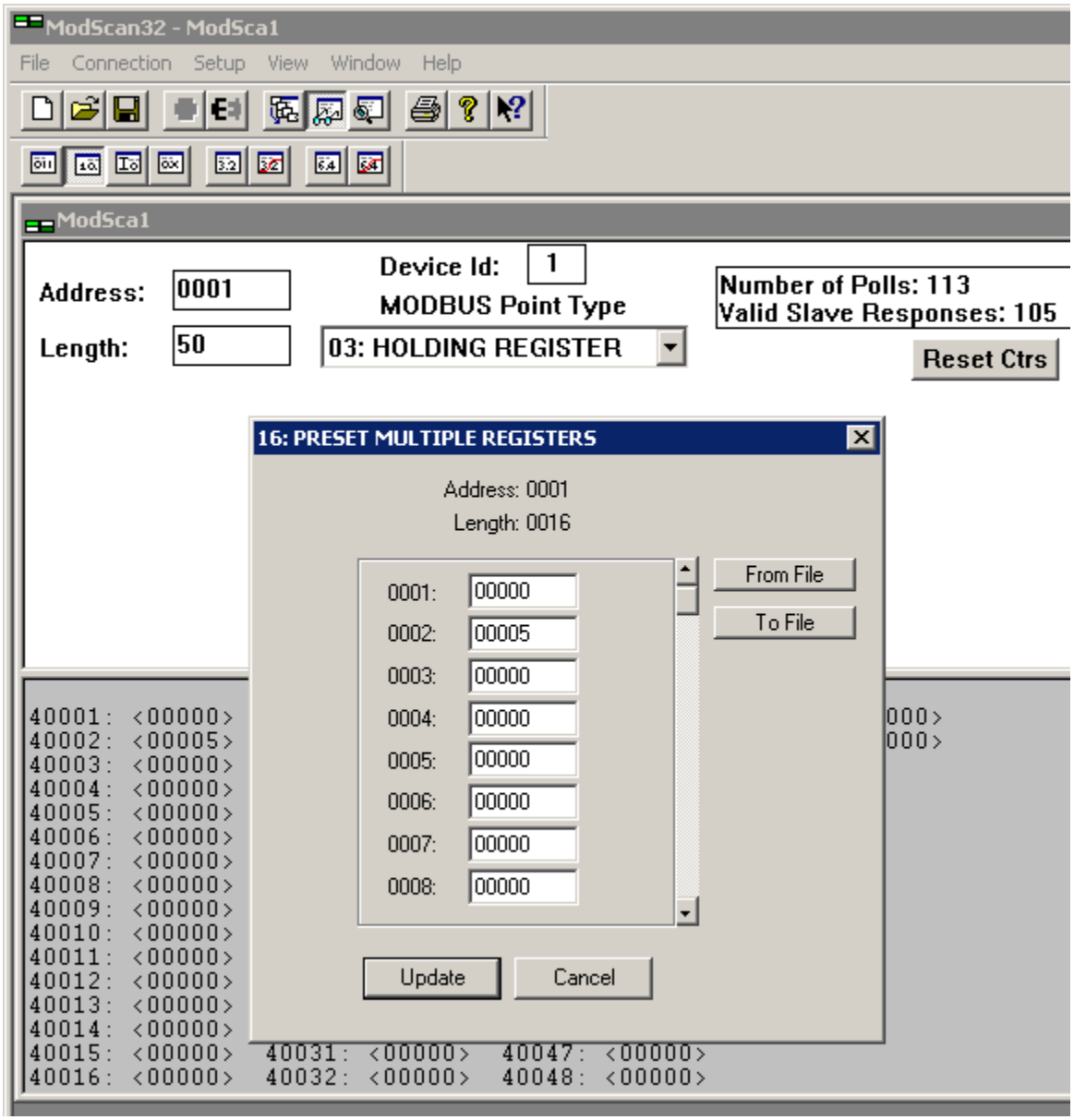


Figure 7.9: Preset Multiple register dialog viewing existing values

Note below, Figure 7.10, that each register value has been changed. Also, we scrolled down so we could get to register 10. Click Update and note the changed register values from the previous display; 40002 is no longer 5 but now 2, Figure 7.11.

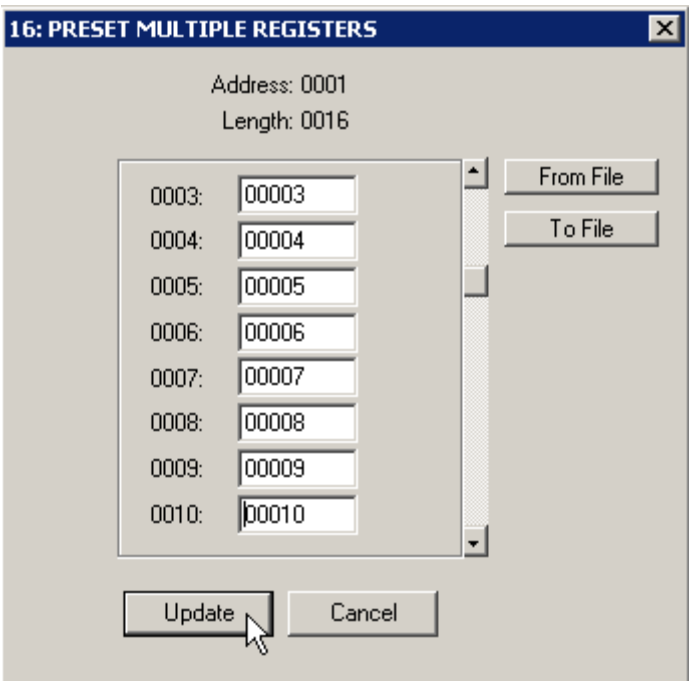


Figure 7.10: Preset Multiple new values entered

Upon clicking the Update key, the new values are written to the controller registers and new values read back using the Read Multiple Register command.

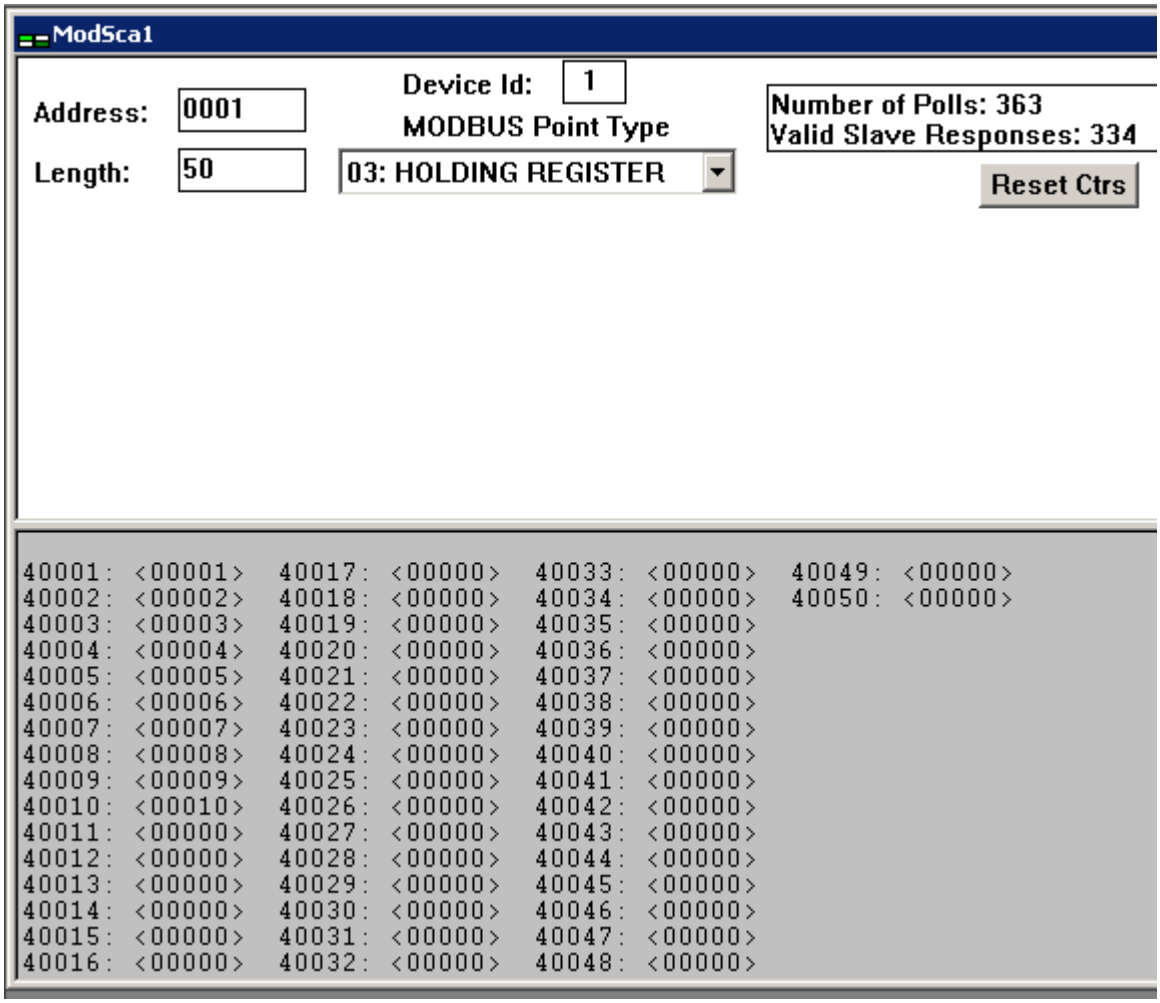


Figure 7.11: New values written and read back, Quickstep registers 1 to 5, Modbus 1 to 10

If any errors occur, a Modbus exception will occur. One such common error is attempting to read too many registers or illegal registers. Below is what is returned if > 120 Modbus registers are attempted:

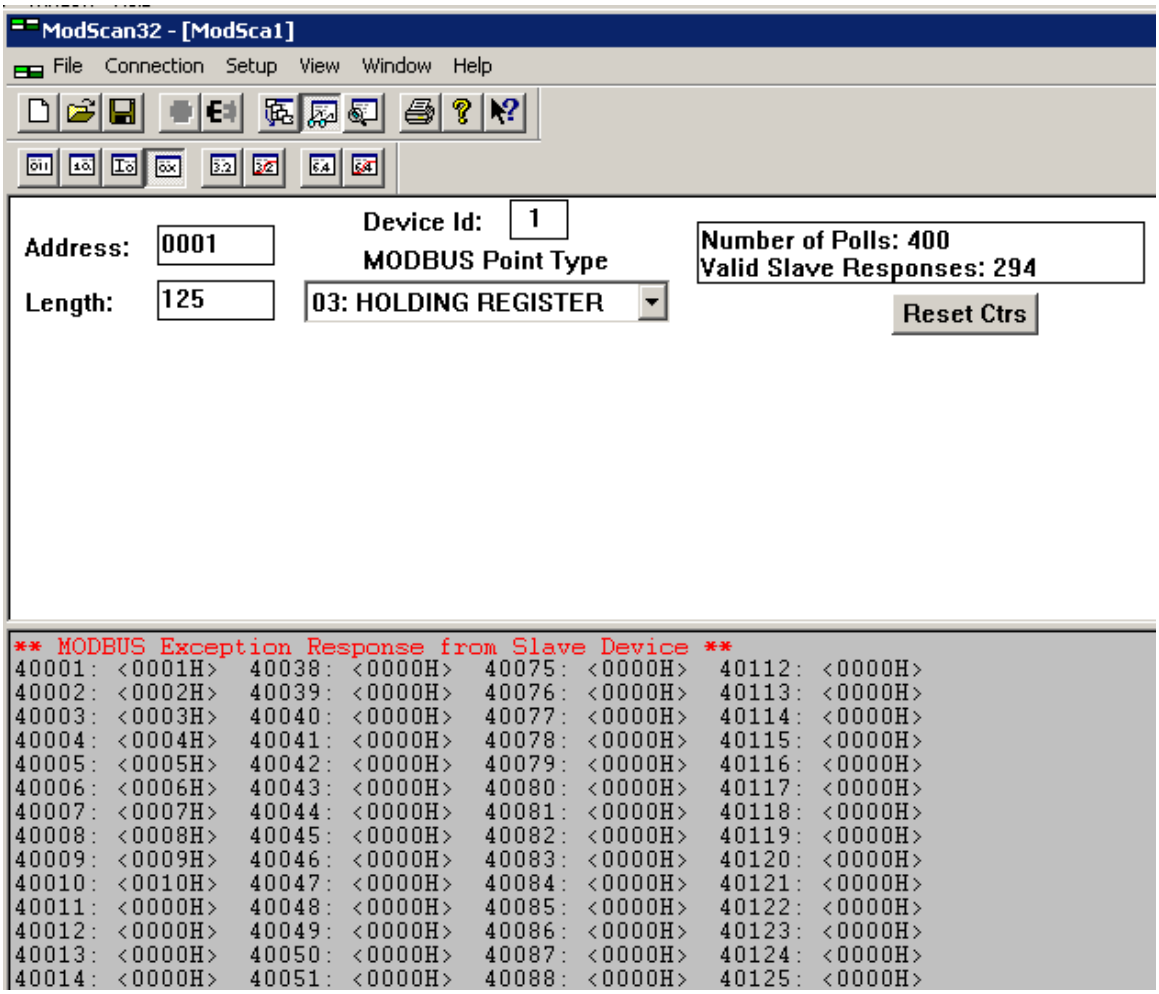


Figure 7.12: Modbus Exception Example > 120 registers

Editing the 125 appropriately will update the error. Below is an example of displaying registers in the 13002 block of the 5200 controller. 13002 is the system millisecond tic counter. Real time clock/date values can also be seen incrementing in other registers dynamically. Note that 26003 is the high 16 bits of 13002 and 26004 (13002 * 2) is the base lower 16 bits.

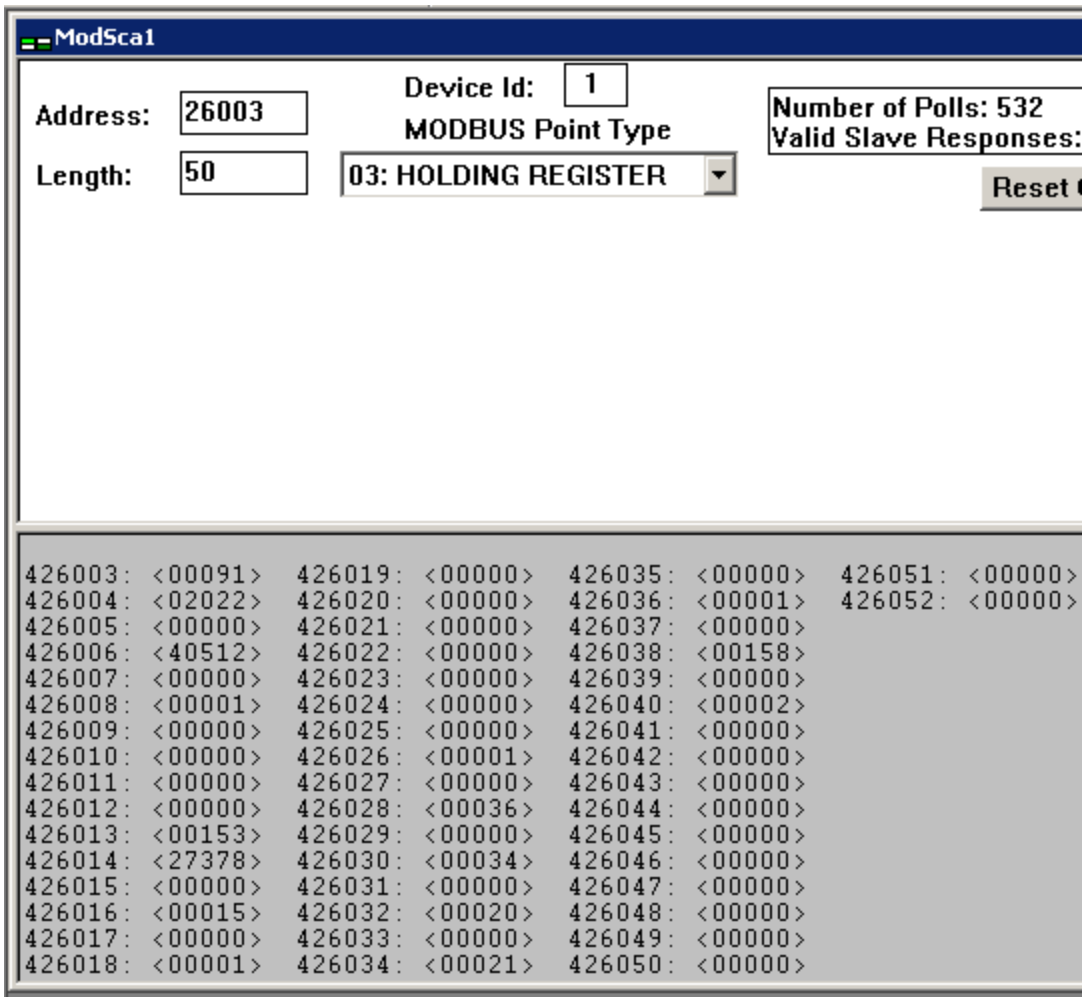


Figure 7.13: Display of controller system tic, dynamically updating, 426003/4

Modbus Slave Serial RTU/ASCII

The Modbus Slave Serial RTU and ASCII protocol functions exactly like that of Modbus TCP Slave with regards to how to access information and ModScan32 operation (see figure 7.14 for serial port setup versus TCP). There are some key differences since an RS232 connection is used versus a network connection.

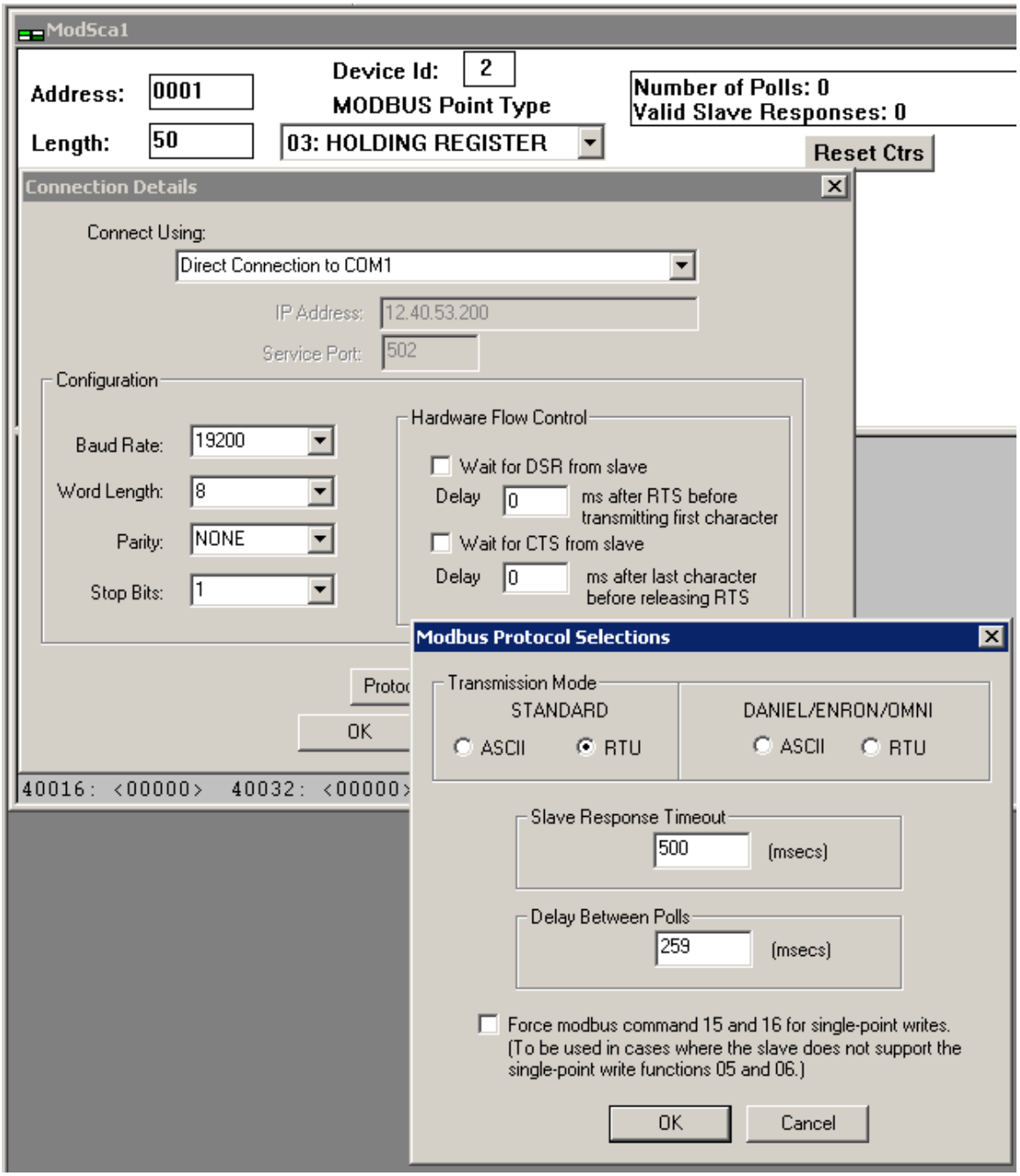


Figure 7.14: ModScan32 Master Scanning Program Serial Connection Setup, select RTU or ASCII Transmission Mode.

They are as follows:

1. The virtual TCP communication ports may also be used except for point to point operations with a single address present. In other words, the communications traffic of other Modbus nodes should not be present on the virtual port, although they can be on COM1/2. This is necessary because Modbus specifies a 3.5 character quiet time between packets and a maximum of a 1.5 inter-character delay during the continuous transmission of a packet data stream in RTU mode (1 second for ASCII mode). The virtual ports cannot guarantee these timing constraints, although from a high level protocol viewpoint, the ports do comply.
2. By default, the Modbus protocol is disabled on the serial and virtual ports. To enable the port, it must be the active port in the 12000 register and the proper Modbus protocol must be written to register 12320. Note that by default the slave port address is 2 and that any value written as the Modbus slave address will be that used on all serial ports, system wide. Note that writing a value of 0 to register 12320 will disable Modbus and return the port to normal CTC protocol operation.

 When Modbus is enabled on a serial port using CTCMON, no further communications will be available on that port except with Modbus. In other words, you will lose your CTCMON link if talking on the same port that is selected as active in register 12000.

Modbus Master TCP RTU & Serial RTU/ASCII

The Modbus Master protocol allows the controller to poll a Modbus TCP or Serial slave device, periodically requesting the registers for a particular device ID. As described in the Modbus TCP Slave section, the protocol allows for interfaces to such things as coils, analog, registers, etc. The 5200 controller is capable of only polling and writing to the Holding Registers of a remote device. Write Single Register (function code 0x06), Write Multiple Registers (function code 0x10), and Read Holding Registers (0x03) commands are supported. Be advised that Modbus Master, as implemented on the 5200 controller, only polls a single device ID. The active device ID register must be changed in order to begin polling a different device. Those who require slow scanning of multiple devices may change the device ID within the Modbus Master Register Control Block (21000-21299, shared with the UDP Peer to Peer register block) by the use of a Quickstep program or it may be remotely modified. This will cause all subsequent polls to use that device ID and hence allow the reading/writing of multiple devices.

A maximum of 256 sequential Modbus registers (16 bit) can be polled, each optionally mapped to a corresponding controller register (32 bit, 21XX8, index 1007). You may also adjust the active start register by changing register 21XX4, described in 3.2.1, dynamically. The controller will read a maximum of 120 RTU (56 ASCII) registers per packet request. This means if the number of registers desired is 50, then 50 will be read with each poll. If the number of registers is greater than 120, then multiple requests are made. If 256 registers are requested in RTU mode, for example, the first 120 are read,

then the next 120, then the remaining 16, all transparently to the user/programmer. When using the remapping register option, all registers will appear sequential within the 23000-24999 register blocks. Simply read and write as desired.

Registers 21000-21299

The 5200 controller can run numerous Modbus TCP Master connections and a single RTU/ASCII Serial connection at the same time, to differing devices, limited only by the performance desired. Each is configured using the Modbus Master Register Control Block (MMRCB. This same block serves multiple purposes and is shared with the UDP Peer to Peer Protocol register block detailed in section 5.1.1.

21XX0	**First Octet IP Address Register (Most Significant) - R/W** This is the first octet of the IP address (XXX.000.000.000) of the Modbus Slave to connect to. If a serial port is used, set to anything other than 0.
21XX1	**Second Octet IP Address Register - R/W** This is the second octet of the IP address (000.XXX.000.000) of the Modbus Slave to connect to. If a serial port is used, set to anything other than 0.
21XX2	**Third Octet IP Address Register - R/W** This is the third octet of the IP address (000.000.XXX.000) of the Modbus Slave to connect to. If a serial port is used, set to anything other than 0.
21XX3	**Fourth Octet IP Address Register (Least Significant) - R/W** This is the fourth octet of the IP address (000.000.000.XXX) of the Modbus Slave to connect to. Once a connection is attempted, you cannot change the IP octet register settings.
21XX4	**Start Register - R/W** This register stores the starting register address that is to be read from the remote Modbus Slave device. It may be modified at any time to select a different register block. Typically address 1 will represent Holding Register 40001 on the device.
21XX5	**Sequential Number Register - R/W** This register stores the number of sequential registers (starting with Register 21XX4) you want to read during a polling session. The value 1 represents a single register and the maximum number of registers allowed is 256. Configure this register before setting up any other registers. Do not change this value during a transaction or all data will be lost and new values will have to be entered. If you modify this register, it lets you reset the connection. All register reads from remote devices will be the same block size.
21XX6	**Poll Timer Register - R/W** Set this register to 0 for a single read request. Specify a value (in units of ms/count) if this register is going to receive periodic updates from the server controller (i.e., the controller sending information to the register). The minimum value allowed is 10 ms. For example; the value 500 would refresh

the data registers with new remote data every ½ second (500 ms). You can access this register at any time once you have initialized the Sequential Number Register (Register 21XX5).

Data registers are mentioned in numerous places throughout the listings below. These registers are represented by Register 21XX9, which is a phantom register. For more information, refer to the 21XX9 listing in this section.

21XX7 **Status Flag Register - Read-Only**

This register reflects the current status of the data registers. Its value is based on any requested operations. Typically, you initiate an operation and then wait for a status of '1'. Possible values are:

STATUS	DESCRIPTION
0	Offline; no connection is present.
1	Last request is successful and completed. Data is available in the data registers if requested.
-1	Requested operation has failed, typically a Modbus Exception error.
-2	Busy; connecting to the desired host.
-3	Busy; reading data.
-4	Busy; writing data.
-5	Timed out, retrying.
-10	Aborted operation; out of local memory or resources.

21XX8 **Index Offset Register - R/W**

This register lets you access each of the requested sequential data registers. It works in conjunction with Register 21XX9 and acts as its pointer. You can store the number of a general or special purpose register in 21XX8 and 21XX9 can then access the resource contained in the pointer. By default 0 points to the very first data element read from the remote device. This would be equivalent to what you set the Start Register to begin with (21XX4). Incrementing this register allows you to access other data elements, like an array. Register 21XX9 can then be read or written accordingly. If using an index register for accessing general data is not desired, the data may also be mapped to sequential registers of your choosing. Refer to the index register 1007 description below. This is the preferred method. However, do not modify the index register while a write is occurring or strange results may

occur.

The index register also has a few special features when you set it to 1000 or above. Modifications are made by writing to the data register and setting the index register appropriately as described below (only registers used by Modbus appear):

1003 - Protocol Index Register - This register tells the data register what protocol to use for setting the peer block registers. You must set this register before setting the Start Register (21XX4). Default mode is 0 for UDP Peer-to-Peer protocol. 2 is used for Modbus TCP Master mode, 3 is used for Modbus Master RTU Serial, and 4 for Modbus Master ASCII Serial.

1004 – TCP/Serial Client Port Index Register - This register points the data register to the destination TCP Port address for your connection, or serial port. You must set this register after setting the Protocol Index Register; otherwise, default values will overwrite any new values. When Protocol Index Register is set to 2 (TCP) the default client port is 502; when set to 3 (Serial), then the client port is set to 1, referencing COM1. For Modbus TCP Master mode, 502 is the industry standard port to connect to. Any client port less than 10 is assumed to be a serial port.

1005 - Modbus Master Unit ID Index Register - This register points the data register to the Unit/Device ID field value used in the Modbus Master request packet. The default ID is 1 but you can set it to any desired value. This ID affects all subsequent transmissions and allows multiplexed devices to be addressed in a Modbus environment.

1006 - Modbus Master Exception Index Register - This register allows you to interrogate the last Modbus Exception error code received from the data register (21XX9). Referencing this register helps to interpret failure types. Typically you would reference this register if a -1 appears as the current status in register 21XX7.

1007 – Register Remapping Start Index Register – This option allows remote registers to be mapped into the 23000 to 24999 consecutive memory space. Previously an index register at 21XX8 needed to be set then data read from 21XX9. This can result in slow operation if a lot of data needs to be transferred. Setting 21XX8 to 1007 and then writing the register value from 23000 to 24999 will allow all data to be remapped to that register block area, consecutively, based upon the block size (21XX5). A write to the remapped area will result in a remote write. By default re-mapping is not active.

1008 - Modbus Master MAX Retries Register – (R/W) This register allows you to change the maximum number of retry attempts on a Unit

	ID before giving up. Default is 2. **1009 - Modbus Master Retry Counter Register** – (R/W) This register allows you to observe and change the current number of message retries to the current Unit ID. **1010 - Modbus Master Timeout Register** – (R/W) This register allows you to change the default Unit ID timeout from 250 milliseconds to that desired, in milliseconds. Note that TCP needs a value > 200 milliseconds when talking to many applications, especially if PC based. If this is not used there will be many timeouts and retries. For example response times of up to 200 milliseconds have been observed the ModSim32 PC programs. The controller can handle smaller values without a problem, it is the PC side that is slow to respond. For Modbus Master RTU Serial the value in 21XX5 is added to the base timeout of 250 milliseconds. 1000 milliseconds is the base timeout for Modbus Master ASCII Serial. **1011 - Modbus Master Block Size Register** – (R/W) This register sets the number of Holding Registers to be accessed. Must be the same or smaller than the Sequential Number Register, defaults to the same. Used to access Unit ID's with varying block sizes when manually changing the Unit ID under program control.
21XX9	**Data Registers - R/W** This phantom register contains peer data that is read or written in a peer transaction. It is a "window" into a register array in the controller. The array size is set by Register 21XX5 and the offset is specified by Register 21XX8. Data integrity is indicated in Register 21XX7. If remapping of registers is not used then set 21XX8 to the array element desired, with 0 being the first.

Example Modbus TCP & RTU Serial Master Initialization

An example of Quickstep initialization code is shown below to set up a connection to the following remote device:

Modbus TCP Master Sample Program

IP address - 12.40.53.168
Device ID - 1
Number of sequential registers to read - 160
Scan time - 100 ms. (set last to initiate)
Starting Register - 1
Re-map registers to consecutive block beginning at registers 23000.
This is the first setup so use 21000, next would be 21010… 21020, etc…

```
[1] Initialize_ModbusMaster
   ;;; This program is used to initialize the TCP port
   ;;; for Modbus TCP Master operation.  A single
   ;;; device is polled using device ID 1 and 160 registers
   ;;; are read and mapped into the 23000 block.  Therefore
   ;;; registers 23000 - 23159 are used, with 23000 referencing
   ;;; Modbus Register #1.  Make sure your Modbus device has
   ;;; at least 160 consecutive registers starting at '1'
   ;;; otherwise Modbus Exceptions will occur.
   ;;; Begin by doing the following:
   ;;; 21005 = Maximum number of registers to read (160)
   ;;; 21000 - 21003 = Set this to be the IP address to
   ;;;                 connect to.  In this example we
   ;;;                 will use 12.40.53.168
   ;;; 21004 = Modbus start register (1)
   ;;; 21008 = 1003  = Set index to point to protocol register
   ;;; 21009 = 2     = Set protocol to Modbus TCP Master
   ;;; 21008 = 1004  = Set TCP port to connect to, default is 502
   ;;; 21009 = 502   = For demo set port to 502 even though default
   ;;; 21008 = 1007  = Set index to point to where to view data
   ;;; 21009 = 23000 = Start remapped area at 23000 for 160 regs.
   ;;; 21008 = 0     = Always set the index back to 0 before begin
   ;;; 21006 = 100   = Set scan poll time to 100 ms./block read,
   ;;;                 min is 50ms.  This also initiates polling.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   store 160 to reg_21005
   store 12 to reg_21000
   store 40 to reg_21001
   store 53 to reg_21002
   store 168 to reg_21003
   store 1 to reg_21004
   store 1003 to reg_21008
   store 2 to reg_21009
   store 1004 to reg_21008
   store 502 to reg_21009
   store 1007 to reg_21008
   store 23000 to reg_21009
   store 0 to reg_21008
   store 100 to reg_21006
   goto Next

[2] Wait_For_Online
   ;;; Once Modbus Master starts to poll we must wait until
   ;;; it is online before proceeding.
   ---------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ---------------------------------------------------------------------------
   if reg_21007=1 goto Modbus_Online
   delay 500 ms goto Wait_For_Online

[3] Modbus_Online
   ;;; It is OK to read and process data now since Modbus
   ;;; is online to the device.  If you wish to monitor another
```

```
;;; device other than Unit ID 1, then change the index register
;;; 21008 to 1005 and write the desired Unit ID to register
;;; 21009, then set 21008 back to 0 and monitor 21007 for
;;; a 1 for online state, once again.  Results will appear
;;; in the 23000 block.
-------------------------------------------------------------------------
<NO CHANGE IN DIGITAL OUTPUTS>
-------------------------------------------------------------------------
delay 1000 ms goto Modbus_Online
```

When Reg_21007 is equal to a 1, then the connection is active and you may interact with the remote device. If a 3 had been written to 1003, then Modbus Master RTU Serial on COM1 would be used.

Modbus RTU Serial Master Sample Program

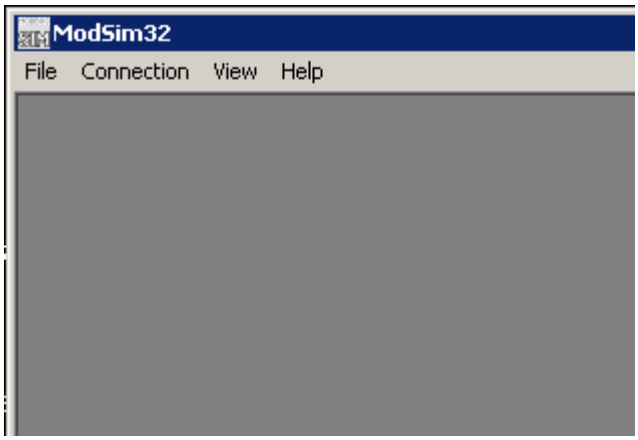
IP address - 12.40.53.168 (can be set to any value other than –1)
Device ID - 1
Number of sequential registers to read - 160
Scan time - 100 ms. (set last to initiate)
Starting Register - 1
Serial Port - COM1
Remap registers to consecutive block beginning at registers 23000.
This is the first setup so use 21000, next would be 21010… 21020, etc…

```
[1] Initialize_ModbusMaster
   ;;; This program is used to initialize the COM1 port
   ;;; for Modbus RTU Serial Master operation.  A single
   ;;; device is polled using device ID 1 and 160 registers
   ;;; are read and mapped into the 23000 block.  Therefore
   ;;; registers 23000 - 23159 are used, with 23000 referencing
   ;;; Modbus Register #1.  Make sure your Modbus device has
   ;;; at least 160 consecutive registers starting at '1'
   ;;; otherwise Modbus Exceptions will occur.
   ;;; Begin by doing the following:
   ;;; 21005 = Maximum number of registers to read (160)
   ;;; 21000 - 21003 = Any value, required to unlock register
   ;;;                 group, on Modbus TCP this is the IP
   ;;;                 address for a connection.
   ;;; 21004 = Modbus start register (1)
   ;;; 21008 = 1003  = Set index to point to protocol register
   ;;; 21009 = 3  = Set protocol to Modbus RTU Serial (4 for ASCII Serial)
   ;;; 21008 = 1004  = Set serial port to use, default is 1
   ;;; 21009 = 1     = For demo set port to 1 even though default
   ;;; 21008 = 1007  = Set index to point to where to view data
   ;;; 21009 = 23000 = Start remapped area at 23000 for 160 regs.
   ;;; 21008 = 0     = Always set the index back to 0 before begin
   ;;; 21006 = 100   = Set scan poll time to 100 ms./block read,
   ;;;                 min is 10ms.  This also initiates polling.
   -------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   -------------------------------------------------------------------------
   store 160 to reg_21005
   store 10 to reg_21000
```

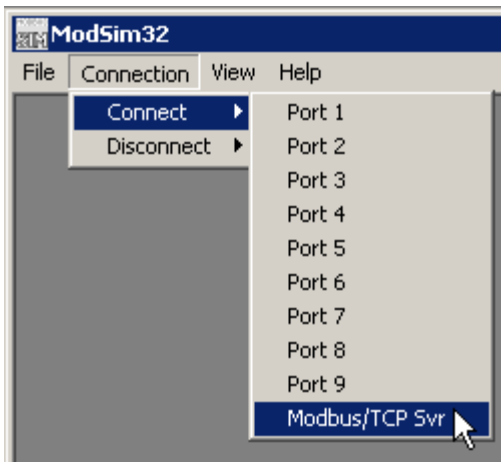
```
   store 10 to reg_21001
   store 10 to reg_21002
   store 10 to reg_21003
   store 1 to reg_21004
   store 1003 to reg_21008
   store 3 to reg_21009
   store 1004 to reg_21008
   store 1 to reg_21009
   store 1007 to reg_21008
   store 23000 to reg_21009
   store 0 to reg_21008
   store 100 to reg_21006
   goto Next

[2] Wait_For_Online
   ;;; Once Modbus Master starts to poll we must wait until
   ;;; it is online before proceeding.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   if reg_21007=1 goto Modbus_Online
   delay 500 ms goto Wait_For_Online

[3] Modbus_Online
   ;;; It is OK to read and process data now since Modbus
   ;;; is online to the device.  If you wish to monitor another
   ;;; device other than Unit ID 1, then change the index register
   ;;; 21008 to 1005 and write the desired Unit ID to register
   ;;; 21009, then set 21008 back to 0 and monitor 21007 for
   ;;; a 1 for online state, once again.  Results will appear
   ;;; in the 23000 block.
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   delay 1000 ms goto Modbus_Online
```

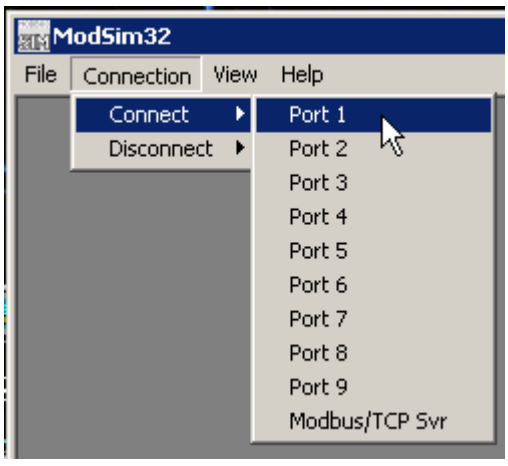
Testing with Win-Tech's ModSim32

As a demonstration of the functionality of the controller Modbus Master interface, this section details the interface of Win-Tech's ModSim32 software and how it applies with regard to our product. It is assumed that the controller Modbus TCP Master or Serial is set up to point to the PC and is attempting a connection. As mentioned before, we only support the Holding Register interface. Upon invoking ModSim32 the screen below will appear.

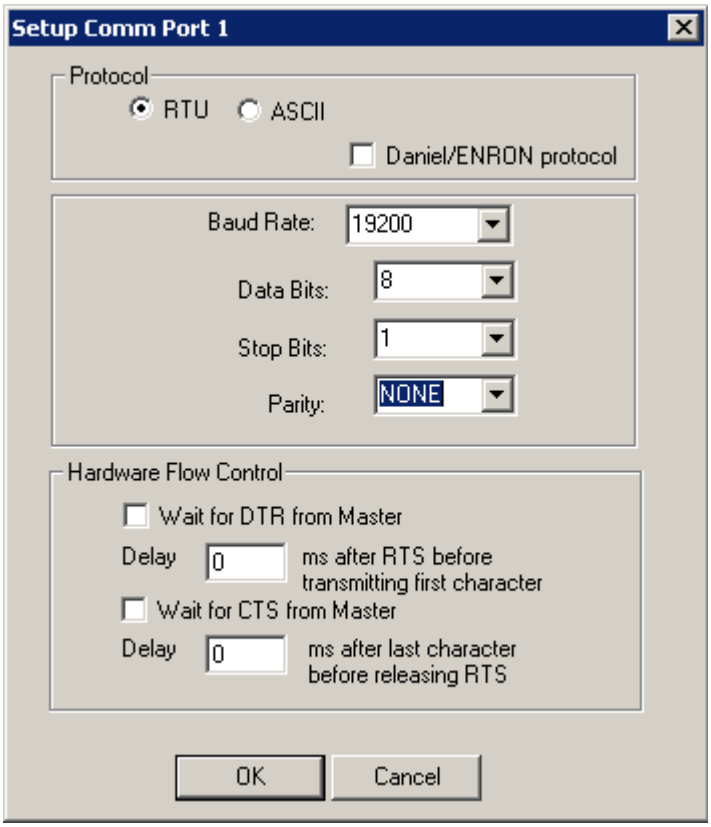


In order to activate the Modbus slave, you must select the Connection menu item and the method of the connection, Modbus/TCP Svr for network or the appropriate Port # for a serial port.

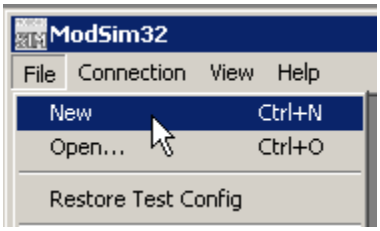


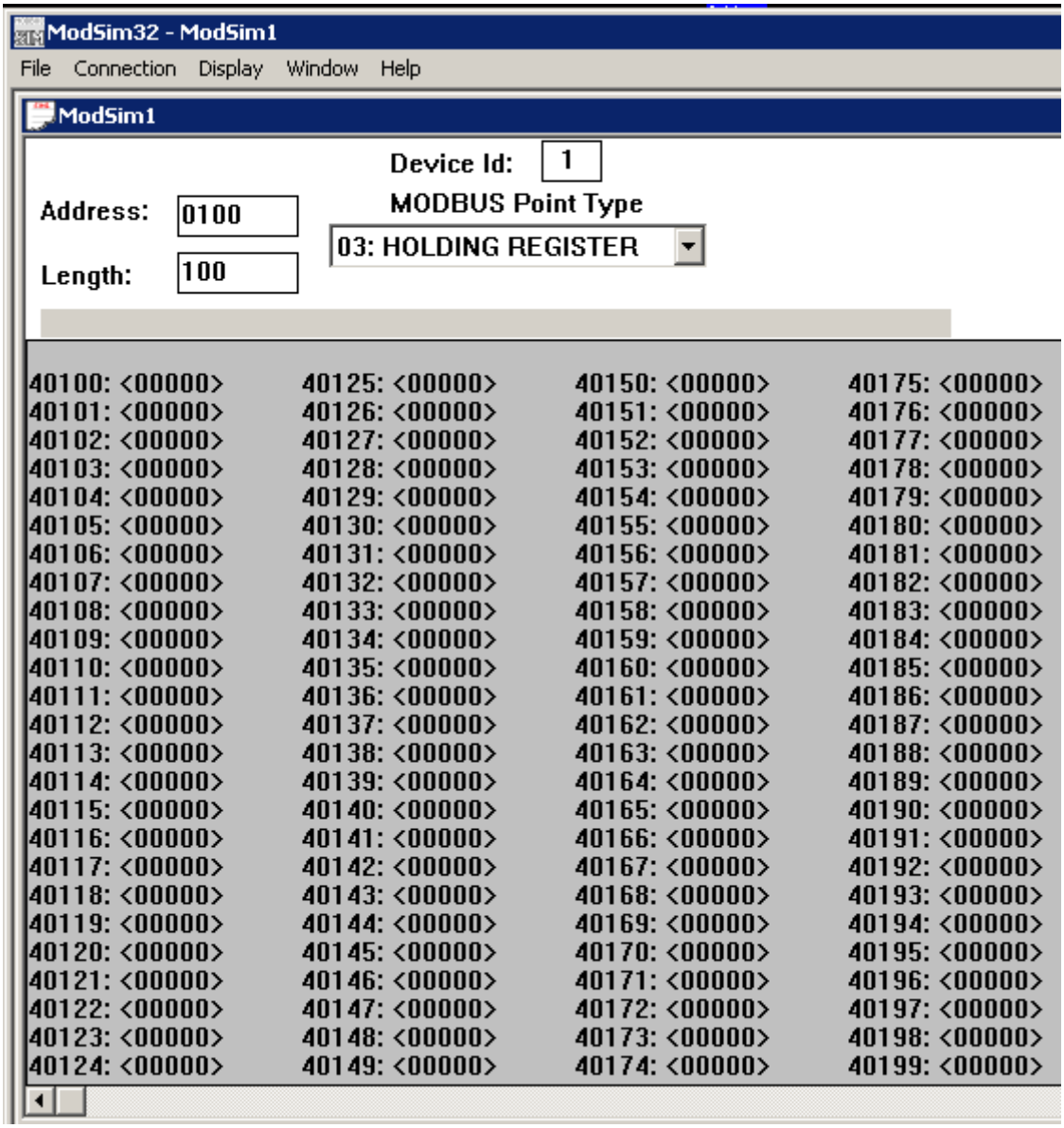


If Serial, select RTU or ASCII and set the baud rate, stop bits, and parity appropriately. Default for the 5200 is 19.2K baud, 8 data bits, 1 stop bit, no parity. However, this is not the default for ModSim and must be changed as shown below:

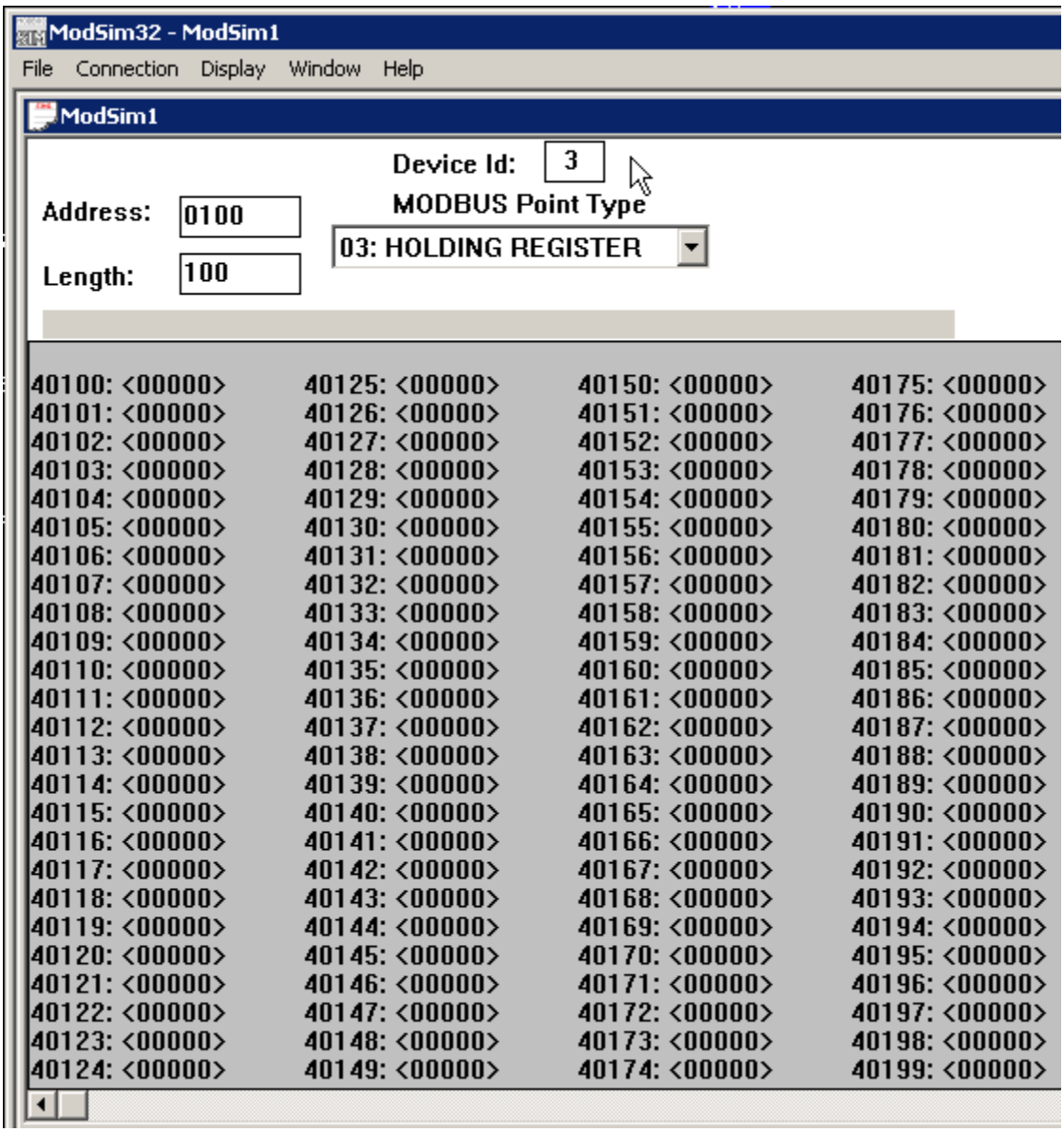


Next devices must be created to listen to the requests. This is done using menu selection: File-> New:

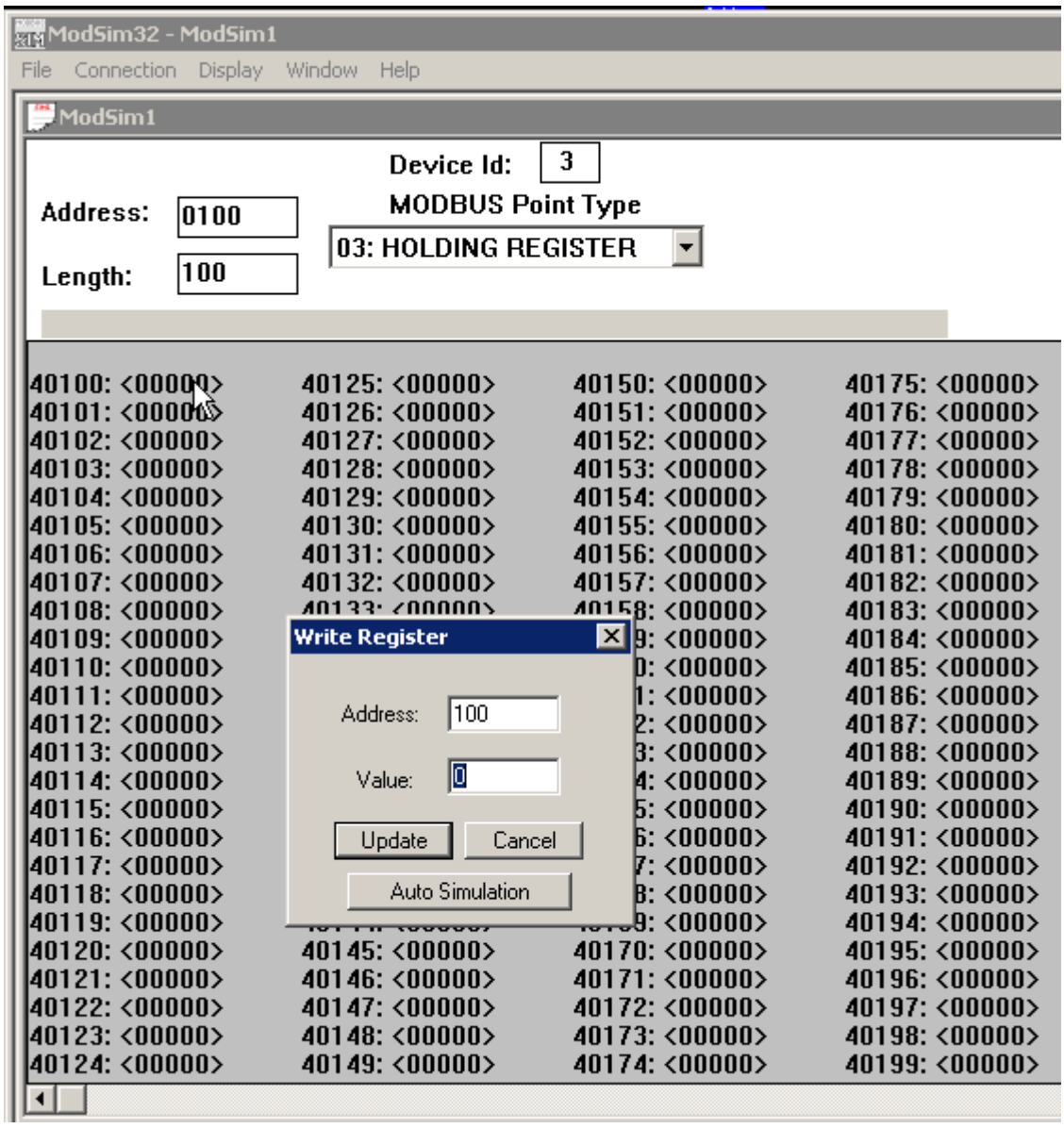




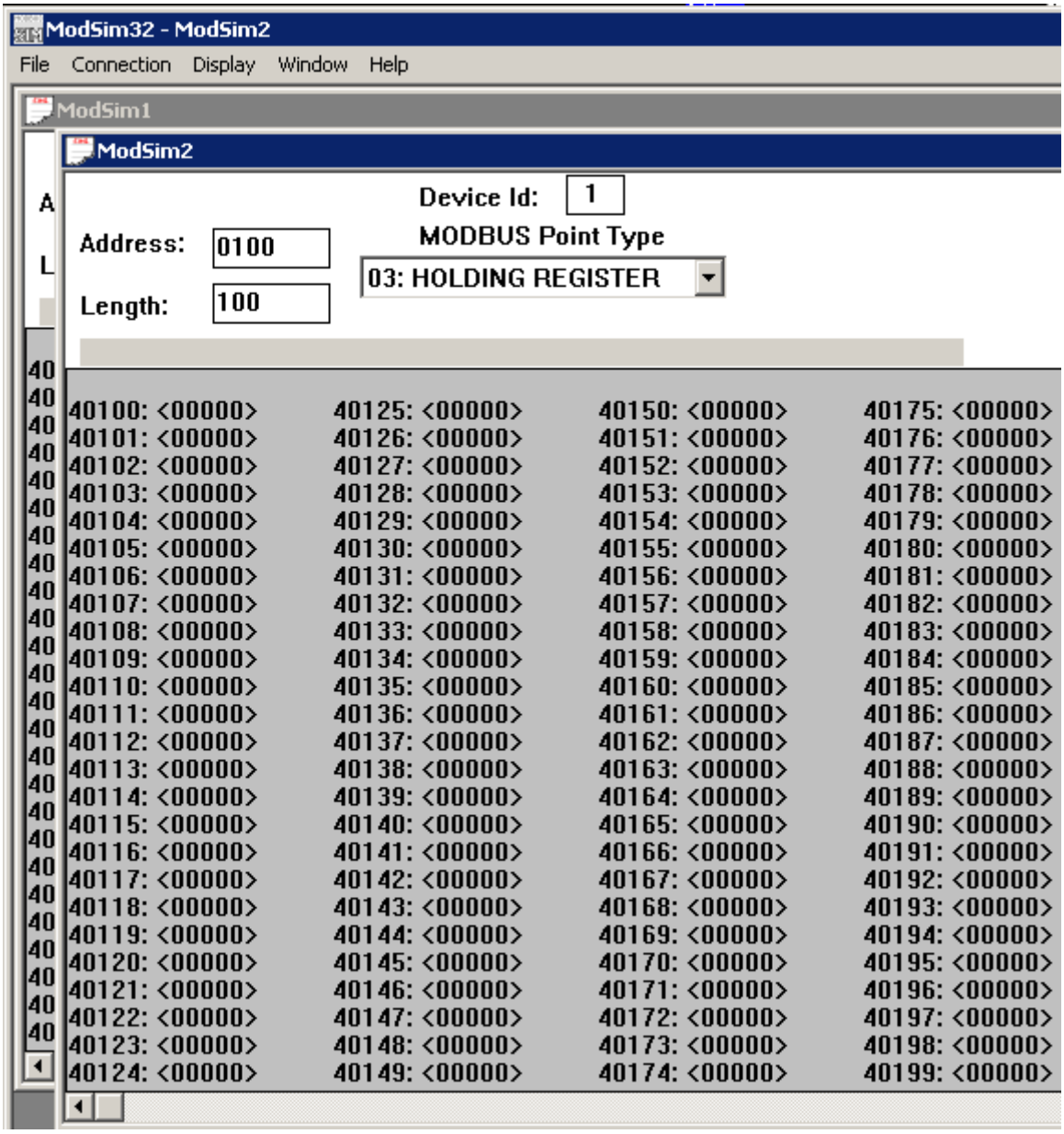
In order to access this device, the controller must have its Device ID set to 1 (the default) and the Starting Address set to 100. If not set correctly, an exception status will be returned upon connection and 21XX7 register will contain a -1. If we want to set the Device ID to a 3, as in our example, modify as below:



Note that the 'Address' field is set to 100, but the display screen starts at 40100. This is Modbus nomenclature. To modify a device Holding Register contents, simply double click on the address and enter the new value in the dialog that appears:



Above shows the modification of address 100. Additional devices can also be created by once again selecting File->New. This allows for the testing of multiple Modbus Slave devices at the same time:



Above shows multiple devices enabled. If there are further questions about the use of ModSim32 simply select the Help menu item and a manual will appear.

CHAPTER 8

SNTP Simple Network Time Protocol

The 5200 controller supports the Simple Network Time Protocol (SNTP) as a client connecting to a server. This protocol provides a means to synchronize a computer system clock to that of the world clock, via the internet. Government agencies provide this service for computers to query the current atomic clock time and adjust their clocks appropriately. For more detailed information reference www.time.gov and www.boulder.nist.gov/timefreq/service/its.htm.

The time returned is based on Coordinated Universal Time (UTC), which is Greenwich Mean Time (GMT). As such, there is no adjustment for daylight savings time or time zones, that must be done locally. To avoid daylight savings time problems it is recommended that you base the controller time on GMT (default) but provisions have been provided to automatically set the clock based on the time zone you are in, using an offset from GMT. Refer to the RTC Tab for further details.

Use of SNTP is not a requirement but typically real time clocks can be expected to drift up to 30 seconds per week. The controller may drift up to 12 seconds per week, depending on the tolerance of crystals, components, etc. Synchronization allows its real time clock to be automatically set with regards to date, year, day of week, and time.

SNTP Register Configuration

SNTP may be configured using either a direct register interface or by individual registers. By default the controller will use the IP address of 192.43.244.18, port 123. The default update frequency is once/day and the default time zone used for clock reset is GMT. These may be changed by modifying the following registers:

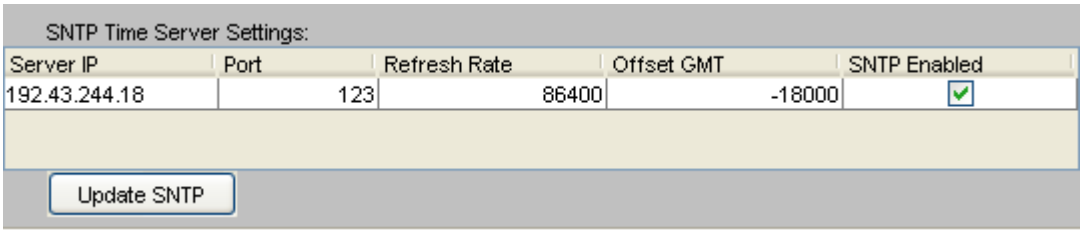
20025	**First Octet IP Address Register (Most Significant) for SNTP Server - R/W** This is the first octet of the IP address (XXX.000.000.000) that is used to connect to the SNTP server. Default is 192.
20026	**Second Octet IP Address Register for SNTP Server - R/W** This is the second octet of the IP address (000.XXX.000.000) that is used to

	connect to the SNTP server. Default is 43.
20027	**Third Octet IP Address Register for SNTP Server - R/W** This is the third octet of the IP address (000.000.XXX.000) that is used to connect to the SNTP server. Default is 244.
20028	**Fourth Octet IP Address Register for SNTP Server - R/W** This is the fourth octet of the IP address (000.000.000.XXX) that is used to connect to the SNTP server. Default is 18. The unit must be reset for a new IP address to take effect.
20041	**SNTP Server Port to connect to - R/W** This register contains the TCP port that should be used for SNTP connections. Default is 123.
20042	**SNTP Update Time - R/W** This register contains the number of seconds before the next synchronization request with the SNTP server. For example 3600 would be an hour, 86400 would be 24 hours. Default is 86400. When a change in time is made to this value it typically takes about 1 minute before the new value will take effect. Power cycling of the controller is not required.
20043	**SNTP Offset from GMT - R/W** This register contains the number of seconds to add or subtract from GMT. The default is 0, which means to set the clock to GMT. –14400 would be the value used for Eastern Standard Time during daylight savings time. Note that the value is both positive and negative.

A 1 must be written to register 20096 whenever the above changes are made in order to store those changes to non-volatile storage. Also, to disable SNTP, simply set the IP address of the SNTP Host to 0.0.0.0.

SNTP WebMON Configuration

WebMON provides a more direct method of updating the SNTP configuration. As with registers, the SNTP Time Server Settings consists of a number of data entry fields, each with their own special functionality:



- Server IP

- Port
- Refresh Rate
- Offset GMT
- SNTP Enabled

By default the controller will use the IP address of 192.43.244.18, port 123. Updates will be performed once/day and the clock is set to GMT.

Server IP

The "Server IP" address designates the host which will provide the time service for the controller. By default the address is 192.43.244.18. Data is entered using the "dot" notation. Entering an IP address of 0.0.0.0 will disable SNTP requests.

Port

The "Port" is the TCP/IP port that the Time Server will be listening on for time requests. Typically this is port 123, and is the factory default.

Refresh Rate

The "Refresh Rate" is the number of seconds before the next synchronization request with the SNTP server. For example 3600 would be an hour, 86400 would be 24 hours (default). When a change in time is made to this value it typically takes about 1 minute before the new value will take effect. Power cycling of the controller is not required.

Offset GMT

"Offset GMT" contains the number of seconds to add or subtract from GMT once the time is received from the server. The default is 0, which means to set the clock to GMT. –18000 (-5 hours) would be the value used for Eastern Standard Time during daylight savings time, -14400 (4 hours) when not. Note that the value is both positive and negative.

SNTP Enabled

If the check box is checked SNTP requests will be enabled and done in the background based upon the above parameters. When deselected the IP address will be forced to 0.0.0.0. If the time service is not being used it is best to ensure this box is not checked, thereby conserving CPU resources.

Blank

CHAPTER 9

SMTP

Simple Mail Transfer Protocol (SMTP), documented in RFC 821, is the Internet's standard host-to-host mail transport protocol which typically operates over TCP port 25. The controller is capable of sending formatted email, using SMTP, under the control of a Quickstep program or by remote communications accessing a data register. Messages may be created either within an ASCII text editor or using WebMON 2.0 (“*WebMON 2.0 User’s Guide, 951-520012).*

For email to operate properly the controller must have an email account on the email server. This will consist of a user account and password. The same account can be shared by multiple controllers.

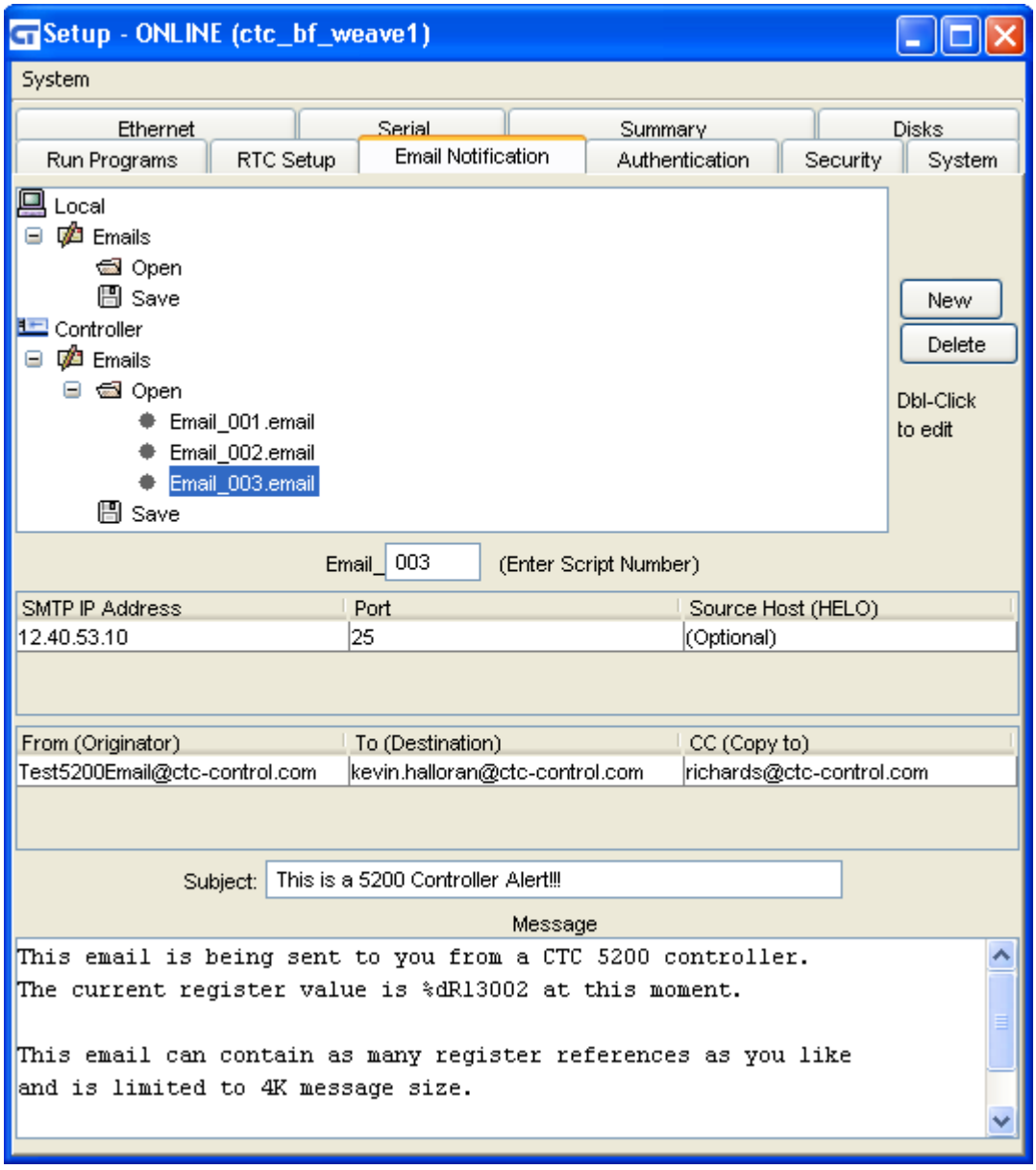
Register Access

Text files created in a specific format and naming convention are stored on the flash disk /_system/Emails subdirectory. Files are stored with a name of “Email_###.email” where ‘###’ references the value which would be written to the SMTP Send Register (12317), to request transmission. For example, a file name of “Email_001.email” would be sent if a ‘1’ was written to the SMTP Send Register. Register 12318 is the SMTP Status Register. The status contents are defined as follows, after a write to the SMTP Send Register:

STATUS	DESCRIPTION
0	Processing
-1	Undefined
0x80000100	General Error, out of memory
0x80000900	Error, parameter error, aborted
0x80001400	Requested operation has failed.
0x80002100	Error, can not connect to host.

Creating Emails using WebMON

The "Email Notification" tab can be used to automatically create, edit, and delete these files.

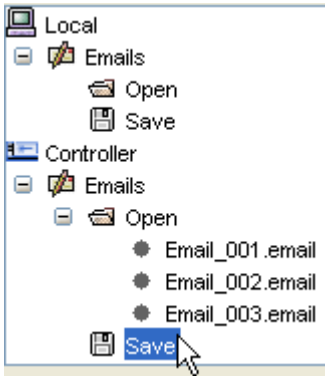


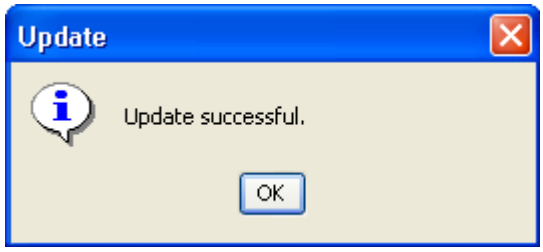
Tree View, Local/Controller

At the top of the Email Notification tab is a tree list. This list is used to access formatted email files either locally or stored on a controller disk. Local->Email references the local disk drive of the computer running WebMON. Selecting Local->Email->Open will cause a dialog box to open and the selection of any email file for editing purposes.

Selecting Local->Email->Save will cause a dialog box to open and an email that is within the form at the bottom, to be saved to the computer's hard disk.

Files that exist within the controller's disk may be individually viewed and selected from the Controller->Email->Open tree node. Each file represents an individual node. To save a file that is created using the email template (form below the tree view), simply double click the Controller->Email->Save node. The file will be saved and named using the Script Number defined within the email template, Email###.email.





Creating/Editing New Email Template

To create a new email, simply select the "New" button to the right of the Email Notification Tree view. This will cause all existing information to be removed from the template form and defaults to be entered. Alternatively an existing email could be loaded and modified as desired, then saved.

A number of data entry fields are available to define the email to be sent by the controller. The top most field, immediately below the tree view, allows the entry of a numeric from 1 to 999. This will become the file sequence number used within the email file name, Email###. Leading 0's will automatically be provided.

The next set of data entry fields is a table whose row defines the SMTP server that is to be used for sending email. Each email may use the same and/or different SMTP servers. Make sure you are authorized for using the server and you are not attempting to relay. Relaying is restricted and occurs when you try to copy an email to someone that is not authorized, outside your domain. For example if the domain was ctc-control.com, you would not be able to send a copy of the email to hotmail.com, using POP3. Mail Servers can be configured to allow for exceptions, if desired. A typical way around this would be to use a distribution list within your mail server, that in turn sends outside the domain.

Available data entry parameters for the first table are:

SMTP IP Address	Port	Source Host (HELO)
12.40.53.10	25	(Optional)

SMTP Server

This is the server IP address of the server handling your email account. It is typically within the same domain as your 'From:' email address. The "dot" notation format is used.

Port

The standard SMTP port used is 25; it may be changed here if desired. This is the port the SMTP server will be listening on for connection requests.

HELO

This is an optional field which can be used to report your domain within the email. It is required by some hosts. For example the domain of www.ctc-control.com would be ctc-control.com.

The second table is used to define who the email is from (FROM:), who it is to be sent to (TO:), and who it is to be copied to (COPY:). Only one address is supported per entry. If larger distributions are required it is suggested that a Distribution List be created on the Email server.

The required format, of each email address, is person@domain.com. Enter each as needed. Note the COPY field is optional:

From (Originator)	To (Destination)	CC (Copy to)
Test5200Email@ctc-control.com	kevin.halloran@ctc-control.com	richards@ctc-control.com

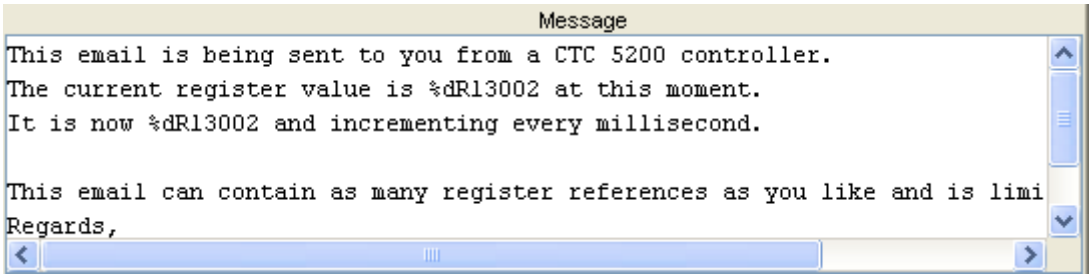
Subject

The Subject line will appear as the summary in an email message. Enter any desired text:

Subject: This is a 5200 Controller Alert!!!

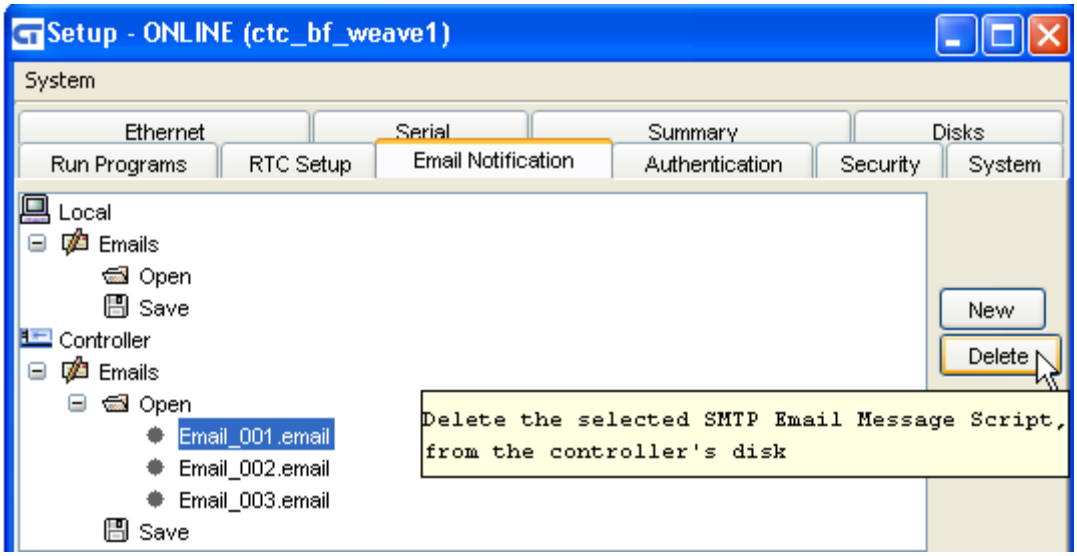
Message

The Message area can contain up to 4K bytes of data. Messages may be any mix of normal text characters and references to Controller registers. Registers are references using "C" style printf directives. For example, to reference the 13002 register and have its contents placed in a message a %dR13002 would be used, optionally %05dR13002 would force at least 5 characters wide with leading 0's as filler. In printf notation %d is decimal, %x is lower case hex, and %X is upper case HEX. These are the only acceptable printf syntaxes currently supported in email messages. Below shows an example of a message which would include the current value of the 13002 register, when sent:



Deleting Email Template

Deleting an email is only supported from a controller disk. To delete a file use the Controller->Email->Open tree view to list the available files. Highlight the one desired and select the "Delete" button. The file will be deleted and the tree updated.



Creating Emails using ASCII Text Editor

The text used to create emails, to be sent by a 5200 controller, requires a specific format. That format includes various 'section headers', used to define the necessary parameters. It is recommended that WebMON be used for the creation of all emails although this section is included for those who desire a further understanding of the format.

There are two section headers. The first, known as [SMTP], must appear in the beginning of the file and is used to define all the specific details of the email message, such as destination, mail server, etc. No spaces are allowed except within the email message itself, designated by the [SMTP_MESSAGE] section header. It is best to use a sample email as an example:

```
# This is a comment
[SMTP]
```

IP=12.40.53.10
PORT=25
HELO=
TO=kevin@ctc-control.com
FROM=Test5200@ctc-control.com
CC=
BCC=
SUBJECT=Test email message
[SMTP_MESSAGE]
Enter Email Message to send, %05dR13002 references register...

- The Pound sign may appear as the first character in any line. All following text on that line will be ignored. It is used to place comments within your email definition document.

[SMTP] – Section header. Required to be on the first line of the file.

IP= This is the SMTP Server IP address of the server handling your email account. It is typically within the same domain as the 'From:' email address. The "dot" notation format is used. No spaces are allowed before or after the '=' sign.

PORT= The standard SMTP port used is 25; it may be changed here if desired. This is the port the SMTP server will be listening on for connection requests.

HELO= Optional field which can be used to report your domain within the email. It is required by some hosts. For example the domain of www.ctc-control.com would be ctc-control.com.

TO= Required field, defining the destination. Only one may be listed per 'TO' although multiple 'TO' fields are allowed.

FROM= Required field, the email address that represents the controller and that can be replied to. This account should exist on the SMTP server, if not then relaying must be enabled.

CC= Optional field, defining the destination to copy the email to. Only one may be listed per 'CC' although multiple 'CC' fields are allowed.

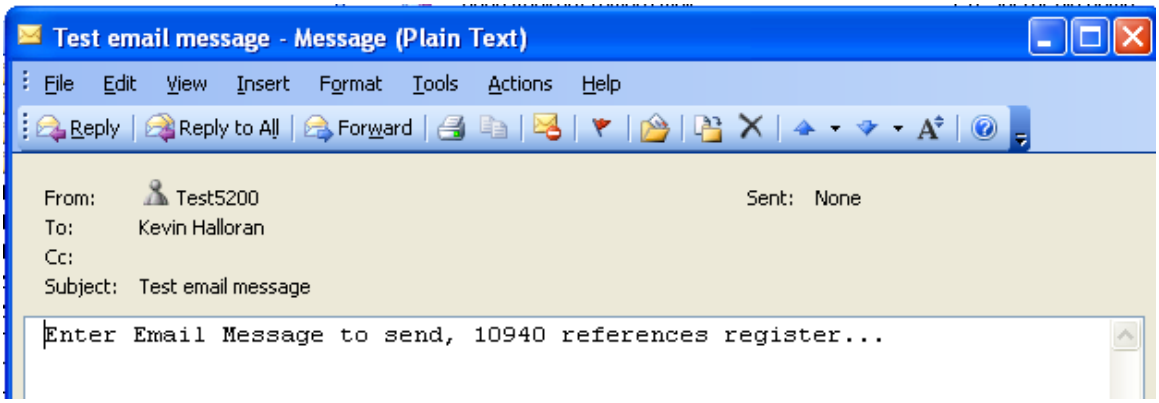
BCC= Optional field, defining the destination to copy the email to. Only one may be listed per 'BCC' although multiple 'BCC' fields are allowed. Typically BCC fields are hidden and will not be displayed when the email is received.

SUBJECT= Required field, specifies the email subject, generally a short summary. Spaces are allowed within the text.

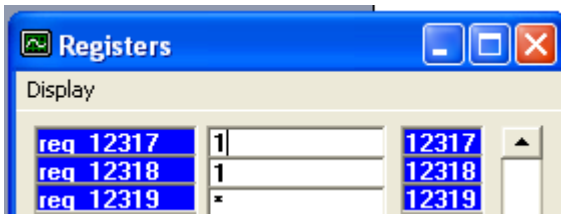
[SMTP_MESSAGE] – Section header. Required prior to the start of the email text message. All following text is assumed to be part of the email. Reference the "Creating Emails using WebMON" section for details on the 'Message' area.

 Ensure that the enter key is entered on the last item in the message, returning the cursor to the far left hand side of the message.

Email message sent and received when the sample email file was stored to Email_001.email within the /_system/Emails sub-directory, and a 1 was written to the SMTP Send Register 12317:



After communications the SMTP Send Register displays the email message number sent along with the results in the SMTP Status Register, 12318. 12318 changed to 0 after the initial write of a 1 to 12317, ending with a 1 after successful transmission:



Monitored with CTCMON

Notice that the %05dR13002 was replaced by the actual register value in the controller at the time the email was composed for transmission.

Blank

CHAPTER

10

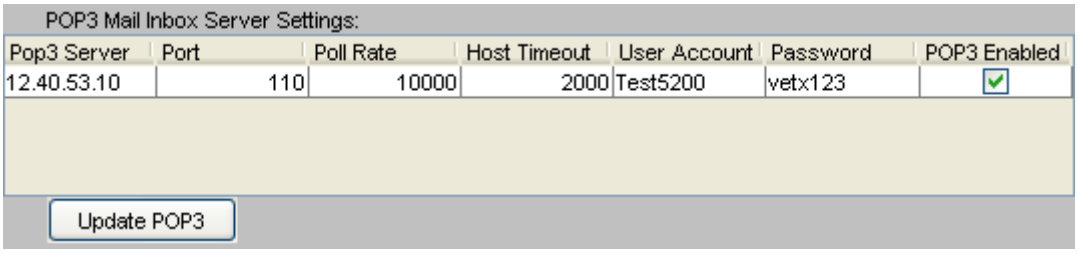
POP3

Post Office Protocol, Version 3 is a set of standardized rules (protocol) by which a client machine can retrieve electronic mail from a mail server (POP server). The server holds the email until the user can retrieve it. POP3 only provides for receiving email, not sending it. SMTP is used for transmission.

For proper operation controllers should be assigned their own email account. You may not share an email account with a controller since each controller will read and delete each email, as it is read and processed.

Mail Inbox Server Configuration

The POP3 Email Server configuration can only be setup using WebMON via the 'Ethernet' Setup tab. It consists of a number of data entry fields, each with their own special functionality:



- POP3 Server
- Port
- Poll Rate
- Host Timeout
- User Name
- Password
- POP3 Enabled

POP3 Server

The "POP3 Server" IP address designates the host which will provide the POP3 mailbox account for the controller. This must be the servers IP address, entered in "dot" notation.

Port

The "Port" is the TCP/IP port that the POP3 Server will be listening on for mail requests. Typically this is port 110, and is the factory default.

Poll Rate

The "Poll Rate" is the time, in milliseconds, that the controller will wait until it checks for available email, within its mailbox. All available email will be read and deleted as processed, in a sequential order. After processing this time delay will occur until the next processing sequence. 10000 milliseconds (10 seconds is the default interval).

Host Timeout

The "Host Timeout" is the time, in milliseconds, that the controller will try to contact its POP3 server and wait for responses for mail requests. It is considered the error timeout. After this period of time the controller will stop trying to contact the server and wait the next poll rate interval before trying again. The default timeout period is 2000 milliseconds (2 seconds).

User Name

The "User Name" is the name needed to log into the mailbox. This is typically the mailbox name but could be set different by the POP3 server. It is limited to 30 characters.

Password

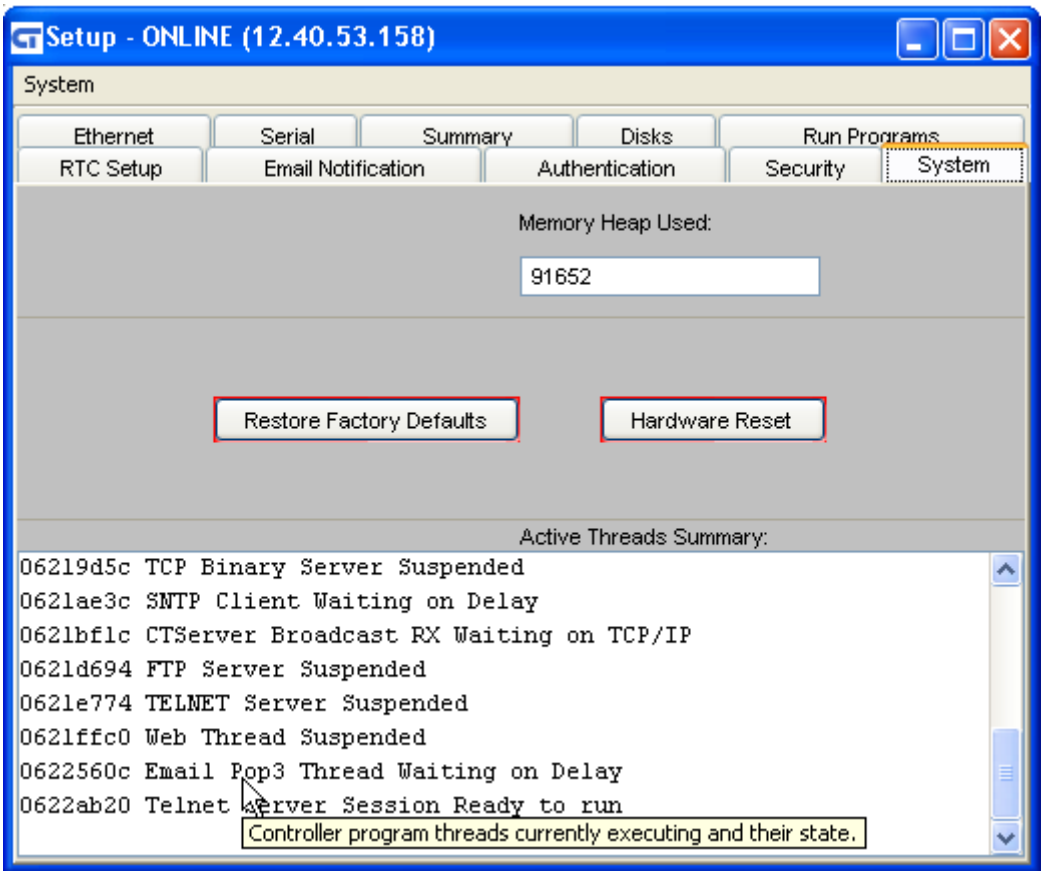
The "Password" is the password required, along with the "User Name" to log into the mailbox being supplied by the POP3 server. It is limited to 30 characters.

POP3 Enabled

A check box is available to enable the POP3 functionality, when checked POP3 is active. Once all changes have been made to the above parameters select the "Update POP3" button to make the changes current in the controller.

 A Hardware reset must be generated whenever the POP3 parameters are changed for them to become active.

To verify that the controller is monitoring a POP3 account the WebMON Setup System tab can be viewed and the execution thread verified:



Email Formatting

Once the 5200 controller email server is configured, enabled, and system restarted, the controller will continually poll the email server for mail. As each mail message is found it will be downloaded, processed, and deleted from the 'inbox'. Processing consists of scanning the email whose messages contain special Section Header character strings and script commands for execution.

Section Headers

The Section Headers are defined as follows exist within the message body of an email:

[CTC_EMAIL_START] – Script commands follow as defined within the *Model 5200 Script Language Guide, 951-520003*. This section header may begin and end as often as required as long as there is a matching [CTC_EMAIL_END], for each. Note that a # sign at the beginning of a line represents a comment.

[CTC_EMAIL_START_ATTACH_ORIGINAL] –Exactly the same as [CTC_EMAIL_START] except that a copy of the original email is appended to the end of the reply email.

[CTC_EMAIL_END] – Script commands end and following should be ignored.

Example Email message text:

```
This line is ignored and can be any information desired in the email.
The next line will signify the start of script processing.
[CTC_EMAIL_START_ATTACH_ORIGINAL]
# This is a comment.
# Request a copy of this email be attached to the original, not needed
# but useful to know what we sent.  Regardless a copy of each of
# these commands and the reply is always sent back as a reply.
# [CTC_EMAIL_START] will not cause original to be attached.

# Let's assume we received an alarm condition via pager or email
# so lets clear it.  Possibly register 1 is used as a flag by the
# program.  Also keep these lines less than 72 characters when
# using Microsoft Exchange as it typically auto-line wraps and
# you will end up with a bad command.
1 = 0
# Now lets get all the version information just to make sure things
# are OK.
get versions
# Restart the controller given to clear the alarm
set restart
# We are all done now so return to normal email text
[CTC_EMAIL_END]

This is just normal email text.  We could issue another command
block if desired following this text.
```

 Emails must be sent as ASCII Plain Text messages, not HTML formatted. Also only Quoted-Printable data encoding is supported within the message body, reference RFC1341.

 Mail Messages should be limited to 4096 bytes, a 9K buffer is available assuming a reply with the original message attached.

 Ensure that the enter key is entered on the last item in the message, returning the cursor to the far left hand side of the message.

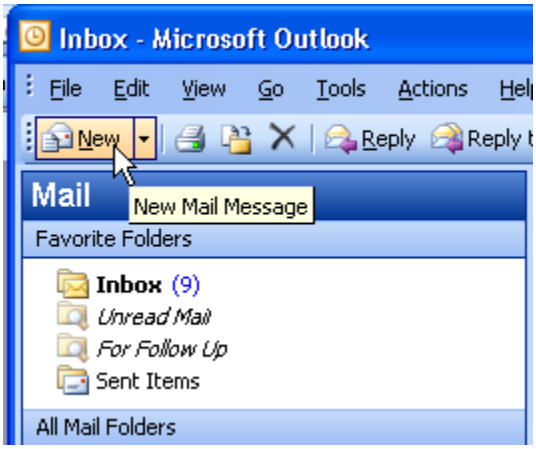
ASCII Text Emails

All emails sent to the controller **MUST** be sent as ASCII Plain Text messages, not HTML formatted. Many email programs allow the selection of HTML, Rich Text, and Plain Text. Plain Text is equivalent to ASCII text messages.

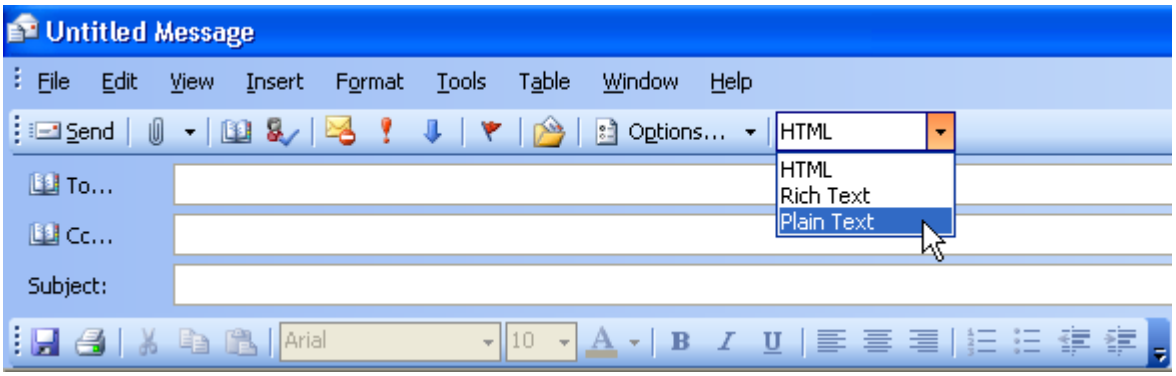
There are a number of ways to make this selection. Using Microsoft Outlook 2003 as an example you may set this as the default to always use or select it on an individual email basis.

Microsoft Outlook Plain Text, Individual Basis

On an individual email basis it may be selected after you open a window for composing a 'New' email:



A window will appear to compose the email, note the pull down box and ensure it is selected to Plain Text.

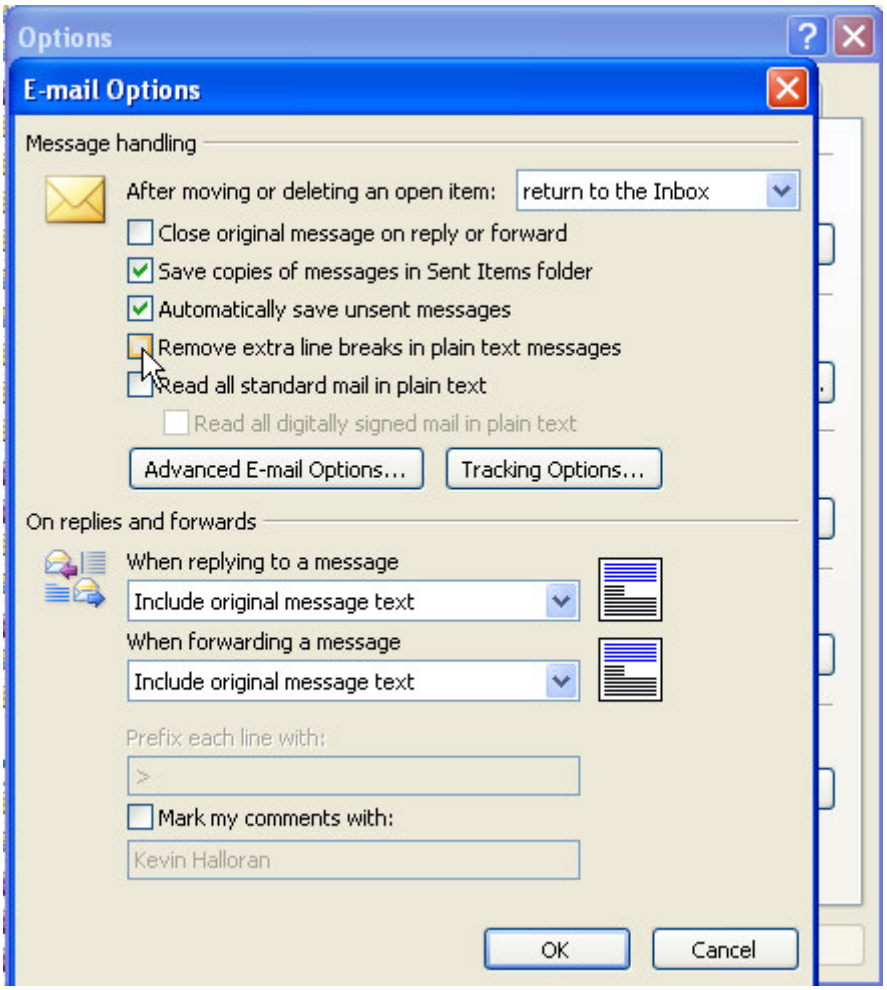


Some email services, such as MSN Hotmail, always send messages in Plain Text format.

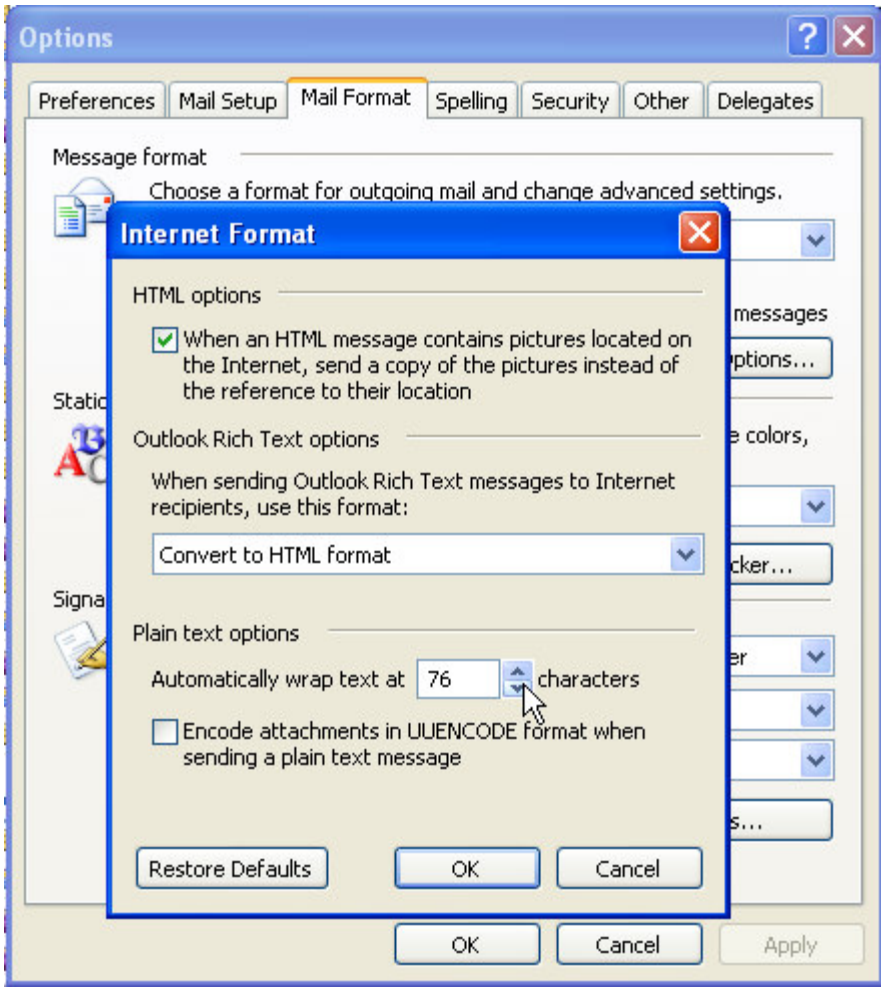
Note that there are a couple of things to be aware of, especially in Outlook 2002. First is that text sent may automatically have line wrapping done. For example Outlook 2002 does it at 64 characters, exchange 72 and Outlook 2003 has a user settable option. The text will appear normal within your Outlook editor but is converted prior to receipt by the

controller. Also when receiving a reply Outlook will remove some of the line feeds making some of your lines appear as one. To remedy this for Plain Text messages there are two option screens under Outlook->Tools->Options, then 'Email Options' button:

Preventing removal of extra line breaks:



Increase the line length before auto-wrapping text, referencing Outlook->Tools->Options->Mail Format Tab, then 'Internet Format' button:



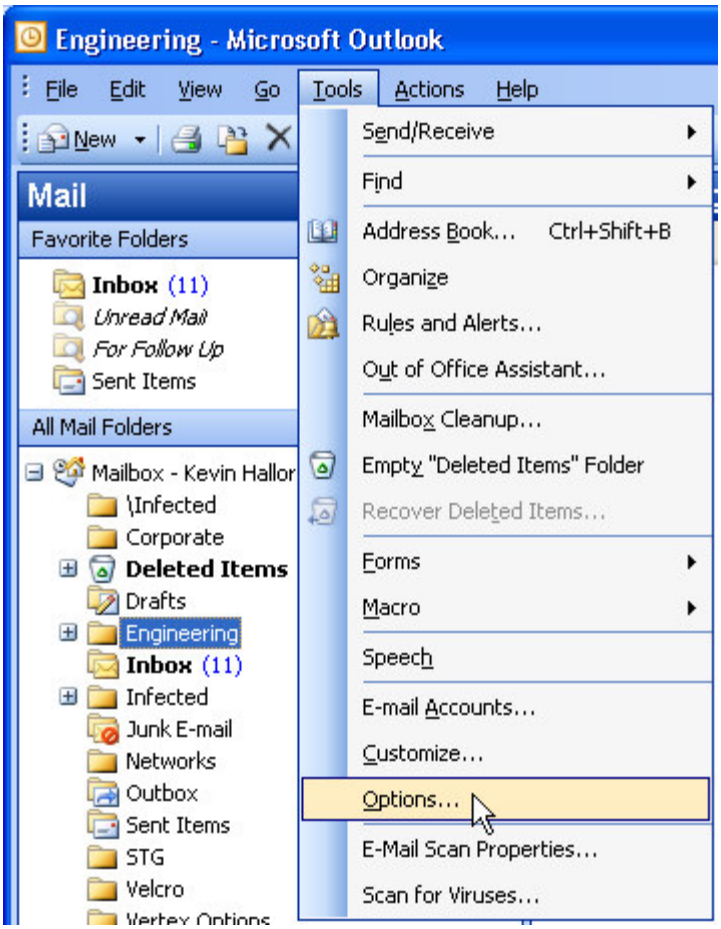
Some Microsoft Knowledgebase Articles worth referencing are 287816 and 327573:

http://support.microsoft.com/default.aspx?scid=kb;EN-US;327573

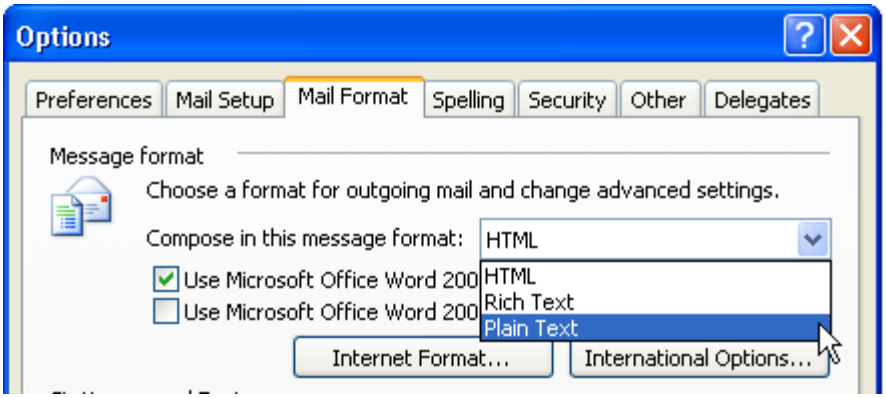
http://support.microsoft.com/default.aspx?scid=kb%3BEN-US%3Bq287816

Microsoft Outlook Plain Text, Default for All

Configuring Microsoft Outlook to always default to Plain Text is done via the Tools menu:



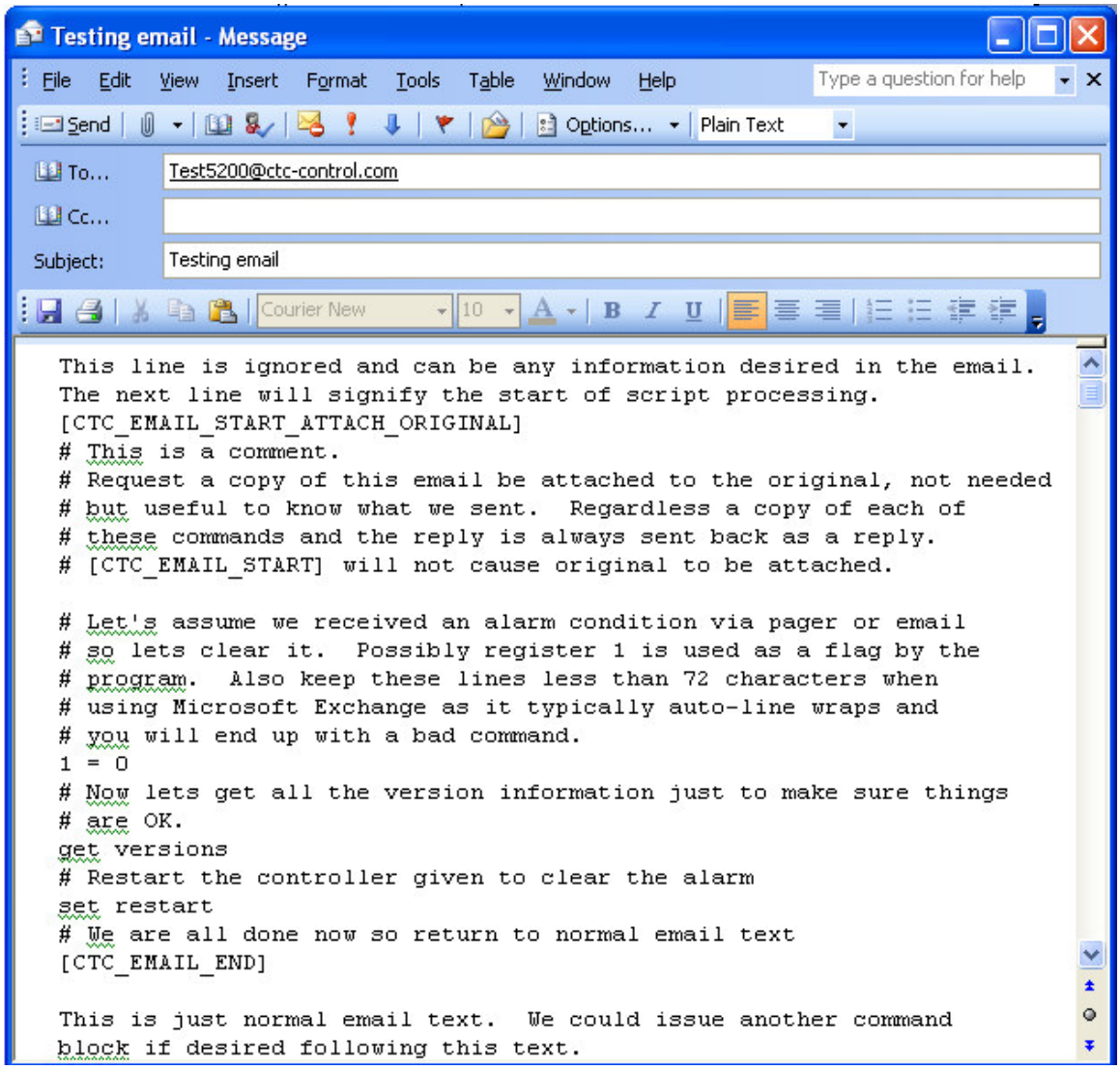
Select the 'Mail Format' tab and set the 'Compose this message format' pull down to 'Plain Text'.



When finished, click OK, the default for all messages is now Plain Text.

Sample Email and Response

The email below was detailed previously and is now shown ready for sending within an Outlook Message box:



Upon clicking 'Send' the email will be sent to mail server where the 'Test5200' account resides. Based on the poll rate the controller will then read the email, process the commands and return a reply since the [CTC_EMAIL_ATTACH_ORIGINAL] parameter is listed. The response received several seconds later is:

```
BlueFusion> 1 = 0
1 = 0

BlueFusion> get versions
```

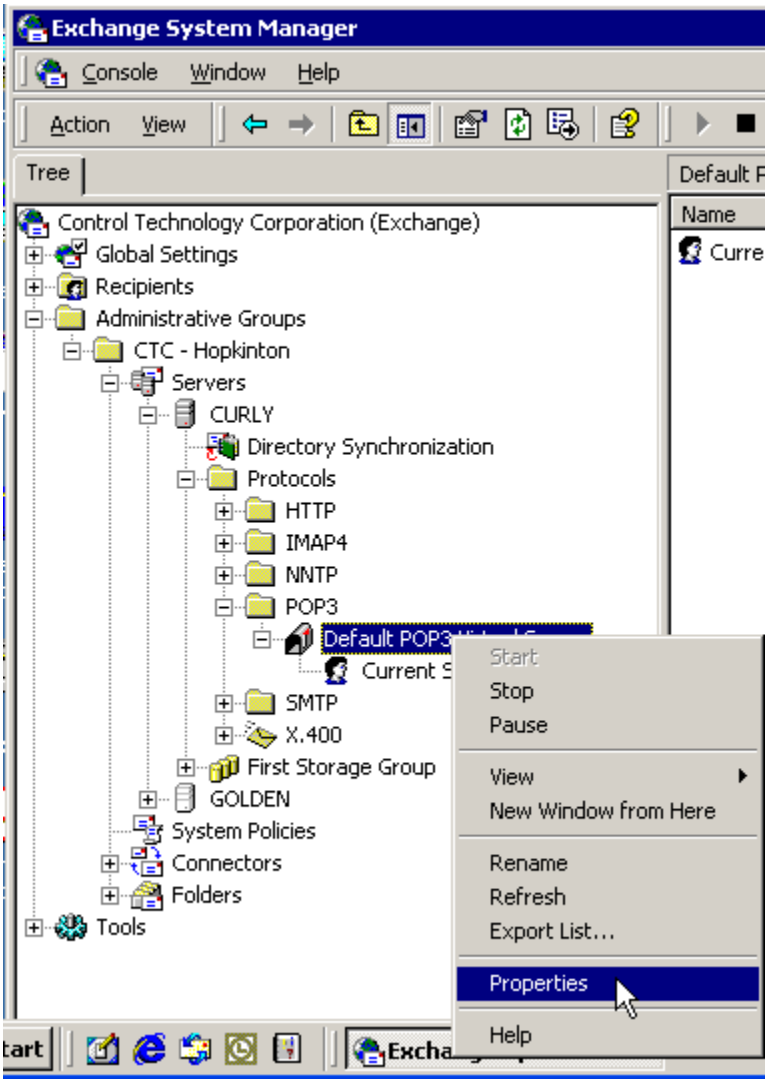
```
*Local 5200 Serial Number = 00063255
  DNS Name: 5200Kev  DHCP active: YES
  Group Name: Sales.DemoUnits
  IP Address = 12.40.53.158 MAC Address = 00C0CB00F717
  Total:  DIN = 4  DOUT = 16  AIN = 8  AOUT = 4  MOTION = 0
  Base Firmware Revisions:
    Quickstep SH2 Application           V05.00.11
    Quickstep SH2 Monitor               V15.15 @
  Slot Firmware Revisions:
    01. M1-30A-Analog 2 I/O              V01.07
       Ain1:  data-32596 offset-32631 spanpos-27218 spanneg-31715
       Ain2:  data-32615 offset-32621 spanpos-25649 spanneg-31771
       Aout1: data-00000 offset-32713 spanpos-31183 spanneg-31268
       Aout2: data-00000 offset-32734 spanpos-31176 spanneg-31261
    02. M1-31A-Analog 4 in               V01.01
       Ain1:  data-32809 offset-32708 spanpos-32747 spanneg-32707
       Ain2:  data-00000 offset-32707 spanpos-32743 spanneg-32704
       Ain3:  data-00000 offset-32706 spanpos-32753 spanneg-32705
       Ain4:  data-65535 offset-32702 spanpos-32756 spanneg-32701
    03. M1-30A-Analog 2 I/O              V01.07
       Ain1:  data-32710 offset-32719 spanpos-31745 spanneg-31731
       Ain2:  data-32707 offset-32715 spanpos-31756 spanneg-31734
       Aout1: data-00000 offset-32700 spanpos-31216 spanneg-31203
       Aout2: data-00000 offset-32709 spanpos-31157 spanneg-31140
    04. Empty                            V00.00
    05. M1-20A-Digital 8 Output          V00.00
       Dout:  0x99
    06. M1-20A-Digital 8 Output          V00.00
       Dout:  0x9F
    07. No Expansion Connected           V00.00
    08. No Expansion Connected           V00.00
    09. No Expansion Connected           V00.00
    10. No Expansion Connected           V00.00
    11. No Expansion Connected           V00.00
    12. No Expansion Connected           V00.00
    13. No Expansion Connected           V00.00
    14. No Expansion Connected           V00.00
    15. No Expansion Connected           V00.00
    16. No Expansion Connected           V00.00
    17. No Expansion Connected           V00.00
    18. No Expansion Connected           V00.00
    19. No Expansion Connected           V00.00
    20. No Expansion Connected           V00.00
    21. No Expansion Connected           V00.00
    22. No Expansion Connected           V00.00
    23. No Expansion Connected           V00.00
    24. No Expansion Connected           V00.00

No Thermocouples.tbl file found.
*

BlueFusion> set restart
SUCCESS: Restart Command completed.


-----Original Message-----
```

```
This line is ignored and can be any information desired in the
email.
The next line will signify the start of script processing.
[CTC_EMAIL_START_ATTACH_ORIGINAL]
# This is a comment.
# Request a copy of this email be attached to the original, not
needed
# but useful to know what we sent.  Regardless a copy of each of
# these commands and the reply is always sent back as a reply.
# [CTC_EMAIL_START] will not cause original to be attached.

# Let's assume we received an alarm condition via pager or email
# so lets clear it.  Possibly register 1 is used as a flag by the
# program.  Also keep these lines less than 72 characters when
# using Microsoft Exchange as it typically auto-line wraps and
# you will end up with a bad command.
1 = 0
# Now lets get all the version information just to make sure
things
# are OK.
get versions
# Restart the controller given to clear the alarm
set restart
# We are all done now so return to normal email text
[CTC_EMAIL_END]

This is just normal email text.  We could issue another command
block if desired following this text.
```

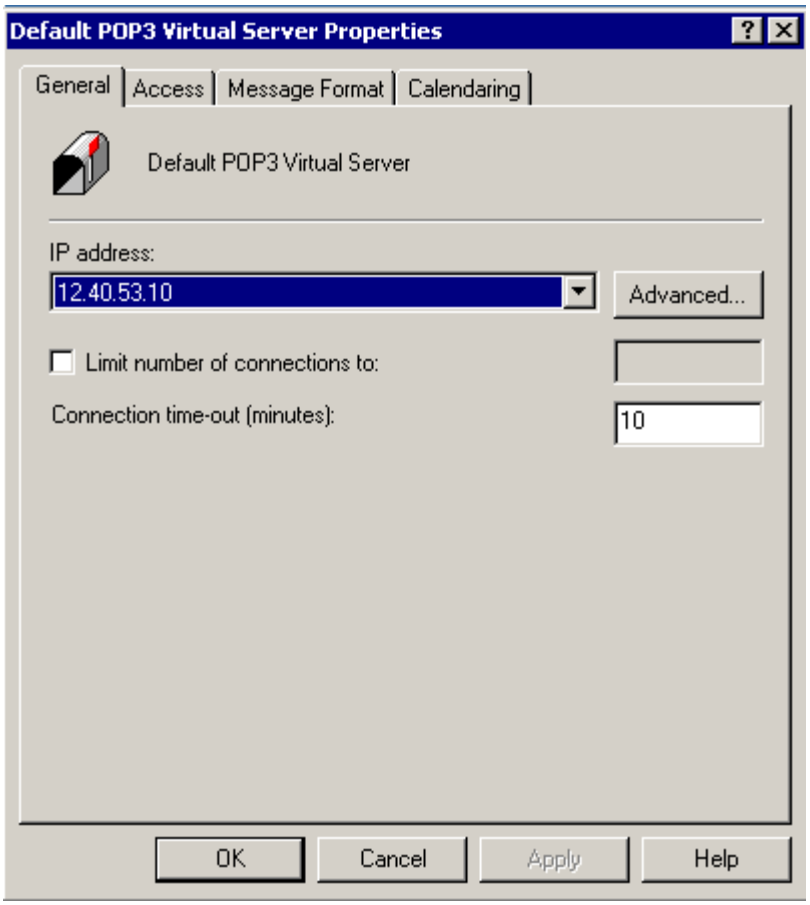
Microsoft Exchange 2000 Setup

All email servers are different in the way they are configured. As an example the setup of Microsoft Exchange 2000 is shown.

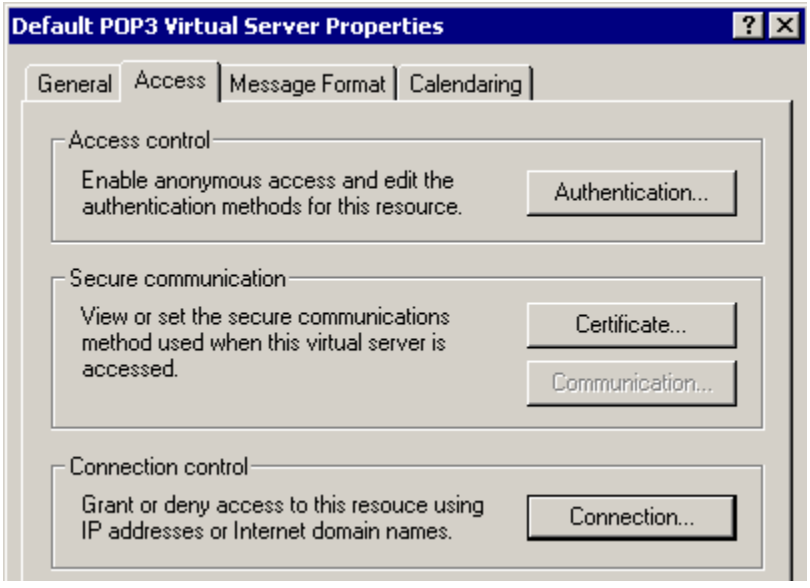
First invoke the Microsoft Exchange System Manager. For your server locate the POP3 protocol under the Administrative Groups, expand the folder and get the properties of the POP3 Virtual Server that you will be using.



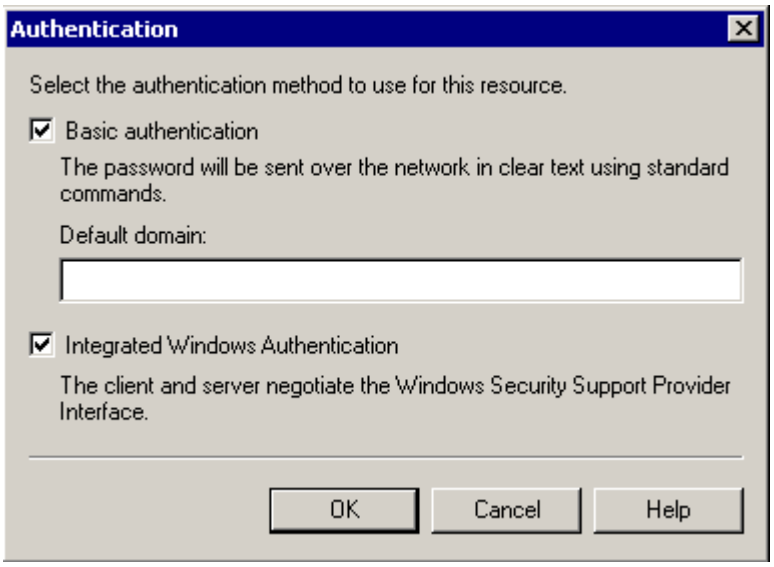
The Properties dialog will now appear:



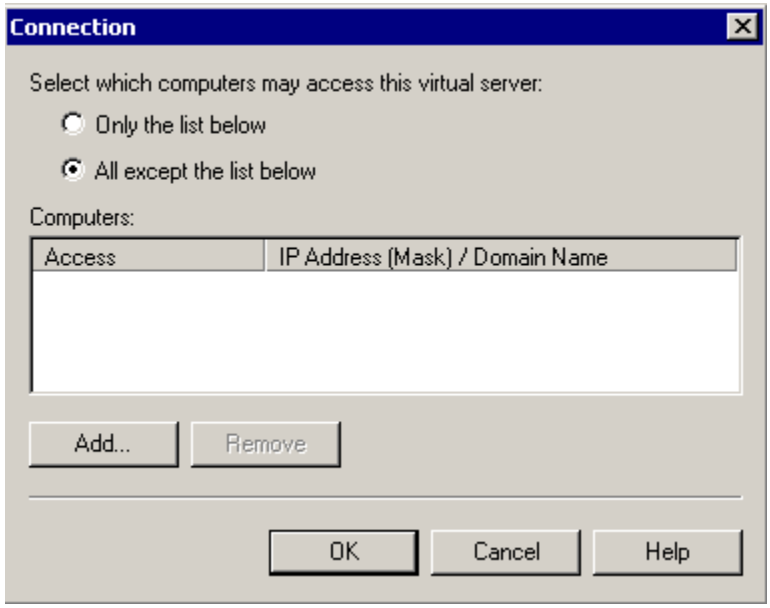
Select the ‘Access’ tab:



Select the Authentication button and ensure Basic Authentication is selected.



When done select OK, then the Connection Button. For security reasons you may only want to allow access from within your Domain. Below allows all connections but by selecting the "Only the list below" radio button you can restrict access.



When complete select the 'OK' button on all open dialogs.

CHAPTER

11

CTNet Binary Protocol (Server)

The CTNet binary protocol is a high-speed, non-routable protocol that has checksum and error reporting capabilities. It is used in cases where data integrity, response time, and processing time are the major criteria. Data transmission is fast for the following reasons:

- Both the commands and data are represented in binary form instead of ASCII.
- The information density is higher and fewer characters are transmitted during large data transfers.
- The controller can use the data "as is" and does not have to perform binary to ASCII conversion.

Therefore use of CTNet results in very short execution times. Note that the binary protocol is non-routable. Non-routable protocols do not contain a networking layer (IP stack), so they cannot cross a router and are limited to local subnets or intranets. However, lack of an IP stack reduces overhead by at least 20 bytes/packet. A smaller packet size increases the transmission rate, which is ideal for industrial controllers. Routable protocols such as TCP/IP result in a larger packet and more processor overhead to process.

CTNet uses a node number in place of an IP address. This node number is defined by writing to Register 20000. You can also determine the node number by reading the value in Register 20000. Set this value within the *_startup.ini* file by defining the CTNET_DEVICENODE parameter.

Binary Protocol

The CTC Binary Protocol may be used to communicate with the 5200 controller via serial ports or a network connection. Regardless of the mode used the basic message layer is the same. On a network the serial port data is simply encapsulated as required. Most users will not require this section and should only refer to the DLL available for use

with Visual Basic. This DLL is discussed in detail within the *"CTC 32-bit Communications Functions Reference Guide"*, available at www.ctc-control.com for download. The CTC Binary Protocol is somewhat more difficult to use than something like the ASCII Protocol, but it can significantly reduce the time required to transfer large blocks of data between a computer and controller and is useful in more demanding applications. The protocol is more efficient, because:

- Both the commands and data are represented in binary form instead of ASCII. The information density is higher and, for large data transfers, fewer characters need to be transmitted.
- The controller does not have to convert the data from ASCII to binary before using it. This results in shorter execution times. Since the computer does not have to convert the data to ASCII, there also may be a significant time savings in the execution of the computer program (the time savings varies between different computer languages).

Serial Port Protocol Framing

To select the CTC Binary Protocol, the first character of the command must be a binary 1 (Ø1H). The controller interprets the rest of the command according to the binary protocol. Use of an ASCII character, on the serial port, will result in the ASCII Protocol being used.

The protocol uses the following format to send messages to and from the controller:

<(Ø1H)> Specifies CTC binary protocol.
<(Ø2H) to (3FH)> Specifies packet length to follow. Packet length is defined as n data bytes + 2.
<data (n bytes)> Consists of function (command) code(s) plus relevant data. For function code and data descriptions, see the section on Binary Protocol Commands.
<checksum> Consists of the complement of the modulo-256 sum of data bytes. This value, when added to the modulo-256 sum of the data packet bytes, equals ØFFH. You can calculate the checksum by adding the data packet bytes and complementing the resulting sum.
<FFH> Required by binary protocol; last byte of packet must be ØFFH. When the controller receives a binary packet, it counts out the number of bytes specified by the packet length. If the last byte is not ØFFH, it returns an error message.

Return communications from the controller to the computer use the same general format, with one exception. The controller does not transmit a leading (Ø1H) byte, since the original message was transmitted using the CTC binary protocol. If the command sent to the controller does not require data from the controller in the return message, the controller sends an acknowledge message like the one shown below:

<(Ø2H) to (3FH)> Specifies packet length to follow. Packet length is defined as n data bytes + 2.
<(64H)> Contains the acknowledge code; equal to decimal 100.
<9BH> Is the value of the checksum of the acknowledge code.
<FFH> Required by binary protocol; last byte of packet must be ØFFH.

When the packet sent to the controller is not correct, it transmits a not acknowledged code. This may happen when the checksum does not calculate correctly or when the last byte of the packet is not ØFFH. A message containing a not acknowledged code is similar to the one shown below:

<(65H)> Contains the not acknowledged code; equal to decimal 101.
<9AH> Is the value of the checksum of the not acknowledged code.

When the format of the message is correct, but the controller cannot execute the command, it sends other error codes. For error code descriptions, see the section on *Binary Protocol Commands*. The following example shows how to create a command in correct format for the CTC binary protocol. It sets flag 4 in the controller.

1. *Send the following command:*

   Ø1H,Ø5H,13H,Ø3H,FHH,EAH,FFH
   Where:
   Ø1H Is the first byte and identifies the packet as using the CTC binary protocol.
   Ø5H Is the second byte and represents the length of the packet.
   13H Is the third byte and contains the function code for a change flag command.
   Ø3H Is the fourth byte and specifies flag 4. Flags 1 through 32 are represented as ØØH through 1FH, and Ø3H specifies flag 4.
   FHH Is the fifth byte and specifies the new state of the flag. ØFHH represents SET and ØØH represents CLEAR.
   EAH Is the sixth byte and contains the checksum value.
   ØFFH Is the seventh and last byte of the packet and signals the end of the message.

2. *To acknowledge the message, the controller sends the following response:*

   Ø3H,64H,9BH,FFH
   Where:
   Ø3H Is the first byte and specifies the packet length
   64H Is the second byte and contains the acknowledge code (decimal 100)
   9BH Is the third byte and contains the checksum value of third byte
   FFH Is the fourth and last byte and signals the end of the message.

Binary Protocol Error Responses

When the controller cannot execute the data transmission from the computer, the controller responds with an error code indicating the nature of the fault. The error code is transmitted using the following format:

Ø3H Packet length.
Error code Error code, see list below.
Checksum The checksum is the complement of the previous byte.
FFH Last byte in packet; signals the end of the message.

Possible error codes are:

64H No error (acknowledgment of transmission
65H Checksum error, or end of packet <> FFH
66H Illegal register number specified
65H Value out of range, for example, input number not present in controller

Binary Protocol Commands

Each CTC binary protocol command has specific format. This section lists the commands and describes their format. The command descriptions also list the following information:

- The type of command
- Format of command sent to the controller
- Format of the controller's response

Not all Control Technology controllers support all of these commands. Contact Control Tech customer support if you have any questions about which of these commands you can use, or if you have any difficulty implementing a command. The following table lists the commands and the controllers which support the command.

Binary Protocol Commands	
Register and Flag Access Commands	
9	Read a register
11	Change a register
17	Read a Flag
19	Change a Flag
75	Read a bank of 50 registers
77	Read a bank of 16 registers
87	Request random registers from list (CTServer)
Input/Output Access Commands	
15	Read a bank of 8 inputs
21	Read a bank of 8 outputs
25	Selectively modify first 128 outputs
29	Read an analog input

31	Read an analog output
33	Change an analog output
71	Get 32 analog inputs
73	Get 32 analog outputs
79	Read a bank of 128 inputs
85	Change multiple analog outputs
91	Read a bank of 128 outputs
Servo Access Commands	
23	Read a servo position
27	Read a servo's dedicated inputs
47	Read a servo error
Data Table Access Commands	
49	Read a data table's dimensions
51	Change a data table's dimensions
53	Read a data table value
55	Change a data table value
57	Read a row of data table values
59	Change a row of data table values
System and Controller Status Access Commands	
13	List counts of inputs, outputs, stepping and servo motors
35	Read controller step
61	Read controller status
63	Change controller status
65	Read system configuration
67	Change system configuration
69	List counts of miscellaneous I/O
105	Shutdown system
107	Get Controller Task Status

The following commands allow you to read and write values to registers and flags. You can read and write values for registers 1 through 65535. Some of the registers in this range are special function registers and you may not be able to read or write to them. Other registers do not exist on certain models and revision levels. Consult *Model 5200 Quick Reference Register Guide (951-520006)* for register specifics.

Register and Flag Access Commands

Binary Protocol Conventions

The binary protocol uses specific conventions for specifying register and flag numbers and values and for checksum error detection.

- When specifying a register number, it is expressed as ØØØ1H through ØFFFFH, corresponding to registers 1 through 65535. For example, register 10 is expressed as ØØØAH.
- You must specify register numbers with the least significant byte first.
- When specifying a flag number, it is expressed as ØØH through ØFH for flags, corresponding to flags 1 through 32. For example, flag 5 is expressed as Ø4H.
- The checksum value is the complement of the previous byte(s). Some commands use the complement of the modulo-256 sum of the previous bytes; see the command description.
- When the controller responds with a register value, it is always a four-byte representation of the register data expressed in 2's complement binary, with the least significant byte transmitted first.

Reading a Numeric Register - Command 9

Command 9 reads the value in any register that allows read access.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø5H Specifies the packet length
Ø9H Indicates the read register function code
LSB - MSB Specifies the register number, ØØØ1H - ØFFFFH. Specified with the least significant byte first.
Checksum Contains the complement of the modulo-256 sum of the previous 3 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø7H Specifies the packet length.
ØAH Indicates the register contents function code
LSB, 3SB, Four-byte representation of register data, expressed in 2's
2SB, MSB complement binary, with the least significant byte transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 5 bytes
FFH Signals the end of the message.

Reading a Bank of 16 Registers - Command 77

Command 77 reads the values in a bank of 16 consecutive registers.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø5H Specifies the packet length
4DH Indicates 16 register group read function code
LSB - MSB Specifies bank of registers to read, ØØØØH - Ø3D9H
Checksum Contains the complement of the modulo-256 sum of the previous 3 bytes

FFH Signals the end of the message

Format of Controller Response

45H Specifies the packet length
E4H Indicates the register contents function code
LSB - MSB Indicates bank of registers, ØØØØH - Ø3D9H
LSB, 3SB, Contains the value of the first register in the group. For a
2SB, MSB description of register data. See the description for single register read.
LSB, 3SB, Contains the value of the second register in the
2SB, MSB group and continues for all 16 registers in the group.
Checksum Contains the complement of the modulo-256 sum of the previous 67 bytes.
FFH Signals the end of the message.

Reading a Bank of 50 Registers - Command 75

Command 75 reads the values in a bank of 50 consecutive registers.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
4BH Indicates 50 register group read function code
ØØH - 13H Specifies the bank of 50 registers to be read, , ØØH - 13H
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Format of Controller Response

CCH Specifies the packet length
4CH Indicates the register contents function code
ØØH - 13H Indicates the bank of 50 register to follow, ØØH - 13H
LSB, 3SB, Contains the value of the first register in the group. For a
2SB, MSB description of register data. See the description for single register read.
LSB, 3SB, Contains the value of the second register in the
2SB, MSB group and continues for all 50 registers in the group
Checksum Contains the complement of the modulo-256 sum of the previous 202 bytes.
FFH Signals the end of the message.

Changing a Register Value - Command 11

Command 11 changes the value in any register that allows write access.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø9H Specifies the packet length

ØBH Indicates the change register value function code
LSB - MSB Specifies the register number, ØØØ1H - ØFFFFH. Specified with the least significant byte first.
LSB, 3SB, Four-byte representation of register data, expressed in 2's
2SB, MSB complement binary, with the least significant byte transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 7 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
Checksum Contains the complement of the previous byte
FFH Signals the end of the message.

Reading a Flag's State - Command 17

Command 17 reads the state of any flag.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
11H Indicates the read flag state function code
Flag Number Specifies the flag number, ØØH - 1FH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
12H Indicates the flag state function code
ØØH or FFH Indicates the flag's status. ØØH if flag is clear and FHH if set. Any other value means that the results are indeterminate.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Changing a Flag's State - Command 19

Command 19 changes the state of any flag.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø5H Specifies the packet length
13H Indicates the change flag state function code
Flag Number Specifies the flag to be changed, ØØH - 1FH

ØØH or FFH Specifies the new state of the flag. ØØH represents CLEAR and FFH represents SET.
Checksum Contains the complement of the previous 3 bytes
ØFFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
Checksum Contains the complement of the previous byte
FFH Signals the end of the message

Digital Input/Output Access Commands

The following commands allow you to read digital input and output states and turn a digital output on or off. Input and output states are read as a group of either 8 or 128.

Binary Protocol Conventions
The binary protocol uses specific conventions for specifying groups of inputs and outputs, their states and for checksum error detection.

- When specifying a bank of inputs or outputs as a group of 8, the first bank of inputs or outputs are specified as ØØH, corresponding to 1 through 8. The second bank is specified as Ø12H, corresponding to 9 through 16, and so on up to 7FH for the 16th bank, corresponding to 121 through 128.
- The checksum value is the complement of the previous byte(s). Some commands use the complement of the modulo-256 sum of the previous bytes; see the command description.
- When the controller responds with a the data for a group of 8 inputs or outputs, the lowest input number is represented by the least significant, the next the 7th least significant bit, and so on.
- For input states, a 1 represents a grounded (on) input.
- For output states, a 1 represents an output that is turned on.

Reading a Bank of 8 Inputs - Command 15

Command 15 reads the state of a group of eight digital inputs. The read inputs function code (ØFH) allows you to read a group of 8 inputs. Inputs are grouped so that the first group of inputs is 1 to 8; the second is 9 to 16, up to 121 to 128 for the 16th and last group.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
ØFH Indicates the read input state function code
Bank Specifies the bank of inputs, ØØH - 7FH

Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
1ØH Indicates the input data function code
ØØH - FFH Contains the data for the eight inputs, The lowest input number is represented by the least significant bit. A 1 indicates a grounded (on) input.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Reading a Bank of 128 Inputs - Command 79

Command 79 reads a bank of 128 inputs.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
4FH Indicates the read 128 inputs request function code
Bank Specifies the input bank to read, ØØH - 7FH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
50H Indicates the input values function code
Bank Input bank to follow, ØØH - 7FH
Inps1-8 Contains the data for the eight inputs, with the lowest input number is represented by the least significant bit. A value of 1 indicates a grounded (on) input.
Inps9-16 Contains the data for the next eight inputs. This continues for a total of 128 inputs.
Checksum Contains the complement of the modulo-256 sum of the previous 18 bytes
FFH Signals the end of the message

NOTE: The controller returns a value of zero (0) for nonexistent inputs with in a bank.

Reading a Bank of 8 Outputs - Command 21

Command 21 reads the state of a group of eight digital outputs. Outputs are grouped in the same manner as inputs.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol

Ø4H Specifies the packet length
15H Indicates the read output state function code
Bank Specifies the bank of outputs, ØØH - 7FH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
16H Indicates the output status function code
ØØH - FFH Contains the data for the eight outputs with the lowest output number represented by the least significant bit. A 1 indicates a that an output is on.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Reading a Bank of 128 Outputs - Command 91

Command 91 reads a bank of 128 digital outputs. The outputs are grouped in the same manner as inputs.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
51H Indicates the read 128 outputs request function code
Bank Specifies the bank of outputs, ØØH - 7FH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

14H Specifies the packet length.
52H Indicates the output values function code
Bank Specifies the bank of outputs, ØØH - 7FH
Outs1-8 Contains the data for the eight outputs, with the lowest output number is represented by the least significant bit. A value of 1 indicates an output is on.
Outs9-16 Contains the data for the next eight output. This continues for a total of 128 output.
Checksum Contains the complement of the modulo-256 sum of the previous 18 bytes
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

NOTE: The controller reports nonexistent outputs within a bank as off, value is 0.

Selectively Changing the First 128 Outputs - Command 25

Command 25 selectively changes the state of a group of 128 digital outputs. This command uses separate on and off masks so you can change specific outputs. For example, an off-mask-Ø of Ø6H (ØØØØ Ø11Ø in binary) would turn off outputs one along with four through eight and outputs two and would remain in their previous state. A subsequent on-mask-Ø of CØH (11ØØ ØØØØ in binary) turns on outputs seven and eight.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
23H Specifies the packet length
19H Indicates the modify outputs function code
off-mask-Ø to Specifies a series of 16 eight-bit masks used to selectively
off-mask-15 turn off any or all of the controller's first 128 outputs. The masks are applied to successive banks of 8 outputs, with the least significant bit of the mask being applied to the lowest numbered output in the bank. A mask value of Ø turns the associated output off. A value of 1 does not change the output.
on-mask-Ø to Specifies a series of 16 eight-bit masks used to selectively
on-mask-15 turn on any or all of the controller's first 128 outputs. The masks are applied to successive banks of 8 outputs, with the least significant bit of the mask being applied to the lowest numbered output in the bank. A mask value of 1 turns the associated output on. A value of Ø does not change the output.
Checksum Contains the complement of the modulo-256 sum of the previous 33 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
Checksum Contains the complement of the previous byte
FFH Signals the end of the message

Analog Input and Output Access Commands

The following commands allow you to read analog input and output states and change the value of an analog output. Input and output states are read individually.

Binary Protocol Conventions
The binary protocol uses specific conventions for specifying analog inputs and outputs, their values and for checksum error detection.

- When specifying an input or output the first input or outputs are specified as ØØH. The last input or output you can specify is 64. Its number is 3FH.
- The checksum value is the complement of the previous byte(s). Some commands use the complement of the modulo-256 sum of the previous bytes; see the command description.

Reading an Analog Input - Command 29

Command 29 reads the value of any one of the first 64 analog inputs.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
1DH Indicates the read analog input function code
Analog Input Specifies the input to be read, ØØH - 3FH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø5H Specifies the packet length.
1EH Indicates the analog input value function code
LSB - MSB Contains the two-byte representation of the analog value, expressed as a number in the range of 0 - 10,000 decimal (ØØØØH - 271ØH), with the least significant byte transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 3 bytes
FFH Signals the end of the message

Reading an Analog Output - Command 31

Command 31 reads the value of any one of the first 64 analog outputs.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
1FH Indicates the read analog output function code
Analog Output Specifies the output to be read, ØØH - 3FH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø5H Specifies the packet length.
1EH Indicates the analog output value function code

LSB - MSB Contains the two-byte representation of the analog value, expressed as a number in the range of 0 - 10,000 decimal (ØØØØH - 271ØH), with the least significant byte transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 3 bytes
FFH Signals the end of the message

Changing an Analog Output - Command 33

Command 33 changes the value of any one of the first 64 analog outputs.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø6H Specifies the packet length
21H Indicates the read analog output function code
Analog Output Specifies the output to be changed, ØØH - 3FH
LSB - MSB Contains the two-byte representation of the analog value, expressed as a number in the range of 0 - 10,000 decimal (ØØØØH - 271ØH), with the least significant byte transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 4 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø5H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
9BH Checksum value. Contains the complement of the previous byte
FFH Signals the end of the message

Servo Access Commands

The following commands allow you to read a servo's position, error and auxiliary inputs.

Binary Protocol Conventions
The binary protocol uses specific conventions for specifying servo axes, their position and error, the state of a servo's auxiliary inputs, and for checksum error detection. You can perform these operations for servos axes 1 - 16.

- When specifying a servo, the first servo axis is specified as ØØH and the 16th specified as ØFH.
- The checksum value is the complement of the previous byte(s). Some commands use the complement of the modulo-256 sum of the previous bytes; see the command description.

Reading a Servo's Position - Command 23

Command 23 reads the position of a servo.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
17H Indicates the read servo position function code
Servo Number Specifies the servo axis to be read , ØØH - ØFH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø7H Specifies the packet length.
18H Indicates the servo position function code
LSB, 3SB, Contains the four byte representation of the servos
2SB, MSB position. The value is expressed in 2's complement binary, with the least significant bye transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 5 bytes
FFH Signals the end of the message

Reading a Servo's Error - Command 47

Command 47 reads a servo's error.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
2FH Indicates the read servo error function code
Servo Number Specifies the servo axis to be read , ØØH - ØFH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø7H Specifies the packet length.
3ØH Indicates the servo position function code
LSB, 3SB, Contains the four byte representation of the servo's error.
2SB, MSB The value is expressed in 2's complement binary, with the least significant bye transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 5 bytes
FFH Signals the end of the message

Reading a Servo's Dedicated Inputs - Command 27

Command 27 reads the status of a servo's dedicated inputs. The controller returns the status of the dedicated input using a one bite code.

- Bit Ø, indeterminate
- Bit 1, Home input
- Bit 2, Start input
- Bit 3, Local/remote input
- Bit 4, Reverse limit input
- Bit 5, Forward limit input
- Bit 6, indeterminate
- Bit 7, indeterminate

Bit Ø is the least significant bit.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
1BH Indicates the read dedicated input status function code
Servo Number Specifies the servo axis to be read, ØØH - ØFH
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø7H Specifies the packet length.
1CH Indicates the servo dedicated input status function code
Status Contains a one byte of the servo's auxiliary input status.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Data Table Access Commands

The following commands allow you to read and change a data table's dimensions; read and change the value of a data table element; read the values in a data table row; and change the values in a data table row.

Binary Protocol Conventions

The binary protocol uses specific conventions for specifying rows and columns of a data table. The manner in which the row or column is specified varies with the command. The checksum value is the complement of the previous byte(s). Some commands use the complement of the modulo-256 sum of the previous bytes; see the command description. The controller may return an error code under the following circumstances:

- The requested data table size is tool large for the controller.
- The requested data table size does not fit in the memory available when stored along with the Quickstep program.
- The command contains a data table column number greater than 32.

Reading a Data Table's Dimensions - Command 49

Command 49 reads the dimensions of a data table. The number of data table columns is ØØH to 2ØH.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø3H Specifies the packet length
31H Indicates the read data table dimensions function code
CEH Contains the checksum of the previous byte
FFH Signals the end of the message.

Format of Controller Response

Ø6H Specifies the packet length.
32H Indicates the data table dimensions function code
LSB, MSB Contains the number of data table rows in the current program, with the least significant bye transmitted first.
cols Contains the number of data table columns.
Checksum Contains the complement of the modulo-256 sum of the previous 4 bytes
FFH Signals the end of the message

Changing a Data Table's Dimensions - Command 51

Command 51 changes a data table's dimensions.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø6H Specifies the packet length
33H Indicates the change data table dimensions function code
LSB, MSB Contains the new number of data table rows, with the least significant bye transmitted first.
columns Contains the new number of data table columns.
Checksum Contains the complement of the modulo-256 sum of the previous 4 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
9BH Contains the checksum, complement of the previous byte
FFH Signals the end of the message

Reading a Data Table Value - Command 53

Command 53 reads the value of a specific data table element by specifying its row and column number.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø6H Specifies the packet length
35H Indicates the read data table location function code
LSB, MSB Contains the row number of data table element, with the least significant bye transmitted first.
columns Contains the column number of data table element.
Checksum Contains the complement of modulo-256 sum of the previous 4 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø5H Specifies the packet length.
36H Indicates the data table data function code
LSB, MSB Contains the data from the data table, expressed as a positive integer. The range is from 0 to 65,535 (decimal) with the least significant bye transmitted first.
Checksum Contains the complement of the modulo-256 sum of the previous 3 bytes
FFH Signals the end of the message

Changing a Data Table Value - Command 55

Command 55 changes the value of a specific data table element by specifying its row and column number.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø8H Specifies the packet length
37H Indicates the change data table location function code
LSB, MSB Contains the row number of data table element, with the least significant bye transmitted first.
columns Contains the column number of data table element.
LSB, MSB Contains the new value for the specified data table element. The new value can range from 0 to 65,535 (decimal) with the least significant bye transmitted first.
Checksum Contains the complement of modulo-256 sum of the previous 6 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
9BH Contains the checksum, complement of the previous byte
FFH Signals the end of the message

Reading a Data Table Row - Command 57

Command 57 reads the values in specific data table row and columns by specifying its row and column number.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø7H Specifies the packet length
39H Indicates the read data table row function code
LSB, MSB Contains the row number, with the least significant byte transmitted first.
First col Indicates the first data table column to read
Quantity Specifies the number of data table columns to read (n); <= 27 columns
Checksum Contains the complement of modulo-256 sum of the previous 5 bytes
FFH Signals the end of the message.

Format of Controller Response

Length Specifies the packet length, (n * 2) + 4, where n = number of columns read.
3AH Indicates the data table row data function code
Quant Specifies the number of data table columns read (n); <= 27 columns
For each of n locations
LSB, MSB Contains the data from the data table, expressed as a positive integer. The range is from 0 to 65,535 (decimal) with the least significant bye transmitted first.
End of location data
Checksum Contains the complement of the modulo-256 sum of the previous (n * 2) + 2 bytes
FFH Signals the end of the message

NOTE: If the number of data table columns specified extends beyond the actual number of columns the controller's response only contains data for the existing columns and the response will be shorter than expected.

Changing a Data Table Row - Command 59

Command 59 changes the values in specific data table row and columns by specifying its row and column number.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
length Specifies the packet length, (n * 2) + 6, where n = number of columns to be changed.
3BH Indicates the change data table row function code
LSB, MSB Contains the row number, with the least significant byte transmitted first.
First col Indicates the first data table column to change
Quantity Specifies the number of data table columns to change (n); <= 27 columns
For each of n locations
LSB, MSB Contains the data from the data table, expressed as a positive integer. The range is from 0 to 65,535 (decimal) with the least significant bye transmitted first.
End of location data
Checksum Contains the complement of modulo-256 sum of the previous (n * 2) + 5 bytes
FFH Signals the end of the message.

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
9BH Contains the checksum, complement of the previous byte
FFH Signals the end of the message

System and Controller Status Access Commands

The following commands allow you to read the status of a controller; start, stop or reset a controller; read or change the configuration of the controller's dedicated inputs; and obtain information about the number and type of controller resources in a particular controller.

Binary Protocol Conventions
The binary protocol uses specific bits for controller status and system configuration information. See the command descriptions for information on how to send and read this information. The checksum value is the complement of the previous byte(s). Some commands use the complement of the modulo-256 sum of the previous bytes; see the command description.

Reading a Controller's Current Status - Command 61

Command 61 reads a controller's status and reports if it is running, stopped, has a software fault, or is in programming mode.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol

Ø3H Specifies the packet length
3DH Indicates the read status byte function code
CEH Contains the checksum of the previous byte
FFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
3EH Indicates the status byte function code
status Indicates the status of the controller, where:
Bit Ø = Ø if running and = 1 if stopped
Bit 1 = Ø in normal mode and = 1 in programming mode
Bit 2 = Ø if status OK and = 1 if there is a software fault
Bit 3 = Ø if in mid-program and =1 if fresh reset.
Bit Ø is the least significant bit and bits 4 through 7 are undefined.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Changing a Controller's Status - Command 63

Command 63 changes a controller's status.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
3FH Indicates the change controller status byte function code
status Indicates the status of the controller, where:
Bit Ø = Ø to start the controller and = 1 to stop it
Bit 3 = 1 to reset the controller and = Ø to continue
Bit Ø is the least significant bit and will always start or stop the controller. All unspecified and undefined bits should be set to Ø.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
Checksum Contains the complement of the previous byte
FFH Signals the end of the message.

Reading a Controller's System Configuration - Command 65

Command 65 reads the configuration of the controller's dedicated inputs.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø3H Specifies the packet length
41H Indicates the read system configuration function code
BEH Contains the checksum of the previous byte
FFH Signals the end of the message.

Format of Controller Response

Ø4H Specifies the packet length.
42H Indicates the system configuration function code
config Indicates the configuration of the controller, where:
Bit Ø = 1 if using input 1 for the start function
Bit 1 = 1 if using input 2 for the stop function
Bit 2 = 1 if using input 3 for the reset function
Bit 3 = 1 if using input 4 for the step function
Bit Ø is the least significant bit and bits 4 through 7 are undefined.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Changing a Controller's System Configuration - Command 67

Command 67 changes the configuration of the controller's dedicated inputs.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
43H Indicates the change system configuration function code
config Indicates the new configuration of the controller, where:
Bit Ø = 1 to use input 1 for the start function
Bit 1 = 1 to use input 2 for the stop function
Bit 2 = 1 to use input 3 for the reset function
Bit 3 = 1 to use input 4 for the step function.
Bit Ø is the least significant bit and bits 4 through 7 are undefined.
Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Format of Controller Response

Ø3H Specifies the packet length.
64H Contains the acknowledge function code (decimal 100)
Checksum Contains the complement of the previous byte
FFH Signals the end of the message.

Listing Counts of Inputs, Outputs, Motion - Command 13

Command 13 obtains information about the number and type of controller resources and reports the information.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø3H Specifies the packet length
ØDH Indicates the I/O count request function code
F2H Contains the checksum of the previous byte
FFH Signals the end of the message.

Format of Controller Response

ØCH Specifies the packet length
ØEH Indicates the I/O count function code
flags Indicates the number of flags, typically 20H
inputs LSB Indicates the number of inputs, LSB: ØØH to F8H
inputs MSB MSB: ØØH to Ø4H
outputs LSB Indicates the number of outputs, LSB: ØØH to F8H
outputs MSB MSB: ØØH to Ø4H
stepping mtrs Indicates the number of stepping motor axes, ØØH to 1ØH
servos Indicates the number of servo axes, ØØH to 1ØH
analog inputs Indicates the number of analog inputs, ØØH to FFH
analog outputs Indicates the number of analog outputs, ØØH to FFH
Checksum Contains the complement of the modulo-256 sum of the previous 10 bytes
FFH Signals the end of the message

Listing Counts of Miscellaneous I/O - Command 69

Command 69 obtains information about the number and type of various controller resources, such as prototyping boards, high-speed counting boards, thumbwheel arrays, and numeric displays and reports it.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø3H Specifies the packet length
45H Indicates the miscellaneous I/O count request function code
BAH Contains the checksum of the previous byte
FFH Signals the end of the message.

Format of Controller Response

Ø7H Specifies the packet length
46H Indicates the I/O count function code

protos Indicates the number of flags, typically 2ØH
h s counters Indicates the number of high-speed counters
twhls Indicates the number of 4-digits thumbwheel arrays
disps Indicates the number of 4-digit numeric displays
Checksum Contains the complement of the modulo-256 sum of the previous 5 bytes
FFH Signals the end of the message

Reading Controller Step Status - Command 35

Command 35 reads the status of tasks in the controller. By executing this command four times, once for each group of eight tasks, you may obtain all the information necessary to reconstruct the hierarchy and status of the controller's tasks. In addition, if software fault has halted execution of your program, the controller's response indicates the type of the fault, the step where it occurred, and any relevant parametric data. As it starts each new task, your Quickstep program assigns a task number from 1 to 32. The main program is always task number one. Each of the 32 tasks, whether they are currently being used or not, reports back a step number along with a 32-bit mask word. If the program is currently using a task number, the mask shows whether the task in currently suspended, waiting for one or more sub-tasks to finish. This is shown by a 1 bit in the bit position of the mask word corresponding to the task for which the current task is waiting. For example, if the main program, task one, called up three sub-tasks, tasks two, three and four, the mask word for task one would be as follows:

ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØ111Ø MSB LSB

To extract the hierarchy of tasks being executed:

1. Start with task one and read its mask word to determine its sub-tasks.
2. Read the mask word of each sub-task, these indicate if any tasks are being executed a the next level down the hierarchy.
3. As you follow the hierarchy of tasks under execution, you may determine the current step being executed by each via the step number data provided. Step number are offset by -1

NOTE: Do not assume that Quickstep allocates task numbers in the order of task hierarchy. The starting and stopping of task numbers in a complex program may result in a scattering of active tasks through out the 32 possible task numbers. The only way to determine the active tasks is to follow the task hierarchy as outlined above. When a controller is stopped because of a software fault the message returned by the controller will contain a software fault code. A list of all fault codes can be found in the Fault Task Handler chapter.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø4H Specifies the packet length
23H Indicates the status request function code
task range Bank of 8 tasks to be read, ØØH to Ø3H, where:

ØØH = tasks 1 through 8
Ø1H = tasks 9 through 16
Ø2H = tasks 17 through 24
Ø3H = tasks 25 through 32

Checksum Contains the complement of the modulo-256 sum of the previous 2 bytes
FFH Signals the end of the message

Format of Controller Response

39H Specifies the packet length
24H - 27H Indicates the controller status function code
Status - If the controller is stopped, it returns a value of ØFFH indicating true. Contains a value of ØØH indicating the controller is running
Fault type - Contains the type code for a software fault, if any are present. If the value is ØØH then no software fault is present.
NOTE: See the table on the previous page for a list of software fault codes.
Fault step – LSB, MSB, 16 bit, where where ØØØØH = step 1, ØØØ1H = step 2
LSB, 3SB, Data relating to software fault if any; otherwise
2SB, MSB unspecified. 48 bytes follow and provide the following data for each of the eight tasks:
LSB, MSB Step number currently being executed by this task, where ØØØØH = step 1, ØØØ1H = step 2, and so forth.
LSB, 3SB, 32 bit mask, indicating with a 1 or Ø for each of the 32
2SB, MSB possible tasks whether this task is waiting for the completion of each task or not. Lowest order bit of LSB represents task 1, etc.
Checksum Contains the complement of the modulo-256 sum of the previous 55 bytes
FFH Signals the end of the message

IP Encapsulation

An option exists which allows the CTC Binary Protocol to be sent over UDP and/or TCP, allowing it to be routed. All Blue Fusion controllers support the raw, low level, non-routable binary protocol, as well as run background servers listening for UDP and TCP connections which support "IP Encapsulation". Simply put, a header is added on to the current serial protocol. The controller listens for UDP requests on IP port 3000 and TCP on port 6000.

```
#define MAXPKTDATALEN 216
#pragma pack(1)
typedef struct ctcIPPacket_s
{

   // Used to validate proper CTC packet versions.
   //
   BYTE version_major;
   BYTE version_minor;
```

```
  // Identifier for each packet sent.  Used to validate incoming packets.
  //
  UINT16 transaction_id;

  // Required within packet.  Only the sender knows for sure the
  // type of the request.  The spare aligns data along word
  // boundaries.
  //
  BYTE type;
  BYTE spare;

  // Number of octets in the CTC binary.
  //
  UINT16 data_size;

  // Up to 216 (maximum in octets) of data.  Note : current maximum
  // packet size is 216 octets + 8 octets or 224 octets or bytes.
  //
  BYTE data[MAXPKTDATALEN];
} CTCPACKET;
#pragma pack()
```

The above structure is aligned on a 1 byte boundry. (#pragma pack(1)).

version_major/version_minor

These two byte fields represent the major and minor software revision of the initiator. The controller side simply returns whatever was received by the host making the request. Typically “version_major” = 0x04 and “version_minor” = 0x00.

transaction_id

The transaction_id is a two byte, little endian format (lsb/msb) field which contains an incrementing number, starting at 0x0001, to track the transaction request by. The controller will return the packet setting the transaction ID to that received, including the response information in the “data” field. Do not use a transaction id of 0x0000.

Type

0x14 – Request
0x15 – Reply

spare

Not used. Alignment purposes only. Set to 0x00.

data_size

This contains the length of the “data” field stored lsb/msb.
The maximum size of the “data” field is 216 bytes.

data

This is the binary protocol transaction which has been encapsulated. Refer to the standard CTC Binary Protocol Documentation. Messages from the host begin with 0x01, that from the controller are the length byte. Both message end with a checksum and 0xff byte. Only the number of bytes defined within “data_size” are contained within “data”, not the full maximum of 216 bytes.

Example register read request of register 0x0002 with transaction ID 0x0001:

```
|---------------------- Header ------------------------|------------ Binary Protocol Msg -----------|
0x04  0x00  0x01  0x00  0x14  0x00  0x07  0x00  0x01  0x05  0x09  0x02  0x00  0xf4  0xff
```

checksum = ~(0x09 + 0x02 + 0x00) = 0xf4

Reply from controller:

```
|---------------------- Header ------------------------|---------------- Binary Protocol Msg ----------------|
0x04  0x00  0x01  0x00  0x15  0x00  0x08  0x00  0x07  0x0a  0x00  0x00  0x00  0x00  0xf5  0xff
```

Register contained 0x00000000. Note that little endian storage is used (lsb first).

Blank

CHAPTER 12

Fault Task Handler

When Quickstep programs encounter problems they fault, removing control from the programmer. A new feature available in Blue Fusion controllers is the "Fault Task Handler". The "Fault Task Handler" is a regular Quickstep task that can be branched to and executed when a soft fault occurs. The Handler is simply a standard Quickstep program. It can be set up as either a separate task that is looping on a 'delay' instruction awaiting the fault, or a main program that sets the "Fault Task Handler" step and continues executing. Later branching in the program can go to the step designated to handle the fault.

There can only be one "Fault Task Handler" active at a time. Any task can be activated as a handler by writing a step number to branch to in register 13038, the TASK_FAULT_STEP_REGISTER. A branch will occur to the designated step when a Fault occurs. You can change which task is the handler or where to branch to at any time, by setting 13038 to a different step, or to 0 to disable the handler. Register 13040, TASK_FAULT_MASK_REGISTER can be set to enable which faults will cause the branch to occur. Each bit is OR'd as required to enable each type of Fault:

FAULT MASK	*FAULT TYPE*
0x0001 (1)	Fatal Errors
0x0002 (2)	Program Errors
0x0004 (4)	Motion Errors
0x0008 (8)	Analog Errors
0x0010 (16)	Digital Errors
0x0020 (32)	Communications Errors

When a Handler is executing it will ignore further soft faults and continue executing. The fault state must be cleared for normal operation to continue. This is controlled by register 13041, the TASK_FAULT_CLEAR_REGISTER (Write Only). This register controls the state of program execution:

Program State	description
1	RESET – Reset the controller only and then stop..
5	RESTART – Reset the controller and begin running again at step 1.
6	STOPPED – Stop the controller but do not reset.
8	RUNNING – Ignore the fault and continue running.
9	FAULT – Continue to fault as usual.
10	SHUTDOWN – Reset the controller and shutdown, requires a power cycle to exit.

Important registers are as follows:

REGISTER	description
13032	Fault Code – (R) Contains the fault code for what caused the fault.
13033	Fault Step – (R) Step in which fault occurred.
13034	Fault Task – (R) Task number, starting at 1, which caused the fault..
13035	Fault Data – (R) Any relevant error data.
13038	Fault Step Register – (R/W) Step to branch to when fault occurs. Write a 0 to disable.
13039	Fault Task Register – (R) Task number that is the active Fault Handler, 0 means none.
13040	Fault Mask Register – (R/W) Bit OR of types of fault that will invoke the handler, by default all enabled (-1) when the Fault Step Register is written
13041	Fault Clear Register – (W) Used to write the recovery state when done processing the Fault.

Fault Codes

Below is a table of all possible fault codes in the 5200:

Fault Value	Fault Mask	Description
1	1	Illegal function
2	1	Bad/corrupt program data
3	2	Destination step is empty
4	Not Used	Bad thumbwheel data
5	1	Step one is empty step
6	2	Too many tasks
7	4	No such stepping motor
8	4	Motor not ready
9	4	Motor not profiled
10	4	No such servo exists
11	4	Servo not ready
12	4	Servo Error
13	2	No such register exists
14	2	No such data table column
15	2	No such data table row
16	Not Used	No such prototyping board
17	Not Used	Illegal sample time
18	8	No such analog input
19	8	No such analog output
20	2	No such display exists
21	16	No such input exists
22	16	No such output exists
23	Not Used	No such thumbwheel exists
24	1	Illegal data table value
25	32	Message transmitting busy
26	1	Divide by zero error
27	1	Data out of range
28	1	Watchdog/hardware fault
29	32	Network error fault
30	Not Used	Network access timeout
31	Not Used	Network access busy
32	Not Used	Network request lost
33	Not Used	Network bad response
34	Not Used	Network bad return message
35	2	No such communications port
36	32	Error in request/reply
37	2	Bad flag number selected
38	2	Bad delay timer selected
39	2	Out of soft counters

40	8	Error in fetching or calculating analog In scaling
41	8	Module not calibrated
42	1	Error re-flashing module
43	2	Error trying to open request file, not exist?
44	1	Error trying to read file, fgets?
45	1	Malloc failed
46	8	Analog module not responding
47	8	Error in fetching or calculating analog Out scaling
48	1	Illegal build of Atmel board
49	32	Lost connection with virtual controller
50	1	Task error
51	1	Task status error
52	Not Used	Time delay not accepted, shorter delay already set (not an error)
53	2	Error accessing Hardware I/O
54	1	Generic IODRIVER error
55	2	Invalid parameter
56	1	Invalid extend data descriptor
57	4	SPI Overrun
58	4	SPI Timeout

Fault Task Handler Example

Symbols:

Registers	*Symbol Name*
10	FaultFlag
13038	FaultStepRegister
13039	FaultTaskRegister
13040	FaultMaskRegister
13041	FaultClearRegister

```
[1] init
   ;;; A Fault Handler is installed in the first
   ;;; step to monitor for communications failure.  The FaultMaskRegister
   ;;; must be set after the FaultStepRegister, otherwise the handler
   ;;; will be invoke for all faults (default).
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   Store 0 to FaultFlag
   store 8 to FaultStepRegister
```

```
   store 32 to FaultMaskRegister
   goto next

[2] v_setup
    …… continue program …….

[8] FaultHandler
   ;;; This step is invoked should a fault occur, such as a
   ;;; network disconnect.  The FaultMaskRegister controls
   ;;; under what circumstances the handler is invoked.  This
   ;;; example is very simple.  It basically shuts all the
   ;;; local outputs off and sets a flag in FaultFlag that
   ;;; has no purpose.  Note that no other tasks will be running
   ;;; in the system nor can this task fault when the handler
   ;;; is invoked.
   ------------------------------------------------------------------------
   <TURN OFF ALL DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 1 to FaultFlag
   delay 3 sec goto ClearFault

[9] ClearFault
   ;;; Now attempt to recover from the fault by issuing a RESTART
   ;;; command
   ------------------------------------------------------------------------
   <NO CHANGE IN DIGITAL OUTPUTS>
   ------------------------------------------------------------------------
   store 2 to FaultFlag
   store 5 to FaultClearRegister
   goto FaultHandler
```

Blank

CHAPTER

13

Formatted Messaging

The 5200 can transmit string-formatted messages, similar to the format supported by the 'C' function 'sprintf'. Each message may consist of just text and/or embedded references to any number of registers, whose values will be substituted just prior to transmission. Message format definitions are stored as records in a file called *message.ini* which is located in the */_system/Messages* subdirectory of the flash disk. Each line of *message.ini* is considered a record, from 1 to a maximum of 50 messages.

Messages are written to the default communications port set in register 12000, which is the standard Serial port selection register in Quickstep. Writing to the ***Message String Transfer Register*** (12316) selects which record to dynamically format and write out the communications port. A read returns the status of the write, with 0 meaning success. The 5200 supports up to 7 communication ports, two of which are dedicated to RS232, while the remaining 5 are assigned by the program as bidirectional TCP redirector ports. The redirector ports appear to Quickstep as RS232 ports, but actually either connect to a remote terminal server or host based application program

Typically a message consists of text with a 'sprintf" formatted specification, followed by `r####`, where `####` is the desired register. Therefore, to read register 8501 to be exactly 5 characters with preceding 0's, `%05dr8501` would be inserted in the text string. Note the `%05d` is the same as a 'printf'/'sprintf' and actually uses the exact same function, only enhanced. This means `%05Xr8501` would cause hexadecimal values to be generated. Sample strings using the previous example could be entered in the *message.ini* file as:

Analog Value = %05dr8500\r\n
Analog Value = %05dr8501\r\n

If the above are the only two entries in the *message.ini* file, then writing a 2 to the ***Message String Transfer Register*** would cause the second line to be processed and the following to be written out the RS232 port if a 583 were in register 8501:

Analog Value = 00583<CR><LF>

Message.ini Extended Formats

As described previously, the 'message.ini' file format is similar in structure to that of the 'C' sprintf function, with additional enhancements. References to registers, data table cells and time/date stamp formats are supported using this extended format:

Register (decimal) - %0#dr<register> or %dr<register>
Example: %05dr13002 (fix size with leading 0's to at least 5 places, reg 13002)

Register (hexidecimal) - %0#xr<register> or %Xr<register>
Example: %05xr13002 (fix size with leading 0's to at least 5 places, reg 13002)

Register (ascii) - %cr<register> or %cr<register>,<length> or %cr<register>,r<register>
Example: %cr12001,r12302 (convert the serial port buffer to ASCII characters)
Example: %cr12001,3 (convert the first 3 serial port buffer registers to ASCII)

Data Table Cell - %0#dD<row>,<col> or %dD<row>,<col>
Example: %05dD1,2 (fix size with leading 0's to at least 5 places, row 1, column 2, from the data table).

Time/Date Stamp - %T!<time/date format>
Example: %T!HH:mm:ss!
%T!MM/DD/YYYY!
Where each below are optional:
HH = hour (24 hour format)
mm = minute
ss = seconds
MM – month
DD – day
YY – year in 2 decimal format, no century.
YYYY – year in 4 decimal format, including century.
E – Day in week, text – Mon, Tue, Wed, Thu, Fri, Sat, or Sun
Z – Time zone information in 5 digit format - <sign>HH:mm from GMT
Note:
- All other characters are treated as filler text, except ending '!'.
- Maximum 48 character Time/Date Stamp string.

'log.ini' in the 'Model 5200 Logging and FTP Client Applications Guide', 951-520015, uses the same formats detailed above.

CHAPTER

14

Network Performance Adjustments

Within a 5200 environment many threads run in parallel, each executing when there is work to do, and then sleeping until it is their time to be serviced once again. At the highest general priority is your Quickstep application program. It must yield in order to allow things like the web server to transfer files, telnet to return key strokes, etc. Quickstep instructions tend to poll I/O or registers, at high rates of speed, until a change of condition occurs, at which point logical branching occurs. At times the time between each step can be critical therefore registers are provided to control the balancing of execution time amongst tasks.

As each Quickstep step is executed a background timer is run, upon timeout, a window is opened allowing other threads to execute, such as the web server. Since there is only one CPU when you service Quickstep you can not be transfer files, when transferring files you can not service Quickstep, hence a decision must be made as to what is the worst case acceptable time allowed between Quickstep steps. Register 13036, Performance Adjustment Register (PAR), is the periodic number of milliseconds the Quickstep execution loop will check to see if any network operations need to take place, if none, Quickstep continues to execute, else it yields control for Register 13037 (Network Service Window, NSW) X 5 milliseconds. Thus PAR controls the network response time for many operations while NSW controls the amount of time the network may run prior to returning control to Quickstep. NSW is the maximum amount of time that typically will occur between Quickstep instructions under heavy network traffic.

By default Quickstep checks to see if the network needs service every 30 milliseconds, allowing the network window to remain open for 30 milliseconds (NSW = 6), thus the worst case time between individual steps. This value may be changed at any time. The minimum value for PAR is 10 milliseconds and NSW is 2 (2 X 5= 10 milliseconds).

Required settings:

- 10 <= PAR <= 250 (smaller PAR > Network Performance)
- 2 <= NSW <=14 (larger NSW > Network Performance)