Control Technology Corp.

CONTROL TECHNOLOGY CORPORATION

Model 5300 Communications & Logging Guide

Model 5300 Communications & Logging 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
5300	All Revisions	>= 5.00.90R69.44

TABLE OF CONTENTS

CHAPTER 1

[1] Communications Summary

With the release of the Model 5300 firmware revision 5.00.90R69.20 and above, numerous 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:

- (4) Serial ports that support the CTNet Binary protocol, CTC ASCII Protocol, User Defined, Modbus RTU/ASCII Master and Slave protocols
- COM1 to COM4 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 *Document No. 951-530001: Remote Administration Guide.*
- HTTP 1.0 Web server for WebMON *(Document No. 951-530012: WebMON 2.0 User's Guide)* diagnostics.
- Modbus/TCP RTU Master and Slave
- UDP Peer to Peer
- TCP client/server raw socket interface, bidirectional (up to 20)
- CTNet Binary protocol
- SMTP support for sending emails
- POP3 inbox support for receiving emails and processing embedded script messages
- Up to 9 serial ports, including 4 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

[2] Serial Communications

The controller contains four RS-232 serial ports. Optionally, COM3 can be ordered with RS-485 in which case COM4 is not available. 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 (default)

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:

Serial Communications Registers

12000	**Select Controller Communications Port**: W access, 1 = COM1, 2 = COM2, 3 = COM3, 4 = COM4, 6 – 25 = TCP raw virtual socket connections (see 22XX0 register descriptions).
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**, Read 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. Use 0 for custom, raw data processing.
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. Not used on Model 5300, communications runs as background thread.
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.

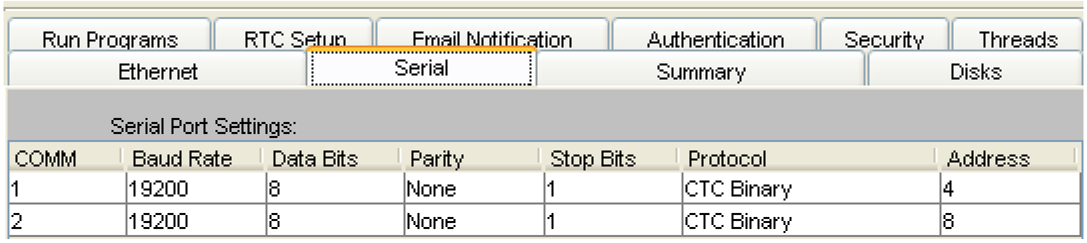
12320	**Serial Active Protocol Selection**: R/W; by default the protocol is set to CTC (0). Write to this port last after setting up any relevant parameters in other register, since this register enables the selected protocol immediately. CTC Binary & ASCII – 0 Modbus Master RTU – 1 (max of 120 16 bit 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 registers) Modbus Slave ASCII – 4 (max of 56 16 bit Modbus Registers or 28 32 bit registers) Diagnostic Terminal – 7 (telnet admin screen active, ^A ESC ESC to activate)
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 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.
12323	**Modbus Endian Swap** – R/W, if set to 1 (0, default) any time a 32 bit even boundary register is read via Modbus the 16 bits is swapped (H/L). 40001/3/5…, is even boundary in Modbus.
12337	**Modbus Bank1 Select** – R/W, used to read variant registers since Modbus does not allow reading great than 32K register and variants start at 36001. Value of 0, disables, else view window 9XXX has this value added to it to reference the actual register. Example if write 27000 here then when read/write 9101 will really be 36101.
12338	**Modbus Bank2 Select** – R/W, used to read variant registers since Modbus does not allow reading great than 32K register and variants start at 36001. Value of 0, disables, else view window 10XXX has this value added to it to reference the actual register. Example if write 26000 here then when read/write 10101 will really be 36101.
12339	**Modbus Bank3 Select** – R/W, used to read variant registers since Modbus does not allow reading great than 32K register and variants start at 36001. Value of 0, disables, else view window 11XXX has this value added to it to reference the actual register. Example if write 25000 here then when read/write 11101 will really be 36101.

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. Refer to *Document No. 951-520012: WebMON User's Guide*, as a review, and note that 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:



Serial Port Settings:

COMM	Baud Rate	Data Bits	Parity	Stop Bits	Protocol	Address
1	19200	8	None	1	CTC Binary	4
2	19200	8	None	1	CTC Binary	8

- 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 Chapter [3] Networking Communications and Chapter [7] Modbus. 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 used 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.

Blank

CHAPTER 3

[3] Networking Communications

The 5300 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 set up and configure the controller for network communications.

CTNet

CTNet is a proprietary, non-routable protocol typically used for legacy communications to the Model 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. Refer to the IP Encapsulation section for further details. TCP encapsulation is limited to 32 simultaneous connections.

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 Model 5300 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 Model 5300 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 Model 5300 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 Model 5300 to receive all traffic on its link. The 5300 is limited to 128 socket connections. Of those 128 a maximum of 32 simultaneous TCP Binary protocol and 32 Modbus TCP Slave.

Configuring a CTNet Node using Registers

Details of the CTNet protocol can be found within the *Guide to CTC Serial Data Communications* and *Document No. MAN-1030-A: CTC Monitor User Guide*, both of which are posted on Control Technology's website (http://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.

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 flashes repeatedly, at a high rate (100ms/second). The S3 LED will stop flashing once the Model 5300 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 5300 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.

Every device (computers, controllers, etc.) that resides on a TCP/IP network must have an IP address assigned. Without DHCP, the IP address must be entered manually at each device, 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.

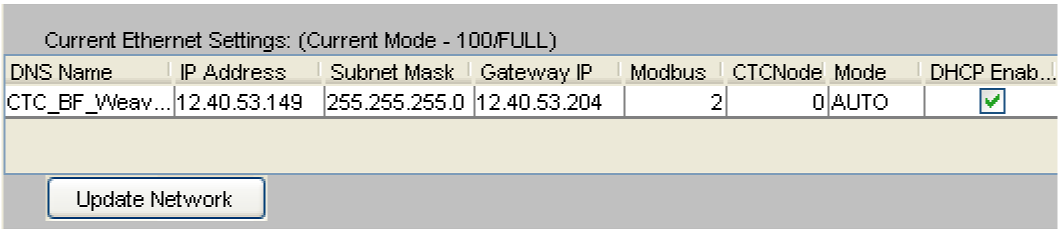
Note that WinPCap only needs to be installed when using the non-routable binary protocol version of CTNet, that used in legacy Model 2700 products using the 2217 Ethernet Controller. Operating CTNet over UDP and TCP can be done using IP Encapsulation and does not required WinPCap. The Model 2700 does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

Network Configuration via WebMON

Instead of directly modifying registers, network parameters may be modified using the WebMON utility. The Ethernet tab in WebMON is used to set various network parameters. Refer to *Document No. 951-520012: WebMON 2.0 User's Guide* for details. 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:



Current Ethernet Settings: (Current Mode - 100/FULL)

DNS Name	IP Address	Subnet Mask	Gateway IP	Modbus	CTCNode	Mode	DHCP Enab...
CTC_BF_Weav...	12.40.53.149	255.255.255.0	12.40.53.204	2	0	AUTO	☑

- DNS Name
- IP Address
- Subnet Mask
- Gateway IP

- Modbus
- CTCNode
- 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.) 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. 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), and 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 your 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 that 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.

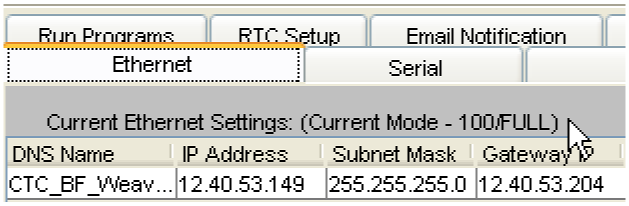
CTCNode

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. The screen capture below shows the current speed is negotiated to 100 full duplex:



Current Ethernet Settings: (Current Mode - 100/FULL)

DNS Name	IP Address	Subnet Mask	Gateway
CTC_BF_Weav...	12.40.53.149	255.255.255.0	12.40.53.204

CHAPTER 4

[4] ASCII Computer/Terminal Protocol

The Model 5300 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 `R10=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 is 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 cannot 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

[5] TCP/IP Raw Sockets

Up to 20 TCP Client/Server RAW Socket sessions are supported by the Model 5300 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 similar.

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 its 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 4 physical serial ports (COM1=1, COM2=2, COM3=3, COM4=4, 0 not used) within the controller. They can also access virtual serial ports 6 to 25, 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:

22000-22199	TCP Raw Socket Session Parameters: R/W, starts at 22000 and repeated every 10 blocks (max of 20 RAW sockets) as follows: **22XX0** – Serial port ID register, offset 0, range 6 – 25. **22XX1** – Client/Server register, offset 1, to initiate connection set to a 0, if controller is a server set to a 1. **22XX2** – Most significant octet of IP Address to connect to if client mode, IPA, offset 2. If server mode this is the default protocol to run as a server, uses same codes as register 12320 (default 0, Modbus Master not supported). **22XX3** – IP Address octet, IPB if client mode, offset 3. If server mode this is the SET/CLR parsing control (similar to register 12303), typically set to 0 for custom, else 1 for normal. **22XX4** – IP Address octet, IPC if client mode, offset 4. If server mode this is the server address to use, typically for Modbus applications, reference register 12321. **22XX5** – Least significant octet of IP Address to connect to, IPD if client mode, offset 5. If server mode write a 0 to this location. **22XX6** – Port to connect to (client) or listen on (server), offset 6 **22XX7** – Connection status register, offset 7, on read, -1 = not initialized, 0 = offline, 1 = online, write a 1 to initiate connection or start server thread. **22XX8** – Index register to offset to data, offset 8. Recommend using serial port buffer, not this interface but available to mimic the peer to peer interface. **22XX9** – Data array, offset 9. Recommend using serial port buffer commands, not this interface but available to mimic the peer to peer interface.

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 = 6        # set up this client connection as controller port 6
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 6 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:

22000-22199	TCP Raw Socket Session Parameters: R/W, starts at 22000 and repeated every 10 blocks (max of 20 RAW sockets) as follows: 22XX0 – Serial port ID register, offset 0, range 6 – 15. 22XX1 – Client/Server register, offset 1, to initiate connection set to a 0, if controller is a server set to a 1. 22XX2 – Most significant octet of IP Address to connect to if client mode, IPA, offset 2. If server mode this is the default protocol to run as a server, uses same codes as register 12320 (default 0, Modbus Master not supported). 22XX3 – IP Address octet, IPB if client mode, offset 3. If server mode this is the SET/CLR parsing control (similar to register 12303), typically set to 0 for custom, else 1 for normal. 22XX4 – IP Address octet, IPC if client mode, offset 4. If server mode this is the server address to use, typically for Modbus applications, reference register 12321. 22XX5 – Least significant octet of IP Address to connect to, IPD if client mode, offset 5. If server mode write a 0 to this location. 22XX6 – Port to connect to (client) or listen on (server), offset 6 22XX7 – Connection status register, offset 7, on read, -1 = not initialized, 0 = offline, 1 = online, write a 1 to initiate connection or start server thread. 22XX8 – Index register to offset to data, offset 8. Recommend using serial port buffer, not this interface but available to mimic the peer to peer interface. 22XX9 – Data array, offset 9. Recommend using serial port buffer commands, not this interface but available to mimic the peer to peer interface.

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. Communication is tunneled over the network to the device. Even a serial port version of CTC Mon 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

[6] UDP Peer to Peer Protocol Overview

Peer to Peer communications allow 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 5100/5200/5300 modules. Model 5300 controllers can also communicate with Model 2700 controllers via the 2717 communications module, but not via the 2217 module. The 5300's peer-to-peer registers let it communicate directly with other Model 5300 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

21000-21299

TCP Peer to Peer and Modbus Master Parameters: R/W, starts at 21000 and repeated every 10 blocks as follows:

21XX0 – First Octet IP Address Register (Most Significant) – R/W

This is the first octet of the IP address (XXX.000.000.000) to connect to.

21XX1 – Second Octet IP Address Register – R/W

This is the second octet of the IP address (000.XXX.000.000) to connect to.

21XX2 – Third Octet IP Address Register – R/W

This is the third octet of the IP address (000.000.XXX.000) to connect to.

21XX3 – Fourth Octet IP Address Register (Least Significant) – R/W

This is the fourth octet of the IP address (000.000.000.XXX) to connect to.

21XX4 – Start Register – R/W

This register stores the starting register address that is to be read from the remote 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 100 for Peer to Peer, 256 for Modbus and CTC Binary Master. 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. For Modbus Master this register is the default; see 21XXX8, 1011 to change when polling differing devices. This register must be written first to define the required storage size; upon initialization all other registers will become available.

21XX6 – Poll Timer Register – R/W

Set this register to 0 for a single read request else set the scan rate in milliseconds. The minimum value allowed is 50 ms for Peer to Peer and 10 ms for Modbus. You can write to this register at any time.

21XX7 – Status Flag Register – R

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:

0 - Offline; no connection is present.

1 - Last request is successful and completed. Data is available in the data registers if requested. Read or Write may now be done.

-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; poll timeout on a device by Modbus Master.

-10 - Aborted operation; out of local memory or resources.

21000-21299 Cont'd	**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 array 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, this register 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. The index register also has a few special features when you set it to 1000 or above. **1000 - Peer Request Time-Out Register** – (R/W) 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** –(R) 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** – (R/W) 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** – (R/W) 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 for Modbus Master RTU Serial on COM1 (TBD), 8 is CTC Binary Master UDP (port 3000), 9 is CTC Binary Master TCP (port 6000). Note the Binary Master protocol allows connection to 2700 series controllers. **1004 - TCP Client Port Index Register** – (R/W) 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 and serial Master mode with a default port number of 502 for TCP and 1 for serial (COM1). Not used for the CTC Binary Master protocol. For that protocol it is fixed to 3000 for UDP and 6000 for TCP.

21000-21299 Cont’d	**1005 – Modbus Master Unit ID Index Register** – (R/W) This register points the data register to the Device/Unit 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** – (R) 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** – (R/W) 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 50 milliseconds to that desired, in milliseconds. **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. **1012 – Random Index Register** – (R/W) CTC Binary Master protocol only. -1 default, with 1013 of -1 means normal peer to peer automatic sequential registers. If 1013 is not -1 then this register may be used to offset to the 1014 register window. All values appear in 21XX9. Writing a -1 to this register clears the registers set by writes to the Random Register Window and restores sequential mode. Typically this register is not accessed. After setting random registers (1014) it can be used to view what has been set. In other words setting this to a 2 would cause the third (0 based) register to be selected in the Random Register Window when that index is set. **1013 – Number of Random Registers Initialized** – (R Only) CTC Binary Master protocol only. Number of random registers initialized, up to 21XX5 max. -1 means none, default (sequential mode). All values appear in 21XX9. Note that this is only if random access mode is being used, else -1 means sequential, reference offset 1014 for further information.

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

[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 Model 5300 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 Model 5300 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 Model 5300 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 Model 5300 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 blue are supported by Model 5300)

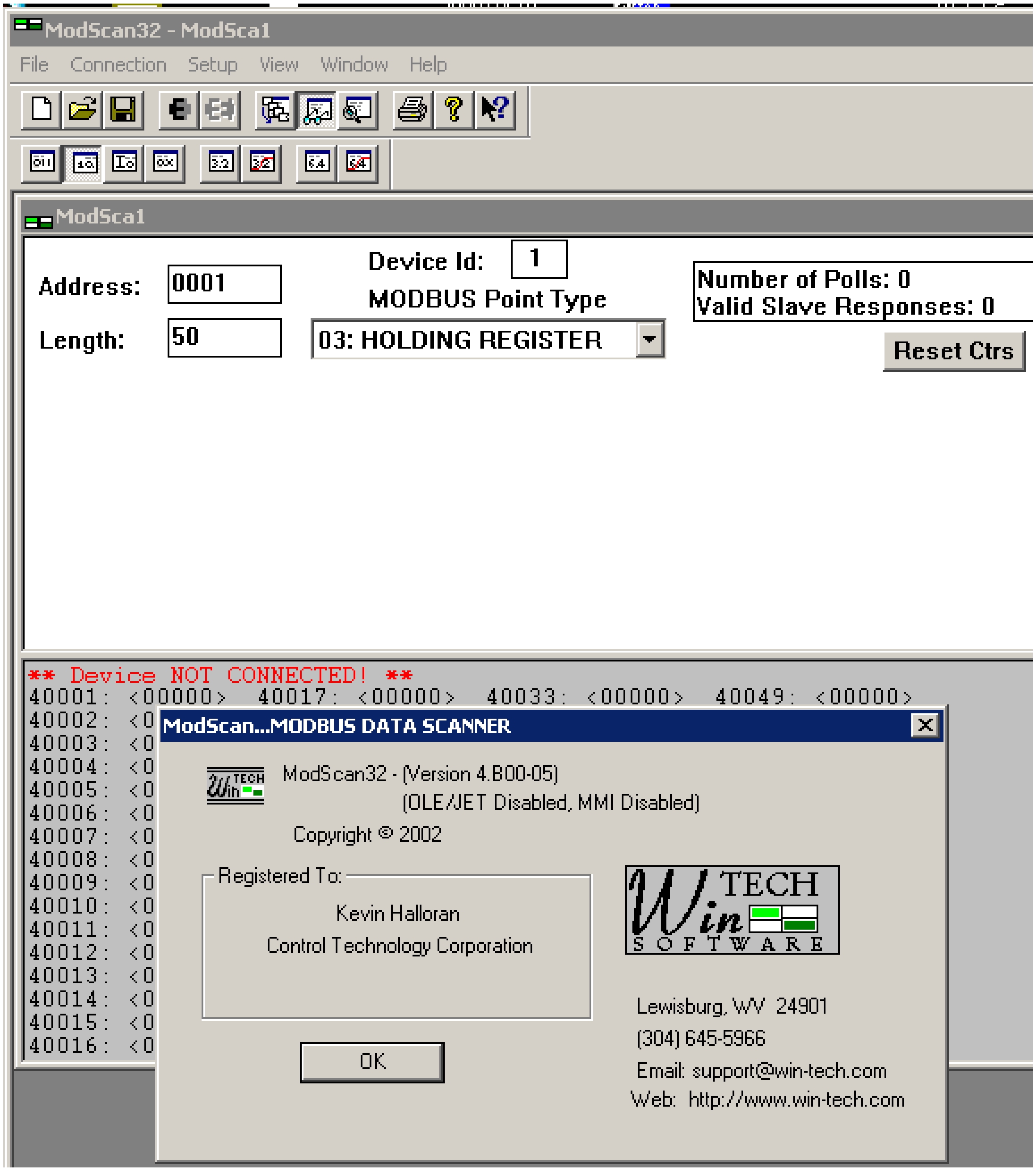
You should also note that Modbus Holding registers are 16 bits in width and those of the Model 5300 controller are 32 bits, since Modbus is Big Endean. This means when reading register 1 in the 5300 controller, the high 16 bits equates to Modbus register 1 and the low 16 bits to Modbus register 2. Modbus 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 5300 registers).
Modbus RTU Serial - 120 Modbus 16 bit registers (60 5300 registers).
Modbus ASCII Serial - 56 Modbus 16 bit registers (28 5300 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 Model 5300 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 the Modbus Slave Serial RTU/ASCII section.

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 Model 5300 controller. As mentioned before, CTC only supports the Holding Register interface. Upon installation of ModScan32, a screen such as Figure 7.3 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).



Address: 0001

Length: 50

Device Id: 1

MODBUS Point Type

03: HOLDING REGISTER

Number of Polls: 0

Valid Slave Responses: 0

Reset Ctrs

```
** Device NOT CONNECTED! **
40001: <00000>  40017: <00000>  40033: <00000>  40049: <00000>
40002: <0
40003: <0
40004: <0
40005: <0
40006: <0
40007: <0
40008: <0
40009: <0
40010: <0
40011: <0
40012: <0
40013: <0
40014: <0
40015: <0
40016: <0
```

ModScan...MODBUS DATA SCANNER

ModScan32 - (Version 4.B00-05)
(OLE/JET Disabled, MMI Disabled)
Copyright © 2002

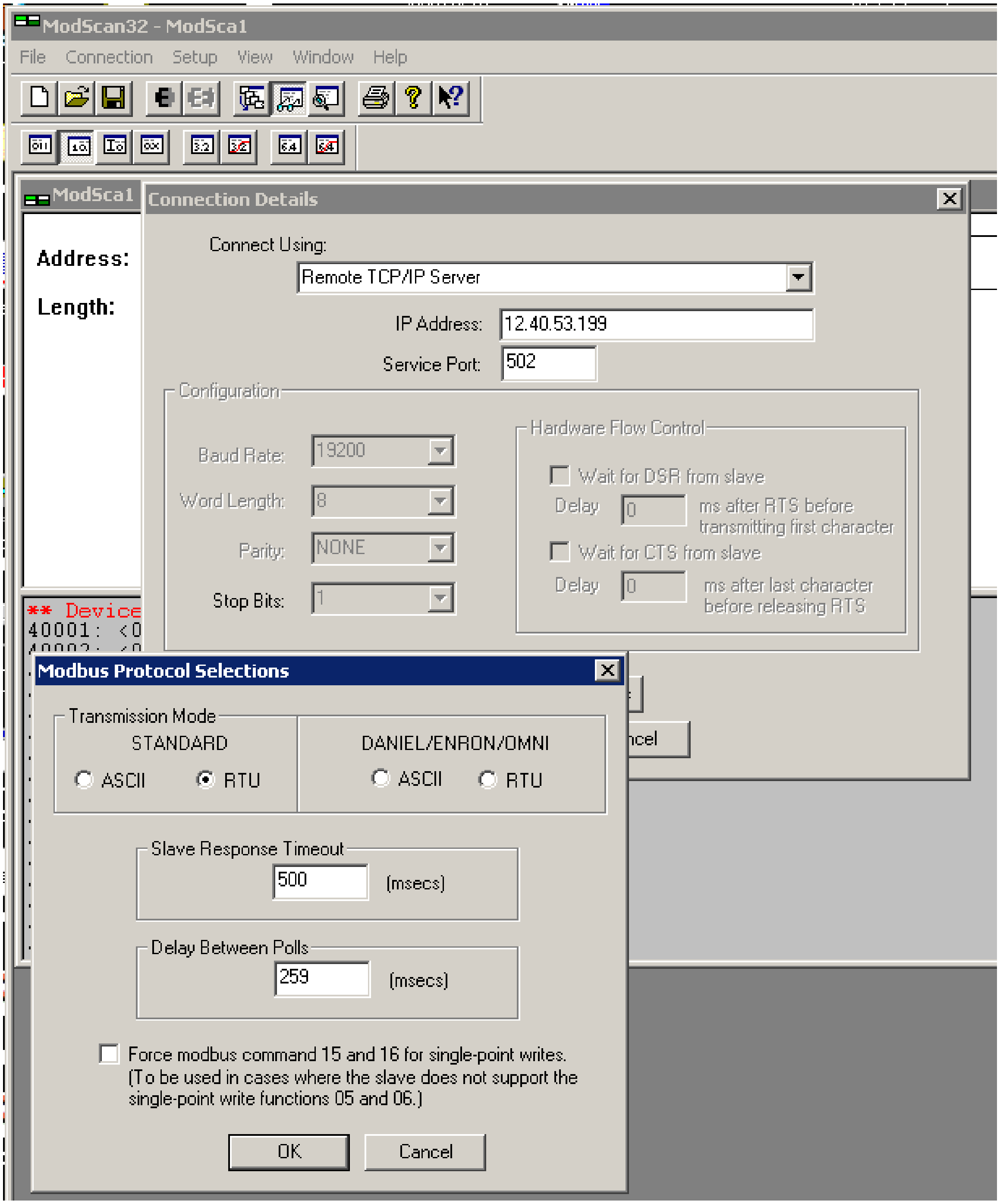
Registered To:
Kevin Halloran
Control Technology Corporation

OK

Lewisburg, WV 24901
(304) 645-5966
Email: support@win-tech.com
Web: http://www.win-tech.com

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:



Connection Details

Connect Using: Remote TCP/IP Server

IP Address: 12.40.53.199

Service Port: 502

Configuration

Baud Rate: 19200

Word Length: 8

Parity: NONE

Stop Bits: 1

Hardware Flow Control

□ Wait for DSR from slave

Delay 0 ms after RTS before transmitting first character

□ Wait for CTS from slave

Delay 0 ms after last character before releasing RTS

Modbus Protocol Selections

Transmission Mode

STANDARD: ○ ASCII ● RTU

DANIEL/ENRON/OMNI: ○ ASCII ○ RTU

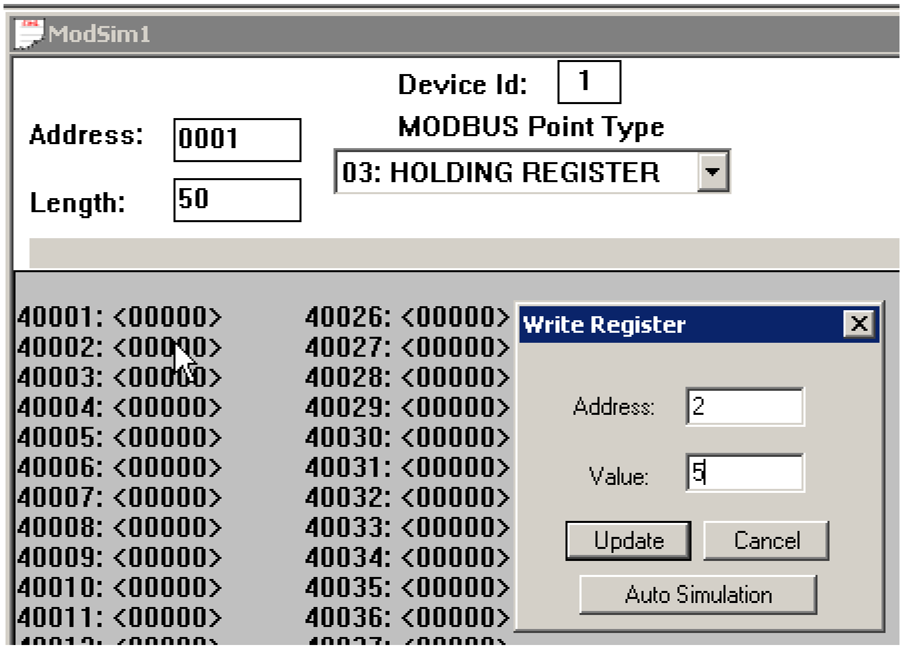
Slave Response Timeout: 500 (msecs)

Delay Between Polls: 259 (msecs)

□ Force modbus command 15 and 16 for single-point writes. (To be used in cases where the slave does not support the single-point write functions 05 and 06.)

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.



Device Id: 1

Address: 0001

MODBUS Point Type: 03: HOLDING REGISTER

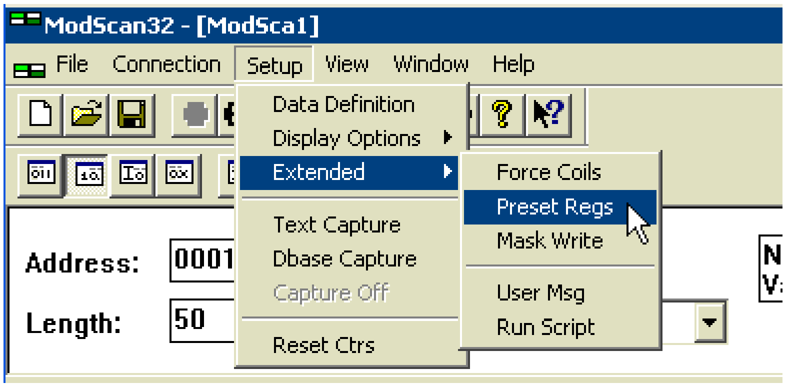
Length: 50

```
40001: <00000>        40026: <00000>
40002: <00000>        40027: <00000>
40003: <00000>        40028: <00000>
40004: <00000>        40029: <00000>
40005: <00000>        40030: <00000>
40006: <00000>        40031: <00000>
40007: <00000>        40032: <00000>
40008: <00000>        40033: <00000>
40009: <00000>        40034: <00000>
40010: <00000>        40035: <00000>
40011: <00000>        40036: <00000>
```

Write Register — Address: 2, Value: 5

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:

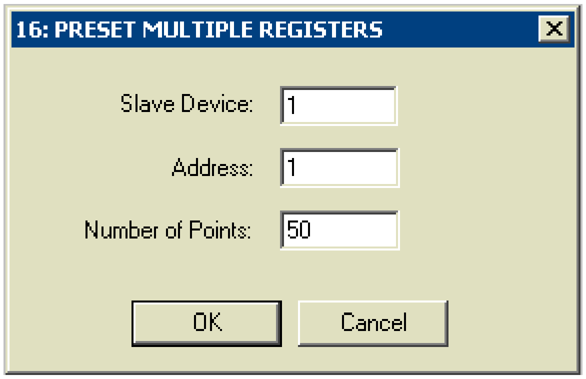


Address: 0001

Length: 50

Figure 7.6: Write Multiple register (Preset Regs) selection

The Preset Multiple Registers pop-up will appear. Note that in TCP, the Model 5300 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:



16: PRESET MULTIPLE REGISTERS

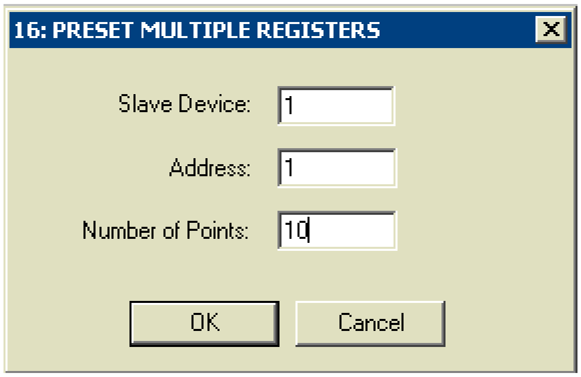
Slave Device: 1

Address: 1

Number of Points: 50

Figure 7.7: Preset Multiple register dialog

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



16: PRESET MULTIPLE REGISTERS

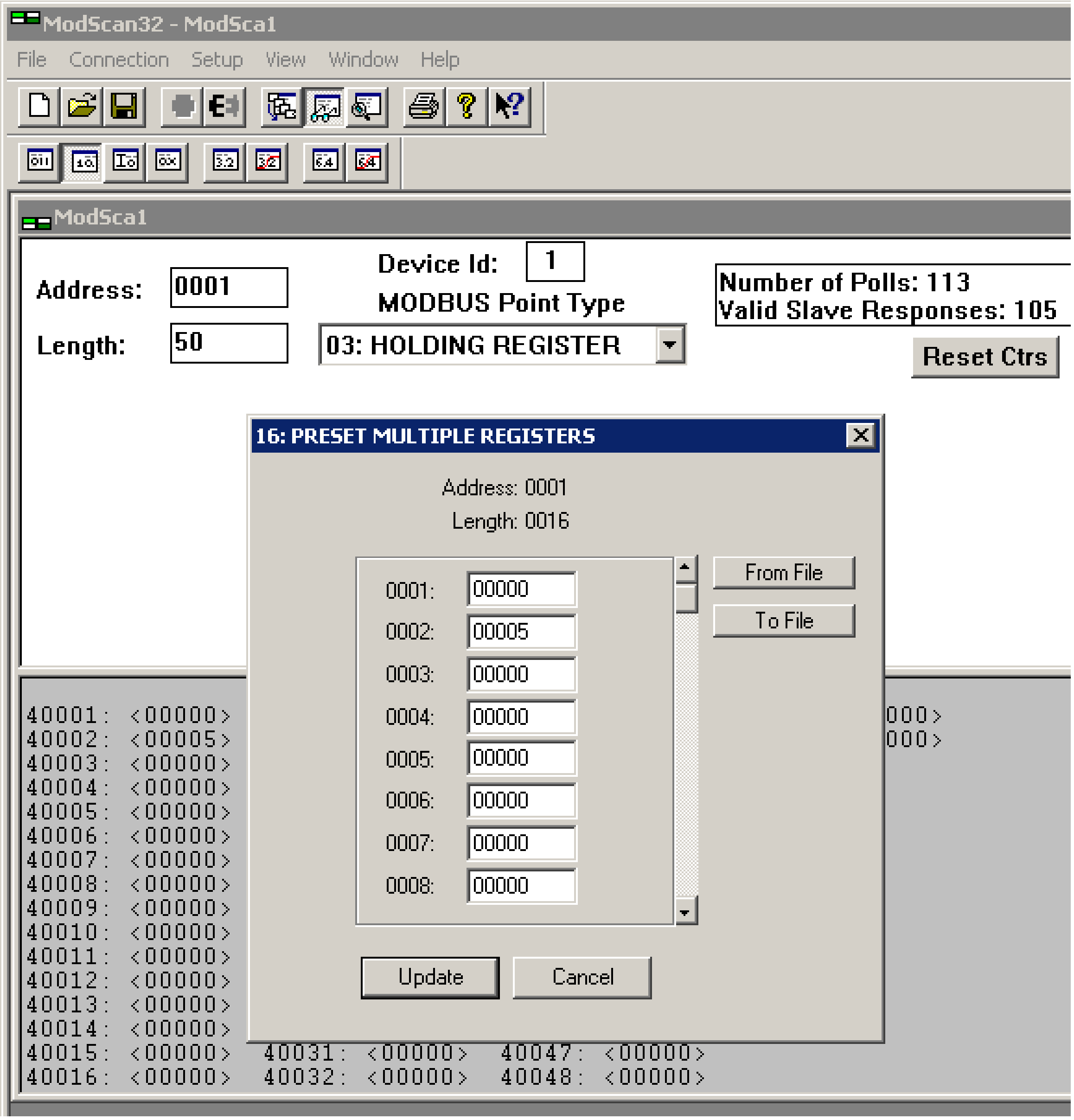
Slave Device: 1

Address: 1

Number of Points: 10

Figure 7.8: Select number of multiple writes to do

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


ModScan32 - ModSca1
File Connection Setup View Window Help
ModSca1
Reset Ctrs
16: PRESET MULTIPLE REGISTERS
From File
To File
Update
Cancel

Address: 0001

Length: 50

Device Id: 1

MODBUS Point Type: 03: HOLDING REGISTER

Number of Polls: 113
Valid Slave Responses: 105

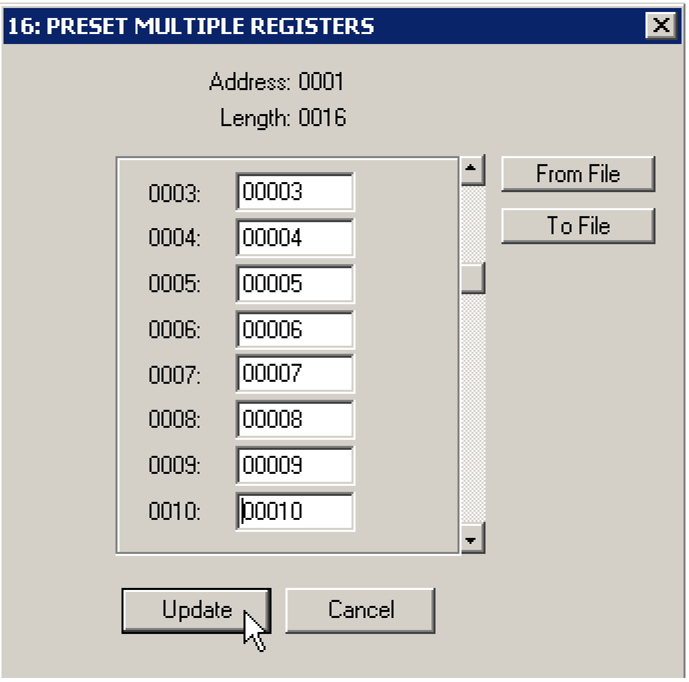
Address: 0001
Length: 0016

0001:	00000
0002:	00005
0003:	00000
0004:	00000
0005:	00000
0006:	00000
0007:	00000
0008:	00000

```
40001: <00000>                                      000>
40002: <00005>                                      000>
40003: <00000>
40004: <00000>
40005: <00000>
40006: <00000>
40007: <00000>
40008: <00000>
40009: <00000>
40010: <00000>
40011: <00000>
40012: <00000>
40013: <00000>
40014: <00000>
40015: <00000>  40031: <00000>  40047: <00000>
40016: <00000>  40032: <00000>  40048: <00000>
```

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.



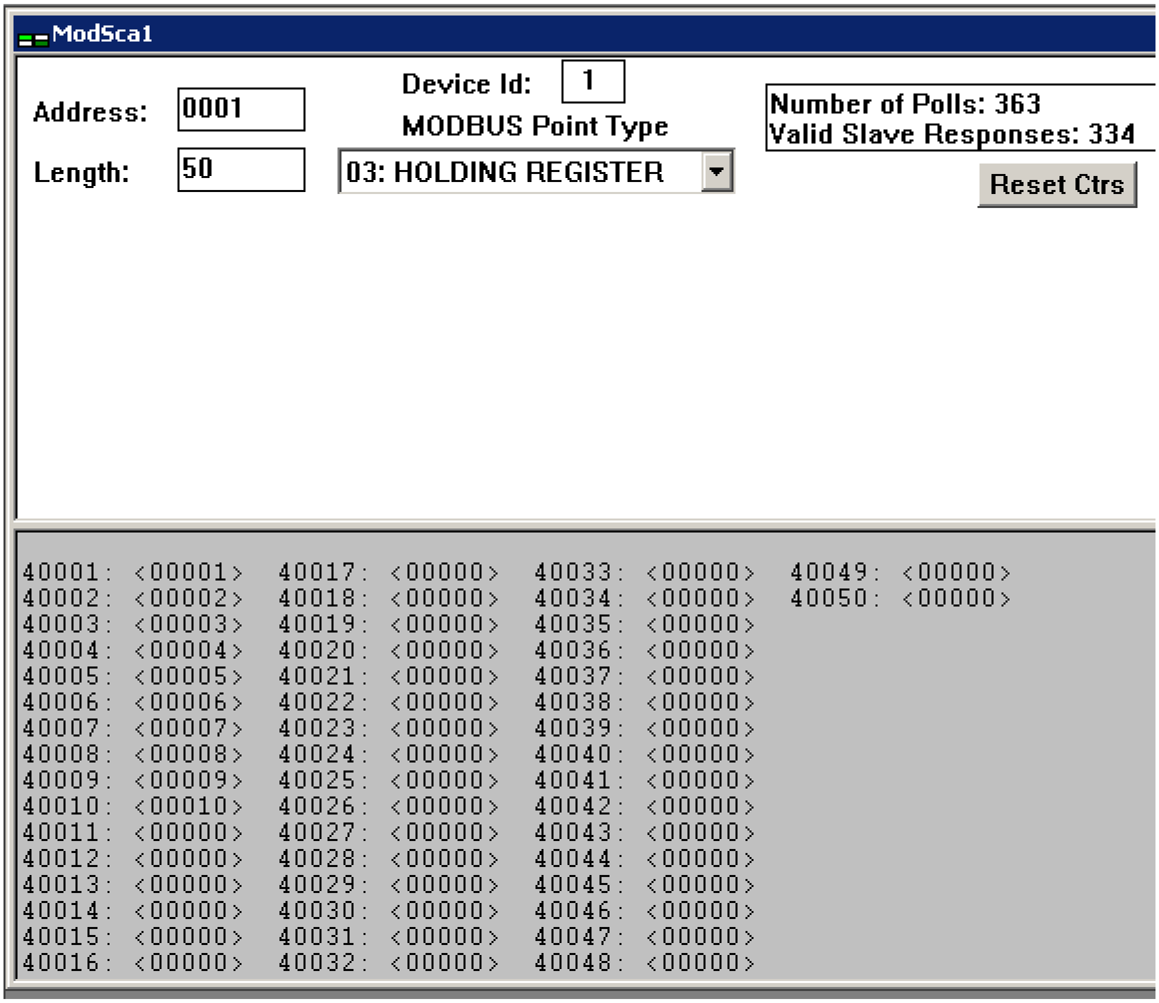
16: PRESET MULTIPLE REGISTERS

Address: 0001
Length: 0016

0003: 00003
0004: 00004
0005: 00005
0006: 00006
0007: 00007
0008: 00008
0009: 00009
0010: 00010

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.



Address: 0001

Length: 50

Device Id: 1

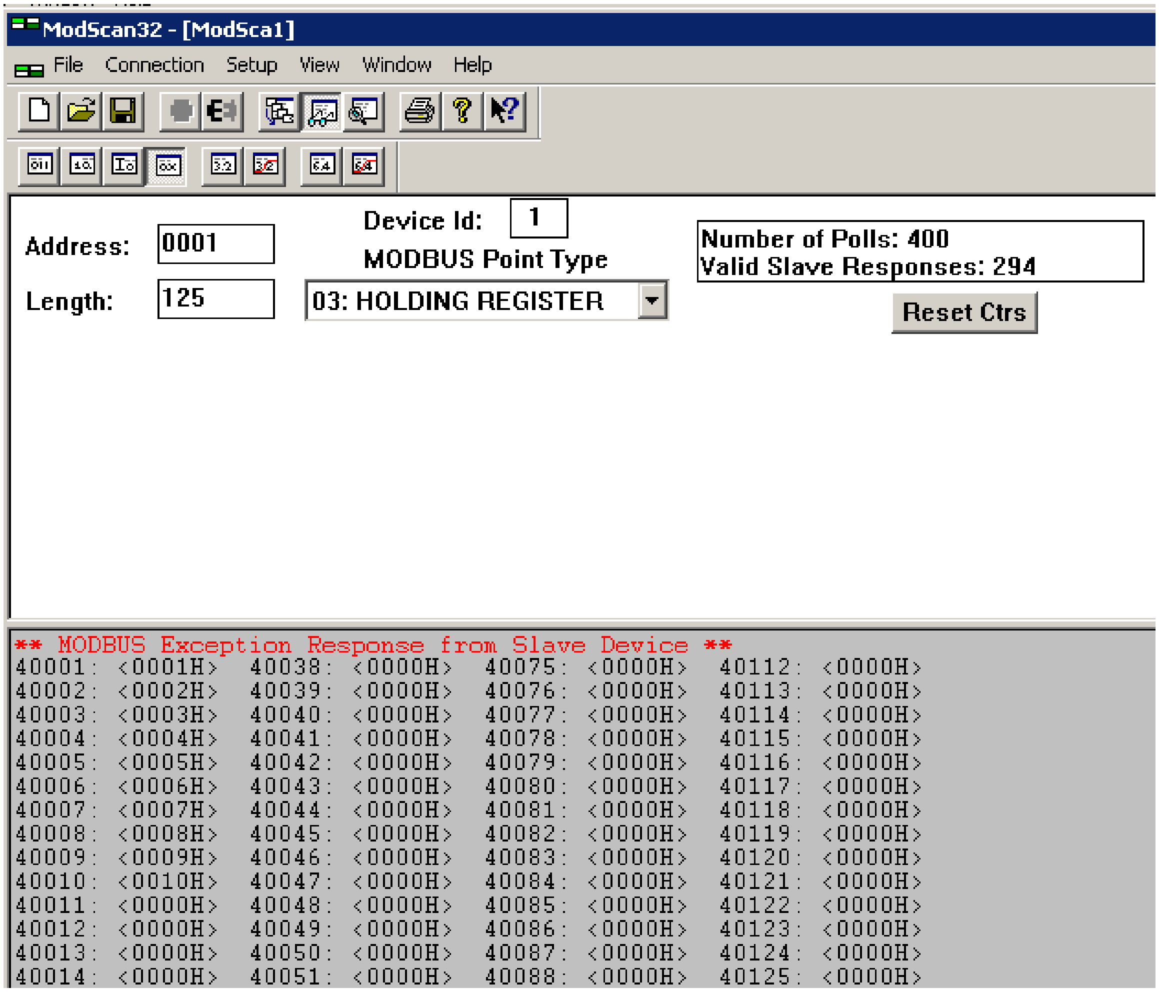
MODBUS Point Type: 03: HOLDING REGISTER

Number of Polls: 363
Valid Slave Responses: 334

```
40001: <00001>  40017: <00000>  40033: <00000>  40049: <00000>
40002: <00002>  40018: <00000>  40034: <00000>  40050: <00000>
40003: <00003>  40019: <00000>  40035: <00000>
40004: <00004>  40020: <00000>  40036: <00000>
40005: <00005>  40021: <00000>  40037: <00000>
40006: <00006>  40022: <00000>  40038: <00000>
40007: <00007>  40023: <00000>  40039: <00000>
40008: <00008>  40024: <00000>  40040: <00000>
40009: <00009>  40025: <00000>  40041: <00000>
40010: <00010>  40026: <00000>  40042: <00000>
40011: <00000>  40027: <00000>  40043: <00000>
40012: <00000>  40028: <00000>  40044: <00000>
40013: <00000>  40029: <00000>  40045: <00000>
40014: <00000>  40030: <00000>  40046: <00000>
40015: <00000>  40031: <00000>  40047: <00000>
40016: <00000>  40032: <00000>  40048: <00000>
```

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:



Address: 0001

Length: 125

Device Id: 1

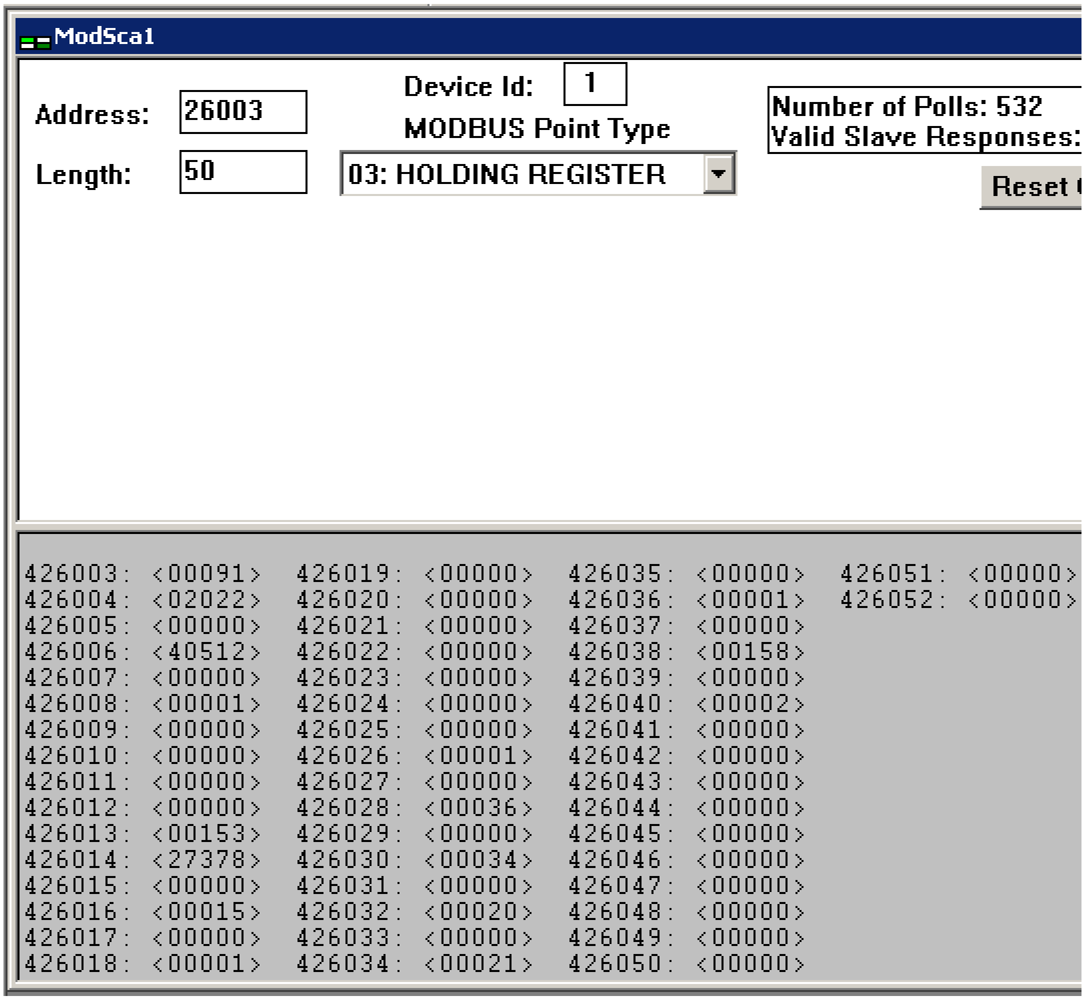
MODBUS Point Type: 03: HOLDING REGISTER

Number of Polls: 400
Valid Slave Responses: 294

```
** MODBUS Exception Response from Slave Device **
40001: <0001H>   40038: <0000H>   40075: <0000H>   40112: <0000H>
40002: <0002H>   40039: <0000H>   40076: <0000H>   40113: <0000H>
40003: <0003H>   40040: <0000H>   40077: <0000H>   40114: <0000H>
40004: <0004H>   40041: <0000H>   40078: <0000H>   40115: <0000H>
40005: <0005H>   40042: <0000H>   40079: <0000H>   40116: <0000H>
40006: <0006H>   40043: <0000H>   40080: <0000H>   40117: <0000H>
40007: <0007H>   40044: <0000H>   40081: <0000H>   40118: <0000H>
40008: <0008H>   40045: <0000H>   40082: <0000H>   40119: <0000H>
40009: <0009H>   40046: <0000H>   40083: <0000H>   40120: <0000H>
40010: <0010H>   40047: <0000H>   40084: <0000H>   40121: <0000H>
40011: <0000H>   40048: <0000H>   40085: <0000H>   40122: <0000H>
40012: <0000H>   40049: <0000H>   40086: <0000H>   40123: <0000H>
40013: <0000H>   40050: <0000H>   40087: <0000H>   40124: <0000H>
40014: <0000H>   40051: <0000H>   40088: <0000H>   40125: <0000H>
```

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 Model 5300 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.



Address: 26003

Length: 50

Device Id: 1

MODBUS Point Type: 03: HOLDING REGISTER

Number of Polls: 532
Valid Slave Responses:

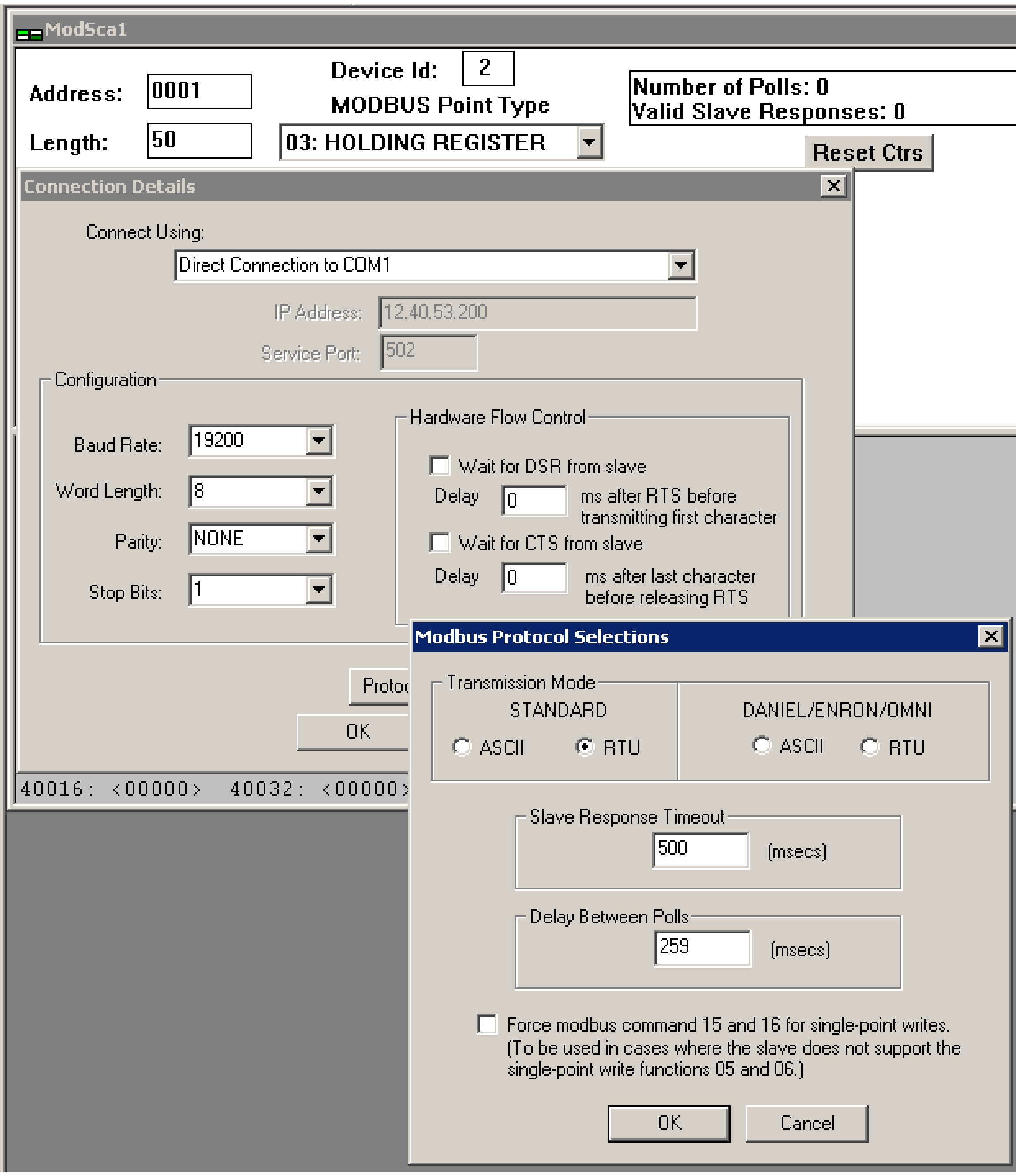
```
426003: <00091>  426019: <00000>  426035: <00000>  426051: <00000>
426004: <02022>  426020: <00000>  426036: <00001>  426052: <00000>
426005: <00000>  426021: <00000>  426037: <00000>
426006: <40512>  426022: <00000>  426038: <00158>
426007: <00000>  426023: <00000>  426039: <00000>
426008: <00001>  426024: <00000>  426040: <00002>
426009: <00000>  426025: <00000>  426041: <00000>
426010: <00000>  426026: <00001>  426042: <00000>
426011: <00000>  426027: <00000>  426043: <00000>
426012: <00000>  426028: <00036>  426044: <00000>
426013: <00153>  426029: <00000>  426045: <00000>
426014: <27378>  426030: <00034>  426046: <00000>
426015: <00000>  426031: <00000>  426047: <00000>
426016: <00015>  426032: <00020>  426048: <00000>
426017: <00000>  426033: <00000>  426049: <00000>
426018: <00001>  426034: <00021>  426050: <00000>
```

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

A maximum of 32 simultaneous Modbus TCP Slave connections are allowed at one time. Idle connections will timeout in about 1 minute.

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.



ModSca1

Address: 0001
Device Id: 2
MODBUS Point Type: 03: HOLDING REGISTER
Length: 50
Number of Polls: 0
Valid Slave Responses: 0

Connection Details

Connect Using: Direct Connection to COM1
IP Address: 12.40.53.200
Service Port: 502

Configuration
Baud Rate: 19200
Word Length: 8
Parity: NONE
Stop Bits: 1

Hardware Flow Control
☐ Wait for DSR from slave
Delay 0 ms after RTS before transmitting first character
☐ Wait for CTS from slave
Delay 0 ms after last character before releasing RTS

40016: <00000> 40032: <00000>

Modbus Protocol Selections

Transmission Mode
STANDARD: ○ ASCII ● RTU
DANIEL/ENRON/OMNI: ○ ASCII ○ RTU

Slave Response Timeout: 500 (msecs)

Delay Between Polls: 259 (msecs)

☐ Force modbus command 15 and 16 for single-point writes. (To be used in cases where the slave does not support the single-point write functions 05 and 06.)

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 CTC MON, no further communications will be available on that port except with Modbus. In other words, you will lose your CTC MON 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. The Model 5300 controller is capable of polling the Holding Registers (R/W, 4XXXX Modbus registers), Input Registers (read only by Modbus definition, 3XXXX Modbus registers), Input Status Registers (read only by Modbus definition, 1XXXX), and Coil Status Registers (R/W, 0XXXX Modbus registers) of a remote device. Write Single Holding Register (function code 0x06), Write Multiple Holding Registers (function code 0x10), Read Holding Registers (0x03), Read Coil Status (function code 0x01), Force Single Coil (function code 0x05), Read Input Status Registers (function code 0x02), and Read Input Registers (0x04) commands are supported. Multidrop mode is supported for serial ports although the exact timings of the Modbus specification may not apply, confirmation with the specific device is required. This typically is not a problem since the 5300 is less strict with character timeouts than the actual specification.

With firmware prior to R44 the Serial Modbus Master did not support multidrop or Coil/Input Registers, and the 5300 only polled a single device ID. The active device ID register also had to be changed in order to begin polling a different device. Those who required slow scanning of multiple devices changed 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/QuickBuilder program. This caused all subsequent polls to use that device ID and hence allow the reading/writing of multiple devices. With release R44 and later this is now automatic by simply defining multiple 21XXX blocks referencing the same serial port as the COM port. The Device ID should not be changed, after initially set. Each 21XXX block can now specify a unique device ID to be polled.

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 Model 5300 controller can run numerous Modbus TCP Master connections and a RTU/ASCII Serial connections 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 the Registers 21000-21299 section.

21000-21299	**TCP Peer to Peer and Modbus Master Parameters:** R/W, starts at 21000 and repeated every 10 blocks as follows: **21XX0 – First Octet IP Address Register (Most Significant) – R/W** This is the first octet of the IP address (XXX.000.000.000) to connect to. **21XX1 – Second Octet IP Address Register – R/W** This is the second octet of the IP address (000.XXX.000.000) to connect to. **21XX2 – Third Octet IP Address Register – R/W** This is the third octet of the IP address (000.000.XXX.000) to connect to. **21XX3 – Fourth Octet IP Address Register (Least Significant) – R/W** This is the fourth octet of the IP address (000.000.000.XXX) to connect to. **21XX4 – Start Register – R/W** This register stores the starting register address that is to be read from the remote 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 100 for Peer to Peer, 256 for Modbus and CTC Binary Master. 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. For Modbus Master this register is the default; see 21XXX8, 1011 to change when polling differing devices. This register must be written first to define the required storage size; upon initialization all other registers will become available. To disable an active connection write a -1 to this port and wait for a -1 to be read back before specifying new parameters. **21XX6 – Poll Timer Register – R/W** Set this register to 0 for a single read request else set the scan rate in milliseconds. The minimum value allowed is 50 ms for Peer to Peer and 10 ms for Modbus. You can write to this register at any time. **21XX7 – Status Flag Register – R** 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: 0 - Offline; no connection is present. 1 - Last request is successful and completed. Data is available in the data registers if requested. Read or Write may now be done.

-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; poll timeout on a device by Modbus Master.

-10 - Aborted operation; out of local memory or resources.

21000-
21299
Cont'd

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 array 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, this register 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. The index register also has a few special features when you set it to 1000 or above.

1000 – Peer Request Time-Out Register – (R/W) 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 –(R) 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 – (R/W) 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 – (R/W) 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 for Modbus Master RTU Serial on COM1 (TBD), 8 is CTC Binary Master UDP (port 3000), 9 is CTC Binary Master TCP (port 6000). Note the Binary Master protocol allows connection to 2700 series controllers.

1004 – TCP Client Port Index Register – (R/W) 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 and serial Master mode with a default port number of 502 for TCP and 1 for serial (COM1). Not used for the CTC Binary Master protocol. For that protocol it is fixed to 3000 for UDP and 6000 for TCP.

21000-
21299
Cont'd

1005 – Modbus Master Unit ID Index Register – (R/W) This register points the data register to the Device/Unit 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 – (R) 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 – (R/W) 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 50 milliseconds to that desired, in milliseconds.

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.

1012 – Random Index Register – (R/W) CTC Binary Master protocol only. -1 default, with 1013 of -1 means normal peer to peer automatic sequential registers. If 1013 is not -1 then this register may be used to offset to the 1014 register window. All values appear in 21XX9. Writing a -1 to this register clears the registers set by writes to the Random Register Window and restores sequential mode. Typically this register is not accessed. After setting random registers (1014) it can be used to view what has been set. In other words setting this to a 2 would cause the third (0 based) register to be selected in the Random Register Window when that index is set.

1013 – Number of Random Registers Initialized – (R Only) CTC Binary Master protocol only. Number of random registers initialized, up to 21XX5 max. -1 means none, default (sequential mode). All values appear in 21XX9. Note that this is only if random access mode is being used, else -1 means sequential, reference offset 1014 for further information.

1014 – Random Register Window – (R/W) CTC Binary Master protocol only. During initialization write the desired sequence to 21XX9 with 21XX8 set to 1014 and that will be the registers scanned, up to 256. If you do nothing sequential mode is use and accessed registers begin with that specified in 21XX4.

Each write automatically increments the index. If a 0 is set, or not all the registers are set to a value, the default will be sequential for the remaining registers. For example if 5 registers are to be scanned and only write 4 times to this register, the first 4 accessed will be what was written, since the last register was not set the sequential mode will be used for that and since this is offset 4, the starting register 21XX4 is referenced, if 21XX4 was 1, 1 + 4 (0 based), register 5 would be accessed for that slot.

Once initialized offset 1012 may be modified to change a register, base 0 (first). Writing a -1 to 1012 will clear the stored registers and put it back to sequential mode. Do not do this while connected.

1015 – Multidrop Indicator Register – (R Only) Used in Modbus Master Serial mode to indicate that the serial port being used has more than one poll definition for it and is currently in multidrop mode. 0 = normal single poll address, 1 = multidrop.

1016 – Modus Access Register – (R/W) By default Modbus Master polls the 4XXXX block of Modbus registers known as Holding Registers. To access an alternate register block set register as follows:

0 – (0XXXX) Coils, each coil represents a 32 bit value, 1 or 0, r/w.

1 – (1XXXX) Input Status (read only), each input represents a 32 bit value, 1 or 0.

3 – (3XXXX) Input Registers (read only), 16 bit value stored in 32 bit register.

4 – (4XXXX) Holding Registers, r/w, **default**, 16 bit value stored in 32 bit register.

1017 – Multidrop Offline Loop Count Register – (R/W) Used in Modbus Master Serial mode to define the number of online poll cycles to complete before attempting an offline node. By default this is 10 polls of online nodes to every one of an offline node. After each cycle a different offline node will be attempted.

Modbus Write Multiple

1996 – Block offset. By default 0, number of words to offset in data buffer array when initiating a block write operation. This can be thought of as the index into the array that would be placed in 21XX8 for 21XX9 to access the data.

1997 – Block Write length. Must be <= 21XX5. This is the number of 16 bit words to write starting at the Block Offset in the data array (21XX9).

1998 – Block Write Control. Must set to 0 in order to load the data array. 1 will cause a Modbus write multiple to occur with the set Block Write length. After processing this will be set to 0 whereupon it can be reloaded and a 1 set again for the next transmission. Monitor 21XX7 for successful completion status. In order to begin receiving again and modify Start Register 21XX4 the Block Write Control must be unlocked by writing a 2 to it, whereupon a 1 will be read back.

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
   ;;;                 if define more than one 21XXX block with
   ;;;                 same serial port will become multidrop
   ;;; 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
```

Modbus RTU Serial Master Multidrop QuickBuilder Initialization

IP address – 172.16.2.26 (can be set to any value other than –1)
Device ID – 1, 2, 3
Number of registers to read – 10 from each with Input Registers from node 3 instead of Holding Registers.
Scan time - 150 ms. (set last to initiate polling)
Starting Register - 3
Serial Port – COM3 (note that the 5300 can be supplied with RS485 COM3).
Remap registers to consecutive block beginning at registers 23001, 23011, 23021.

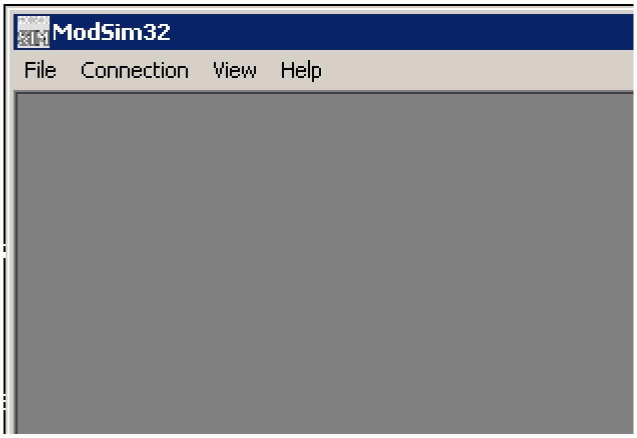
```
ip1 = 2;
ip2 = 26;
polltime = 150;  // Time between polls, this is about the fastest for USB to serial
                 // converters
/*
CTC Binary Protocol UDP:
        protocol - 8
        commport - 3000

CTC Binary Protocol TCP:
        protocol - 9
        commport - 6000

Modbus RTU Serial:
        protocol - 3
        commport - 1 for COM1, 2 for COM2, 3 for COM3, 4 for COM4

Modbus ASCII Serial:
        protocol - 4
        commport - 1 for COM1, 2 for COM2, 3 for COM3, 4 for COM4

Modbus RTU TCP:
        protocol - 2
        commport - 502
*/
commport = 3;
protocol = 3;  // RTU
timeout = 250; // Amount of milliseconds to wait till receive response.
```

```
                // This must be TX packet + RX packet time since timer starts at
                // beginning of transmission while awaiting full receive response.
                // Thus more registers read the larger this value needs to be.

// Number of multidrop nodes on this commport
numNodes = 3;

// Number of poll loops until check an offline node, default is 10, if all offline
// then every poll.
offlineLoops = 5;

// Must reset the connection on each node and wait until it is shutdown in case
// this is a restart
baseReg = 21005;  // Number of Ports definition register
for i = 0 to numNodes repeat {
        // Clear connections for each node defined
        if ($REGISTERS[baseReg + (i*10)] != -1) then
        {
                $REGISTERS[baseReg + (i*10)] = -1;   // Set Number of registers to read
                                                     // to -1 to clear old connection
                while ($REGISTERS[baseReg + (i*10)] != -1) repeat { };
        }
};

basereg = 21000;
$REGISTERS[basereg+5] = 10;           //Number of registers to read
$REGISTERS[basereg] = 172;            // IP Address, if serial set to anything
$REGISTERS[basereg+1] = 16;           // IP Address
$REGISTERS[basereg+2] = ip1;          // IP Address
$REGISTERS[basereg+3] = ip2;          // IP Address
$REGISTERS[basereg+4] = 1;            // Modbus start register 0000H
$REGISTERS[basereg+8] = 1003;         // Set index to point to protocol register
$REGISTERS[basereg+9] = protocol;     // Set protocol to that defined
$REGISTERS[basereg+8] = 1004;         // Set index to port definition
$REGISTERS[basereg+9] = commport;     // Port to that defined
$REGISTERS[basereg+8] = 1005;         // Set index to Modbus Slave ID to poll parameter
$REGISTERS[basereg+9] = 1;            // Set Slave ID to poll
$REGISTERS[basereg+8] = 1007;         // Set index to Remap register definition
$REGISTERS[basereg+9] = 23001;        // Set register to remap Modbus access to
$REGISTERS[basereg+8] = 1010;        // Set index to packet timeout
$REGISTERS[basereg+9] = timeout;      // Set packet timeout value in milliseconds.
$REGISTERS[basereg+8] = 1017;        // Set index to offline loop counter
$REGISTERS[basereg+9] = offlineLoops;// Set number of poll cycles before attempt an
                                     // offline node.
$REGISTERS[basereg+8] = 0;            // Set index to 0 when done
$REGISTERS[basereg+6] = polltime;     // Set Poll Rate in mS for this node (delay from
                                      // last packet sent to any node).
                                      // This will start everything

basereg = 21010;
$REGISTERS[basereg+5] = 10;        //Number of registers to read
$REGISTERS[basereg] = 172;         // IP Address, if serial set to anything
$REGISTERS[basereg+1] = 16;        // IP Address
$REGISTERS[basereg+2] = ip1;       // IP Address
$REGISTERS[basereg+3] = ip2;       // IP Address
$REGISTERS[basereg+4] = 11;        // Modbus start register 0000H
$REGISTERS[basereg+8] = 1003;      // Set index to point to protocol register
$REGISTERS[basereg+9] = protocol;     // Set protocol to that defined
$REGISTERS[basereg+8] = 1004;         // Set index to port definition
$REGISTERS[basereg+9] = commport;     // Port to that defined
$REGISTERS[basereg+8] = 1005;         // Set index to Modbus Slave ID to poll parameter
$REGISTERS[basereg+9] = 2;            // Set Slave ID to poll
$REGISTERS[basereg+8] = 1007;         // Set index to Remap regsiter definition
$REGISTERS[basereg+9] = 23011;        // Set register to remap Modbus access to
$REGISTERS[basereg+8] = 1010;         // Set index to packet timeout
$REGISTERS[basereg+9] = timeout;      // Set packet timeout value in milliseconds.
$REGISTERS[basereg+8] = 1017;         // Set index to offline loop counter
```

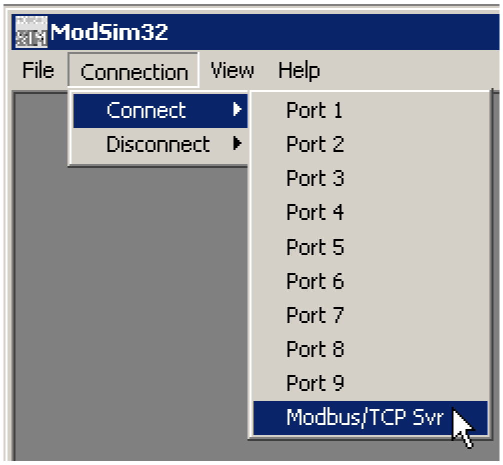
```
$REGISTERS[basereg+9] = offlineLoops;// Set number of poll cycles before attempt an
offline node.
$REGISTERS[basereg+8] = 0;           // Set index to 0 when done
$REGISTERS[basereg+6] = polltime;    // Set Poll Rate in mS for this node (delay from
                                     // last packet sent to any node).  This will start
                                     // everything

basereg = 21020;
$REGISTERS[basereg+5] = 10;          //Number of registers to read
$REGISTERS[basereg]  = 172;          // IP Address, if serial set to anything
$REGISTERS[basereg+1] = 16;          // IP Address
$REGISTERS[basereg+2] = ip1;         // IP Address
$REGISTERS[basereg+3] = ip2;         // IP Address
$REGISTERS[basereg+4] = 21;          // Modbus start register 0000H
$REGISTERS[basereg+8] = 1003;        // Set index to point to protocol register
$REGISTERS[basereg+9] = protocol;    // Set protocol to that defined
$REGISTERS[basereg+8] = 1004;        // Set index to port definition
$REGISTERS[basereg+9] = commport;    // Port to that defined
$REGISTERS[basereg+8] = 1005;        // Set index to Modbus Slave ID to poll parameter
$REGISTERS[basereg+9] = 3;           // Set Slave ID to poll
$REGISTERS[basereg+8] = 1007;       // Set index to Remap regsiter definition
$REGISTERS[basereg+9] = 23021;      // Set register to remap Modbus access to
$REGISTERS[basereg+8] = 1010;        // Set index to packet timeout
$REGISTERS[basereg+9] = timeout;     // Set packet timeout value in milliseconds.
$REGISTERS[basereg+8] = 1017;       // Set index to offline loop counter
$REGISTERS[basereg+9] = offlineLoops; // Set number of poll cycles before attempt an
                                      // offline node.
$REGISTERS[basereg+8] = 1016;         // Set index to Input Register Selection (3XXXX)
$REGISTERS[basereg+9] = 3;            // Set to 3XXXX.
$REGISTERS[basereg+8] = 0;            // Set index to 0 when done
$REGISTERS[basereg+6] = polltime;      // Set Poll Rate in mS for this node (delay from
                                      // last packet sent to any node).  This will start
                                      // everything
```

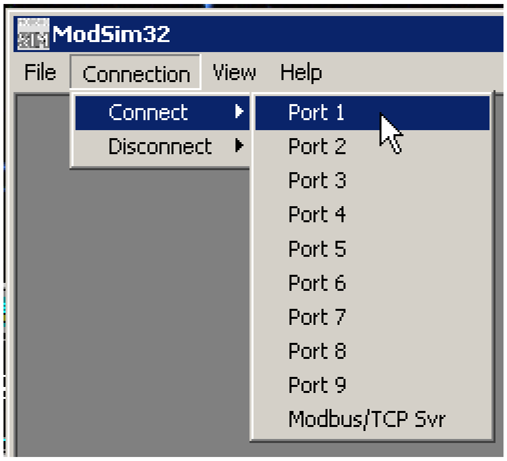
Testing with Win-Tech's ModSim32

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.

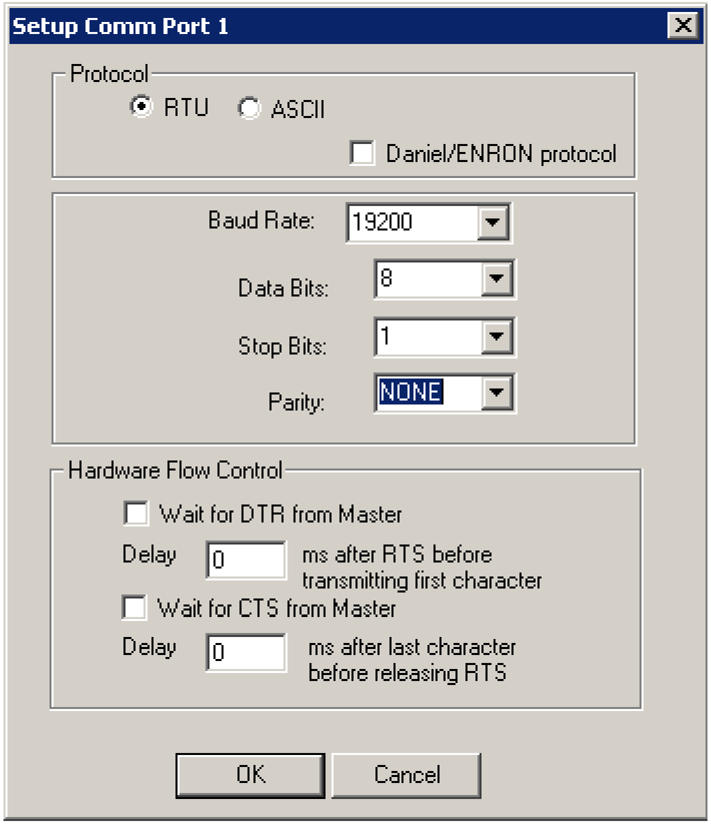


In order to activate the Modbus slave, you must select the Connection menu item and the method of the connection, Modbus/TCP Svr for network or the appropriate Port # for a serial port.





If Serial, select RTU or ASCII and set the baud rate, stop bits, and parity appropriately. Default for the Model 5300 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:



Setup Comm Port 1

Protocol

● RTU ○ ASCII

□ Daniel/ENRON protocol

Baud Rate: 19200

Data Bits: 8

Stop Bits: 1

Parity: NONE

Hardware Flow Control

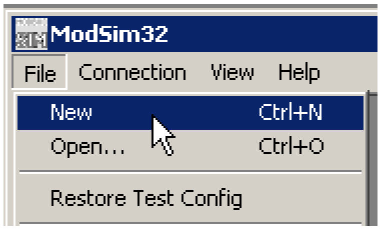
□ Wait for DTR from Master

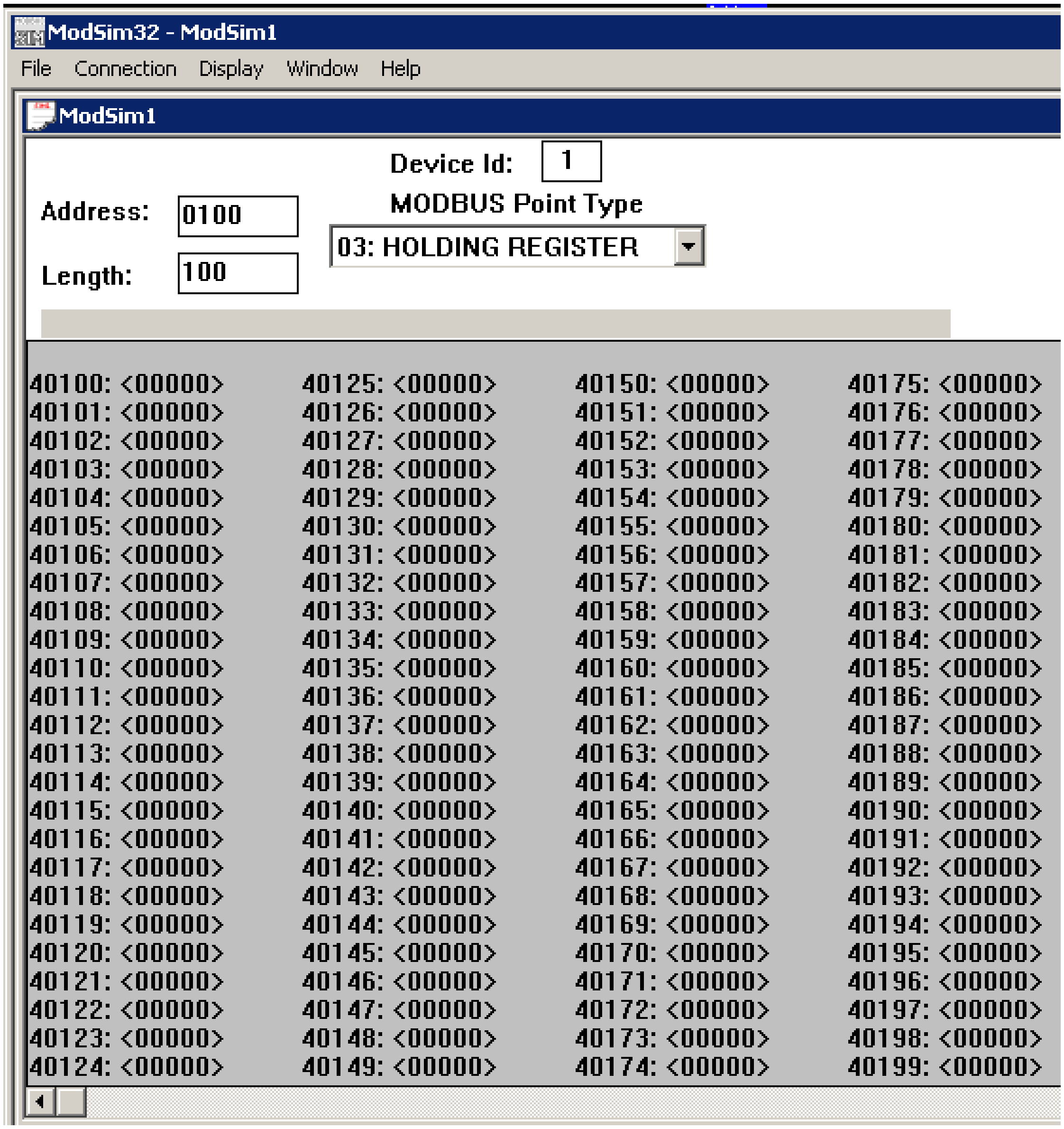
Delay 0 ms after RTS before transmitting first character

□ Wait for CTS from Master

Delay 0 ms after last character before releasing RTS

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





Device Id: 1

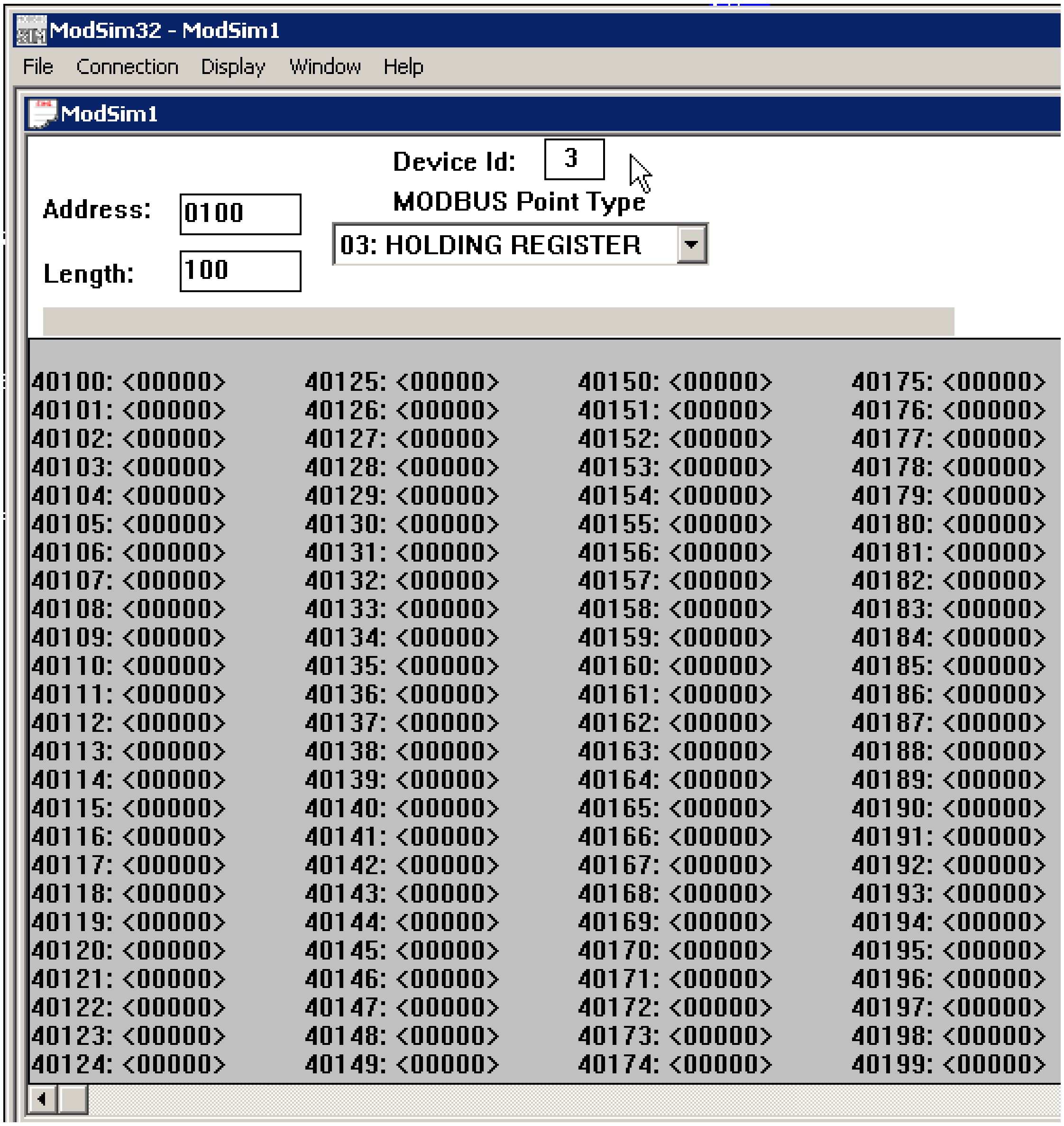
Address: 0100

MODBUS Point Type: 03: HOLDING REGISTER

Length: 100

```
40100: <00000>    40125: <00000>    40150: <00000>    40175: <00000>
40101: <00000>    40126: <00000>    40151: <00000>    40176: <00000>
40102: <00000>    40127: <00000>    40152: <00000>    40177: <00000>
40103: <00000>    40128: <00000>    40153: <00000>    40178: <00000>
40104: <00000>    40129: <00000>    40154: <00000>    40179: <00000>
40105: <00000>    40130: <00000>    40155: <00000>    40180: <00000>
40106: <00000>    40131: <00000>    40156: <00000>    40181: <00000>
40107: <00000>    40132: <00000>    40157: <00000>    40182: <00000>
40108: <00000>    40133: <00000>    40158: <00000>    40183: <00000>
40109: <00000>    40134: <00000>    40159: <00000>    40184: <00000>
40110: <00000>    40135: <00000>    40160: <00000>    40185: <00000>
40111: <00000>    40136: <00000>    40161: <00000>    40186: <00000>
40112: <00000>    40137: <00000>    40162: <00000>    40187: <00000>
40113: <00000>    40138: <00000>    40163: <00000>    40188: <00000>
40114: <00000>    40139: <00000>    40164: <00000>    40189: <00000>
40115: <00000>    40140: <00000>    40165: <00000>    40190: <00000>
40116: <00000>    40141: <00000>    40166: <00000>    40191: <00000>
40117: <00000>    40142: <00000>    40167: <00000>    40192: <00000>
40118: <00000>    40143: <00000>    40168: <00000>    40193: <00000>
40119: <00000>    40144: <00000>    40169: <00000>    40194: <00000>
40120: <00000>    40145: <00000>    40170: <00000>    40195: <00000>
40121: <00000>    40146: <00000>    40171: <00000>    40196: <00000>
40122: <00000>    40147: <00000>    40172: <00000>    40197: <00000>
40123: <00000>    40148: <00000>    40173: <00000>    40198: <00000>
40124: <00000>    40149: <00000>    40174: <00000>    40199: <00000>
```

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. To set the Device ID to a 3, as in our example, modify as below:



Device Id: 3

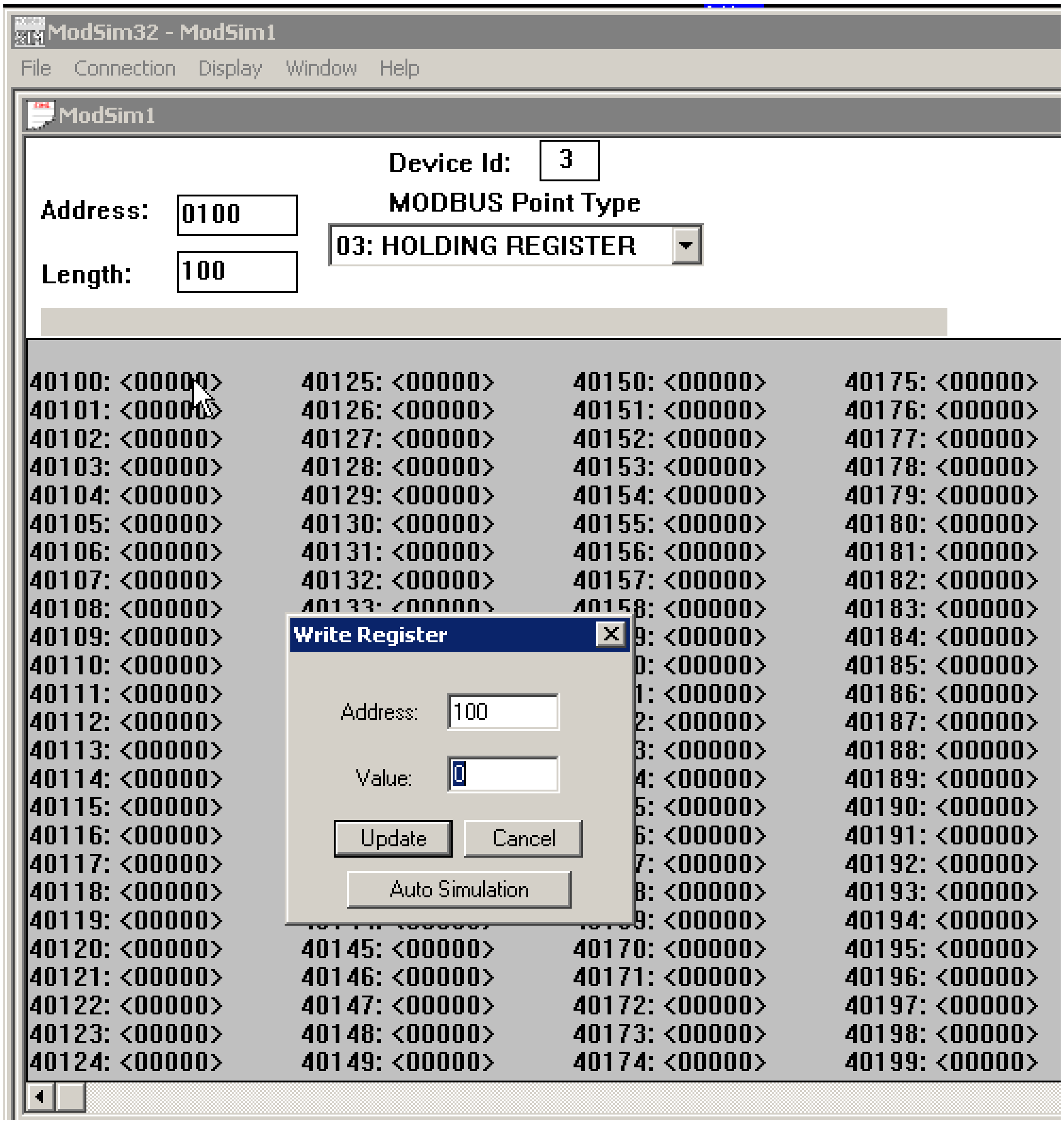
Address: 0100

Length: 100

MODBUS Point Type: 03: HOLDING REGISTER

```
40100: <00000>    40125: <00000>    40150: <00000>    40175: <00000>
40101: <00000>    40126: <00000>    40151: <00000>    40176: <00000>
40102: <00000>    40127: <00000>    40152: <00000>    40177: <00000>
40103: <00000>    40128: <00000>    40153: <00000>    40178: <00000>
40104: <00000>    40129: <00000>    40154: <00000>    40179: <00000>
40105: <00000>    40130: <00000>    40155: <00000>    40180: <00000>
40106: <00000>    40131: <00000>    40156: <00000>    40181: <00000>
40107: <00000>    40132: <00000>    40157: <00000>    40182: <00000>
40108: <00000>    40133: <00000>    40158: <00000>    40183: <00000>
40109: <00000>    40134: <00000>    40159: <00000>    40184: <00000>
40110: <00000>    40135: <00000>    40160: <00000>    40185: <00000>
40111: <00000>    40136: <00000>    40161: <00000>    40186: <00000>
40112: <00000>    40137: <00000>    40162: <00000>    40187: <00000>
40113: <00000>    40138: <00000>    40163: <00000>    40188: <00000>
40114: <00000>    40139: <00000>    40164: <00000>    40189: <00000>
40115: <00000>    40140: <00000>    40165: <00000>    40190: <00000>
40116: <00000>    40141: <00000>    40166: <00000>    40191: <00000>
40117: <00000>    40142: <00000>    40167: <00000>    40192: <00000>
40118: <00000>    40143: <00000>    40168: <00000>    40193: <00000>
40119: <00000>    40144: <00000>    40169: <00000>    40194: <00000>
40120: <00000>    40145: <00000>    40170: <00000>    40195: <00000>
40121: <00000>    40146: <00000>    40171: <00000>    40196: <00000>
40122: <00000>    40147: <00000>    40172: <00000>    40197: <00000>
40123: <00000>    40148: <00000>    40173: <00000>    40198: <00000>
40124: <00000>    40149: <00000>    40174: <00000>    40199: <00000>
```

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:



Device Id: 3

Address: 0100

Length: 100

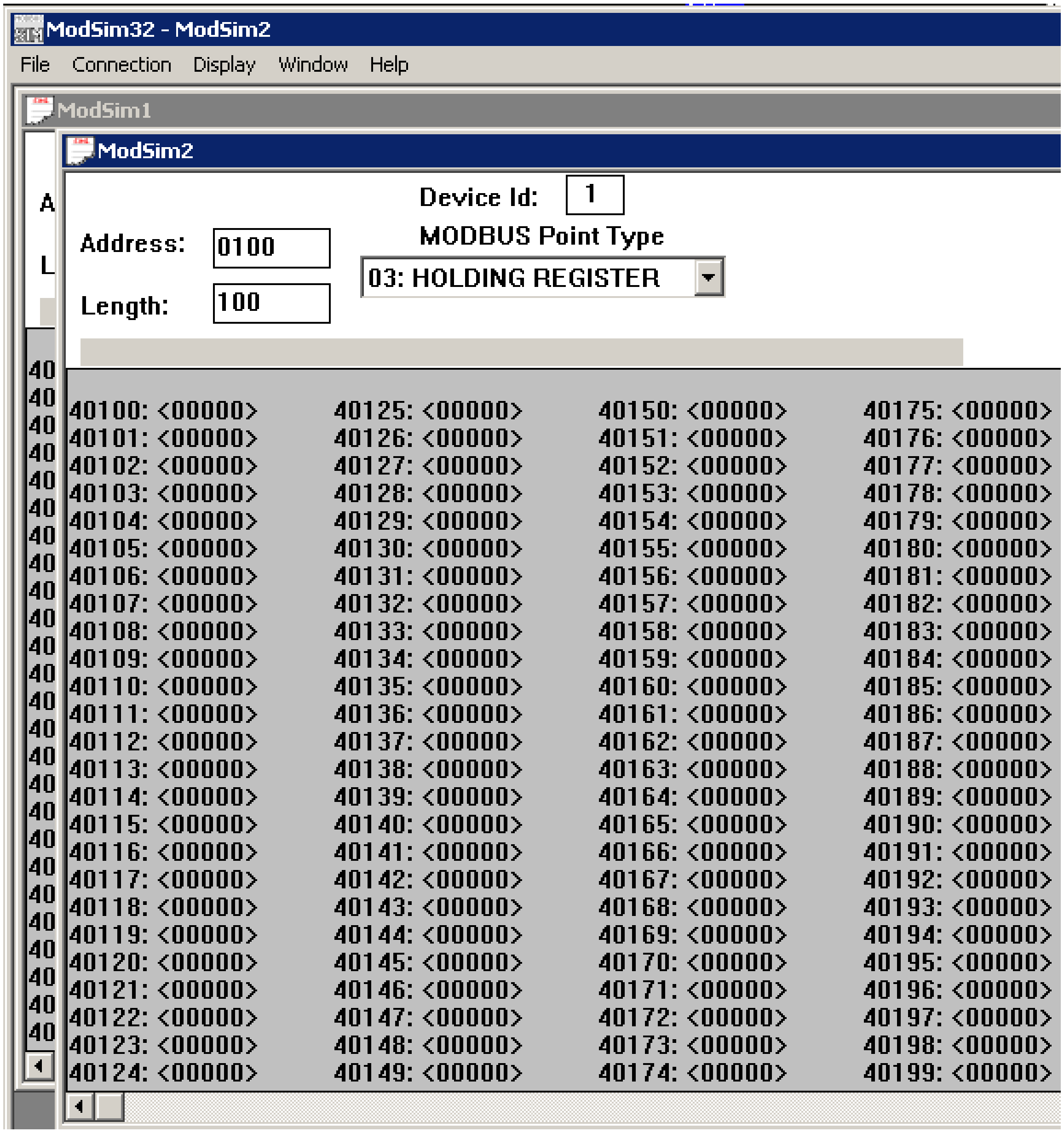
03: HOLDING REGISTER

```
40100: <00000>   40125: <00000>   40150: <00000>   40175: <00000>
40101: <00000>   40126: <00000>   40151: <00000>   40176: <00000>
40102: <00000>   40127: <00000>   40152: <00000>   40177: <00000>
40103: <00000>   40128: <00000>   40153: <00000>   40178: <00000>
40104: <00000>   40129: <00000>   40154: <00000>   40179: <00000>
40105: <00000>   40130: <00000>   40155: <00000>   40180: <00000>
40106: <00000>   40131: <00000>   40156: <00000>   40181: <00000>
40107: <00000>   40132: <00000>   40157: <00000>   40182: <00000>
40108: <00000>   40133: <00000>   40158: <00000>   40183: <00000>
40109: <00000>                        9: <00000>   40184: <00000>
40110: <00000>                        0: <00000>   40185: <00000>
40111: <00000>                        1: <00000>   40186: <00000>
40112: <00000>                        2: <00000>   40187: <00000>
40113: <00000>                        3: <00000>   40188: <00000>
40114: <00000>                        4: <00000>   40189: <00000>
40115: <00000>                        5: <00000>   40190: <00000>
40116: <00000>                        6: <00000>   40191: <00000>
40117: <00000>                        7: <00000>   40192: <00000>
40118: <00000>                        8: <00000>   40193: <00000>
40119: <00000>                        9: <00000>   40194: <00000>
40120: <00000>   40145: <00000>   40170: <00000>   40195: <00000>
40121: <00000>   40146: <00000>   40171: <00000>   40196: <00000>
40122: <00000>   40147: <00000>   40172: <00000>   40197: <00000>
40123: <00000>   40148: <00000>   40173: <00000>   40198: <00000>
40124: <00000>   40149: <00000>   40174: <00000>   40199: <00000>
```

Address: 100

Value: 0

The screen capture 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:



Device Id: 1

MODBUS Point Type: 03: HOLDING REGISTER

Address: 0100

Length: 100

```
40100: <00000>    40125: <00000>    40150: <00000>    40175: <00000>
40101: <00000>    40126: <00000>    40151: <00000>    40176: <00000>
40102: <00000>    40127: <00000>    40152: <00000>    40177: <00000>
40103: <00000>    40128: <00000>    40153: <00000>    40178: <00000>
40104: <00000>    40129: <00000>    40154: <00000>    40179: <00000>
40105: <00000>    40130: <00000>    40155: <00000>    40180: <00000>
40106: <00000>    40131: <00000>    40156: <00000>    40181: <00000>
40107: <00000>    40132: <00000>    40157: <00000>    40182: <00000>
40108: <00000>    40133: <00000>    40158: <00000>    40183: <00000>
40109: <00000>    40134: <00000>    40159: <00000>    40184: <00000>
40110: <00000>    40135: <00000>    40160: <00000>    40185: <00000>
40111: <00000>    40136: <00000>    40161: <00000>    40186: <00000>
40112: <00000>    40137: <00000>    40162: <00000>    40187: <00000>
40113: <00000>    40138: <00000>    40163: <00000>    40188: <00000>
40114: <00000>    40139: <00000>    40164: <00000>    40189: <00000>
40115: <00000>    40140: <00000>    40165: <00000>    40190: <00000>
40116: <00000>    40141: <00000>    40166: <00000>    40191: <00000>
40117: <00000>    40142: <00000>    40167: <00000>    40192: <00000>
40118: <00000>    40143: <00000>    40168: <00000>    40193: <00000>
40119: <00000>    40144: <00000>    40169: <00000>    40194: <00000>
40120: <00000>    40145: <00000>    40170: <00000>    40195: <00000>
40121: <00000>    40146: <00000>    40171: <00000>    40196: <00000>
40122: <00000>    40147: <00000>    40172: <00000>    40197: <00000>
40123: <00000>    40148: <00000>    40173: <00000>    40198: <00000>
40124: <00000>    40149: <00000>    40174: <00000>    40199: <00000>
```

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.

Blank

CHAPTER

8

[8] SNTP Simple Network Time Protocol

The Model 5300 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, which must be done locally. To avoid daylight savings time problems it is recommended that you base the controller time on GMT (default) and then use the provisions in the RTC Setup tab to automatically set the clock based on the time zone you are in, using an offset from GMT. Refer to *Document No. 951-520012: WebMON 2.0 User's Guide* for further details on the RTC Setup tab.

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:

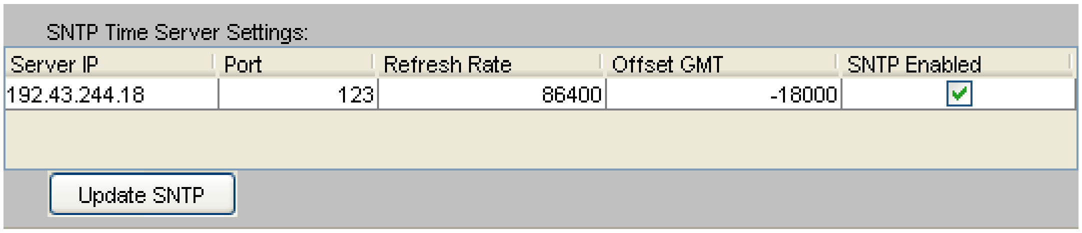
20025-20028	**SNTP Server IP Address**: R/W, with 20025 being the first Octet, X.0.0.0, 20026 the second, 0.X.0.0, 20027 is the third, 0.0.X.0, and 20028 is the fourth, 0.0.0.X. Default is 192.43.244.18 (standard).
20041	**SNTP Server Port**: R/W, default is 123.

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, number of seconds to add or subtract from GMT, default is 0.

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 consist of a number of data entry fields, each with their own special functionality:



SNTP Time Server Settings:

Server IP	Port	Refresh Rate	Offset GMT	SNTP Enabled
192.43.244.18	123	86400	-18000	☑

- 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 that 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, which 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 SNTP Enabled 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

[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. Refer to *Document 951-520012: WebMON 2.0 User's Guide* for additional information on creating messages with WebMON.

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.

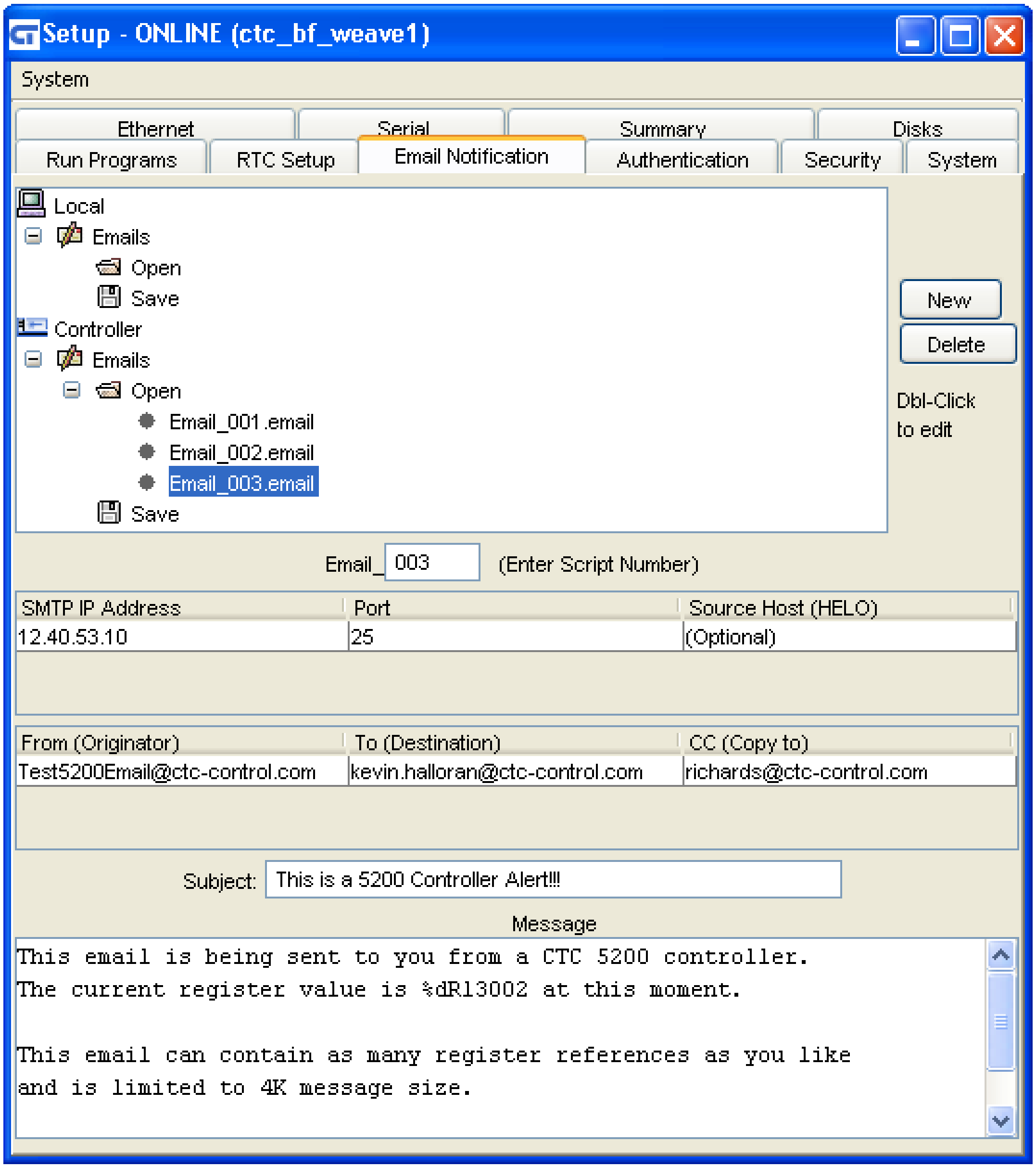
Register Access

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, cannot connect to host.

Creating Emails using WebMON

The Email Notification tab in WebMON can be used to automatically create, edit, and delete email files.



Email_ 003 (Enter Script Number)

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

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

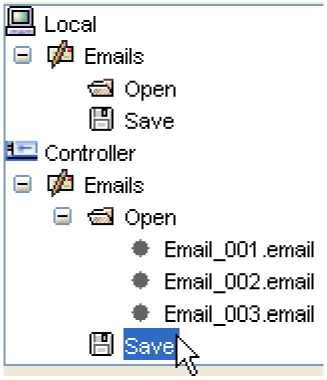
Subject: This is a 5200 Controller Alert!!!

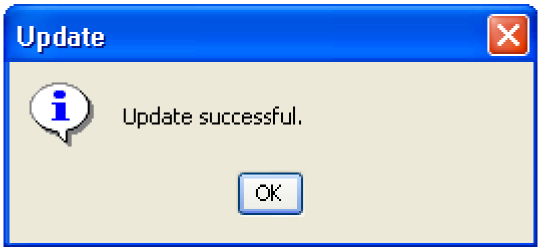
```
This email is being sent to you from a CTC 5200 controller.
The current register value is %dR13002 at this moment.

This email can contain as many register references as you like
and is limited to 4K message size.
```

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->Emails references the local disk drive of the computer running WebMON. Selecting Local->Emails->Open will cause a dialog box to open and the selection of any email file for editing purposes. Selecting Local->Emails->Save will cause a dialog box to open and an email that is within the form at the bottom will 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->Emails->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->Emails->Save node. The file will be saved and named using the Script Number defined within the email template, `Email###.email`.





Update successful.

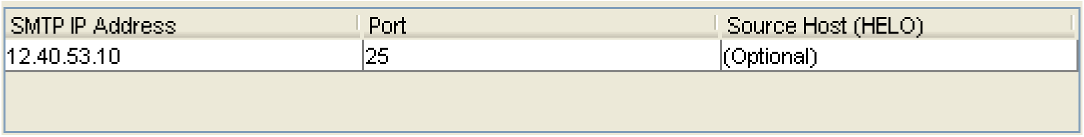
Creating/Editing New Email Template

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 IP Address (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

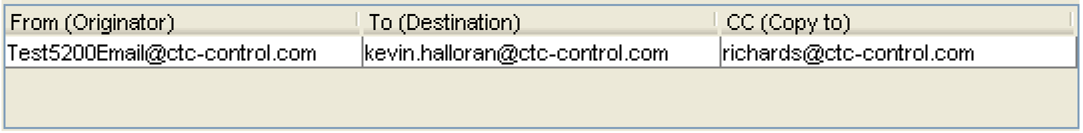
The standard SMTP port used is 25; but it may be changed here if desired. This is the port the SMTP server will be listening on for connection requests.

Source Host (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(Originator)], who it is to be sent to [TO(Destination)], and who it is to be copied to [CC(Copy to)]. 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.

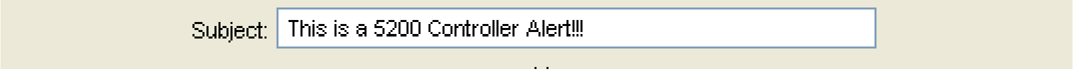
The required format of each email address is person@domain.com. Enter each as needed. Note the CC (Copy to) field is optional:



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

Subject

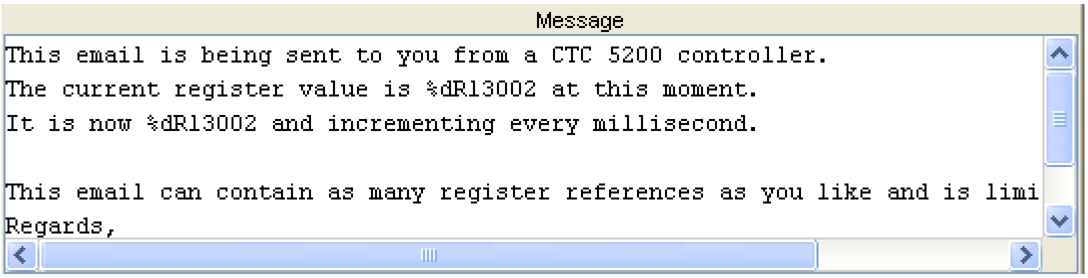
The Subject line will appear as the summary in an email message. Enter any desired text:



Subject: This is a 5200 Controller Alert!!!

Message

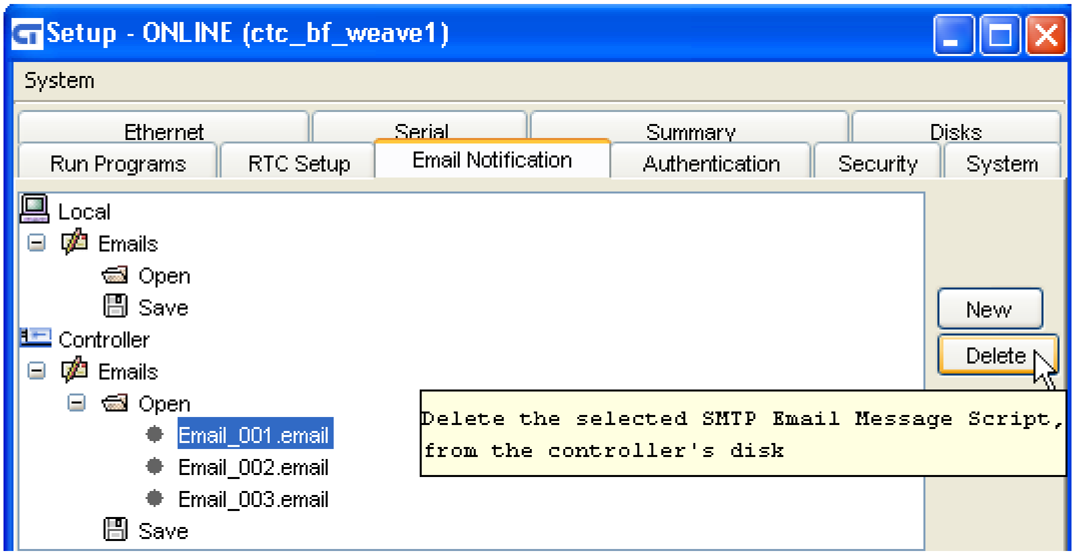
The Message area can contain up to 4K bytes of data. Messages may be any mix of normal text characters and references to Controller registers. Registers are references using "C" style printf directives. For example, to reference the 13002 register and have its contents placed in a message a %dR13002 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:



```
This email is being sent to you from a CTC 5200 controller.
The current register value is %dR13002 at this moment.
It is now %dR13002 and incrementing every millisecond.

This email can contain as many register references as you like and is limi
Regards,
```

Deleting Email Template

Deleting an email is only supported from a controller disk. To delete a file use the Controller->Emails->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.



```
Delete the selected SMTP Email Message Script,
from the controller's disk
```

Creating Emails using ASCII Text Editor

The text used to create emails, to be sent by a Model 5300 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=Test5300@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 name, it may be a DNS resolvable name or an 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 that 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 addressee may be listed per `TO` entry, 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. Otherwise, relaying must be enabled.

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

BCC= Optional field, defining the destination to copy the email to. Only one addressee 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 by other addressees.

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. Refer to 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.

[SMTP] - New to the 5300 firmware release R69.20 is the support of the AUTH LOGIN sequence used by a number of public email sites. One in particular is www.gmx.com. This web site can be used for both POP3 and SMTP email. Unlike sites like gmail and hotmail it does not require SSL/TLS encryption. To enable the AUTH LOGIN feature simply specify a USER and PASSWORD definition within the [SMTP] section:

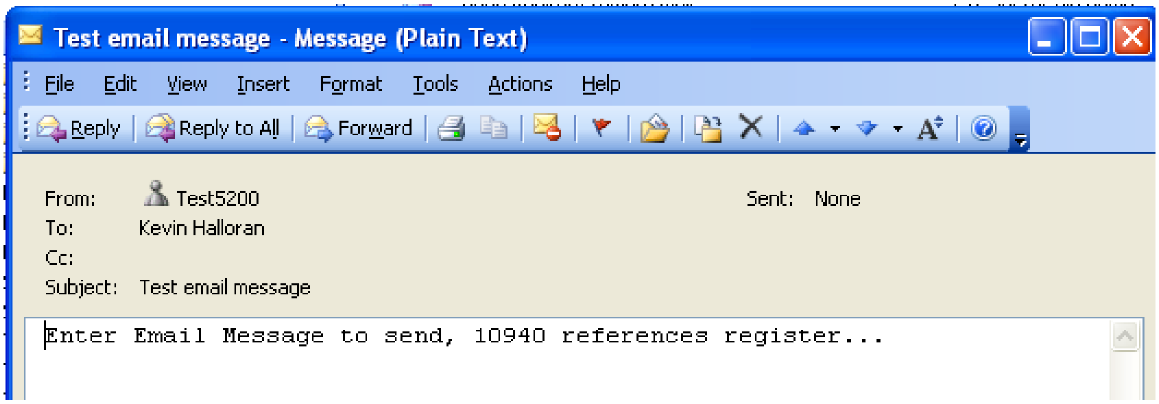
USER= User account name, limited to 80 characters. For GMX mail server this is your email address.

PASSWORD= Email account password, limited to 80 characters.

GMX email account sample reference:

```
[SMTP]
IP=smtp.gmx.com
PORT=25
HELO=
TO=somename@gmail.com
FROM=bluefusion@gmx.com
CC=
SUBJECT=test GMX Email
USER=bluefusion@gmx.com
PASSWORD=BlueFusion
[SMTP_MESSAGE]
Testing GMX email, %dR13002 references register...
```

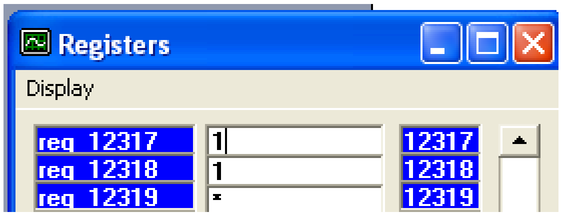
Below is an Email message sent and received when the sample email file was stored to Email_001.email within the /_system/Emails sub-directory, and a 1 was written to the SMTP Send Register 12317:


Test email message - Message (Plain Text)
File Edit View Insert Format Tools Actions Help
Reply
Reply to All
Forward

From: Test5200
Sent: None
To: Kevin Halloran
Cc:
Subject: Test email message

```
Enter Email Message to send, 10940 references register...
```

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:



```
reg 12317   1   12317
reg 12318   1   12318
reg 12319   *   12319
```

Monitored with CTCMON

Notice that the %05dR13002 was replaced by the actual register value in the controller (10940) at the time the email was composed for transmission.

SMTP Email Diagnostics

Trying to resolve SMTP problems can be difficult without network tracing capability. To help in this matter the ‘test esmtp’ and ‘test smtp’ commands can be executed via the telnet administrator interface. These commands allow you to send a sample email and monitor what is being sent and received, typically yielding better diagnostic information:

```
test smtp <host> <from> <to>     (no login is used, internal email server)

test esmtp <host> <from> <to> <username> <password>    (uses the AUTH LOGIN sequence)
```

Example of Diagnostic Trace:

```
BlueFusion/>test esmtp smtp.gmx.com bluefusion@gmx.com someone@gmail.com
bluefusion@gmx.com BlueFusion
220 mail.gmx.com GMX Mailservices ESMTP {mp-us002}
helo
250 mail.gmx.com GMX Mailservices {mp-us002}
AUTH LOGIN
334 VXNlcm5hbWU6
Ymx1ZWZ1c2lvbkBnbXguY29t
334 UGFzc3dvcmQ6
Qmx1ZUZ1c2lvbg==
235 2.7.0 Go ahead {mp-us002}
mail from:<bluefusion@gmx.com>
250 2.1.0 ok {mp-us002}
rcpt to:<someone@gmail.com>
250 2.1.5 ok {mp-us002}
data
354 mail.gmx.com Go ahead {mp-us002}
Date: 10/14/2010 20:24:29
From: bluefusion@gmx.com
To: someone@gmail.com
CC:
```

```
BCC:
Subject: 5300 Email Test
This is the first line.
This is the second.
This is register 13002 = 0625
.
250 2.6.0 Message accepted {mp-us002}
quit
221 2.0.0 GMX Mailservices {mp-us002}
SUCCESS
```

CHAPTER 10

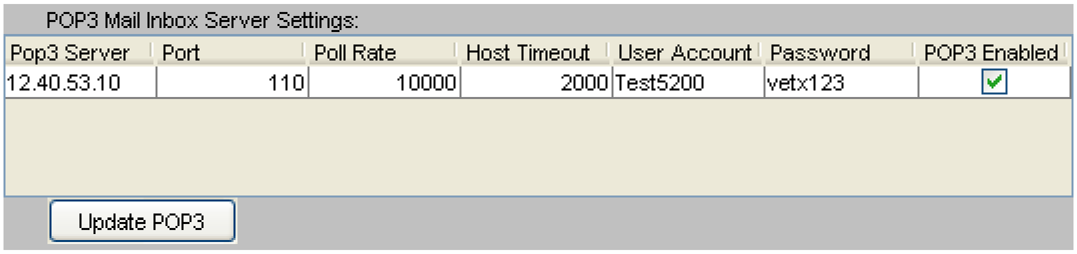
[10] POP3

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 set up using WebMON via the Ethernet Setup tab. It consists of a number of data entry fields, each with their own special functionality:



POP3 Mail Inbox Server Settings:

Pop3 Server	Port	Poll Rate	Host Timeout	User Account	Password	POP3 Enabled
12.40.53.10	110	10000	2000	Test5200	vetx123	☑

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

POP3 Server

The POP3 Server IP address designates the host that will provide the POP3 mailbox account for the controller. This must be the server's 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, which 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 Account

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

Password

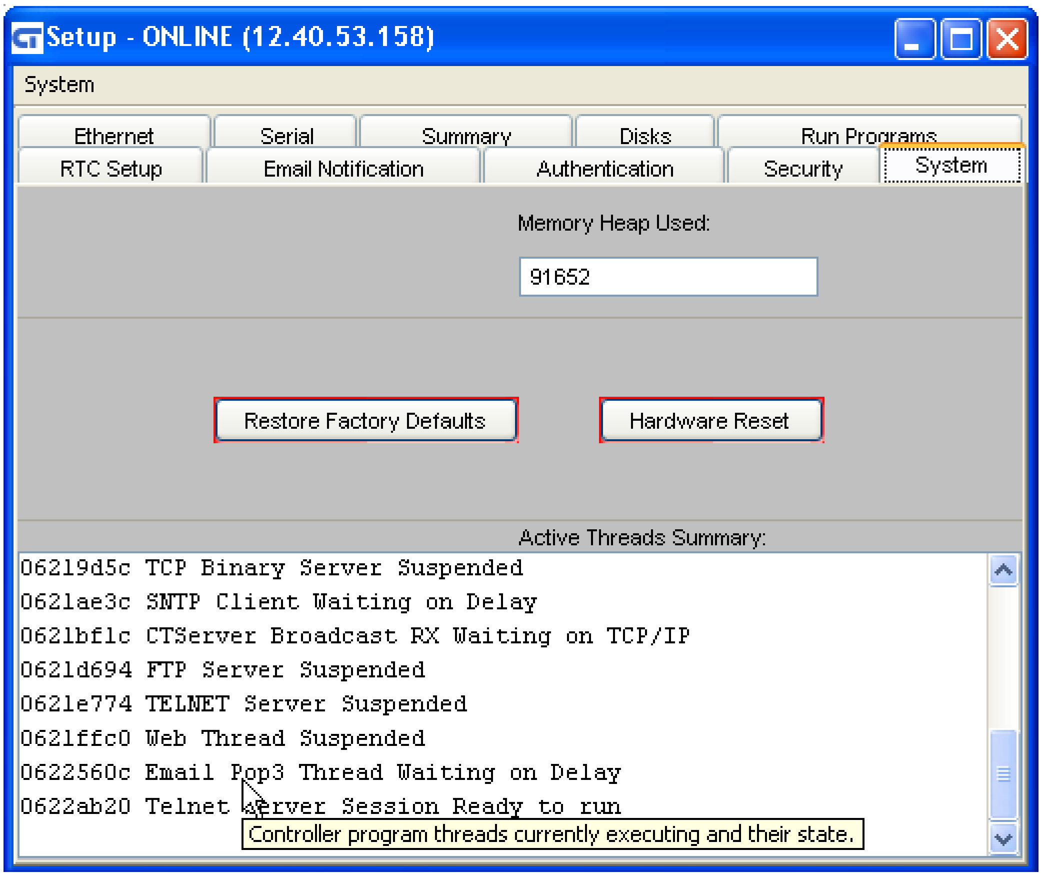
The Password is the password required, along with the User Account, to log into the mailbox being supplied by the POP3 server. It is limited to 30 characters.

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:



Memory Heap Used:

91652

Active Threads Summary:

```
06219d5c TCP Binary Server Suspended
0621ae3c SNTP Client Waiting on Delay
0621bf1c CTServer Broadcast RX Waiting on TCP/IP
0621d694 FTP Server Suspended
0621e774 TELNET Server Suspended
0621ffc0 Web Thread Suspended
0622560c Email Pop3 Thread Waiting on Delay
0622ab20 Telnet Server Session Ready to run
```

Controller program threads currently executing and their state.

Email Formatting

Once the Model 5300 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 that exist within the message body of an email are defined as follows:

[CTC_EMAIL_START] – Script commands follow as defined within *Document No. 951-530003: Model 5300 Script Language Guide*. 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 any text 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.

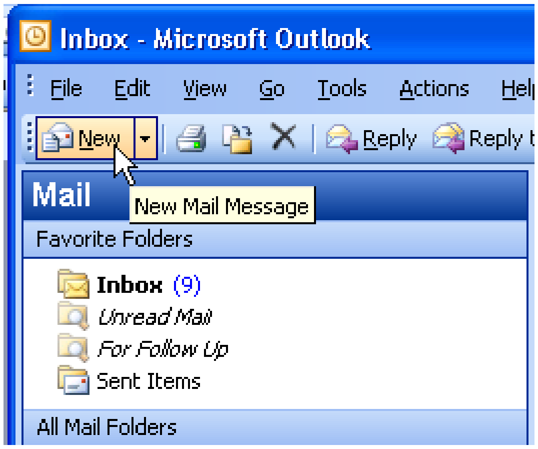
ASCII Text Emails

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.

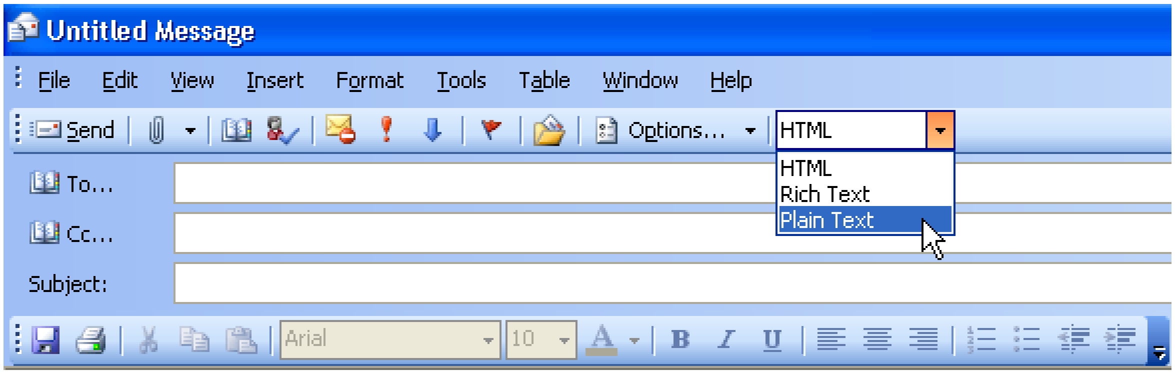
There are a number of ways to make this selection. Using Microsoft Outlook 2003 as an example you may set this as the default to always use or select it on an individual email basis.

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:



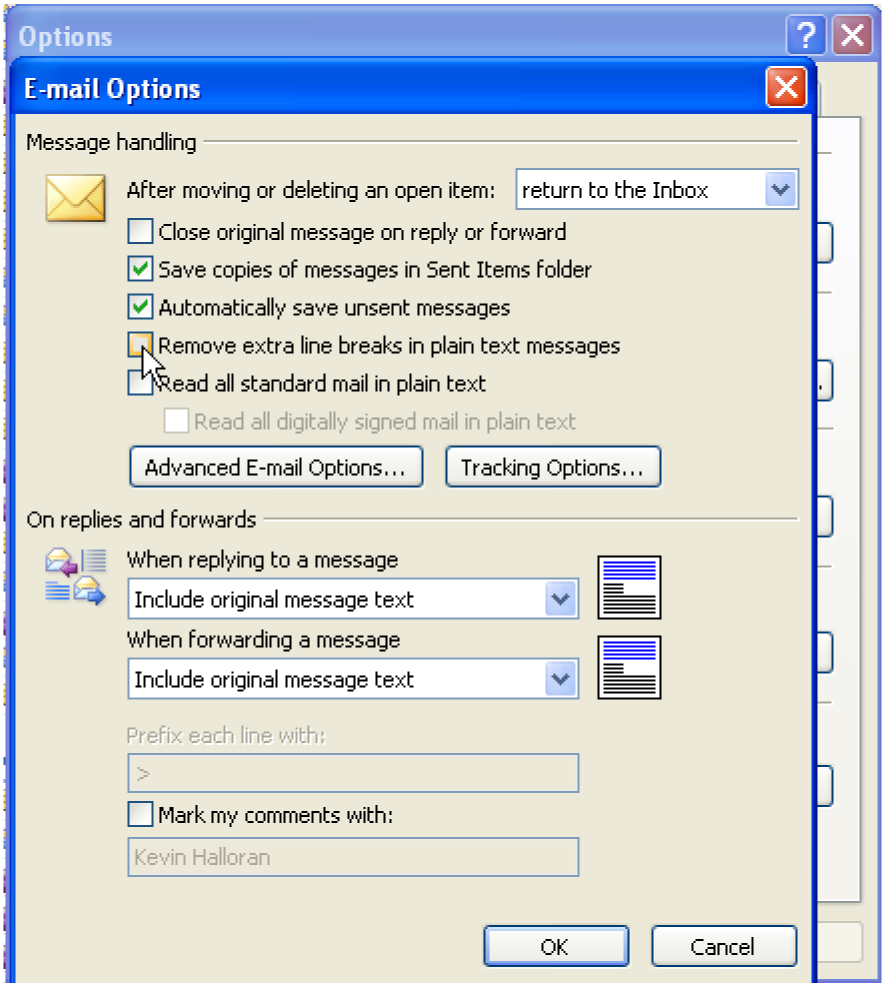
A window will appear to compose the email, note the pull down box and ensure it is selected to Plain Text.



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 sets it at 72 characters 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:



Message handling

After moving or deleting an open item: return to the Inbox

□ Close original message on reply or forward

☑ Save copies of messages in Sent Items folder

☑ Automatically save unsent messages

□ Remove extra line breaks in plain text messages

□ Read all standard mail in plain text

□ Read all digitally signed mail in plain text

On replies and forwards

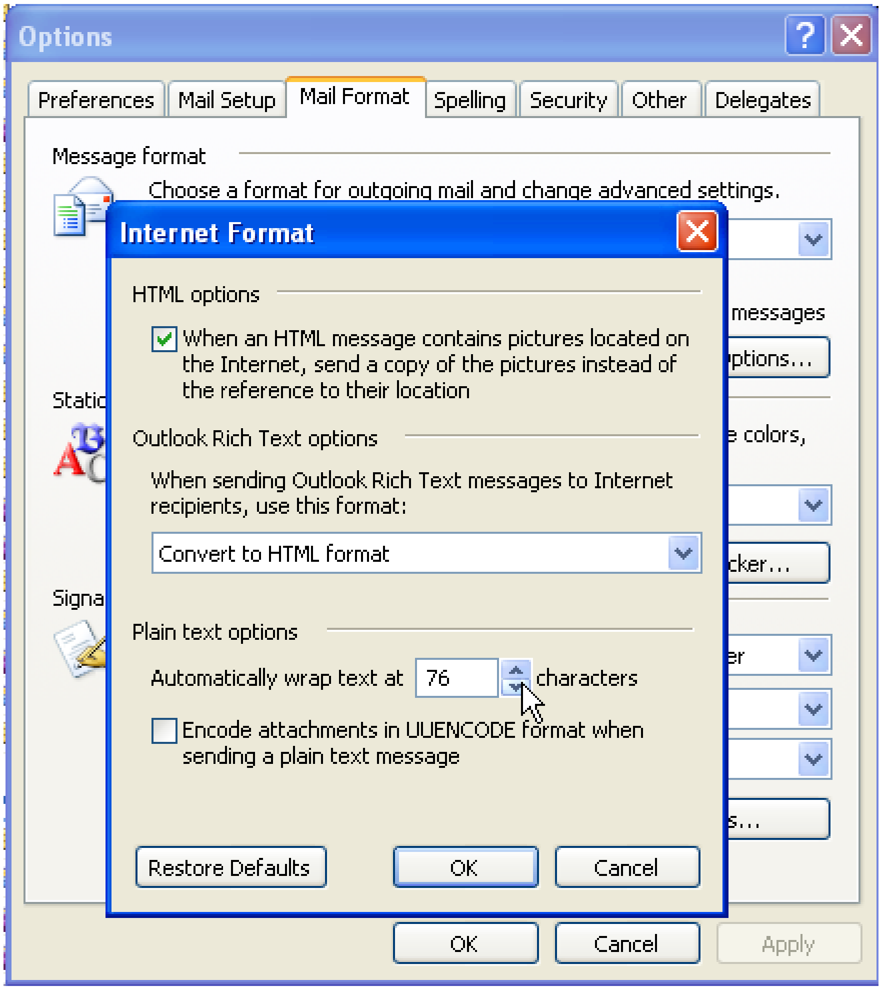
When replying to a message: Include original message text

When forwarding a message: Include original message text

Prefix each line with: >

□ Mark my comments with: Kevin Halloran

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



Choose a format for outgoing mail and change advanced settings.

HTML options

☑ When an HTML message contains pictures located on the Internet, send a copy of the pictures instead of the reference to their location

Outlook Rich Text options

When sending Outlook Rich Text messages to Internet recipients, use this format:

Convert to HTML format

Plain text options

Automatically wrap text at 76 characters

□ Encode attachments in UUENCODE format when sending a plain text message

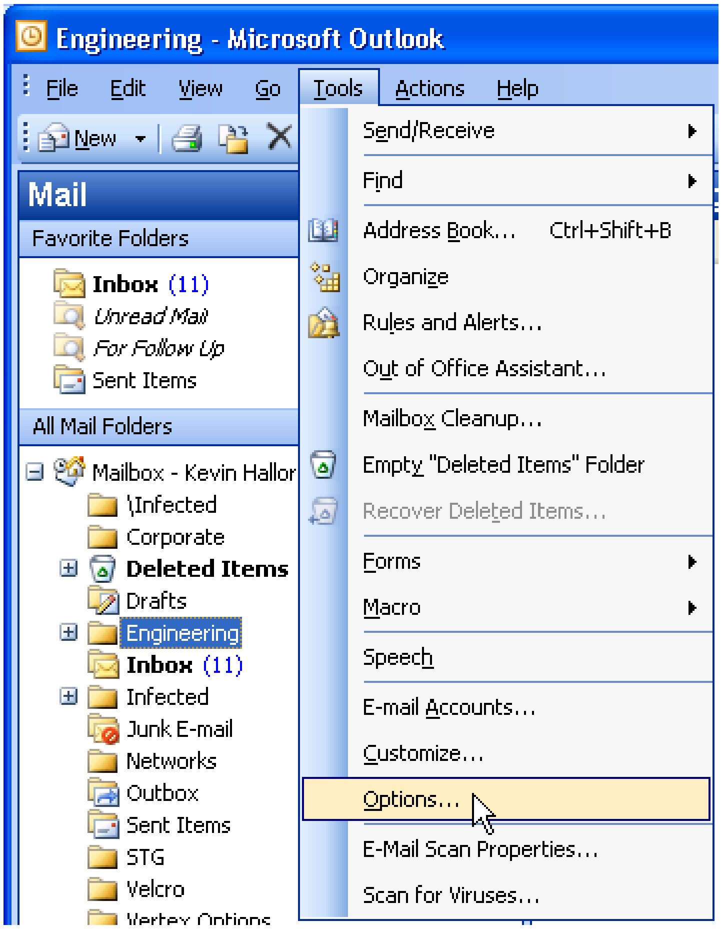
Some Microsoft Knowledgebase Articles worth referencing are 287816 and 327573:

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

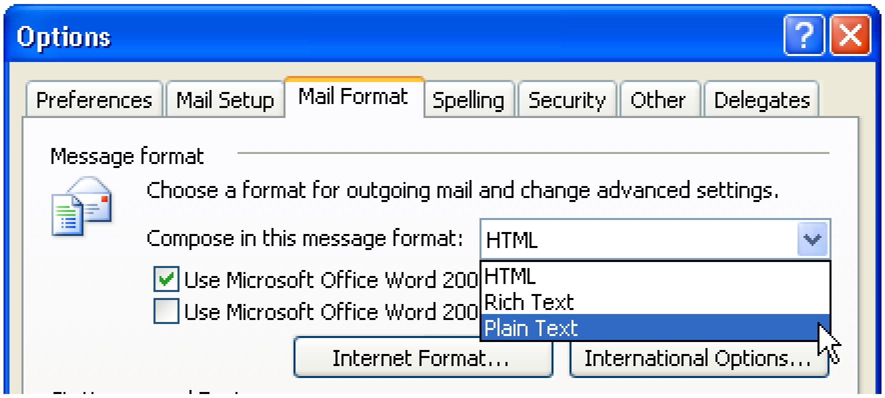
http://support.microsoft.com/default.aspx?scid=kb%3BEN-US%3Bq287816

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 in this message format pull down to Plain Text.



Choose a format for outgoing mail and change advanced settings.

Compose in this message format: HTML

HTML
Rich Text
Plain Text

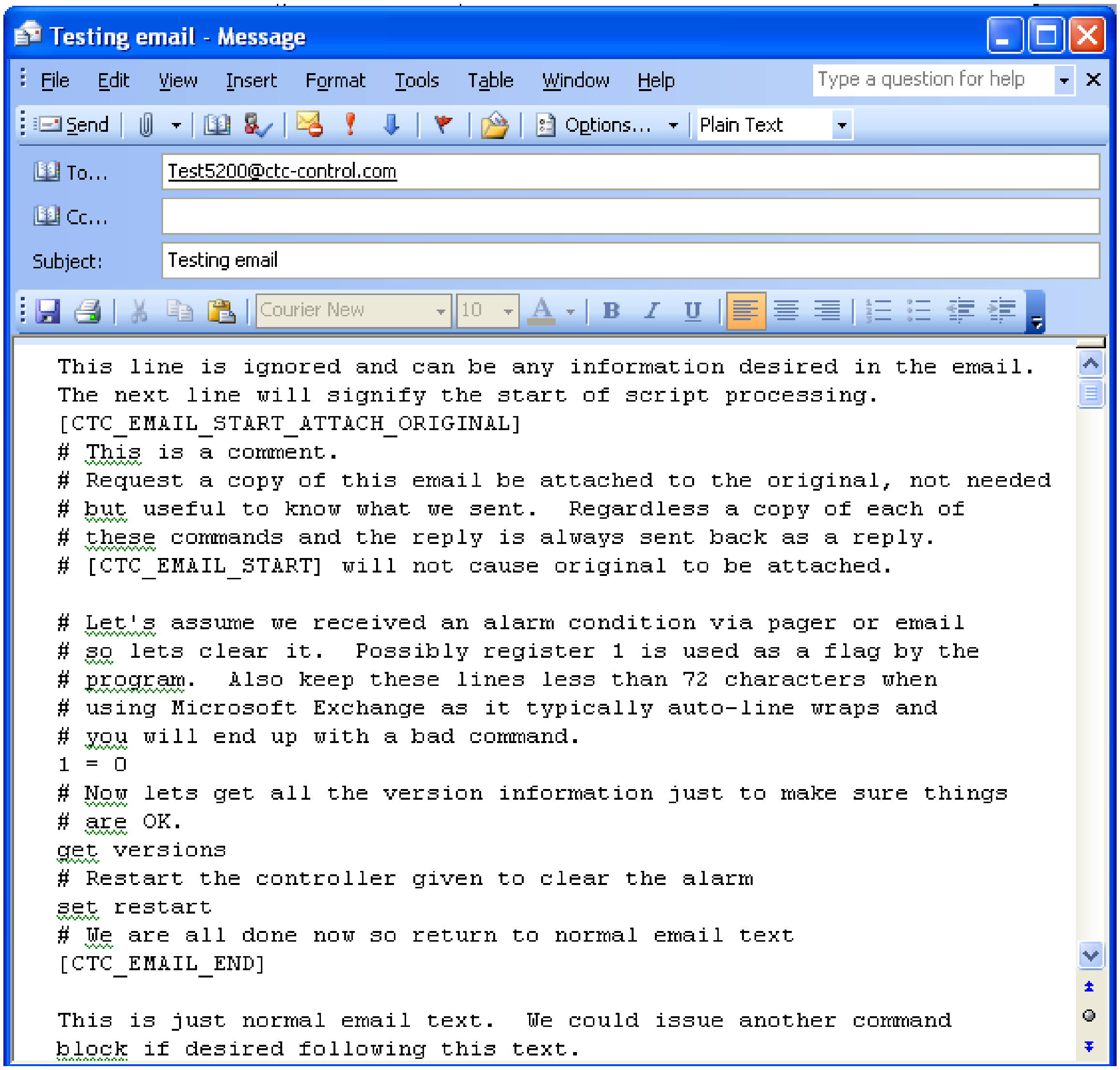
☑ Use Microsoft Office Word 200

☐ Use Microsoft Office Word 200

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:



To... Test5200@ctc-control.com

Cc...

Subject: Testing email

```
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.
```

Upon clicking Send the email will be sent to mail server where the ‘Test5300’ 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
```

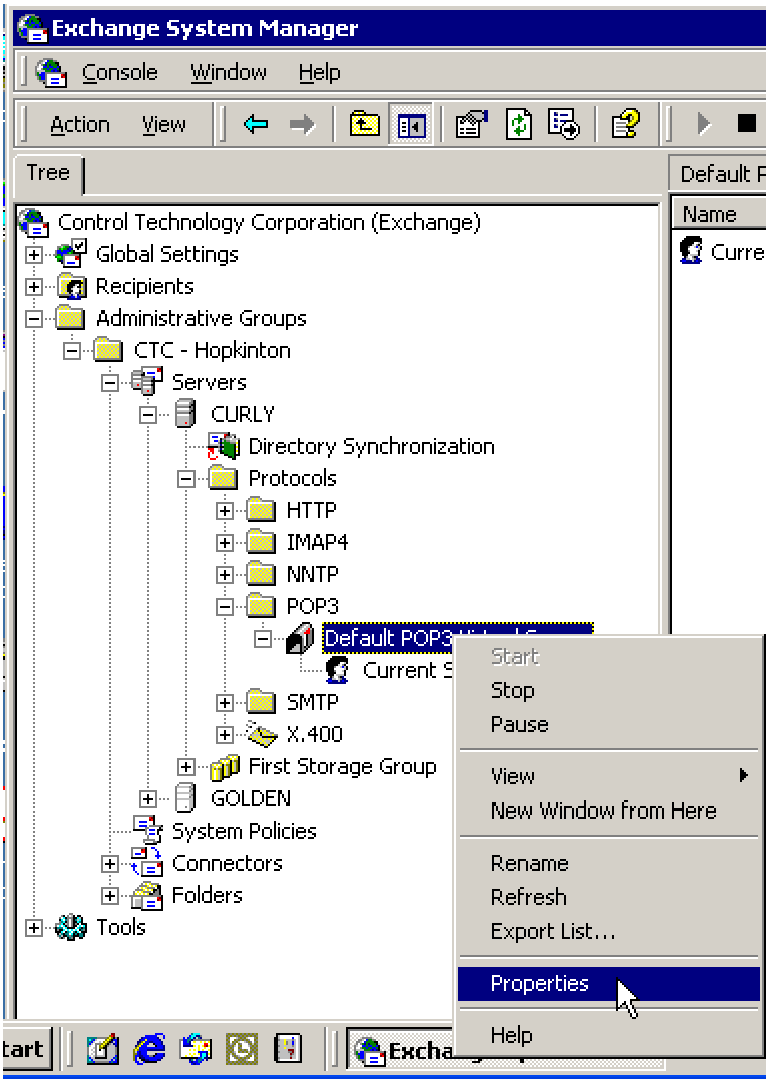
```
*Local 5300 Serial Number = 00063255
  DNS Name: 5300Kev  DHCP active: YES
  Group Name: Sales.DemoUnits
  IP Address = 12.40.53.158 MAC Address = 00C0CB00F717
  Total:  DIN = 4  DOUT = 16  AIN = 8  AOUT = 4  MOTION = 0
  Base Firmware Revisions:
    Quickstep SH2 Application           V05.00.11
    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.
```

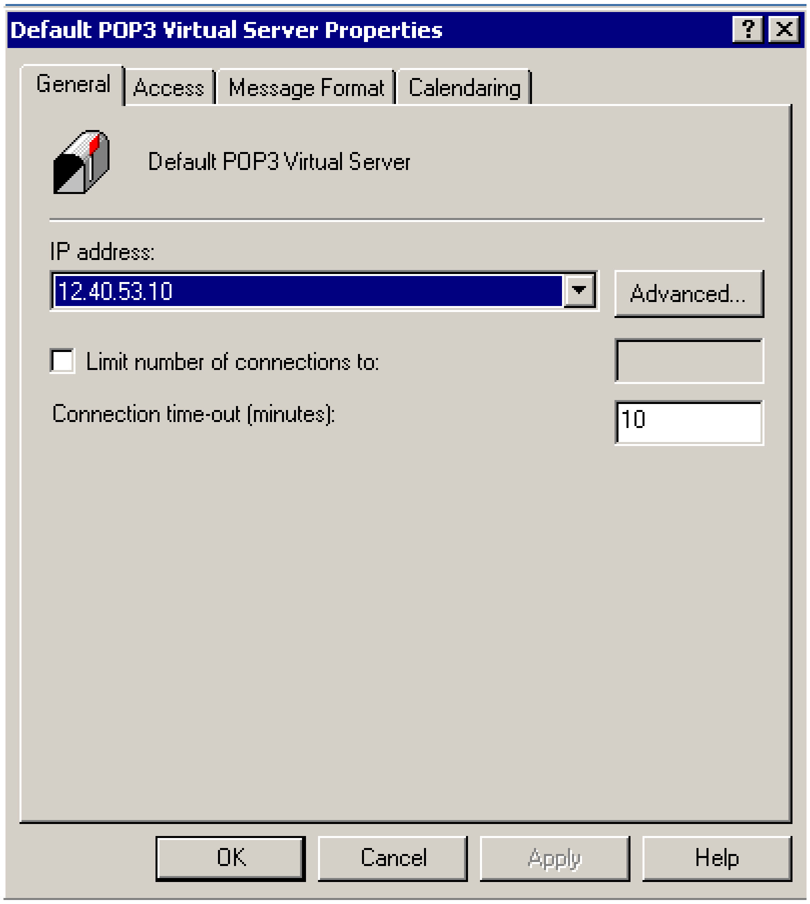
Microsoft Exchange 2000 Setup

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:



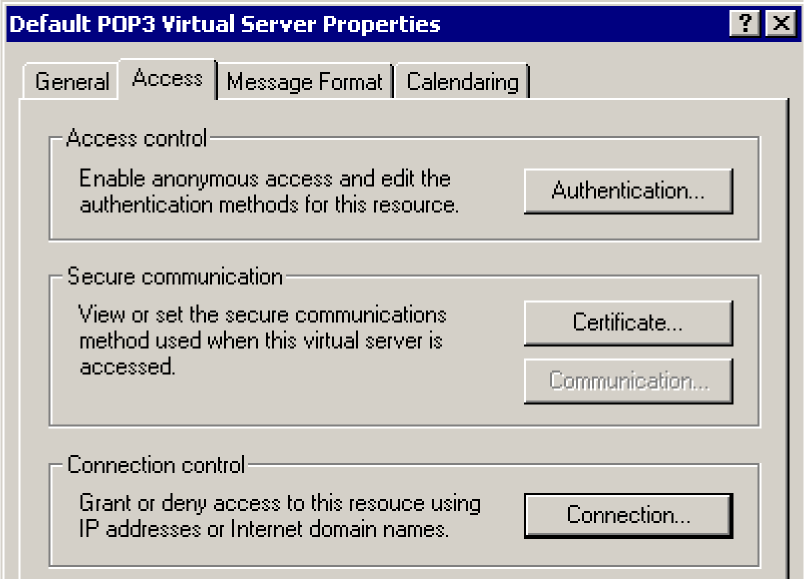
Default POP3 Virtual Server

IP address: 12.40.53.10

☐ Limit number of connections to:

Connection time-out (minutes): 10

Select the Access tab:



Access control

Enable anonymous access and edit the authentication methods for this resource.

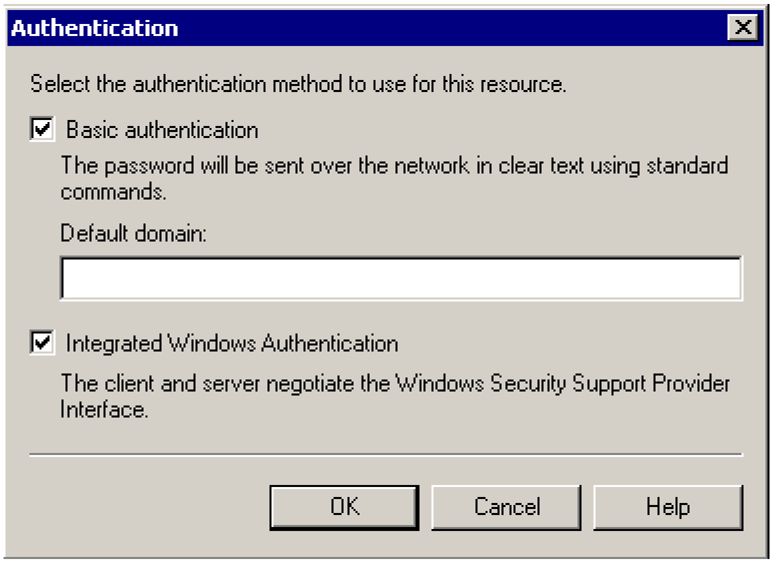
Secure communication

View or set the secure communications method used when this virtual server is accessed.

Connection control

Grant or deny access to this resouce using IP addresses or Internet domain names.

Select the Authentication button and ensure Basic Authentication is selected.



Authentication

Select the authentication method to use for this resource.

☑ Basic authentication

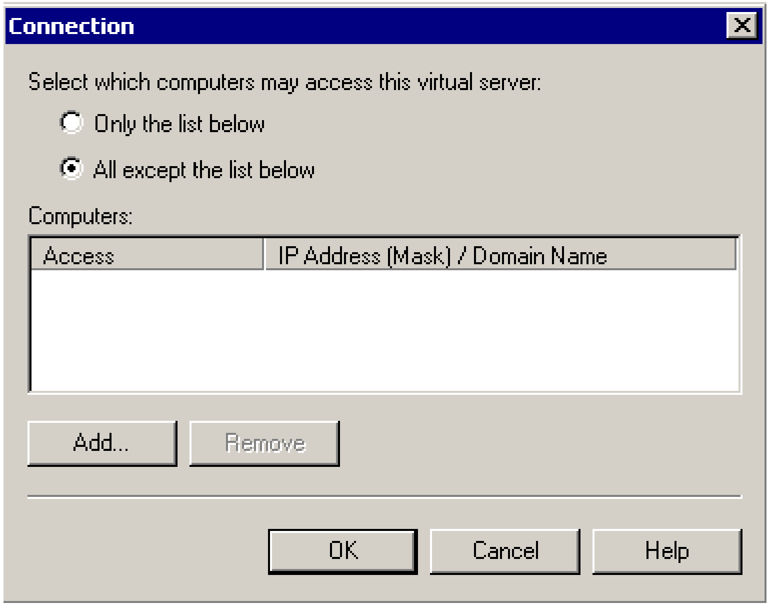
The password will be sent over the network in clear text using standard commands.

Default domain:

☑ Integrated Windows Authentication

The client and server negotiate the Windows Security Support Provider Interface.

When done select OK, then click 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.



Connection

Select which computers may access this virtual server:

○ Only the list below

● All except the list below

Computers:

Access	IP Address (Mask) / Domain Name

When complete select the OK button on all open dialogs.

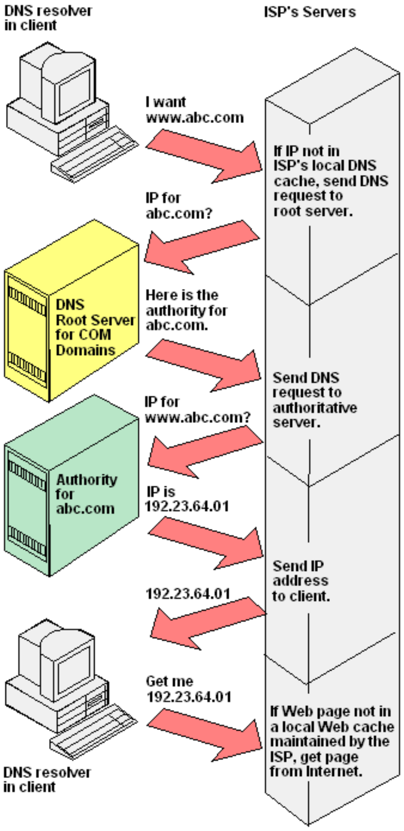
CHAPTER

11

[11] DNS Support

(DNS - **D**omain **N**ame **S**ystem) A system for converting host names and domain names into IP addresses on the Internet or on local networks that use the TCP/IP protocol. For example, when a Web site address is given to the DNS either by typing a URL in a browser or behind the scenes from one application to another, DNS servers return the IP address of the server associated with that name.

From Computer Desktop Encyclopedia
© 2005 The Computer Language Co. Inc.



DNS and the Model 5300

The Model 5300 controller is capable of using DNS to resolve names used in place of IP addresses. The actual DNS IP address used will be that returned by the DHCP server, or when using static IP Addresses, 20128 to 20131 allow the setting of a static DNS server reference.

DNS IP resolution is supported anywhere IP addresses are used within scripts. Both names and IP '.' nomenclature may be intermixed within commands.

Script commands that support DNS:

```
ftpconnect <host name or IP Address>
test esmtp <host> <from> <to> <username> <passwd>
dnslookup <host name>
dnsRlookup
     dnsRlookup <Reg #> <host name>
     dnsRlookup 5 www.ctc-control.com
     Assuming this resolved to 12.40.53.10, this would
     store the following:
          Register 5 = 12
          Register 6 = 40
          Register 7 = 53
          Register 8 = 10
```

CHAPTER 12

[12] Quickstep & QuickBuilder Symbols

This section discusses the symbol file generated by Quickstep 2 and the QuickBuilder tools. These symbols can be imported into HMI displays and used to symbolically monitor assigned registers. Additionally the recommended CTNet Binary Protocol commands for both legacy and newer controllers are discussed.

Quickstep Symbol Table

Supported Controllers: Models 2700, 5100, 5200, 5300.

The symbol format used by Quickstep 2 consists of an ASCII text file with tab delimited fields, each line representing a record entry. Each record is terminated by a 0x0d 0x0a combination. There are four fields:

TYPE – This field determines the resource type. It consists of a single bit set as follows:

1 – Constant
2 – Analog Input
4 – Analog Output
8 – Counter
16 – Data table column
32 - Display
64 – Flag
128 – Digital Input
256 - Stepper
1024 – Register
2048 – Servo
8192 – Step Name
65536 – Unknown Step name
131072 – Digital Output
524288 – Thumb Wheel

RESOURCE – Assigned resource number for access.

STATE – State references the active state, normally open or closed, only used for digital resources. If 0 then normally open and active state is a 1, if 1 then normally closed and active state is a 0.

NAME – Symbolic name.

Example:

1024	38	0	bZeroBatchCount
1024	39	0	bEditJob
1024	9	0	bTest
128	1	0	reot
128	12	0	buckleSensors
128	13	0	sawVFDStatus
131072	13	0	servoReset
131072	14	0	servoEnable
131072	15	0	runSaw
4	1	0	sawSpeed
4	2	0	cnvyrSpeed

Referencing the symbol sawSpeed, its TYPE is a 4, meaning Analog Output. The RESOURCE is 1, first analog output in the controller. STATE field is ignored since that is only for digital.

For HMI access purposes only the following TYPE fields are typically supported (32 bit integers); the rest can be ignored:

2 – Analog Input (resource 1 to N)
4 – Analog Output (resource 1 to N)
64 – Flag (resource 1 to N)
128 – Digital Input (resource 1 to N)
1024 – Register (resource 1 to N)
131072 – Digital Output (resource 1 to N)

Quickstep HMI Communications

Numerous commands are available within the CTNET Binary Protocol as described in the previous chapter.

Note: 2700, 5100, and 5200 controllers do not support Variants.

To simplify access all resources can be accessed by using registers, adding the resource number to the base value:

Refer to *Document No. 951-530006: Model 5300 Quick Reference Register Guide.*

Analog Input, base register 8500
Analog Output, base register 8000
Flag, base register 13200
Digital Input, base register 2000
Register, base register 0
Digital Output, base register 1000

For example, if the TYPE is a 2, designating an Analog Input, with a RESOURCE number of 5, reading register 8505 (8500 + 5) will result in the Analog Input value.

Referencing *Document No. 951-530001: Model 5300 Enhancements Overview*, the primary commands of interest are:

Command	***Description***
9	Read a register
11	Change a register
75	Read a bank of 50 registers (limited from 1 to 1000)
77	Read a bank of 16 registers
87	Request random registers from list (not supported 2701E/2601)

QuickBuilder Symbol Table

Supported Controller: Model 5300

Two symbol table formats available, that described below as well as the Quickstep 2 table format for backward compatibility to tools like CTC MON. Note that the Quickstep 2 format has reduced symbolic information. The symbol file can be found in the project sub-directory with a `.sym` file extension. Two are created upon translation that, with the base name ending with `_QS2.sym`, is in the Quickstep 2 format.

The symbol format used by QuickBuilder consists of an ASCII text file with fixed field sizes, padded with spaces, each line representing a record entry. Each record is terminated by a 0x0d 0x0a combination. There are eight fields:

SYMBOL – Symbolic name, starting in record position 1.

GROUP – Storage group, starting in record position 51. Available groups are:

`double` – 64 bit double in Microsoft format externally via CTC binary protocol, gcc internally (32 bit words swapped). Quickbuilder will reference this as a float (float in Quickbuilder world is actually 64 bits).
`boolean` – 32 bit integer with 0 or 1 value, false/true.

`int` – 32 bit integer.

`string` – array of characters.

TYPE – Resource type, starting in record position 61. Available types are:

`axis` – Servo or stepper axis.

`var` – Variable, volatile.

`nvar` – Variable, non-volatile

`din` – Digital Input

`dout` – Digital Output

`ain` – Analog Input

`aout` – Analog Output

`pid` – PID Algorithm, where RESOURCE is input of PID and REGISTER is output of PID.

RESOURCE – Assigned resource number for access, starting in record position 71.

REGISTER – Assigned register number for access, starting in record position 81. Typically used instead of RESOURCE.

STORAGE – Storage type, starting in record position 91. Available types are:

`scalar` – Single item.

`vector` – One dimensional array.

`table` – Two dimensional array.

Note: Arrays are allocated dynamically. Thus size can change during runtime.

MODULE – Controller module model number referenced, for informational purposes only, starting in record position 101. All spaces if variable or pid.

SLOT – Controller slot module is expected in, for informational purposes only, starting in record position 111. All spaces if variable or pid.

Example:

```
buckleSensors                                     boolean   din       12        2012      scalar    M3-11     1
bZeroBatchCount                                   boolean   var       0         38        scalar
cnvyrSpeed                                        int       aout      2         8002      scalar    M3-34     2
cnvyrSpeed                                        int       ain       1         8501      scalar    M3-32     3
COFFEE_POT                                        boolean   dout      3         1003      scalar    M3-10     4
conveyorVFDStatus                                 boolean   din       14        2014      scalar    M3-11     1
cSaw                                              int       var       0         16        scalar
HEATER                                            boolean   dout      2         1002      scalar    M3-10     4
nvar1                                             double    nvar      0         36701     scalar
PID_PWM                                           int       pid       8502      5903      scalar
PID1                                              int       pid       8501      8001      scalar
PID2                                              int       pid       8502      8017      scalar
PWM1                                              boolean   dout      1         1001      scalar    M3-10     4
var1                                              double    var       0         36101     scalar
```

QuickBuilder HMI Communications

Numerous commands are available within the CTNET Binary Protocol as described in the previous chapter.

Note: QuickBuilder makes extensive use of Variants, which are only supported in the Model 5300 controller..

To simplify access all resources can be accessed by referencing the assigned REGISTER.

Referencing *Document No. 951-530001: 5300 Enhancements Overview*, the primary commands of interest are:

Legacy Register Commands (scalar integers)

Command	***Description***
9	Read a register (integer only, else converts if a Variant)
11	Change a register (integer only)
75	Read a bank of 50 registers (integer only, 1 to 1000)
77	Read a bank of 16 registers (integer only)
87	Request random registers from list (integer only, else converts if a Variant)

Variant Commands (integers, floats, strings, scalar & arrays)

Command	***Description***
91	Get properties (only needed if dynamic array size needed)
93	Read a variant
95	Change a variant
109	Read a variant array block (consecutive)
113	Read a block of variants, randomly

For optimized performance integer access should use the Legacy Register Commands. Variants can be of any type and have a greater overhead.

Blank

CHAPTER

13

[13] 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 Model 5300:

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 14

[14] Formatted Messaging

The Model 5300 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 Model 5300 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.

the `log.ini` file in *Document No. 951-530015: Model 5300 Logging and FTP Client Applications Guide* uses the same formats detailed above.

CHAPTER

15

[15] Network Performance Adjustments

Within a Model 5300 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 interval between each step can be critical, so 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 cannot be transferring files, when transferring files you cannot 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 need to be done, 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), which becomes 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)

Blank

CHAPTER 16

[16] Data Logging

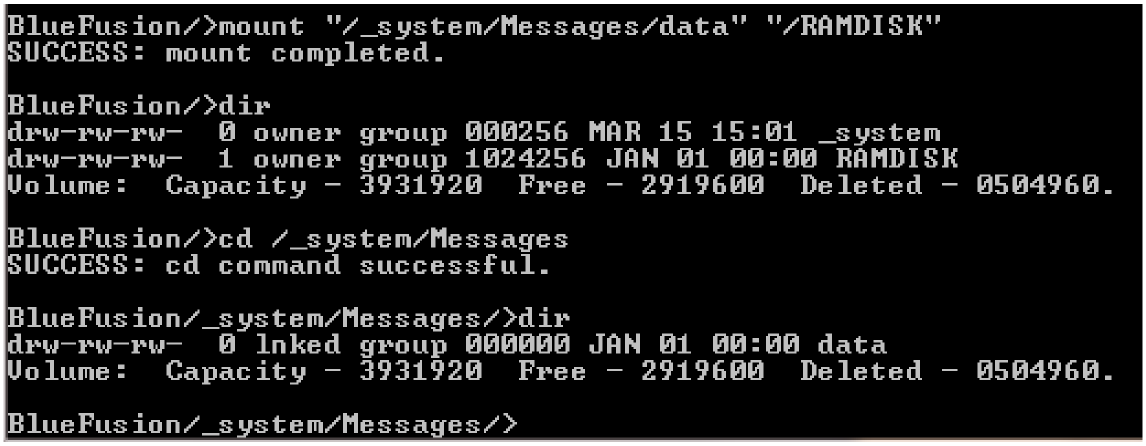
Data Logging on a Model 5300 controller consists of the process of periodically recording collected information in a pre-determined file format. Data may consist of any combination of register contents, a data table cell, time/date information, and/or constant string text. Data is logged in a record format derived from a definition file called `log.ini`. This definition file lists all the logging record formats required, each individually selectable at run-time.

Logging Controller Setup

Data Logging causes a file to be reopened during each write operation, at which time records are appended to it. The flash drive does not support the appending of records to an existing file therefore only the NVRAM is supported for logging. An NVRAM disk resides at the `/RAMDISK` directory of the Model 5300. Both `/` (root) and `/_system` are FLASH. The NVRAM disk is referenced as a virtual sub-directory called `data`, residing within the `/_system/Messages` FLASH directory.

Virtual Directory Creation

Log files are expected to exist in a virtual directory linked to the `/_system/Messages` FLASH directory, called `data`. This allows you to reference the main flash disk file structure but in actuality be redirected to a NVRAM disk of your choice and size. The `mount` command is used to create a virtual directory. Reference *Document No. 951-520001: Model 5200 Remote Administration Guide*. In summary, the following example is used:



```
BlueFusion/>mount "/_system/Messages/data" "/RAMDISK"
SUCCESS: mount completed.

BlueFusion/>dir
drw-rw-rw-  0 owner group 000256 MAR 15 15:01 _system
drw-rw-rw-  1 owner group 1024256 JAN 01 00:00 RAMDISK
Volume:  Capacity - 3931920  Free - 2919600  Deleted - 0504960.

BlueFusion/>cd /_system/Messages
SUCCESS: cd command successful.

BlueFusion/_system/Messages/>dir
drw-rw-rw-  0 lnked group 000000 JAN 01 00:00 data
Volume:  Capacity - 3931920  Free - 2919600  Deleted - 0504960.

BlueFusion/_system/Messages/>
```

All log commands will operate on the `data` sub-directory, with the full path being `/_system/Messages/data`. The following section will describe this process in more detail.

Virtual directories are volatile and must be re-created upon every power cycle, typically by the use of a script file (`_startup.ini`*).*

Logging Record Format and Operation

A predefined format file must reside within the `/_system/Messages` directory called `log.ini`. This file contains individual records defining the desired contents to be written to a log file during a logging sequence.

The content of the `log.ini` file follows the same format as that of the `message.ini` file. This string-formatted message is similar to the format supported by the 'C' function `sprintf`. Each message may consist of either text and/or embedded references to any number of registers, where the values will be substituted just prior to writing to the log file. A maximum completed record size (each line in a log file) of 512 bytes is supported. Message format definitions are stored as records in a file called `log.ini` which is located in the `/_system/Messages` subdirectory of the flash disk. Each line of `log.ini` is considered a record, from 1 to a maximum of 50 pre-defined formats.

The logging of data is controlled by two registers, the *Log Selection Register* (12325) and the *Log String Transfer Register* (12326). The *Log Selection Register* determines the name of the log file to be written, `Log###.log`, where ### is the register contents at the time of an operation. For example writing a 1 to the register defines the name of the file accessed to be `Log001.log`, a 2 would be `Log002.log`, etc. Up to 999 different sequences may be used.

Writing to the *Log String Transfer Register* (12326) causes the actual write operation to occur, selecting which record in the `log.ini` file to dynamically format and write to the

NVRAM disk file. A read on the *Log String Result Register* (12327) returns the status of the write, as defined below:

- 0 – Success
- 43 – File error, either `log.ini` is missing or `Log###.ini` created an error.
- 44 – No such record in the `log.ini` file
- 53 – Write error, check available disk space
- -1 – Default value after setting the *Log Selection Register*

Script and Flash Disk Registers

<table>
<tr><td>12311</td><td>Script Execution: R/W; writing a numeric to this register will cause the corresponding script to be executed. For example: writing a 4 to this register will cause <code>Script004.ini</code> to be executed. Adding 1000 to the value will cause the script to execute as a background thread. For example 1004 will run <code>Script004.ini</code> except as a background thread. Only scripts 000 to 019 may be run this way, reference 12340/12360/12380 register blocks for result information.</td></tr>
<tr><td>12312</td><td>Script Execution Result: Read only, 0 = busy, 1 = successfully executed, else error code (reference 951-520015).</td></tr>
<tr><td>12314</td><td>Flash Disk Selection: (R/W), 0 = root, 1 – n = drive mounted in sequence. Determines volume in which Flash Disk Space Register (12315) returns information.</td></tr>
<tr><td>12315</td><td>Flash Disk Space: Read only, contains the approximate free space available on the flash disk.</td></tr>
<tr><td>12324</td><td>Script Line Result: Read only, contains the last error code that caused a foreground script to stop executing (Script #001 to 999).</td></tr>
<tr><td>12325</td><td>Log File Name Selection: R/W, writing a numeric to this register will define the Log/Data table file name to be used:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><b>0-999</b></td><td>Normal, Log###.log file written using log.ini</td></tr>
<tr><td><b>1000-1999</b></td><td>Variant array is written to log file, Log###.log, log.ini not referenced</td></tr>
<tr><td><b>2000-2999</b></td><td>Reserved</td></tr>
<tr><td><b>3000-3999</b></td><td>Variant array is written to QS2 data table format using name datatable###.tab, log.ini not referenced.</td></tr>
<tr><td><b>4000-4999</b></td><td>Reserved</td></tr>
<tr><td><b>5000-5999</b></td><td>Varant array is loaded/read from QS2 data table format using name datatable###.tab</td></tr>
</table>
</td></tr>
<tr><td>12326</td><td>Log String Transfer: R/W, write record number of 'log.ini' format file to reference and begin writing formatted record to <code>Log###.log</code> file. If using Variant then this becomes the variant register number.</td></tr>
<tr><td>12327</td><td>Log String Result: R, result of logging operation, 0 = success, -1 = busy.</td></tr>
<tr><td>12328</td><td>Log Deletion: R/W, write the numeric value of the <code>Log###.log</code> file to delete.</td></tr>
<tr><td>12329</td><td>Snap Execution: R/W, write the numeric value of the <code>Log###.log</code> file to rename to <code>Snap###.log</code>.</td></tr>
<tr><td>12330</td><td>Snap Result: Read only, result of logging operation, 0 = success, 53 = failed, or not exist.</td></tr>
<tr><td>12331</td><td>Snap Deletion: R/W, write the numeric value of the <code>Snap###.log</code> file to delete.</td></tr>
<tr><td>12340-12359</td><td>Script Thread Result: Read only, 0 = busy, 1 = successfully executed, else error code (reference 951-520015).</td></tr>
<tr><td>12360-12379</td><td>Script Line Thread Result: Read only, contains the last error code that caused a foreground script to stop executing.</td></tr>
<tr><td>12380-12399</td><td>Script Line Thread Result: Read only, contains the line specific error code returned upon abort of script, (reference 951-520015).</td></tr>
</table>

Log.ini Format

As described previously, the `log.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.

Typically a log record consists of text with a `sprintf` formatted specification, intermixed, as required, with the format detailed above. 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 `log.ini` file as:

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

If the above are the only two entries in the `log.ini` file, then writing a 2 to the ***Log String Transfer Register*** will cause the second line to be processed and the following to be written to the disk if a 583 is in register 8501:

```
Analog Value = 00583<CR><LF>
```

Log Format Example

Assume a record format of the following is desired:

- Comma delimited format
- Field 1 – MM/DD/YYYY
- Field 2 – HH:mm
- Field 3 – Time tic register 13002
- Field 4 – Analog Input 1
- Field 5 – Analog Input 5
- Field 6 – New line separator <CR> <LF>

The format for this in a `log.ini` file would be a record inserted with the following format:

```
%T!MM/DD/YYYY!, %T!HH:mm!, %dR13002, %dR8501, %dR8505\r\n
```

If desired, constant text could be added or merged around the above data, although in this example it was not needed. Additional records could be added to the `log.ini` file to represent other formats to be logged. In this example only one is required thus it will be referenced as the first record in the `log.ini` file.

The sequence of events to write a record to a log file with the name `Log001.log` would be:

1. Set the *Log Execution Register (12325)* to a 1, which will set the *Log String Result Register* to a -1: 12325 = 1
2. Write a 1, for the first record of the `log.ini` format file, to the *Log String Transfer Register (12326)* to actually do the write operation: 12326 = 1
3. Monitor *Log String Result Register (12327)* for a change from -1 to another value, 0 signifying success. Note that if background threads (Advanced Scripting, chapter 18) are not used the result and write operation occurs immediately upon writing to the Log String Transfer Register and control is returned to the task only after the write is totally complete. This means status is immediately complete and valid:

   0 – Success

43 – Could not open the file, either `log.ini` is missing or `Log###.ini` created an error.
44 – No such record in the `log.ini` file
53 – Write error, check available disk space

4. Loop Back to step 2 to write the next record if the file name desired has not changed.

An example of three records of data would be:

07/06/2010, 13:41, 234567, 800, 1200
07/06/2010, 13:52, 246000, 801, 1198
07/06/2010, 13:58, 250007, 808, 1190

SNAPSHOT

Snapshot capability is available which renames a file while it is actively being appended to, allowing for real time uploads. For example, if `Log001.log` is being created, and records appended, a single write to the *Snapshot Execution Register* (12329) renames `Log001.log` to `Snap001.log`. The host may then upload `Snap001.log`, which contains all of the logged data to that instant, while in the background records are still being written to a new `Log001.log` file. This allows for automatic synchronization of recorded data.

The *Snapshot Result Register* (12330) reflects the results of the operation:

SUCCESS = 0
ERROR_IOACCESS = 53 (log file does not exist or there is an existing Snap file of the desired name)

Log File Deletion

Both Log and Snap files may be deleted by writing the file number to a special deletion register:

```
LOG_DELETION_REGISTER - 12328
SNAP_DELETION_REGISTER - 12331
```

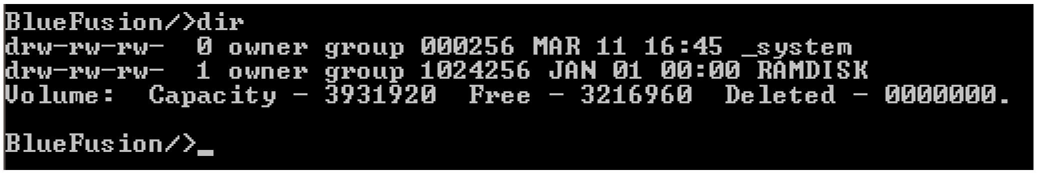
Upon deletion the respective status register will contain either a ‘0’ for success, or a 53 (ERROR_IOACCESS, no file existed).

```
LOG_STRING_RESULT_REGISTER (12327)
SNAP_RESULT_REGISTER (12330)
```

Log Disk Maintenance

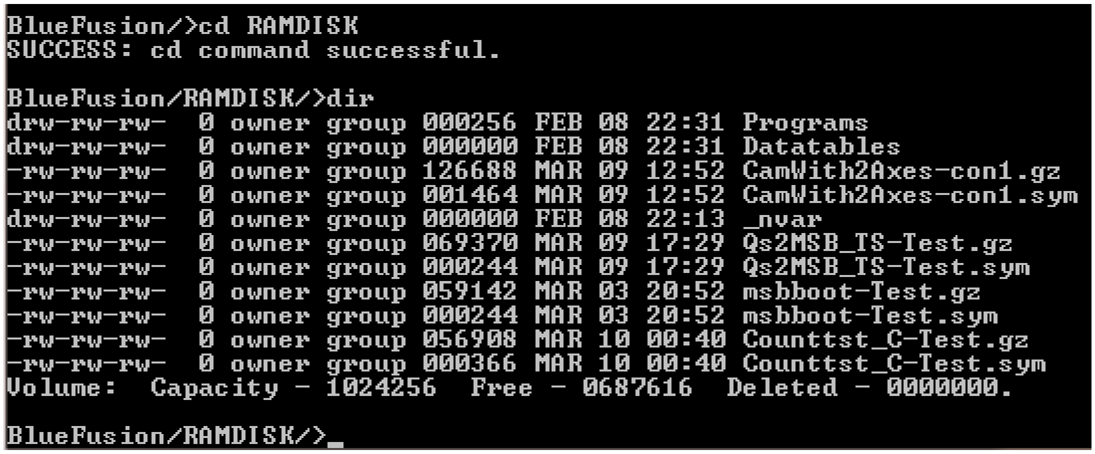
The FLASHDISK_SELECTION_REGISTER (register 12314) is used to select the active disk volume whose remaining space is to appear in the FLASHDISK_SPACEAVAIL_REGISTER (12315). By default it is 0, reflecting the root volume. As each volume is mounted it becomes a new volume and is assigned a number, incrementally. To determine a volume number, simply view a directory at the root level. The second column is the volume number; you will see a 0 in the `_system` row and a 1 in the `mounted ramdisk` row. Reading register 12315 will retrieve the actual bytes remaining.

Directory of the root drive gives the free space on that drive:



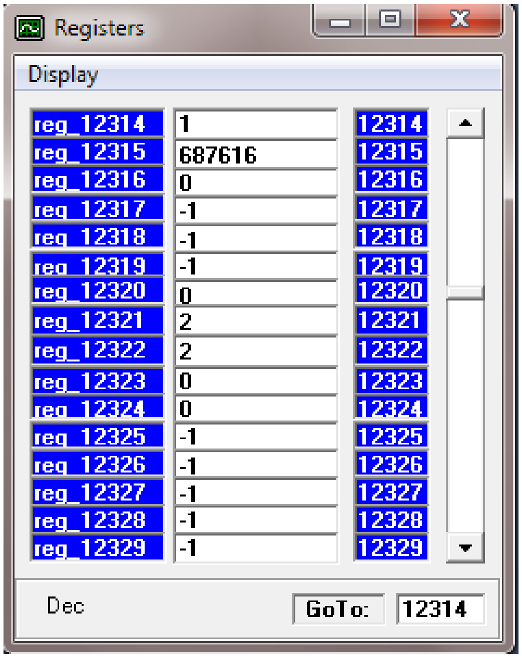
```
BlueFusion/>dir
drw-rw-rw-  0 owner group 000256 MAR 11 16:45 _system
drw-rw-rw-  1 owner group 1024256 JAN 01 00:00 RAMDISK
Volume:  Capacity - 3931920  Free - 3216960  Deleted - 0000000.

BlueFusion/>_
```

Changing to the ramdisk drive shows the free space on that drive:



```
BlueFusion/>cd RAMDISK
SUCCESS: cd command successful.

BlueFusion/RAMDISK/>dir
drw-rw-rw-  0 owner group 000256 FEB 08 22:31 Programs
drw-rw-rw-  0 owner group 000000 FEB 08 22:31 Datatables
-rw-rw-rw-  0 owner group 126688 MAR 09 12:52 CamWith2Axes-con1.gz
-rw-rw-rw-  0 owner group 001464 MAR 09 12:52 CamWith2Axes-con1.sym
drw-rw-rw-  0 owner group 000000 FEB 08 22:13 _nvar
-rw-rw-rw-  0 owner group 069370 MAR 09 17:29 Qs2MSB_TS-Test.gz
-rw-rw-rw-  0 owner group 000244 MAR 09 17:29 Qs2MSB_TS-Test.sym
-rw-rw-rw-  0 owner group 059142 MAR 03 20:52 msbboot-Test.gz
-rw-rw-rw-  0 owner group 000244 MAR 03 20:52 msbboot-Test.sym
-rw-rw-rw-  0 owner group 056908 MAR 10 00:40 Counttst_C-Test.gz
-rw-rw-rw-  0 owner group 000366 MAR 10 00:40 Counttst_C-Test.sym
Volume:  Capacity - 1024256  Free - 0687616  Deleted - 0000000.

BlueFusion/RAMDISK/>_
```

Below, using CTCMON, it is shown that volume 1 is being read, which is represented by the value of ‘1’ in register 12314. Note the value in register 12315, which refers to the size in bytes available in the NVRAM disk labeled ‘RAMDISK’. It is the same as what was displayed in the Telnet session, above.



reg_12314	1	12314
reg_12315	687616	12315
reg_12316	0	12316
reg_12317	-1	12317
reg_12318	-1	12318
reg_12319	-1	12319
reg_12320	0	12320
reg_12321	2	12321
reg_12322	2	12322
reg_12323	0	12323
reg_12324	0	12324
reg_12325	-1	12325
reg_12326	-1	12326
reg_12327	-1	12327
reg_12328	-1	12328
reg_12329	-1	12329

Dec

GoTo: 12314

Blank

CHAPTER

17

[17] FTP Client

The Model 5300 Controller supports simultaneous FTP Server and/or Client sessions. FTP provides a standard means to transfer files to/from the controller from a host computer. When using the built-in FTP Server, the remote computer is the master, initiating file transfers. This section details the use of the 5300 FTP Client mode, where the controller is capable of initiating its own file transfers to a remote host FTP Server. Simple files may be uploaded/downloaded, as well as firmware updates and new Quickstep application programs. Additional information with regards to the FTP Server capability may be found in *Document No. 951-520001: Model 5200 Remote Administration Guide.*

When using FTP Client commands within a script it must be executed as a background threaded operation. Background execution of a script would occur by adding 1000 to the base file script number. For example, a 1001 written to the 'Script Execution Register' (12311) results in `Script001.ini` *executing as a background thread. Ftp Client operations should only be run from a command line within telnet or as a background thread, not as part of a Quickstep task (file numbers, 001 – 999).*

Setup

An FTP Server may reside on virtually any host or workstation computing environment. Software is available for Windows 2000, XP Pro, 2003 Server, and Unix/Linux environments. Unix/Linux, Windows 2000/2003 Servers, and XP Professional contain the service as part of the installation CD. Windows 2000 must use a third party package such as can be found at the following web links:

http://www.bpftppro.com (reference Appendix A)
http://www.serv-u.com/
http://www.candc1.com/ftpservu/index.cfm
http://www.abraxis.com/ipswitch/wsftp-server.htm

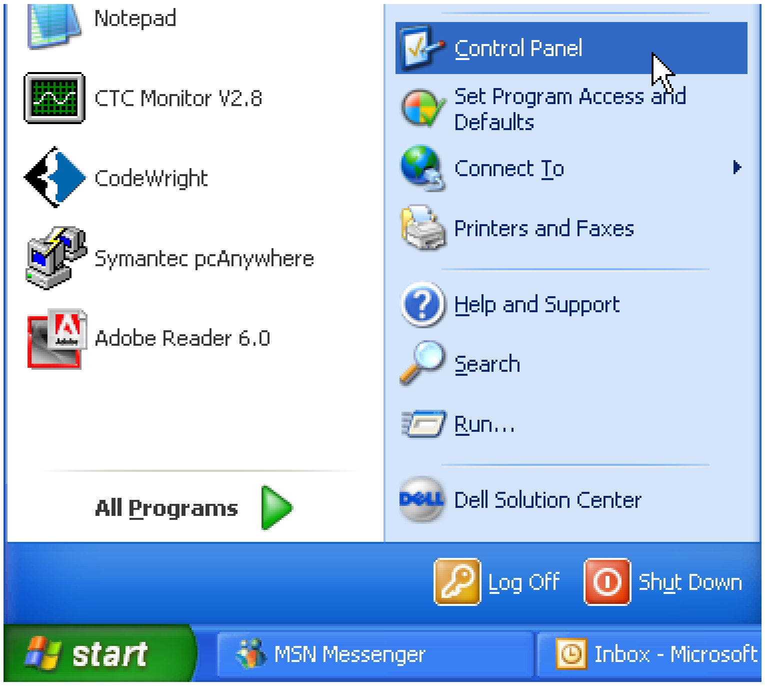
Specifications of what to use, and how to configure the environment, is beyond the scope of this document. A very good reference for installing the resident FTP Server, and its use, on a Windows XP Professional computer is available online from PCSTATS. The title of their article is called: "Beginner's Guides: Setting up a FTP Server in WinXP". It is strongly suggest that you read this article if you are not familiar with FTP Servers. The web link is:

http://www.pcstats.com/articleview.cfm?articleID=1491

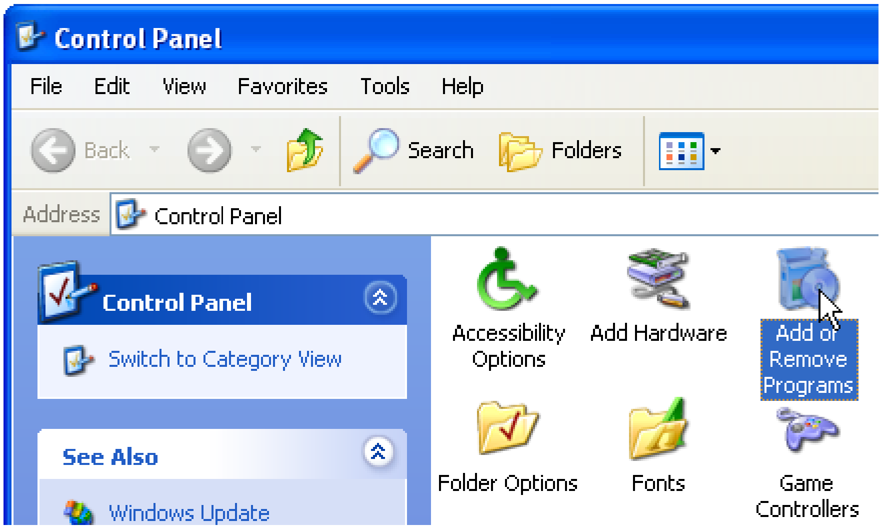
The article provides not only step by step procedures but also a very good background on ftp itself, as well as security considerations.

A quick summary of the installation steps involved are as follows:

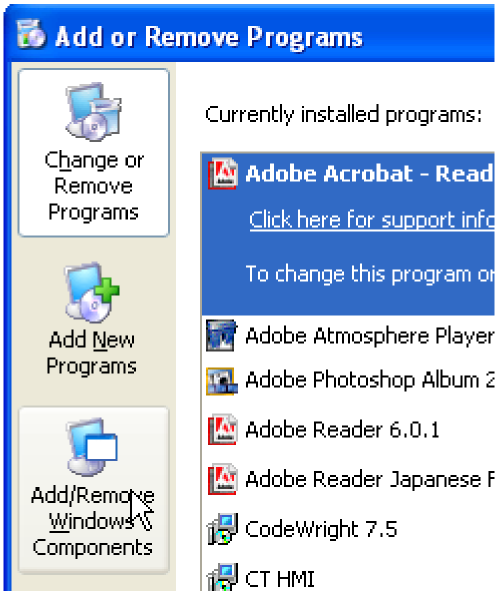
1. While logged in with administrative rights open a Control Panel Window.



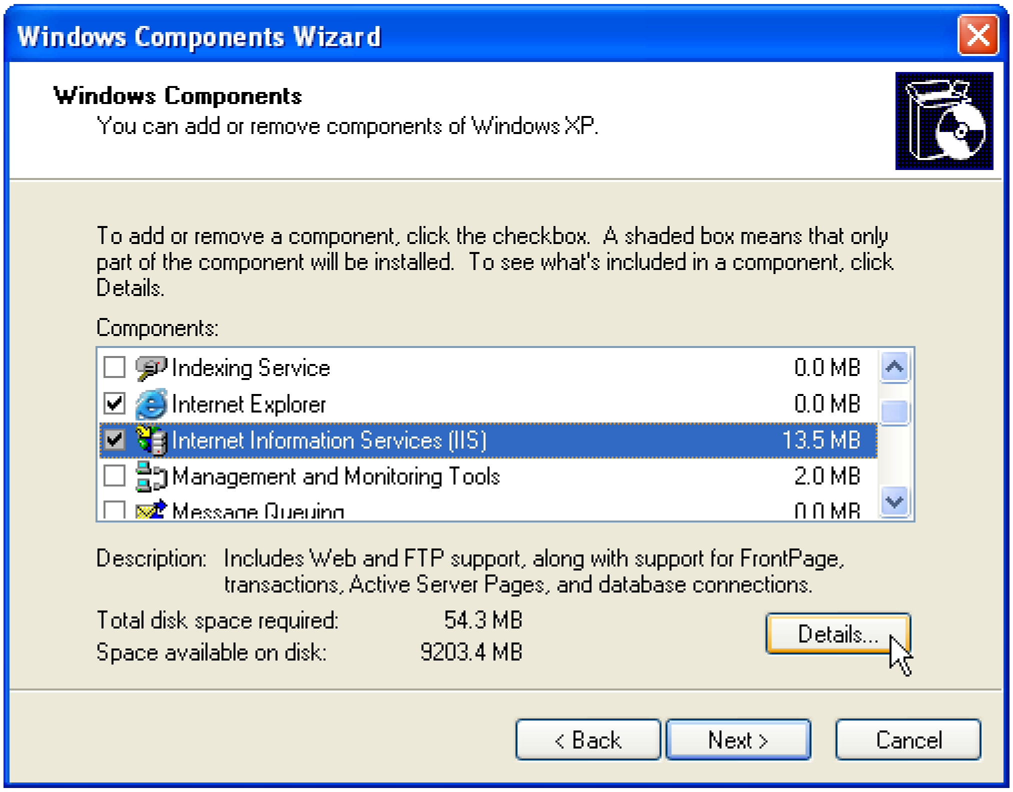
2. Double-click Add or Remove Programs.



3. At the left side of the window click Add/Remove Windows Components.



4. The Ftp Server option is listed under the IIS component. Select it then the Details button.



Windows Components Wizard

Windows Components

You can add or remove components of Windows XP.

To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Components:

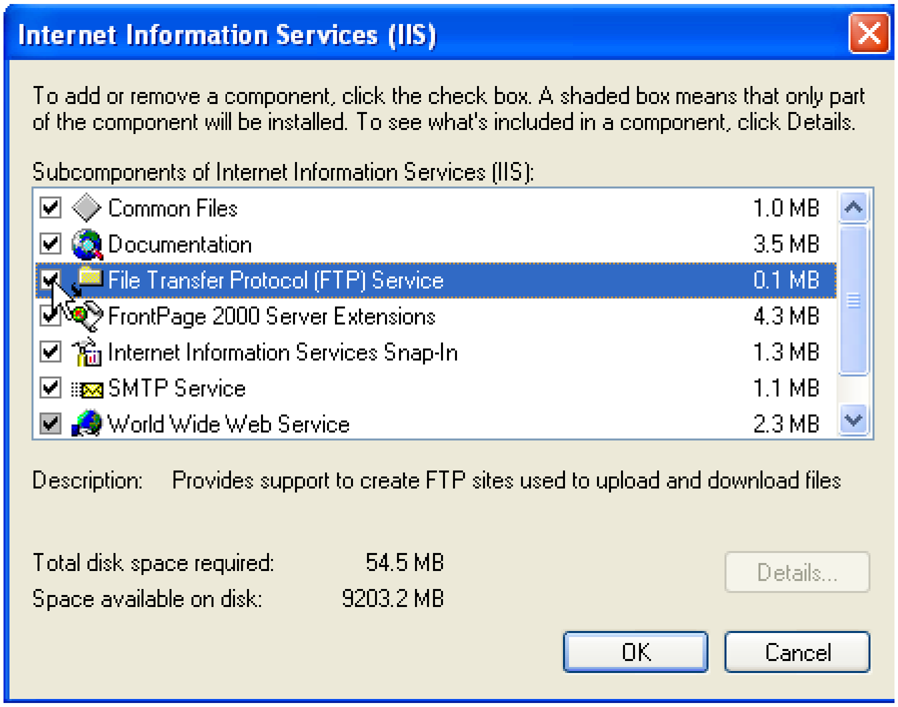
☐	Indexing Service	0.0 MB
☑	Internet Explorer	0.0 MB
☑	Internet Information Services (IIS)	13.5 MB
☐	Management and Monitoring Tools	2.0 MB
☐	Message Queuing	0.0 MB

Description: Includes Web and FTP support, along with support for FrontPage, transactions, Active Server Pages, and database connections.

Total disk space required: 54.3 MB

Space available on disk: 9203.4 MB

5. Make sure the check boxes are as below and then click OK.



Internet Information Services (IIS)

To add or remove a component, click the check box. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Subcomponents of Internet Information Services (IIS):

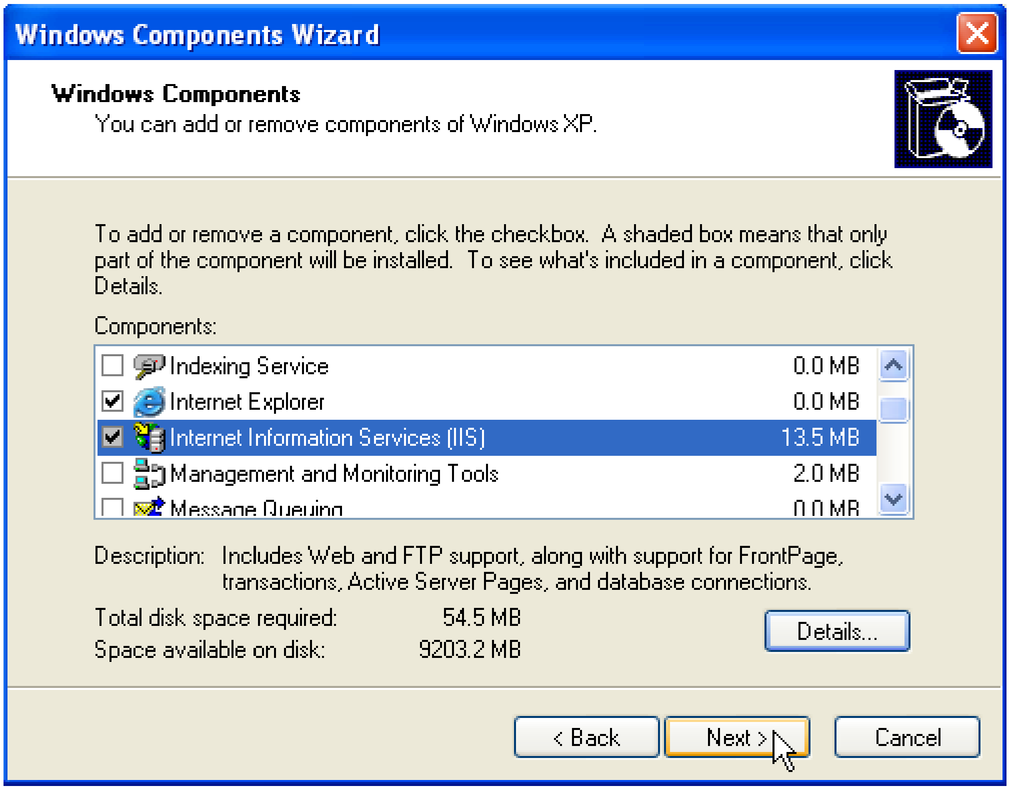
☑	Common Files	1.0 MB
☑	Documentation	3.5 MB
☑	File Transfer Protocol (FTP) Service	0.1 MB
☑	FrontPage 2000 Server Extensions	4.3 MB
☑	Internet Information Services Snap-In	1.3 MB
☑	SMTP Service	1.1 MB
☑	World Wide Web Service	2.3 MB

Description: Provides support to create FTP sites used to upload and download files

Total disk space required: 54.5 MB

Space available on disk: 9203.2 MB

6. Click Next to proceed.



Windows Components

You can add or remove components of Windows XP.

To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Components:

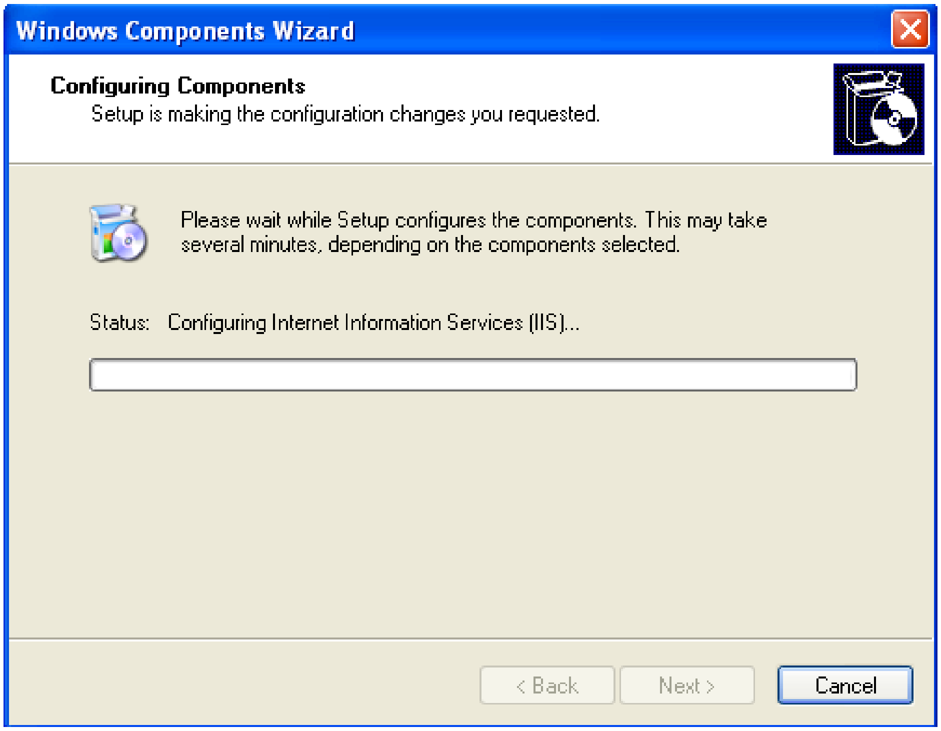
□	Indexing Service	0.0 MB
☑	Internet Explorer	0.0 MB
☑	Internet Information Services (IIS)	13.5 MB
□	Management and Monitoring Tools	2.0 MB
□	Message Queuing	0.0 MB

Description: Includes Web and FTP support, along with support for FrontPage, transactions, Active Server Pages, and database connections.

Total disk space required: 54.5 MB

Space available on disk: 9203.2 MB

7. IIS components will begin to install.



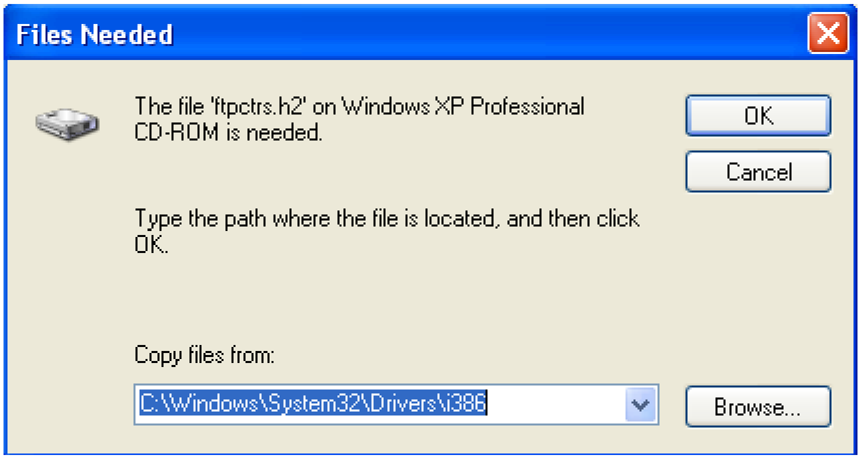
Configuring Components

Setup is making the configuration changes you requested.

Please wait while Setup configures the components. This may take several minutes, depending on the components selected.

Status: Configuring Internet Information Services (IIS)...

8. The prompt below may appear, requiring the XP Professional install CD.



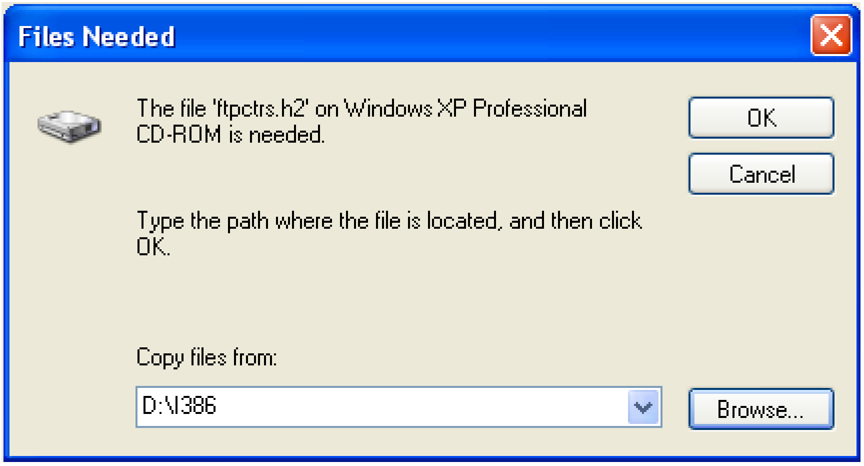
Files Needed

The file 'ftpctrs.h2' on Windows XP Professional CD-ROM is needed.

Type the path where the file is located, and then click OK.

Copy files from: C:\Windows\System32\Drivers\i386

9. Insert the CD and set the directory as appropriate.



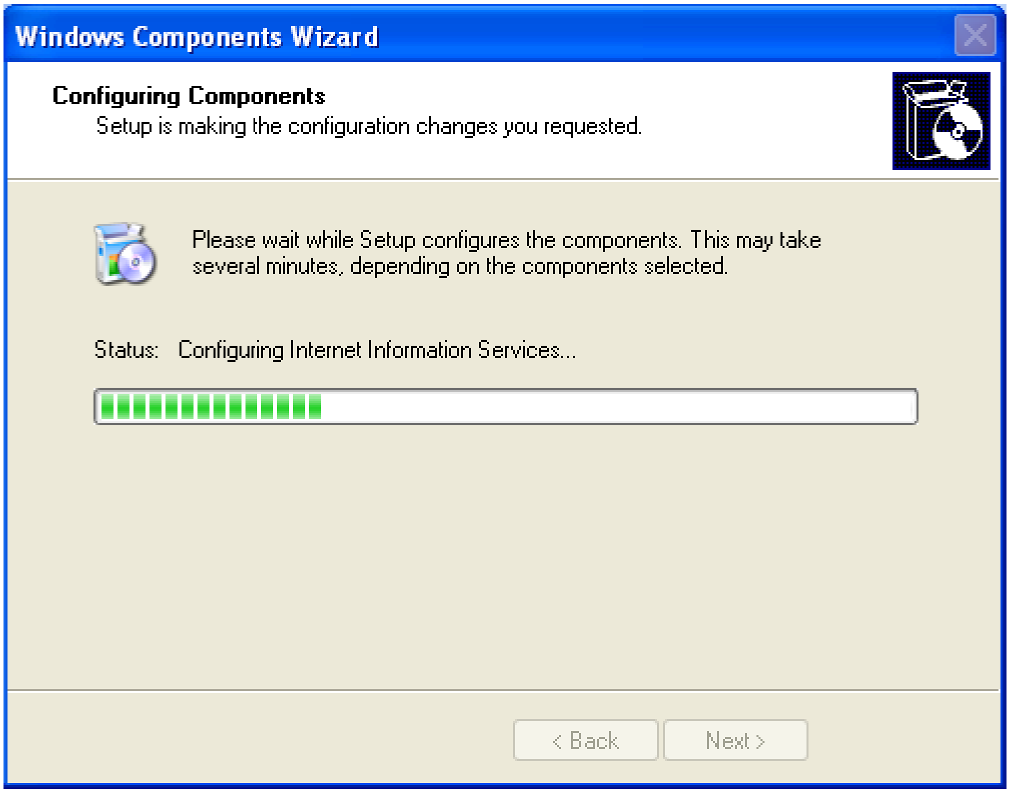
Files Needed

The file 'ftpctrs.h2' on Windows XP Professional CD-ROM is needed.

Type the path where the file is located, and then click OK.

Copy files from: D:\I386

10. Click OK when the path is correct and installation will begin.



Windows Components Wizard

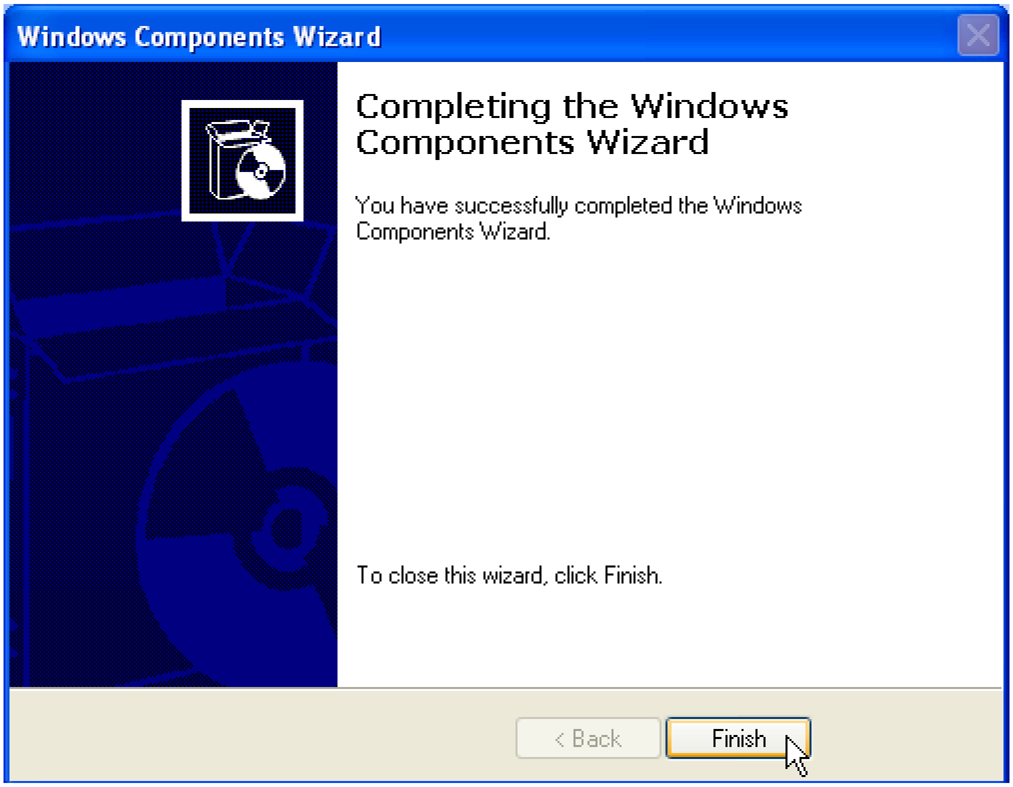
Configuring Components

Setup is making the configuration changes you requested.

Please wait while Setup configures the components. This may take several minutes, depending on the components selected.

Status: Configuring Internet Information Services...

11. Once complete click Finish.

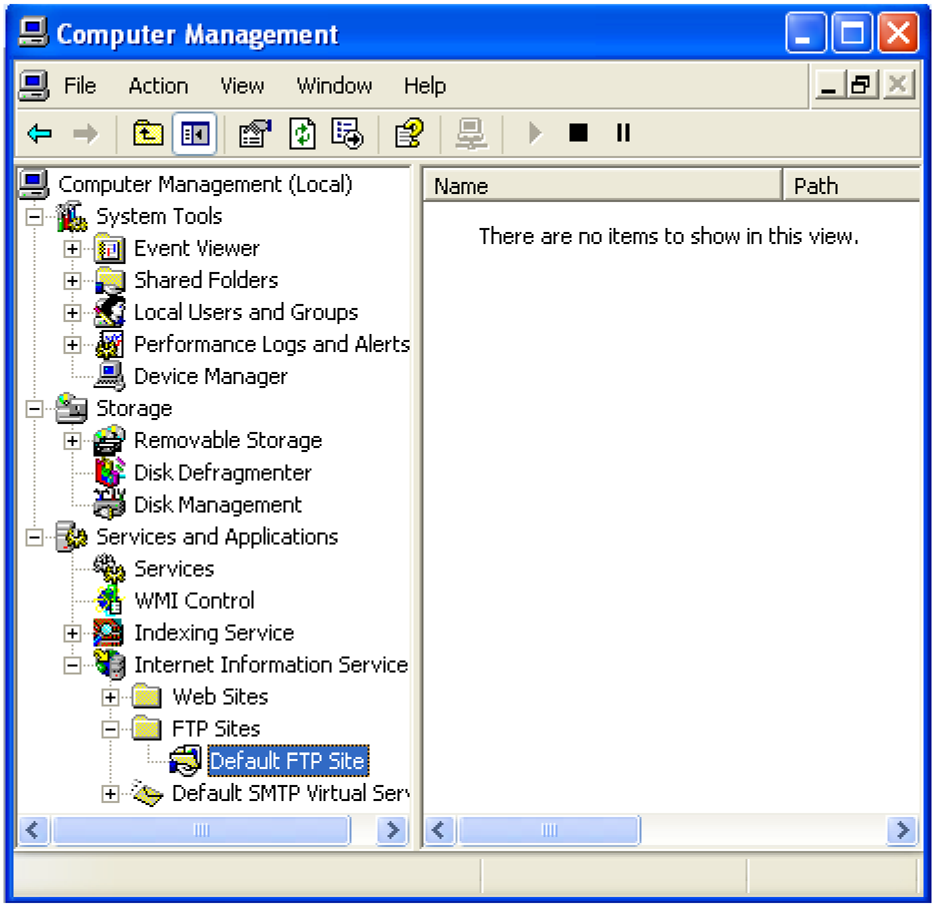


Completing the Windows Components Wizard

You have successfully completed the Windows Components Wizard.

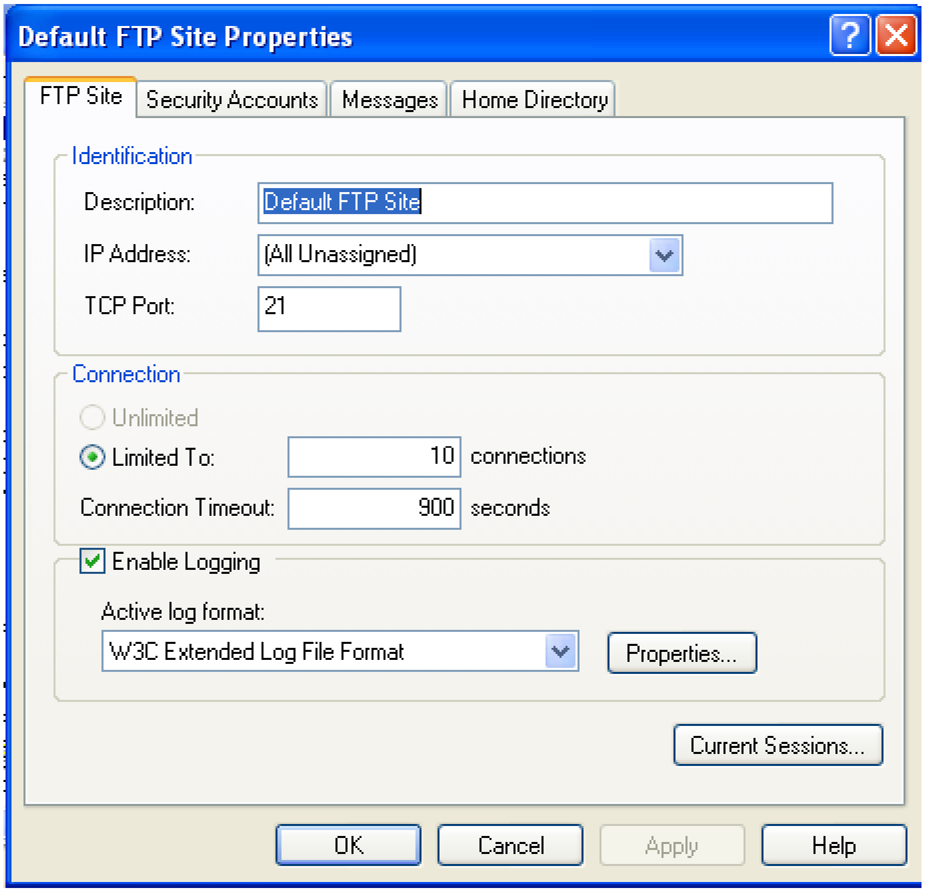
To close this wizard, click Finish.

12. From Control Panel->Administrative Tools, select Computer Management and expand the Internet Information Service folders until the Default FTP Site appears. This verifies that ftp is installed.



There are no items to show in this view.

13. Right click the Default FTP Site and select the Properties menu item to view the current settings. Reference the PCSTATS web site, previously discussed, for suggestions on how to adjust security, add user accounts for access, etc. Note that by default only **local** computer user accounts with **Administrative privileges** may access files.



Identification

Description: Default FTP Site

IP Address: (All Unassigned)

TCP Port: 21

Connection

○ Unlimited

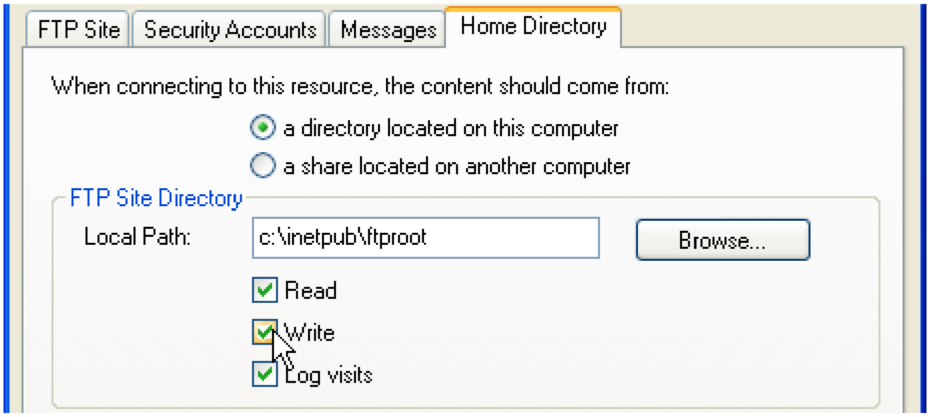
● Limited To: 10 connections

Connection Timeout: 900 seconds

☑ Enable Logging

Active log format: W3C Extended Log File Format

If both upload and download are to be done, Write must be enabled under the Home Directory tab:



When connecting to this resource, the content should come from:

● a directory located on this computer

○ a share located on another computer

FTP Site Directory

Local Path: c:\inetpub\ftproot

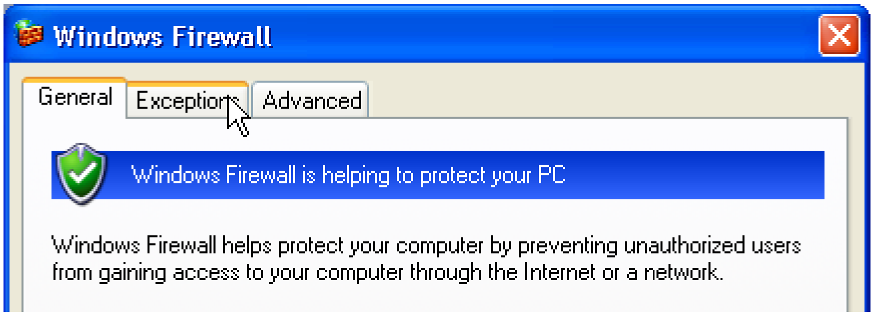
☑ Read

☑ Write

☑ Log visits

14. Windows XP has additional firewall security built into the product. The FTP Server must be specifically enabled to allow incoming connections and bypass the firewall. From the Control Panel select the Windows Firewall icon:

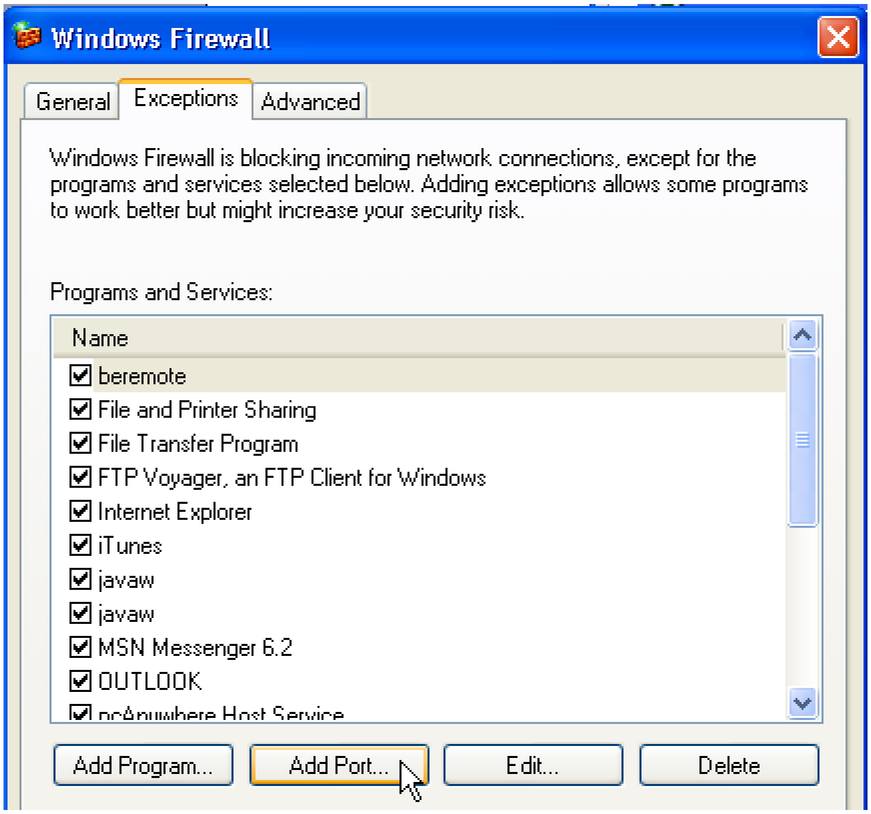
15. Select the Firewall Exceptions Tab:



Windows Firewall is helping to protect your PC

Windows Firewall helps protect your computer by preventing unauthorized users from gaining access to your computer through the Internet or a network.

16. Select Add Port:

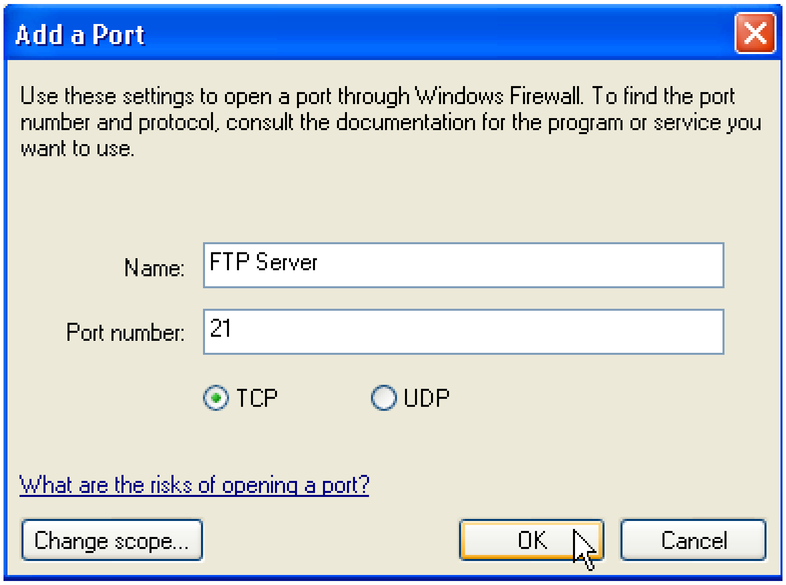


Windows Firewall is blocking incoming network connections, except for the programs and services selected below. Adding exceptions allows some programs to work better but might increase your security risk.

Programs and Services:

Name
☑ beremote
☑ File and Printer Sharing
☑ File Transfer Program
☑ FTP Voyager, an FTP Client for Windows
☑ Internet Explorer
☑ iTunes
☑ javaw
☑ javaw
☑ MSN Messenger 6.2
☑ OUTLOOK
☑ pcAnywhere Host Service

17. Fill out the dialog box as below and click OK:



Add a Port

Use these settings to open a port through Windows Firewall. To find the port number and protocol, consult the documentation for the program or service you want to use.

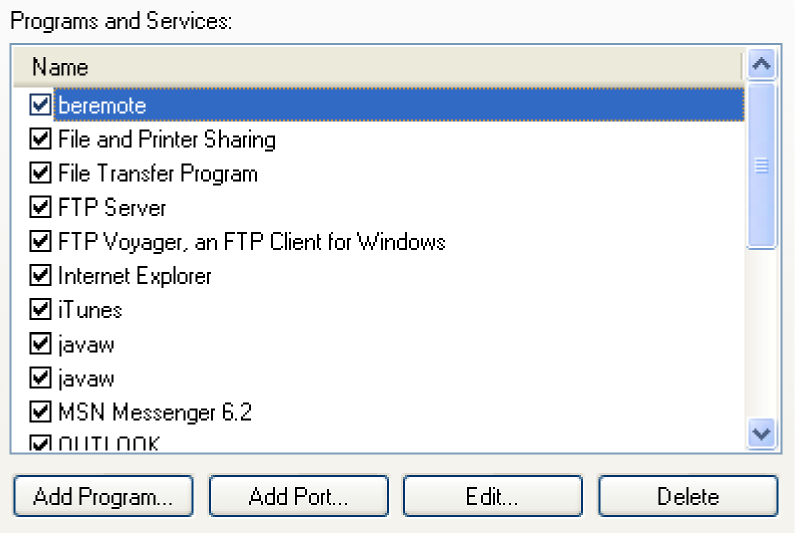
Name: FTP Server

Port number: 21

● TCP ○ UDP

What are the risks of opening a port?

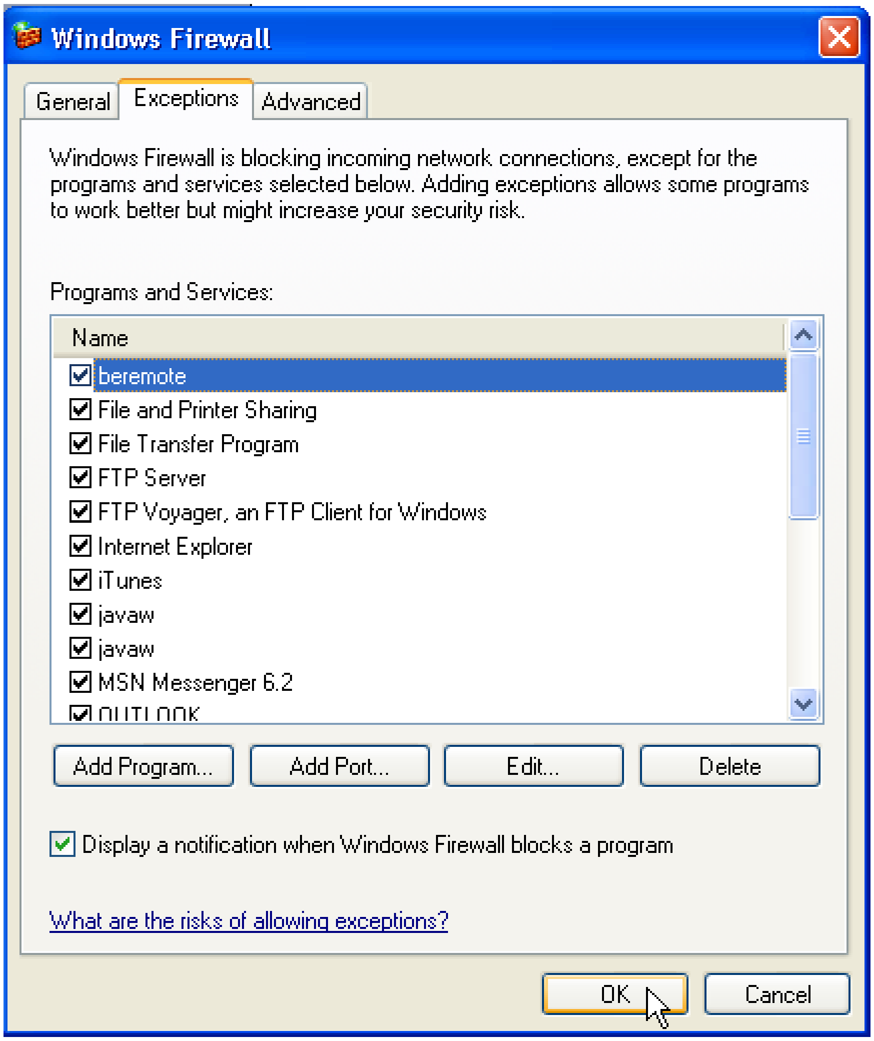
18. The FTP Server Name should appear in the list of checked Programs and Services:



Programs and Services:

Name
☑ beremote
☑ File and Printer Sharing
☑ File Transfer Program
☑ FTP Server
☑ FTP Voyager, an FTP Client for Windows
☑ Internet Explorer
☑ iTunes
☑ javaw
☑ javaw
☑ MSN Messenger 6.2
☑ OUTLOOK

19. Click OK at the bottom of the dialog to exit:



Windows Firewall is blocking incoming network connections, except for the programs and services selected below. Adding exceptions allows some programs to work better but might increase your security risk.

Programs and Services:

Name
- ☑ beremote
- ☑ File and Printer Sharing
- ☑ File Transfer Program
- ☑ FTP Server
- ☑ FTP Voyager, an FTP Client for Windows
- ☑ Internet Explorer
- ☑ iTunes
- ☑ javaw
- ☑ javaw
- ☑ MSN Messenger 6.2
- ☑ OUTLOOK

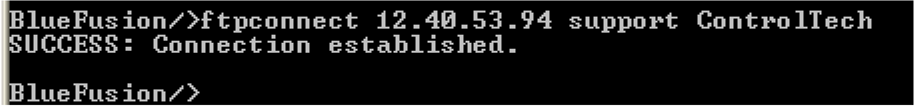
☑ Display a notification when Windows Firewall blocks a program

What are the risks of allowing exceptions?

Commands

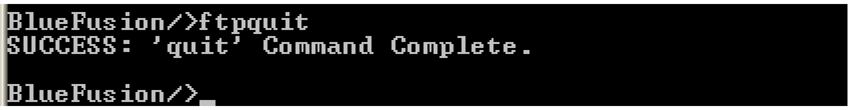
Once a server is available, the Model 5300 controller provides two means to access the external server via its FTP client capabilities. The first is via the command line using Telnet, the second by the use of advanced scripting detailed in Chapter 18: Advanced Scripting. The following demonstrates the FTP commands when using Telnet to interact with a Windows 2003 Server:

- **`Ftpconnect`** `<ip address> <User Name> <Password>`
  Provides initial connection to the remote host computer running the FTP Server.



```
BlueFusion/>ftpconnect 12.40.53.94 support ControlTech
SUCCESS: Connection established.

BlueFusion/>
```

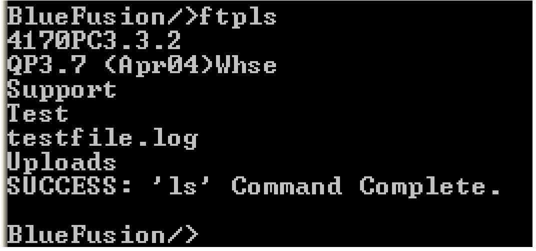
- **`Ftpquit`**
  Closes an FTP session after a successful `Ftpconnect`.



```
BlueFusion/>ftpquit
SUCCESS: 'quit' Command Complete.

BlueFusion/>_
```

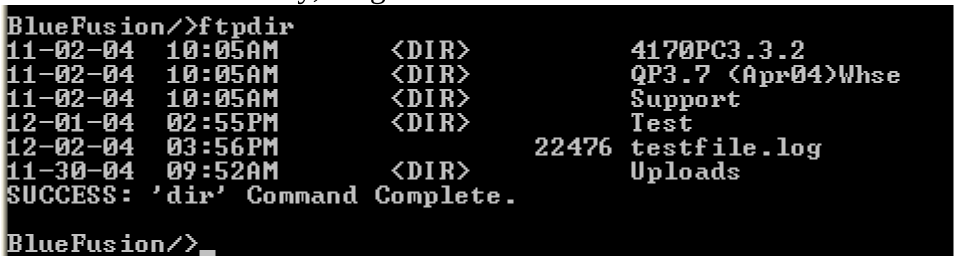
The following commands require the FTP connection to be active:

- **`Ftpls`** `<optional path>`
  Get the current directory, short format.



```
BlueFusion/>ftpls
4170PC3.3.2
QP3.7 (Apr04)Whse
Support
Test
testfile.log
Uploads
SUCCESS: 'ls' Command Complete.

BlueFusion/>
```

- **`Ftpdir`** `<optional path>`
  Get the current directory, long format.



```
BlueFusion/>ftpdir
11-02-04  10:05AM       <DIR>          4170PC3.3.2
11-02-04  10:05AM       <DIR>          QP3.7 (Apr04)Whse
11-02-04  10:05AM       <DIR>          Support
12-01-04  02:55PM       <DIR>          Test
12-02-04  03:56PM                22476 testfile.log
11-30-04  09:52AM       <DIR>          Uploads
SUCCESS: 'dir' Command Complete.

BlueFusion/>_
```

- **`Ftpsend`** `<source path> <optional destination path/name>`
  Send a file to the remote host. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.

- **`Ftpappend`** `<source path> <optional destination path/name>`
  Send a file to the remote host, if it exists append to it else create a new file. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.

- **`Ftpget`** `<source path> <optional destination path/name>`
  Get a file from the remote host. Firmware may be re-flashed or new programs loaded, bypassing flash storage by directing it to the root directory. Only one ftp server or client session can do this at a time since reserved SDRAM storage space is used as a temporary buffer. Example:

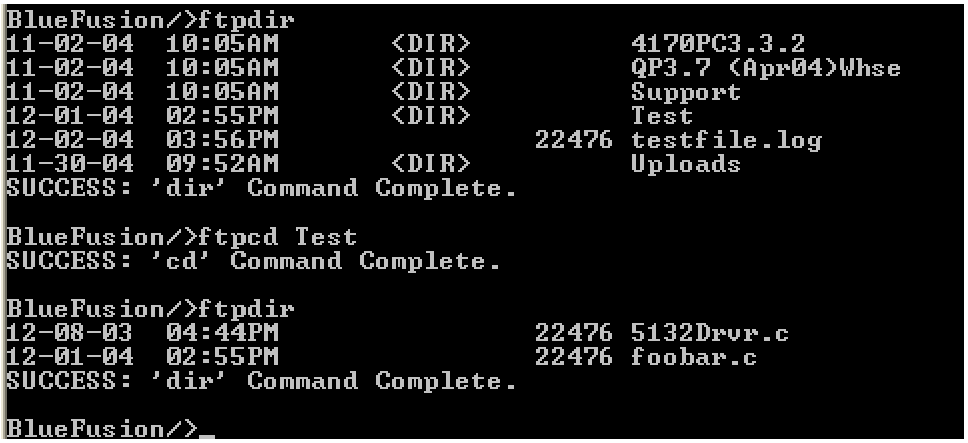
```
ftpget BF5300V05009068.elf /BF5300V05009068elf
```

  Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format. Example:

```
ftpget hostfile.fil "/mydir/Script%03dR1.ini"
```

where the contents of R1 will be substituted into the filename.

- **Ftpcd** `<path>`
  Change the current directory, on the host, to that specified. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.



```
BlueFusion/>ftpdir
11-02-04  10:05AM       <DIR>          4170PC3.3.2
11-02-04  10:05AM       <DIR>          QP3.7 (Apr04)Whse
11-02-04  10:05AM       <DIR>          Support
12-01-04  02:55PM       <DIR>          Test
12-02-04  03:56PM                22476 testfile.log
11-30-04  09:52AM       <DIR>          Uploads
SUCCESS: 'dir' Command Complete.

BlueFusion/>ftpcd Test
SUCCESS: 'cd' Command Complete.

BlueFusion/>ftpdir
12-08-03  04:44PM                22476 5132Drvr.c
12-01-04  02:55PM                22476 foobar.c
SUCCESS: 'dir' Command Complete.

BlueFusion/>_
```

- **Ftpmkdir <directory name>**
  Makes a directory on the host computer in the current directory. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.

- **Ftprename <source path/file> <new name>**
  Renames the specified source file to the new name. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.

- **Ftprmdir <directory name>**
  Removes the specified directory on the host computer. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.

- **Ftpdelete <source path/file>**
  Deletes a file on the host computer. Paths enclosed in quotes (" ") allow the embedding of register contents using the `log.ini` format.

Telnet Error Codes

When executing ftp commands from the telnet command line the returned message will typically begin with `SUCCESS:` . At times, a failure will occur, causing an error message to appear. The message will be displayed in the Telnet session on a new line using the following format:

```
ERROR: <message> <code>
```

Where `<message>` is a description of the failure and `<code>` is detailed below:

FTP_ERROR – 0xD0
Internal ftp error.

FTP_TIMEOUT – 0xD1
Timeout occurred.

FTP_FAILED – 0xD2
General ftp failure.

FTP_NOT_CONNECTED – 0xD3
Attempted to execute a command but was not connected to the host.

FTP_NOT_DISCONNECTED – 0xD4
Host closed session during command execution.

FTP_NOT_OPEN – 0xD5
FTP failure on opening a file for transfer.

FTP_NOT_CLOSED – 0xD6
Attempted to open a file when a previous was not closed

FTP_LOGIN – 0xD9
Attempt to log onto the host failed, security violation

FTP_NOT_FOUND – 0xDA
Request was not executed (typically 550 FTP return code). Typically returned when a file/directory does not exist, or there is a security access violation blocking access.

FTP_RETURN_CODE – 0xDB
Unknown return code from host ftp server.

CHAPTER
18

[18] Advanced Scripting

Document #951-520003: Model 5200 Script Language Guide details the operation of scripting within the Model 5200/5300 controller. This section discusses features beyond that of the Model 5200, such as automated file transfers, data table & file operations, and background threaded script execution. A thorough knowledge of scripts is assumed.

Some advanced features available within scripts are:

- **Background Execution** - Scripts may not only be run as part of a step, they may also execute as separate background threads, in parallel to Quickstep execution. Writing a 1 to 999 to the *Script Execution Register* (12311) causes a script to run, within a Step, much like a Do statement. An advanced feature allows a programmer to write a 1001 to 1020, executing Script001 to Script020 as a thread, much like a future Begin Quickstep statement. These scripts fully execute as an independent background thread.
- **Continuous Execution** - Scripts may execute continuously until either a fatal error or an end command is executed, terminating the script.
- **Script Nesting** - Scripts may also invoke other Scripts (one call level supported).
- **Branching & Conditionals** - Scripts support branching, program labels, and if conditionals.
- **Error Tracking** - Major file and communication instructions set a status error code, private to each script, which may be referenced from an if conditional or onerror command. The variable is referenced as ERRORCODE and allows for advanced retry and monitor operations.
- **Execution Time Control** - The Alarm instruction allows the script to sleep until a specific time, automatically waking at a defined time such as every Friday afternoon at 3PM. Use theDelay instruction for pauses based on millisecond values.

Data Table

load datatable [Variant regnum] [filename]

The `load datatable` script command can be used to load a variant array from a file. This file is stored in the same format as the `QS2.TAB` file format except that floats have a decimal point and strings are enclosed in quotes. A current QS2 file would be read in as integers, given that that is a limitation of the QS2 data table. The enhanced table that uses variants for storage is shown below:

```
data_table[4][6] =
{
      84      104      105      115       32      105
     115       32      114      111      119       32
      49        0        0        0        0        0
       0        0        0        0      6.789   "string"
}
```

Note that tabs or spaces may be used as a separator, as all whitespace characters < ‘0’ are ignored except LF, which designates the end of a line/row. There is no restriction on the line/row length except as required by each Variant cell (string 223 bytes). The first 2 lines are ignored and the actual table size is set by the data found in the file. A CR LF combination should follow the last ‘}’ to denote the end of the file.

The last two cells show examples of a float, 6.789, and a string format “string”. Remember the string may reference other registers using the `%d, message.ini`, format, but an extra % is needed (i.e., `%%d).` The first % will be stripped when the string is parsed.

In the above example, 84 would be loaded into array location [0][0], and “string” would be loaded into [3][5].

The load/save datatable commands list the array size as [row][column] for compatibility with QS2.

save datatable [Variant regnum] [filename]

The `save datatable` script command operates exactly the same as the `load datatable` command except that the Variant register contents are written to a file. Any unknown cells will contain a “?”. Two separate formats are available, QS2 compatibility mode and CSV (comma delimited, similar to a log file).

In QS2 compatibility mode, seven spaces will be placed between cell data and a `.tab` file extension must be used. Upon writing any existing file will be first deleted. The use of any other file extension will cause the CSV format to be used, where each cell is separated by a command and a single space. End of line is the same as the QS2 format, CR LF. There is no header information in CSV format thus the first array location will be the first bytes of the file.

The `load/save datatable` *commands list the array size as [row][column] for compatibility with QS2.*

Diagnostics

disktest 1 [file size] [block size] [/path/file]

This command is used to perform a test of the file system. A file of the size [`file size`] will be written using blocks of size [`block size`] to the file [`/path/file`]. An incrementing byte pattern is written and verified. The complete write is done first, file closed and then read back and verified. For example, to test a 1MB file with a block size of 512 bytes:

Example: `disktest 1 1000000 512 /SDISK/test1.bin`

disktest 2 [file size] [block size] [/path/file]

This command is used to perform a test of the file system. A file of the size [`file size`] will be written using blocks of size [`block size`] to the file [`/path/file`]. An incrementing byte pattern is written and verified. The complete write is done first, file closed and then read back and verified. For example, to test a 1MB file with a block size of 512 bytes:

Example: `disktest 2 1000000 512 /SDISK/test1.bin`

Quickstep

enable quickstep2

This command enables QS2 operation at the next reboot, assuming a 1 is written to register 20096. Quickstep may be enabled and disabled from normal operation in order to conserve CPU cycles.

System default is enabled.

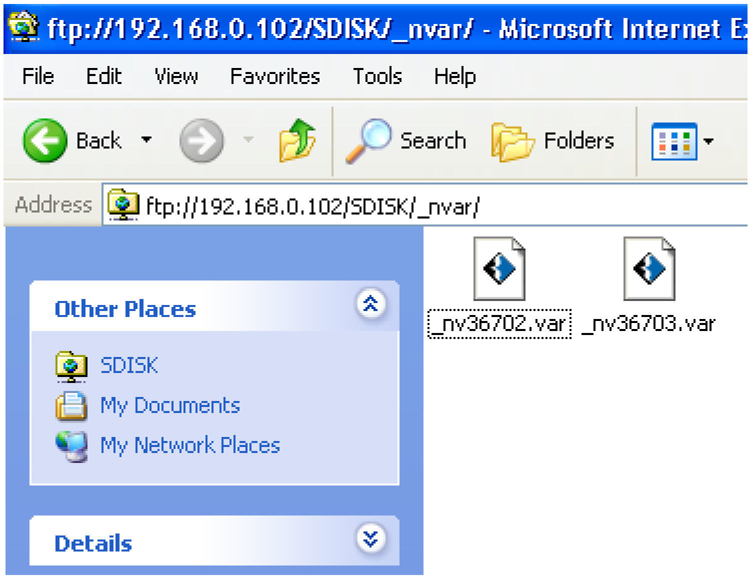
disable quickstep2

This command disables QS2 operation at the next reboot, assuming a 1 is written to register 20096. Quickstep may be enabled and disabled from normal operation in order to conserve CPU cycles.

File System

set close nvariant [Variant #]"

Non-Volatile array variants are stored in files that may be transferred among controllers, deleted, copied, etc. The contents of the non-volatile variant are independent of the actual register number. The file name determines its assignment. Hence `_nv36702` will be used for register 36702.



If that file was copied to file `_nv36703`, then the data has now been duplicated and register 36702 and 36703 now have the same data. When replacing a file it is important to close it first. Not closing it means it cannot be deleted. You may copy a non-closed file but make sure no task is writing to it or the data may be changing in the background.

Closing a file is also important if you are replacing non-volatile variants data, since upon access if the file is closed it will re-attempt to open it and in this instance find the new file. Open files cannot be replaced or deleted.

Example: `set close nvariant 36702`

This would close the file `_nv36702` should it be open using the default path of `/SDISK/_nvar`.

To close all open non-volatile specify a -1 as the [`Variant #`]. Should an error occur the `ERROR_OO_RANGE` flag is set.

set logpath [path]

Sets an alternate log file storage path. Power up default is `/_system/Messages`.

Example: `set logpath /SDISK/myLogDir`

set scriptspath [path]

Sets an alternate script file storage path. Power up default is `/_system/Scripts`.

Example: `set scriptspath /SDISK/myScriptsDir`

set nvariantpath [path]

Sets an alternate nvariant file storage path. Power up default is `/SDISK/_nvar` which will automatically be created at power up if it does not exist (one level only in directory tree).

Example: `set nvariantpath /SDISK/mynvariants`

set emailspath [path]

Sets an alternate email file storage path. Power up default is `/_system/Emails`.

Example: set emailspath /SDISK/myEmailsDir

set webpath [path]

Sets an alternate web file storage path. Power up default is `/_system/Web`.

Example: `set webpath /SDISK/myWebDir`

set firmwarepath [path]

Sets an alternate firmware file storage path. Power up default is `/_system/Firmware`.

Example: `set firmwarepath /SDISK/myFirmwareDir`

set programspath [path]

Sets an alternate program file storage path. Power up default is `/_system/Programs`.

Example: `set programspath /SDISK/myProgramsDir`

set datatablespath [path]

Sets an alternate data tables file storage path. Power up default is `/_system/Datatables`.

Example: `set datatablespath /SDISK/myDatatablesDir`

copy [source path/file] [destination path/file]

This command is used to copy a file from one location to a new location, creating a new file and overwriting any existing.

Example: `copy /SDISK/_nvar/_nv36702 /SDISK/_nvar/_nv36750`

A new non-volatile Variant, 36750 now exists with the same contents of register 36702. `SCRIPT_ERRMASK_FILE` error bit is set if an error occurs.

Monitor

mon tfs init

Initialize and clear the controller monitor file system.

mon tfs rm

Remove a file from the controller monitor file system.

mon tfs ls

List all the files within the controller monitor file system, typically just one, which is the application program to execute.

mon reboot

Exit the controller application and reboot into the monitor, re-loading the entire controller program once again. Basically a full reset where VBIAS will be shut off and execution of application programs immediately terminated without notification.

Miscellaneous

get vproperties [Variant #]

Since there are currently no utilities to view variant information, such as how big it is, array size, floating point precision, etc, this command will display that information:

Example: `get vproperties 36702`

```
SUCCESS:  Nonvolatile Variant 36702, Size:  rows - 50 / columns -
200, precision:  6.
```

printf [format string…]

The `printf` command allows for a string format to be tested prior to inclusion in a variant cell or `message.ini` file. The string will be parsed exactly as it would when used in these applications. This command is typically used for testing only as it has no effect other than visual final string presentation.

Example: `printf "The contents of register 100 = %dR100"`

The " " are required.

clear startup project

This command will clear the current default project that is invoked at reset or power up, thus none will be executed upon power up.

get project

This command will display the currently active project that is running.

get project info [project file]

This command is used to determine the contents of a QuickBuilder project file.

get startup project

This command will display the project set to run at power up or reset.

run project [opt. project file]

This command is used to load and run a QuickBuilder project file. The controller is restarted. If no project is specified the last saved project will be run.

set startup project [opt. project file]

This command is used to save a specific project file name/location upon which to run at power up and reset, as the default. If none is specified then the last executed path/name will be saved.

Advanced Commands

Inc <Register>

Increment the contents of the reference register by 1. Typically used in counter operations such as retrying communications.

Example: `inc 200`

If the contents of register 200 was 99, it would become 100 after execution.

Dec <Register>

Decrement the contents of the reference register by 1. Typically used in counter operations such as retrying communications.

Example: `dec 200`

If the contents of register 200 was 99, it would become 98 after execution.

If <Resource> <Logic> <Resource> goto <Label>

`<Resource>` - Register reference (R####), decimal or hex constant (0x00000000), or ERRORCODE.

`<Logic>` - >, >=, <, <=, !=, ==, and & resulting in a Boolean result of true or false.

`<Label>` - Single line within script file containing a preceding colon ':' followed by a unique character string. Example – `:myLabel`

Example:

```
if R200 >= 55 goto tooBig
if ERRORCODE & 0x00200000 goto fileFailure
```

:<Label>

A label consists of a single, independent line, within a script text file, to which a name is assigned, typically as a destination for a branch operation (if or goto).

Example: `:myLabel`

Onerror <optional error mask> goto <Label>

Set where to automatically branch should an error occur. Execution of 'Onerror' with no terms clears the branching option. Only file and communication instructions, along with syntax errors specifically set error flags. The 'optional error mask' allows you to activate individual errors. Upon return from and instruction a global ERRORCODE variable has its individual flags set should a problem occur, else ERRORCODE = 0.

Example:

```
onerror   (clears any error branching)
onerror goto myLabel       (if any error
occurs branch to myLabel)
onerror 0x00200000 goto myLabel  (branch if
file error)
```

Goto <Label>

Immediately branch to the desired `<Label>`.

End

Exit the Script.

Delay <Register or constant – milliseconds>

Delay the designated number of milliseconds. Milliseconds may either be a decimal constant, hexadecimal notation (0x00000000), or reference the contents of a register (R200 for register 200).

Example:

```
delay 2000      (delay 2 seconds or 2000 milliseconds)
delay R1  (delay by the number of milliseconds in register 1)
```

Alarm <TIME=HH:MM:SS> <optional day of week, DOW=Mon…>

Pause execution until the designated time occurs. A 'TIME' and optional day of week, DOW, from 0 (Monday) to 6 (Sunday) is listed. Each parameter may be contained within a register. For example the hour and/or minute could reference register 15 while the minute is a constant: R15:00. When referencing a specific day of the week, enter the following: Mon, Tue, Wed, Thu, Fri, Sat, or Sun, for the desired day, or a number from 0 to 6. Seconds may also be included, yielding a format of <HH:MM:SS>.

Example:

```
alarm TIME=23:00 DOW=Mon  (sleep until 11PM on Monday)
alarm TIME=23:00    (sleep until 11PM)
alarm TIME=R1:00    (sleep until hour contents in register 1)
alarm TIME=23:00:05 DOW=0  (where 0 = Monday)
alarm TIME=23:00:05 DOW=R10  (where contents R10 = 0 - 6)
```

ERRORCODE

`ERRORCODE` is a universal variable, private to each script. Upon execution of script commands status bits are set should problems occur. `if` conditional and `onerror` commands can branch accordingly.

SCRIPT_ERRMASK_FATAL - 0x00800000
Fatal error, such as a memory allocation failure. Currently this can only be returned by an `ftpconnect` command.

SCRIPT_ERRMASK_SYNTAX - 0x00400000
Syntax error, bad parameter passed as part of a command.

SCRIPT_ERRMASK_FILE - 0x00200000
General File failure. For example, `rename` of file or creation of a new directory (`mkdir`) failed. Affected commands are:

- mkdir
- rename
- cd
- rmdir
- format flash
- delete
- load symbols
- get symbols

- mount
- umount

SCRIPT_ERRMASK_FTP_CONNECT - 0x00100000

Returned if an `ftpconnect` attempt fails or there is an attempt to use a command that requires a previous connection and it does not exist.

SCRIPT_ERRMASK_FTP_LOSTCONNECT - 0x00080000

Lost connection while attempting an FTP command. Affected commands are:

- Ftpls
- Ftpdir
- Ftpmkdir
- Ftprmdir
- Ftpcd
- Ftpget
- Ftpsend
- Ftpappend
- Ftpdelete

SCRIPT_ERRMASK_FTP_COMMAND - 0x00040000

Unknown FTP response code returned. Affected commands are:

- Ftpls
- Ftpdir
- Ftpmkdir
- Ftprmdir
- Ftpcd
- Ftpget
- Ftpsend
- Ftpappend
- Ftpdelete

SCRIPT_ERRMASK_FTP_NOTFOUND - 0x00020000

Operation failed, file or directory does not exist on host. Affected commands are:

- Ftpls
- Ftpdir
- Ftpmkdir
- Ftprmdir
- Ftpcd
- Ftpget
- Ftpsend
- Ftpappend
- Ftpdelete

SCRIPT_ERRMASK_FTP_SECURITY - 0x00010000
User name or Password failed. This only applies to `ftpconnect`.

Script Example

The following sample script, Script001, shows how to connect to an ftp server, send a file called `Log001.log` and download a file called `anyfile.log` on the host to a file called `newfile.log` on the controller. A maximum of 3 retries, with a 5 second interval will be attempted before aborting. Register 2 is used as an arbitrary status register, which can be monitored by Quickstep, and should be set to 0 prior to invoking the script. Its status is as follows:

1 = Complete and successful
-1 = Busy running script
-2 = General failure
-3 = File did not exist on the Controller
-4 = File did not exist on the Host

The execution of the following script would occur by writing a 1001 to the *Script Execution Register* (12311). **Adding a 1000 to the script file number causes it to execute as a background thread. Ftp Client operations should only be run from a command line within telnet or as a background thread, not as part of a Quickstep task (001 – 999).**

Script001.ini:

```
# This script tests the automated transfer of data from
# the controller to a remote ftp server.
# Register 1 is used as a retry counter
# Register 2 is used to notify Quickstep program of completion
# status
# Writing to the Script Execution Register (12311) will cause
# execution to occur.  Script file name is currently
Script001.ini
# thus writing 1001 will cause this file to run as a thread in
the
# background, writing a 1 will cause it to run within a Quickstep
# step.
# Threaded execution (background) is the preferred method
#
:start
# Clear the retry counter
      1 = 0
# Clear our completion flag register
      2 = -1
      goto firstTime

:retry
# Delay for 5 seconds and then try again
```

```
      delay 5000
# Bump the retry counter and see if done with retries...
      inc R1
      if R1 > 3 goto abort

:firstTime
# First set up where to branch if an error should occur
      onerror goto errorOccurred

# Lets connect to the remote host
      ftpconnect 12.40.53.94 support ControlTech

# Now lets upload a file and then download a file
# If an error occurs we will exit automatically due to the
# 'onerror' command
# SEND to host from controller
      ftpsend /_system/Messages/data/Log001.log
# RECEIVE from host and store to controller under different name
      ftpget anyfile.log /_system/Messages/data/newfile.log
# All done so close host session gracefully
      ftpquit
# Set completion flag to 0 indicating we are done and successful
      2 = 1

# If we get here we are all done
      end

# Process error if should occur
:errorOccurred

# Lets see what type of error occurred
# First check to see if initial connection failed
      if ERRORCODE & 0x00100000 goto retry
# Next see if we failed during a transfer
      if ERRORCODE & 0x00080000 goto retry
# Next see if we failed because the file did not exist on the
controller
      if ERRORCODE & 0x00200000 goto nofileController
# Next see if we failed because the file did not exist on the
host
      if ERRORCODE & 0x00020000 goto nofileHost
# Was a fatal failure so give up
:abort
      2 = -2
      end
# File did not exist on the controller
:nofileController
      2 = -3
      end
# File did not exist on the host
:nofileHost
      2 = -4
      end
```

CHAPTER 19

[19] CTNet Binary Protocol (Server Interface)

This section discusses the CTNet Binary Protocol, at the packet level, as is supported by the controller. The CTNet binary protocol is a high-speed 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.

Consequently, use of CTNet results in very short execution times. Note that CTNet used to be non-routable (2700 with 2217 Ethernet controllers). 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.

Non-routable 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.

Provisions have been made to extend the CTNet protocol by encapsulating it in a UDP/TCP packet. In this case the IP address becomes the destination and Register 20000 is ignored. Port 3000 is for UDP and port 6000 is for TCP connections. UDP/TCP is fully routable. Refer to the last section of this chapter for how to encapsulate. In short the discussion that follows fully applies to the encapsulated packet. Serial port communications are also supported for all CTNet packets; again, Register 20000 does not apply in that case either since only point to point communications are supported.

A maximum of 32 simultaneous TCP Binary protocol connections are allowed at one time. Idle connections will timeout in about 1 minute.

Binary Protocol

The CTC Binary Protocol may be used to communicate with the Model 5300 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 under Windows 2000/XP. This DLL is discussed in detail within *Document No. MAN1080A: 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).

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.
<length (1 byte)> Specifies packet length to follow. Packet length is defined as n data bytes + 2 (checksum and 0xff).
<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.

```
//
// Generate a checksum for a packet
// Parameters:  p - pointer to start of data section
//len - length of data only section (not length, checksum
or
//0xff)
// Returns:  <checksum>
unsigned char  Packet_Check(unsigned char * p, int len)
{
  unsigned int c = 0;
  int i;

      for( i=0; i<len; i++)
            c = (c + *(p + i)) & 255;
      return( (char)~c);
}
```

<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.

All communications are in Little Endian format.

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:

<03H)> 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:

<03H)> Specifies packet length to follow. Packet length is defined as n data bytes + 2.
<(65H)> Contains the not acknowledged code; equal to decimal 101.
<9AH> Is the value of the checksum of the not acknowledged code.
<FFH> Required by binary protocol; last byte of packet must be ØFFH.

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,FFH,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.
   FFH Is the fifth byte and specifies the new state of the flag. FFH 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 (controller response is command + 1)	
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)
Variant Commands	
91	Get properties
93	Read a variant
95	Change a variant
109	Read a variant array block (consecutive)
111	Write a variant array block (consecutive)
113	Read a block of variants, randomly
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 *Document No. 951-530006: Model 5300 Quick Reference Register Guide* for register specifics.

Variant Packets

A number of commands are available to interface with variant storage within the Model 5300. When communicating with the controller a packed data structure is used. Two separate structures are used, that for individual read/writes, VARIANT_STORAGE, or for block access VARIANT_STORAGE_BLOCK (VARIANT_STORAGE_BLOCK_SERIAL if serial port). When using block transfers the total size (number of elements) is dependent upon whether Ethernet or serial communications is being used. Ethernet allows for a larger packet and when using UDP and TCP the packet itself provides a CRC. Thus the checksum field is not really needed and not used on the larger block transfers.

When using variants the packet structure is identical except that the data portion is the packed variant structure:

<(Ø1H)> Specifies CTC binary protocol.
<length (1 byte)> Specifies packet length to follow. Packet length is defined as n data bytes + 2 (checksum and 0xff). Checksum is not used on packet type 109/110, 111/112, and 113/114 when using Ethernet communications (length set to 5 on request, response length is 3), it is used on serial since a reduced packet size is used.
<Command/Response Code>
<LSB Register #> Register of interest low byte unless random read, in which case ignored.
<MSB Register #> Register of interest high byte unless random read, in which case ignored.
<packed variant structure> Valid structures:

VARIANT_STORAGE
VARIANT_STORAGE_BLOCK
VARIANT_STORAGE_BLOCK_SERIAL.

<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.

Register and Flag Access Command/Response definitions

```
// GET is request, GOT is controller response
// binary protocol message types
#define MSG_LOAD_PROGRAM_PACKET                  ((BYTE) 0)
#define MSG_ENTER_PROGRAM_MODE                   ((BYTE) 1)
#define MSG_LEAVE_PROGRAM_MODE                   ((BYTE) 2)
#define MSG_UNLOAD_PROGRAM_PACKET                ((BYTE) 3)
#define MSG_PROGRAM_PACKET                       ((BYTE) 4)
#define MSG_GET_ID_CODES                         ((BYTE) 5)
#define MSG_GOT_ID_CODES                         ((BYTE) 6)
#define MSG_OLD_GET_STATUS                       ((BYTE) 7)
#define MSG_OLD_GOT_STATUS                       ((BYTE) 8)
#define MSG_GET_REGISTER                         ((BYTE) 9)
#define MSG_GOT_REGISTER                        ((BYTE) 10)
#define MSG_SET_REGISTER                        ((BYTE) 11)
#define MSG_12                                  ((BYTE) 12)
#define MSG_GET_IO_COUNTS                       ((BYTE) 13)
#define MSG_GOT_IO_COUNTS                       ((BYTE) 14)
#define MSG_GET_INPUTS                          ((BYTE) 15)
#define MSG_GOT_INPUTS                          ((BYTE) 16)
#define MSG_GET_FLAG                            ((BYTE) 17)
#define MSG_GOT_FLAG                            ((BYTE) 18)
#define MSG_SET_FLAG                            ((BYTE) 19)
#define MSG_20                                  ((BYTE) 20)
#define MSG_GET_OUTPUTS                         ((BYTE) 21)
#define MSG_GOT_OUTPUTS                         ((BYTE) 22)
#define MSG_GET_SERVO_POSITION                  ((BYTE) 23)
#define MSG_GOT_SERVO_POSITION                  ((BYTE) 24)
#define MSG_SET_OUTPUTS                         ((BYTE) 25)
#define MSG_26                                  ((BYTE) 26)
#define MSG_GET_SERVO_INPUT                     ((BYTE) 27)
#define MSG_GOT_SERVO_INPUT                     ((BYTE) 28)
#define MSG_GET_ANALOG_INPUT                    ((BYTE) 29)
#define MSG_GOT_ANALOG_INPUT                    ((BYTE) 30)
#define MSG_GET_ANALOG_OUTPUT                   ((BYTE) 31)
#define MSG_GOT_ANALOG_OUTPUT                   ((BYTE) 32)
#define MSG_SET_ANALOG_OUTPUT                   ((BYTE) 33)
#define MSG_34                                  ((BYTE) 34)
#define MSG_GET_STATUS                          ((BYTE) 35)
```

```
#define MSG_GOT_STATUS_1of4                     ((BYTE) 36)
#define MSG_GOT_STATUS_2of4                     ((BYTE) 37)
#define MSG_GOT_STATUS_3of4                     ((BYTE) 38)
#define MSG_GOT_STATUS_4of4                     ((BYTE) 39)
#define MSG_SET_EA_OUTPUT                       ((BYTE) 40)
#define MSG_LOAD_EA_PROGRAM_PACKET              ((BYTE) 41)
#define MSG_42                                  ((BYTE) 42)
#define MSG_UNLOAD_EA_PROGRAM_PACKET            ((BYTE) 43)
#define MSG_EA_PROGRAM_PACKET                   ((BYTE) 44)
#define MSG_DUMP_USER_MEMORY                    ((BYTE) 45)
#define MSG_USER_MEMORY                         ((BYTE) 46)
#define MSG_GET_SERVO_ERROR                     ((BYTE) 47)
#define MSG_GOT_SERVO_ERROR                     ((BYTE) 48)
#define MSG_GET_DATA_TABLE_SIZE                 ((BYTE) 49)
#define MSG_GOT_DATA_TABLE_SIZE                 ((BYTE) 50)
#define MSG_SET_DATA_TABLE_SIZE                 ((BYTE) 51)
#define MSG_52                                  ((BYTE) 52)
#define MSG_GET_DATA_TABLE_ELEMENT              ((BYTE) 53)
#define MSG_GOT_DATA_TABLE_ELEMENT              ((BYTE) 54)
#define MSG_SET_DATA_TABLE_ELEMENT              ((BYTE) 55)
#define MSG_56                                  ((BYTE) 56)
#define MSG_GET_DATA_TABLE_ROW                  ((BYTE) 57)
#define MSG_GOT_DATA_TABLE_ROW                  ((BYTE) 58)
#define MSG_SET_DATA_TABLE_ROW                  ((BYTE) 59)
#define MSG_60                                  ((BYTE) 60)
#define MSG_GET_CONTROLLER_STATE                ((BYTE) 61)
#define MSG_GOT_CONTROLLER_STATE                ((BYTE) 62)
#define MSG_SET_CONTROLLER_STATE                ((BYTE) 63)
#define MSG_64                                  ((BYTE) 64)
#define MSG_GET_SYSCONFIG_BYTE                  ((BYTE) 65)
#define MSG_GOT_SYSCONFIG_BYTE                  ((BYTE) 66)
#define MSG_SET_SYSCONFIG_BYTE                  ((BYTE) 67)
#define MSG_68                                  ((BYTE) 68)
#define MSG_GET_OTHER_IO_COUNTS                 ((BYTE) 69)
#define MSG_GOT_OTHER_IO_COUNTS                 ((BYTE) 70)
#define MSG_GET_32_ANALOG_INS                   ((BYTE) 71)
#define MSG_GOT_32_ANALOG_INS                   ((BYTE) 72)
#define MSG_GET_32_ANALOG_OUTS                  ((BYTE) 73)
#define MSG_GOT_32_ANALOG_OUTS                  ((BYTE) 74)
#define MSG_GET_50_REGISTERS                    ((BYTE) 75)
#define MSG_GOT_50_REGISTERS                    ((BYTE) 76)
#define MSG_GET_16_REGISTERS                    ((BYTE) 77)
#define MSG_GOT_16_REGISTERS                    ((BYTE) 78)
#define MSG_GET_128_INPUTS                      ((BYTE) 79)
#define MSG_GOT_128_INPUTS                      ((BYTE) 80)
```

```
#define MSG_GET_128_OUTPUTS                     ((BYTE) 81)
#define MSG_GOT_128_OUTPUTS                     ((BYTE) 82)
#define MSG_SET_64_ANALOG_OUTS                  ((BYTE) 85)
#define MSG_GET_N_REGISTERS                     ((BYTE) 87)
#define MSG_GOT_N_REGISTERS                     ((BYTE) 88)

// special message for 2217 v3.8 data structure
#define MSG_GET_2217_DATA                       ((BYTE) 83)
#define MSG_GOT_2217_DATA                       ((BYTE) 84)

// Variant data types
#define MSG_GET_VREGISTERROW                    ((BYTE) 89)
#define MSG_GOT_VREGISTERROW                    ((BYTE) 90)
#define MSG_GET_VPROPERTIES                     ((BYTE) 91)
#define MSG_GOT_VPROPERTIES                     ((BYTE) 92)
#define MSG_GET_VREGISTER                       ((BYTE) 93)
#define MSG_GOT_VREGISTER                       ((BYTE) 94)
#define MSG_SET_VREGISTER                       ((BYTE) 95)
#define MSG_GET_RUNCOMMAND                      ((BYTE) 97)
#define MSG_GOT_RUNCOMMAND                      ((BYTE) 98)
#define MSG_GET_VREGISTER_BLOCK                ((BYTE) 109)
#define MSG_GOT_VREGISTER_BLOCK                ((BYTE) 110)
#define MSG_SET_VREGISTER_BLOCK                ((BYTE) 111)

#define MSG_GET_VREGISTER_RANDOM_BLOCK         ((BYTE) 113)
#define MSG_GOT_VREGISTER_RANDOM_BLOCK         ((BYTE) 114)

// devicenet and/or distributed io messages
#define MSG_UNLOAD_REMOTE_DATA                 ((BYTE) 101)
#define MSG_REMOTE_DATA_PACKET                 ((BYTE) 102)
#define MSG_LOAD_REMOTE_DATA                   ((BYTE) 103)
#define MSG_104                                ((BYTE) 104)
#define MSG_GET_REMOTE_IO                      ((BYTE) 105)
#define MSG_GOT_REMOTE_IO                      ((BYTE) 106)
```

Variant Structures

The distribution file `Ctccom32v2.h` is available from the Downloads page on Control Technology's web site and contains the definitions for the structures used with the CTC communications DLL. The DLL conforms to the packet structure discussed within this document. In summary below are the definitions. Note the structures are packed, aligned on a byte boundary:

```
#define BIT0 0x0001
#define BIT1 0x0002
#define BIT2 0x0004
#define BIT3 0x0008
#define BIT4 0x0010

#define VARIANT_MAX_STRING 223
#define VARIANT_INTEGER BIT0
#define VARIANT_UINTEGER BIT1
#define VARIANT_STRING BIT2
#define VARIANT_FLOAT BIT3
#define VARIANT_DOUBLE BIT4

typedef struct
{
        int numAccess; // number of items to access
        int rowInc;           // row increment, if 0 just read columns based upon colInc.
        int colInc;           // col increment, if 0 just increment rows.
        int arraysizeCols;    // Used on write operation, -1 do not expand existing
                              // columns, else columns desired.  Rows will automatically
                              // grow as needed
} BLOCKACCESS;

typedef struct
{
   int type;   // type of storage being used or requested
                       // If -1 on read then return current, else set to type want.
                       // On write must set to type that is stored within this structure
   unsigned char precision;  // double to string conversion precision %.6f default
                              // On read is what is presently set, write what want.
   unsigned char flags;// special flags for processing so far only
                       // VARIANT_INDIRECTION_FLAG used, can be used to set property
                       // in ->settings on write operation, no effect on read.  Written
                       // value becomes register to reference for further operations.
   unsigned char cmd;// 00, no operation other than read/write specified, else do defined
               // operation.  Currently have write for properties access to 'settings'
                       // VARIANT_CMD_SET_INDIRECTION and VARIANT_CMD_CLEAR_INDIRECTION,
                       // write value ignored.
   unsigned char pad;
   unsigned short taskHandle; // task number (offset in task array + 1, where 0 is 1) or
                              / handle thus usable from remote or 'C' API, 4096 to
                              // 65535, set to 0 for public reg.
   unsigned short slength;    // this is reserved for later use and possible string
                              // length if want unsigned char, 0 - 255 values,
                              // VARIANT_BYTE, future type

   unsigned int indexCol;       // Column dimension index reference
   unsigned int indexRow;       // Row dimension index reference
   union                      // Data that was read or has been written of 'type'
   {
        int iValue;
        unsigned int uiValue;
        float fValue;
        double dValue;
        unsigned int dSwap[2]; // used to swap doubles for PC access
        char sValue[VARIANT_MAX_STRING+1];
   } data;
} VARIANT_STORAGE;

#define MAX_VARIANT_BLOCK_32BITS      346     // 346 integers
#define MAX_VARIANT_BLOCK_64BITS      173     // 173 doubles
#define MAX_VARIANT_RANDOM_BLOCK      (MAX_VARIANT_BLOCK_32BITS/3)  // 115 items

#define MAX_VARIANT_BLOCK_32BITS_SERIAL 50
```

```
#define MAX_VARIANT_BLOCK_64BITS_SERIAL (MAX_VARIANT_BLOCK_32BITS_SERIAL/2)// 25 doubles
#define MAX_VARIANT_RANDOM_BLOCK_SERIAL (MAX_VARIANT_BLOCK_32BITS_SERIAL/3)// 16 items


typedef struct
{
        int reg;  // may at some point reserve the upper 16 bits of this
                 // integer for 'type' req.
        int row;
        int col;
} VARIANT_ACCESS_REQUEST;

// Allow for block reads
typedef struct
{
   int type;   // type of storage being used or requested
               // If -1 on read then return current, else set to type want.
               // On write must set to type that is stored within this structure
               // write not supported for block access
               // If block access type field will be 0 if error else type of first cell.
               // slength will be the number of elements returned within data.block.?[n]
   unsigned char precision;  // double to string conversion precision %.6f default
                              // On read is what is presently set, write what want.
   unsigned char flags;       // special flags for processing so far only
                      // VARIANT_INDIRECTION_FLAG used, can be used to set property
                      // in ->settings on write operation, no effect on read.  Written
                      // value becomes register to reference for further operations.
   unsigned char cmd;// 00, no operation other than read/write specified, else do defined
               // operation.  Currently have write for properties access to 'settings'
               // VARIANT_CMD_SET_INDIRECTION and VARIANT_CMD_CLEAR_INDIRECTION,
               // write value ignored.
   unsigned char pad;
   unsigned short taskHandle; // task number (offset in task array + 1, where 0 is 1) or
                              // handle thus usable from remote or ‘C’ API, 4096 to
                              // 65535, set to 0 for public reg.

   unsigned short slength;    // this is reserved for later use and possible string
                              // length if want unsigned char, 0 – 255 values,
                              // VARIANT_BYTE, future type
   unsigned int indexCol;     // Column dimension index reference
   unsigned int indexRow;     // Row dimension index reference
   union                      // Data that was read or has been written of 'type'
   {
        int iValue;
        unsigned int uiValue;
        float fValue;
        double dValue;
        unsigned int dSwap[2]; // used to swap doubles for PC access
        char *psValue;
        char sValue[VARIANT_MAX_STRING+1];
        // will be stored in same VARIANT_STORAGE upon return, thus
        // data.blockread.numAccess * sizeof(variant type)
        // if BLOCKACCESS then iValue[n], fValue[n], or dValue[n] up to
        // MAX_VARIANT_READBLOCK_SIZE

        struct
        {
                BLOCKACCESS blockaccess;      // Defines block read of variant cells, data
                                              // storage must be big enough since
                union
                {
                        int ibValue[MAX_VARIANT_BLOCK_32BITS];
                        float fbValue[MAX_VARIANT_BLOCK_32BITS];
                        double dbValue[MAX_VARIANT_BLOCK_64BITS];
                      union
                      {
                              int ibValue;
                              float fbValue;
                              double dbValue;
                      } random[MAX_VARIANT_RANDOM_BLOCK];
                      VARIANT_ACCESS_REQUEST request[MAX_VARIANT_RANDOM_BLOCK];
               };
       } block;
   } data;
} VARIANT_STORAGE_BLOCK;

typedef struct
{
   int type;   // type of storage being used or requested
               // If -1 on read then return current, else set to type want.
               // On write must set to type that is stored within this structure
               // write not supported for block access
               // If block access type field will be 0 if error else type of first cell.
               // slength will be the number of elements returned within data.block.?[n]
   unsigned char precision;  // double to string conversion precision %.6f default
                              // On read is what is presently set, write what want.
   unsigned char flags;       // special flags for processing so far only
                      // VARIANT_INDIRECTION_FLAG used, can be used to set property
                      // in ->settings on write operation, no effect on read.  Written
                      // value becomes register to reference for further operations.
   unsigned char cmd;// 00, no operation other than read/write specified, else do defined
               // operation.  Currently have write for properties access to 'settings'
               // VARIANT_CMD_SET_INDIRECTION and VARIANT_CMD_CLEAR_INDIRECTION,
               // write value ignored.
   unsigned char pad;
   unsigned short taskHandle; // task number (offset in task array + 1, where 0 is 1) or
                              // handle thus usable from remote or ‘C’ API, 4096 to
                              // 65535, set to 0 for public reg.

   unsigned short slength;    // this is reserved for later use and possible string
                              // length if want unsigned char, 0 - 255 values,
                              // VARIANT_BYTE, future type
   unsigned int indexCol;     // Column dimension index reference
   unsigned int indexRow;     // Row dimension index reference
   union                      // Data that was read or has been written of 'type'
   {
       int iValue;
       unsigned int uiValue;
       float fValue;
       double dValue;
       unsigned int dSwap[2]; // used to swap doubles for PC access
       char *psValue;
       char sValue[VARIANT_MAX_STRING+1];
       // will be stored in same VARIANT_STORAGE upon return, thus
       // data.blockread.numAccess * sizeof(variant type)
       // if BLOCKACCESS then iValue[n], fValue[n], or dValue[n] up to
       // MAX_VARIANT_READBLOCK_SIZE

       struct
       {
               BLOCKACCESS blockaccess;      // Defines block read of variant cells, data
                                             // storage must be big enough since
               union
               {
                      int ibValue[MAX_VARIANT_BLOCK_32BITS_SERIAL];
                      float fbValue[MAX_VARIANT_BLOCK_32BITS_SERIAL];
                      double dbValue[MAX_VARIANT_BLOCK_64BITS_SERIAL];
                      union
```

```
                       {
                              int ibValue;
                              float fbValue;
                              double dbValue;
                       } random[MAX_VARIANT_RANDOM_BLOCK_SERIAL];
                       VARIANT_ACCESS_REQUEST request[MAX_VARIANT_RANDOM_BLOCK_SERIAL];
               };
        } block;
   } data;
} VARIANT_STORAGE_BLOCK_SERIAL;
```

Variant Access Commands

Get Properties - Command 91

Command 91 reads the current properties of a variant which includes its number of rows and columns as well as default floating point precision (typically 6).

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
05H Specifies the packet length
5BH Get Properties function code
LSB - MSB Specifies the variant register number whose properties are desired. Specified with the least significant byte first.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Format of Controller Response

ØAH Specifies the packet length.
5CH Get Properties response code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
LSB- MSB Number of columns.
LSB- MSB Number of rows.
<Precision Byte> - Floating point precision currently set.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Read a Variant - Command 93

Command 93 reads a Variant cell. If the Variant is not an array simply set the row and column to 0 in the structure.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
<sizeof(VARIANT_STORAGE) + 5> Specifies the packet length
5DH Read a Variant function code

LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE structure> Variant storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Format of Controller Response

<sizeof(VARIANT_STORAGE) + 3> Specifies the packet length
5EH Read a Variant response code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE structure> Variant storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Example Structure initialization:
Read 36201[2][5] as a double – (36201 is the LSB/MSB in the message sent)

```
VARIANT_STORAGE v;
memset((void *)&v,0,sizeof(VARIANT_STORAGE));
v.indexCol = 5;
v.indexRow = 2;
v.precision = 6;
v.type = VARIANT_DOUBLE;
```

Depending upon which type you are accessing the returned Variant will be accessed as follows where `rp` is a pointer to the receive buffer.

```
      // Got the data
      memcpy((void *)&v,rp+4,sizeof(VARIANT_STORAGE));
      switch(v.type)
      {
            case VARIANT_FLOAT:
                  variant->FloatVar = v.data.fValue;
                  break;
            case VARIANT_DOUBLE:
                  variant->DoubleVar = v.data.dValue;
                  break;
            case VARIANT_STRING:
                  variant->slength = v.slength;
                  if (variant->slength > VARIANT_MAX_STRING)
                  {
                        // too big
                        return FAILURE;
                  }
                  memcpy(variant->StringArray, v.data.sValue, variant-
>slength);
```

```
                // null terminate
                variant->StringArray[variant->slength] = 0x00;
                break;
        case VARIANT_INTEGER:
                variant->LongVar = v.data.iValue;
                break;
        default:
                return FAILURE;  // unknown type
}
return SUCCESS;
```

Change a Variant - Command 95

Command 95 writes a Variant cell. If the Variant is not an array simply set the row and column to 0 in the structure.

Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
<sizeof(VARIANT_STORAGE) + 5> Specifies the packet length
5FH Change a Variant function code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE structure> Variant storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
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

Example Structure initialization:
Write 36201[2][5]– (36201 is the LSB/MSB in the message sent)

```
VARIANT_STORAGE v;
memset((void *)&v,0,sizeof(VARIANT_STORAGE));
v.indexCol = 5;
v.indexRow = 2;
v.precision = 6;
```

For each type of data writing where variant is user structure (reference previous section):

```
switch(variant->type)
{
        case VARIANT_FLOAT:
                v.data.fValue = variant->FloatVar;
                break;
```

```
            case VARIANT_DOUBLE:
                  v.data.dValue = variant->DoubleVar;
                  break;
            case VARIANT_STRING:
                  if (variant->slength > VARIANT_MAX_STRING)
                  {
                        // too big
                        return FAILURE;
                  }
                  memcpy(v.data.sValue, variant->StringArray,variant-
>slength);
                  // null terminate
                  v.data.sValue[variant->slength] = 0x00;
                  break;
            case VARIANT_INTEGER:
                  v.data.iValue = variant->LongVar;
                  break;
            default:
                  return FAILURE;
      }
```

… Send data packet and await ACK …

Read a Variant Array Block - Command 109

Command 109 performs a read starting at a specific row/column position in a Variant array and reads the requested number of cells sequentially or until there are no more cells.

Format of Message Sent to Controller
UDP/TCP

Ø1H Identifies the packet as using the CTC binary protocol
< 5> Specifies the packet length for the message, without the Variant area since including it would make the message exceed byte storage size.
6DH Reads a Variant Block function code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field, **not used**.
FFH Signals the end of the message.

Serial Port:

Ø1H Identifies the packet as using the CTC binary protocol
<sizeof(VARIANT_STORAGE_BLOCK_SERIAL) + 5> Specifies the packet length.
6DH Reads a Variant Block function code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK_SERIAL structure> Variant block storage area.

Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Format of Controller Response
UDP/TCP

< 3> Specifies the packet length for the message, without the Variant area since including it would make the message exceed byte storage size.
6EH Reads a Variant Block function code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK structure> Variant storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field, **not used**.
FFH Signals the end of the message.

Serial Port:
<sizeof(VARIANT_STORAGE_BLOCK_SERIAL) + 3> Specifies the packet length.
6EH Reads a Variant Block function code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK_SERIAL structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Example Structure initialization:
Read 36201[0][0] as an integer, 5 consecutive cells – (36201 is the LSB/MSB in the message sent). There are 35 columns in each row.

```
VARIANT_STORAGE_BLOCK v;
if (ctc->connType == SERIAL)
{
      sz = sizeof(VARIANT_STORAGE_BLOCK_SERIAL);
      length = sz+5;    // packet length

}
else
{
      sz = sizeof(VARIANT_STORAGE_BLOCK);
      length = 5;
}
// initialize the variant structure
memset((void *)&v.type,0,sz);
v.indexCol = 0;
v.indexRow = 0;
```

```
v.precision = 6;
v.type = VARIANT_INTEGER;  // String not supported
v.data.block.blockaccess.colInc = 1;
v.data.block.blockaccess.rowInc = 1;
v.data.block.blockaccess.numAccess = 5;
v.data.block.blockaccess.arraysizeCols = 35;  // tells when to
                                        // increment row number
```

Depending upon which type you are accessing the returned Variant will be accessed as follows where `rp` is a pointer to the receive buffer. `sz` is the size of the structure used, that of VARIANT_STORAGE_BLOCK or VARIANT_STORAGE_BLOCK_SERIAL.

```
// Got the data
memcpy((void *)&v,rp+4,sz);
// move the data into the vb structure
variant->type = v.type;             // type of data read
variant->slength = v.slength; // number read
switch(variant->type)
{
      case VARIANT_FLOAT:
            memcpy(variant->block.fbValue, v.data.block.fbValue,
                  sizeof(float) * variant->slength);
            break;
      case VARIANT_DOUBLE:
            memcpy(variant->block.dbValue, v.data.block.dbValue,
                  sizeof(double) * variant->slength);
            break;
      case VARIANT_INTEGER:
      case VARIANT_UINTEGER:
            memcpy(variant->block.ibValue, v.data.block.ibValue,
                  sizeof(int) * variant->slength);
            break;
      default:
            return FAILURE;  // unknown type
}
return SUCCESS;
```

Write a Variant Array Block - Command 111

Command 111 performs a write starting at a specific row/column position in a Variant array and writes the requested number of cells sequentially or until there are no more cells.

Format of Message Sent to Controller
UDP/TCP

Ø1H Identifies the packet as using the CTC binary protocol
< 5> Specifies the packet length for the message, without the Variant area since including it would make the message exceed byte storage size.
6FH Writes a Variant Block function code

LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field, **not used**.
FFH Signals the end of the message.

Serial Port:

Ø1H Identifies the packet as using the CTC binary protocol
<sizeof(VARIANT_STORAGE_BLOCK_SERIAL) + 5> Specifies the packet length.
6DH Writes a Variant Block function code
LSB - MSB Specifies the variant register number. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK_SERIAL structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
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

Example Structure initialization:
Write 36201[0][0] as a floats, 5 consecutive cells – (36201 is the LSB/MSB in the message sent). There are 35 columns in each row.

```
VARIANT_STORAGE_BLOCK v;
if (ctc->connType == SERIAL)
{
      sz = sizeof(VARIANT_STORAGE_BLOCK_SERIAL);
      length = sz+5;    // packet length

}
else
{
      sz = sizeof(VARIANT_STORAGE_BLOCK);
      length = 5;
}
// initialize the variant structure
memset((void *)&v.type,0,sz);
v.indexCol = 0;
v.indexRow = 0;
v.precision = 6;
v.type = VARIANT_FLOAT;  // String not supported
```

```
      v.data.block.blockaccess.colInc = 1;
      v.data.block.blockaccess.rowInc = 1;
      v.data.block.blockaccess.numAccess = 5;  // assume variant-
>numAccess is 5
      v.data.block.blockaccess.arraysizeCols = 35;
      // move the data in now where 'variant' is a user structure of
choice
      memcpy((void *)&v.data.block.fbValue, variant->fbValue,
sizeof(float) * variant->numAccess);
```

… Send data packet and await ACK …

Read a Block of Variants Randomly - Command 113

Command 113 reads variants in a user defined order, rather than sequentially, this includes any cell (row/column) or different variant. All will be returned of the same type, integer, float, or double. String is not supported.

Format of Message Sent to Controller
UDP/TCP

Ø1H Identifies the packet as using the CTC binary protocol
< 5> Specifies the packet length for the message, without the Variant area since including it would make the message exceed byte storage size.
71H Reads a random Variant Block function code
LSB - MSB Specifies the variant register number, may be any value such as 0x0000. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field, **not used**.
FFH Signals the end of the message.

Serial Port:

Ø1H Identifies the packet as using the CTC binary protocol
<sizeof(VARIANT_STORAGE_BLOCK_SERIAL) + 5> Specifies the packet length.
71H Reads a random Variant Block function code
LSB - MSB Specifies the variant register number, may be any value such as 0x0000. Specified with the least significant byte first.
<VARIANT_STORAGE_BLOCK_SERIAL structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field.
FFH Signals the end of the message.

Format of Controller Response
UDP/TCP

< 3> Specifies the packet length for the message, without the Variant area since including it would make the message exceed byte storage size.

72H Read a random Variant Block response code
LSB - MSB returns what was sent.
<VARIANT_STORAGE_BLOCK structure> Variant storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field, **not used**.
FFH Signals the end of the message.

Serial Port:

<sizeof(VARIANT_STORAGE_BLOCK_SERIAL) + 3> Specifies the packet length.
72H Reads a random Variant Block response code
LSB - MSB returns what was sent.
<VARIANT_STORAGE_BLOCK_SERIAL structure> Variant block storage area.
Checksum Contains the complement of the modulo-256 sum of all bytes after the length field
FFH Signals the end of the message.

Example Structure initialization:
Read 36301[0][0], 36301[0][1], 36301[0][2], 36301[1][0], 36302[0][0] as doubles.

```
VARIANT_STORAGE_BLOCK v;
// initialize the variant structure
if (ctc->connType == SERIAL)
{
      sz = sizeof(VARIANT_STORAGE_BLOCK_SERIAL);
      length = sz+5;    // packet length

}
else
{
      sz = sizeof(VARIANT_STORAGE_BLOCK);
      length = 5;
}
// initialize the variant structure
memset((void *)&v.type,0,sz);
v.precision = 6;
v.type = VARIANT_DOUBLE;
v.data.block.blockaccess.numAccess = 5;
// 36301[0][0]
v.data.block.request[0].reg = 36301;
v.data.block.request[0].row = 0;
v.data.block.request[0].col = 0;
// 36301[0][1]
v.data.block.request[1].reg = 36301;
v.data.block.request[1].row = 0;
v.data.block.request[1].col = 1;
// 36301[0][2]
v.data.block.request[2].reg = 36301;
v.data.block.request[2].row = 0;
v.data.block.request[2].col = 2;
```

```
// 36301[1][0]
v.data.block.request[3].reg = 36301;
v.data.block.request[3].row = 1;
v.data.block.request[3].col = 0;
// 36302[0][0]
v.data.block.request[4].reg = 36302;
v.data.block.request[4].row = 0;
v.data.block.request[4].col = 0;
```

Depending upon which type you are accessing, the returned Variant will be accessed as follows where `rp` is a pointer to the receive buffer. `sz` is the size of the structure used, that of VARIANT_STORAGE_BLOCK or VARIANT_STORAGE_BLOCK_SERIAL.

```
// Got the data
memcpy((void *)&v,rp+4,sz);
// move the data into the vb structure
variant->type = v.type;            // type of data read
variant->slength = v.slength; // number read
memcpy(&variant->block.random[0], &v.data.block.random[0],
      sizeof(v.data.block.random) * variant->slength);
return SUCCESS;
```

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 7FH for flags, corresponding to flags 1 through 128. 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 individual command descriptions earlier in this chapter for more information.
- 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 format), 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

4EH Indicates the register contents function code

LSB - MSB (2 bytes) Indicates bank of registers, ØØØØH - Ø3D9H

LSB - MSB (4 bytes) Contains the value of the first register in the group. For a description of register data, see the description for single register read.

LSB - MSB (4 bytes) Contains the value of the second register in the group.

Additional LSB - MSB lines follow for the remainder of the 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, limited from 1 to 1000.

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 registers to follow, ØØH - 13H
LSB - MSB (4 bytes) Contains the value of the first register in the group. For a description of register data, see the description for single register read.
LSB - MSB (4 bytes) Contains the value of the second register in the group.
Additional LSB - MSB lines follow for the remainder of the 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

Request Random Registers from List - Command 87

Command 87 reads the values of up to 50 random registers from a list.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
??H Specifies the packet length, all following bytes, including checksum but not ending FFH.
57H Indicates Random Register Read function code
NUMREGS – Single byte from 1 to 50 representing number of following random registers to read. Registers are listed as 2 byte shorts (16 bits), lsb/msb, results are returned as 32 bit integers.
LSB – MSB1 First register number to read, 16 bits
LSB – MSB2 Second register number to read, 16 bits
…
LSB – MSBN Last register number to read, 16 bits
Checksum Contains the complement of the modulo-256 sum of all the bytes after the packet length bytes
FFH Signals the end of the message

Format of Controller Response

??H Specifies the packet length

58H Indicates the register contents function code
NUMREGS – Single byte from 1 to 50 representing number of following random registers results which are being returned. Registers' results are returned as 32 bit integers, lsb to msb.
LSB - MSB (4 bytes) Contains the value of the first register in the group. For a description of register data, see the description for single register read.
LSB - MSB (4 bytes) Contains the value of the second register in the group. Additional LSB - MSB lines follow for the remainder of the NUMREGS registers in the group.
…
Checksum Contains the complement of the modulo-256 sum of the previous bytes, excluding packet length
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 (2 bytes) Specifies the register number, ØØØ1H - ØFFFFH. Specified with the least significant byte first.
LSB - MSB (4 bytes) Four-byte representation of register data, expressed in 2's 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 - 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
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 - 7FH
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

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

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 12H, 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 individual command descriptions earlier in this chapter for more information.
- When the controller responds with the data for a group of 8 inputs or outputs, the lowest input number is represented by the least significant bit, 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 Inputs 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, where 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 within 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 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 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 81

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, where 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 outputs. This continues for a total of 128 outputs.
Checksum Contains the complement of the modulo-256 sum of the previous 18 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 three 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 off-mask-15 Specifies a series of 16 eight-bit masks used to selectively 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 on-mask-15 Specifies a series of 16 eight-bit masks used to selectively 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 output is 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 individual command descriptions earlier in this chapter for more information.

Reading an Analog Input - Command 29

Command 29 reads the value of any of the 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 - FFH
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 of the 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 - FFH
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 of the 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 - FFH
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

Change Multiple Analog Outputs - Command 85

Command 85 changes the value of up to 64 sequential analog outputs.
Format of Message Sent to Controller

Ø1H Identifies the packet as using the CTC binary protocol
Ø6H Specifies the packet length
55H Indicates the Write Multiple Analog Output function code
Analog Output Start Specifies the first output to be changed, Ø1H - FFH
Length Specifies the number of sequential analog outputs to change 01H - 40H
LSB – MSB First 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
…
LSB-MSB Last
Checksum Contains the complement of the modulo-256 sum of the previous 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 - MSB (4 bytes) Contains the four-byte representation of the servo's position. The value is expressed in 2's (complement binary format), 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 - MSB Contains the four-byte representation of the servo's error. The value is expressed in 2's (complement binary format), 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-bit 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 code 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 individual command descriptions earlier in this chapter for more information. The controller may return an error code under the following circumstances:

- The requested data table size is too 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 byte transmitted first
columns 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 byte 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 the data table element, with the least significant byte transmitted first
columns Contains the column number of the 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 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 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 the data table element, with the least significant byte transmitted first.
columns Contains the column number of the 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 a specific data table row and columns by specifying their row and beginning 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

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 57 changes the values in a specific data table row and columns by specifying their row and beginning 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) + 4, where n = number of columns to be changed
3AH 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 individual command descriptions earlier in this chapter for more information.

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 80H
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 8ØH
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 it is 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 is currently suspended or 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, which indicate if any tasks are being executed at 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 numbers are offset by -1.

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 throughout 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. If the controller is running, it returns a value of ØØH.
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.

. For additional information on fault codes, see Chapter 13: Fault Task Handler

Fault step – LSB, MSB, 16 bit, where where ØØØØH = step 1, ØØØ1H = step 2
LSB MSB (4 bytes) Data relating to software fault if any; otherwise 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 - MSB (4 bytes) 32 bit mask, indicating with a 1 or Ø for each of the 32 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 that 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, and additionally run background servers listening for UDP and TCP connections that 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 boundary. (#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 in a two-byte, little endian format (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 Chapter 19: CTNet Binary Protocol for additonal information on the standard CTC Binary Protocol. Messages from the host begin with 0x01, that from the controller are the length of the message in bytes. Both messages 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).

APPENDIX

A

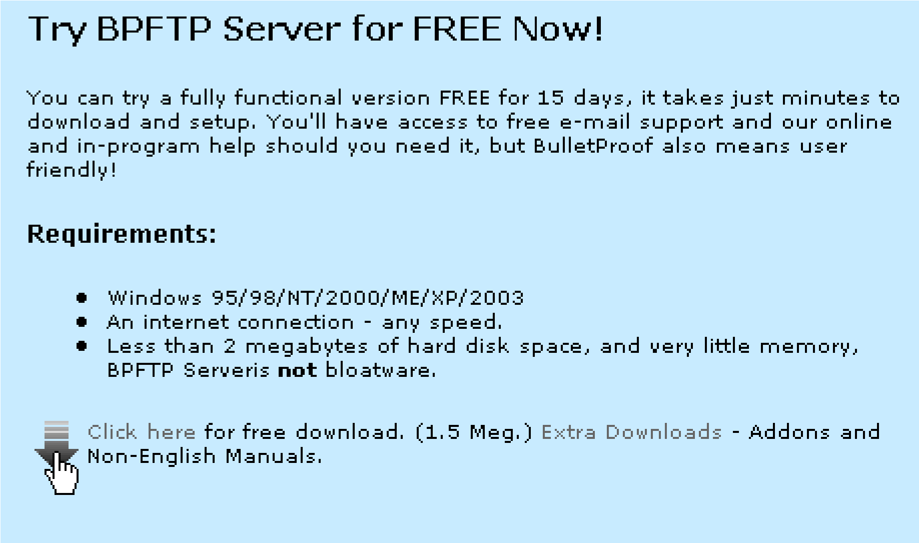
[A] BulletProof FTP Server

BulletProof Software has available a low cost FTP Server ($34.95, 15 day free trial) which can be installed on Windows systems for communications with the Model 5300. This appendix is provided as an initial quick start guide detailing its installation and initial setup. Detailed information and additional support is provided within their manual. The program may be downloaded from their web site at:

http://www.bpftpserver.com/download.php.

Installation

1. From their web site click the download Icon and save the file to a desired directory:



Try BPFTP Server for FREE Now!

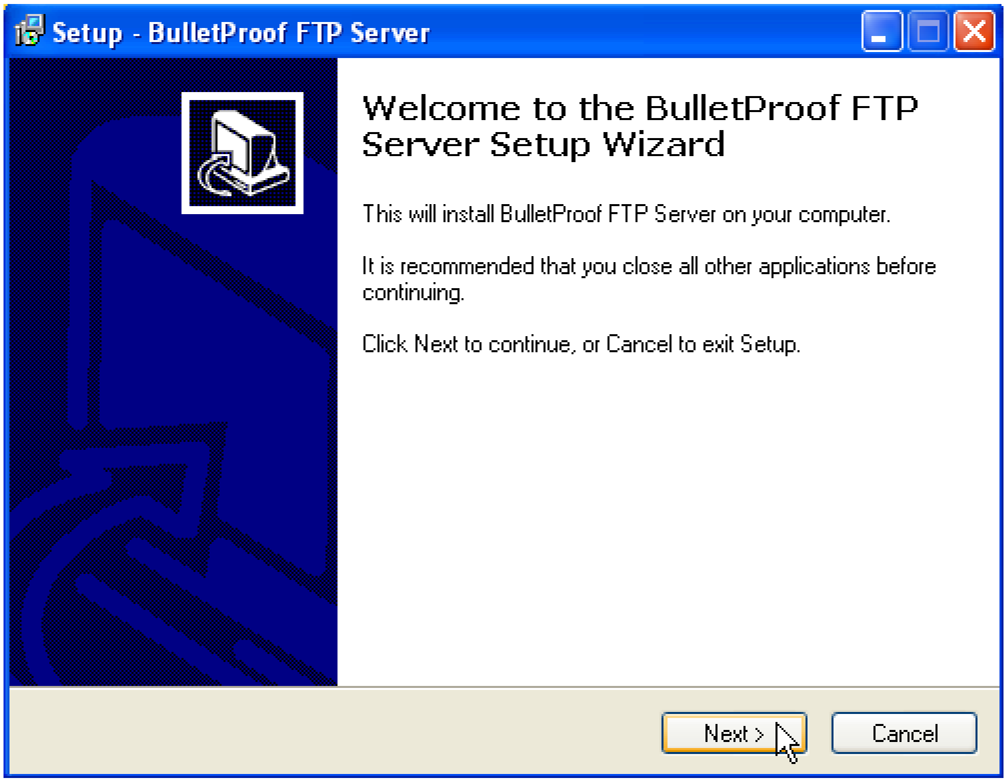
You can try a fully functional version FREE for 15 days, it takes just minutes to download and setup. You'll have access to free e-mail support and our online and in-program help should you need it, but BulletProof also means user friendly!

Requirements:

- Windows 95/98/NT/2000/ME/XP/2003
- An internet connection - any speed.
- Less than 2 megabytes of hard disk space, and very little memory, BPFTP Serveris **not** bloatware.

Click here for free download. (1.5 Meg.) Extra Downloads - Addons and Non-English Manuals.

2. Once downloaded execute their "ftpsetup.exe" file. At the welcome screen click the Next button:



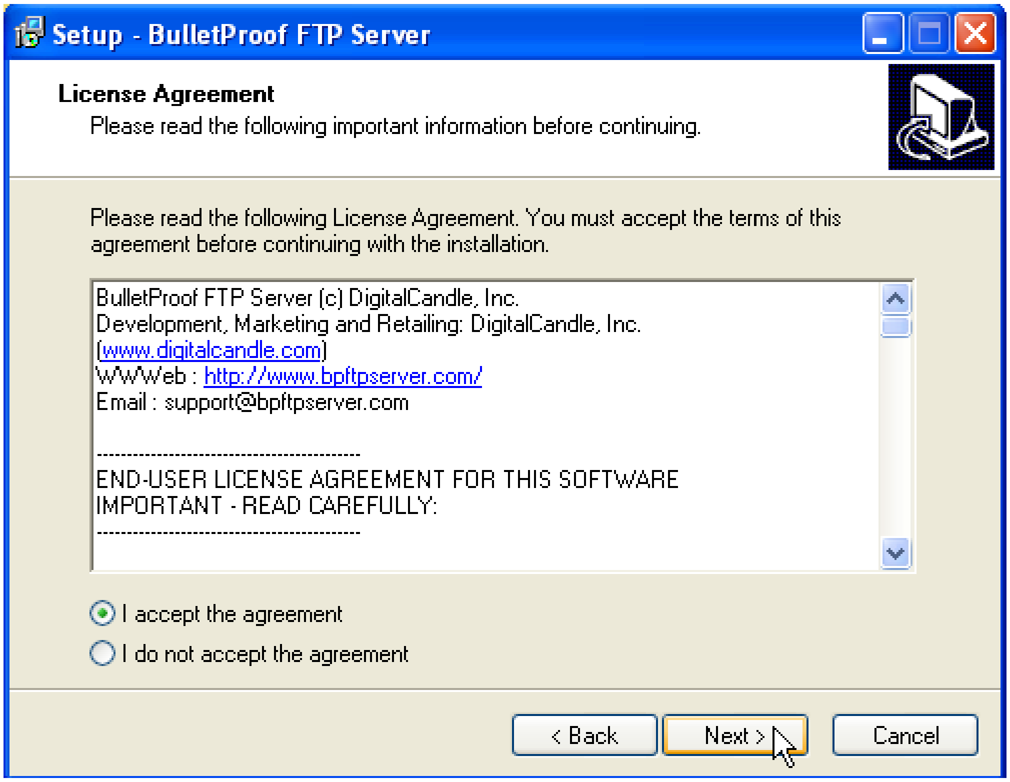
Welcome to the BulletProof FTP Server Setup Wizard

This will install BulletProof FTP Server on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

3. Accept their agreement and click Next:



License Agreement

Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

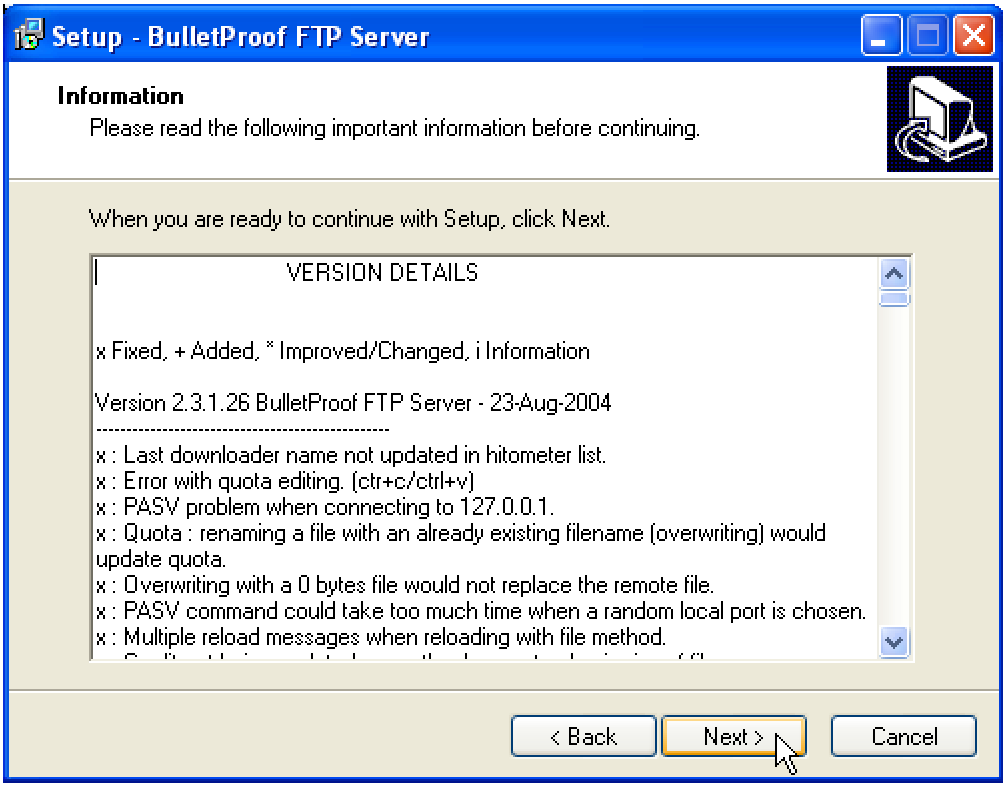
BulletProof FTP Server (c) DigitalCandle, Inc.
Development, Marketing and Retailing: DigitalCandle, Inc.
(www.digitalcandle.com)
WWWeb : http://www.bpftpserver.com/
Email : support@bpftpserver.com

END-USER LICENSE AGREEMENT FOR THIS SOFTWARE
IMPORTANT - READ CAREFULLY:

● I accept the agreement

○ I do not accept the agreement

4. Click Next at the Information screen:



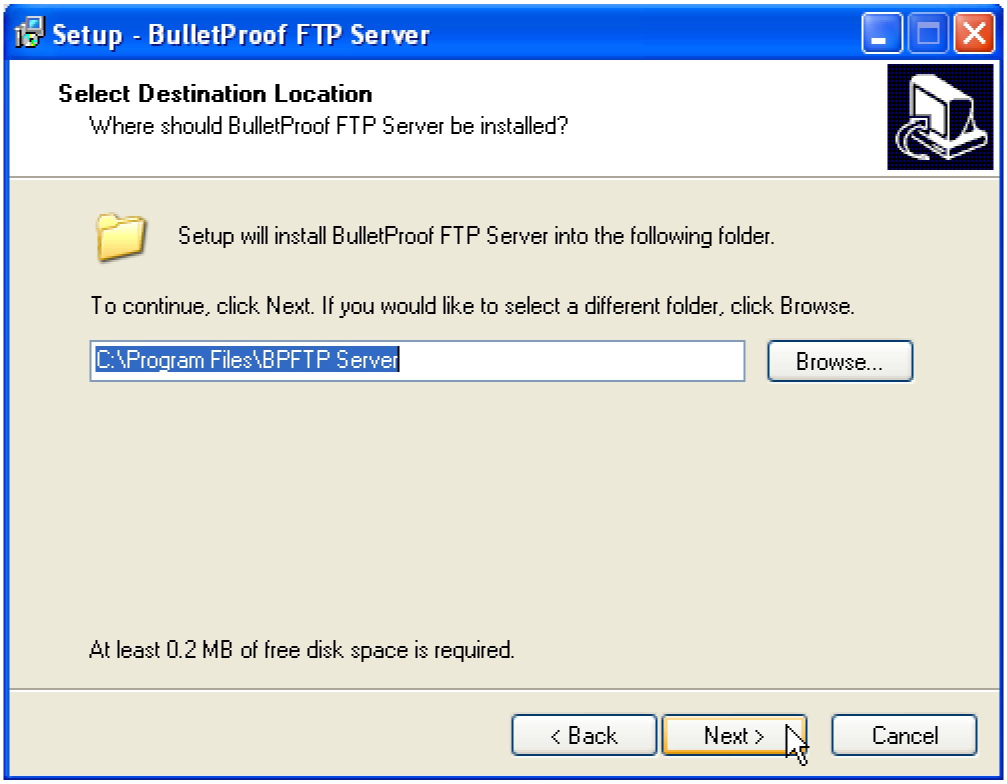
Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

```
                    VERSION DETAILS


x Fixed, + Added, * Improved/Changed, i Information

Version 2.3.1.26 BulletProof FTP Server - 23-Aug-2004
---------------------------------------------
x : Last downloader name not updated in hitometer list.
x : Error with quota editing. (ctr+c/ctrl+v)
x : PASV problem when connecting to 127.0.0.1.
x : Quota : renaming a file with an already existing filename (overwriting) would
update quota.
x : Overwriting with a 0 bytes file would not replace the remote file.
x : PASV command could take too much time when a random local port is chosen.
x : Multiple reload messages when reloading with file method.
```

5. Change the installation directory if required and click Next:



Select Destination Location

Where should BulletProof FTP Server be installed?

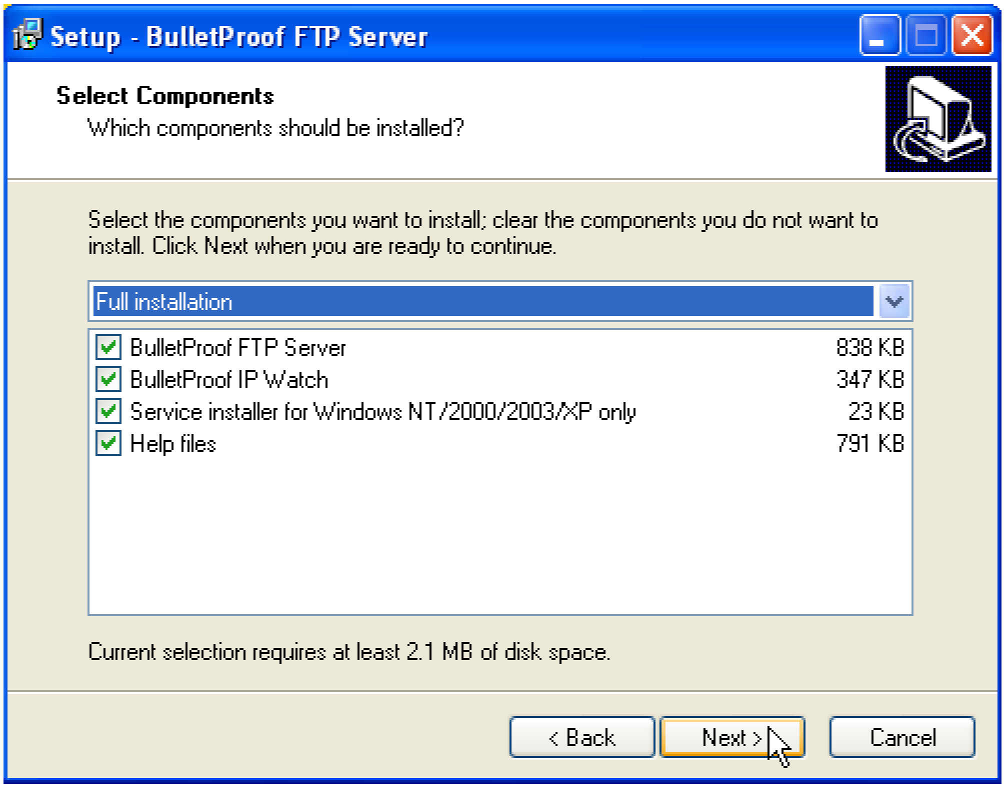
Setup will install BulletProof FTP Server into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\BPFTP Server

At least 0.2 MB of free disk space is required.

6. Accept the defaults of the next few screens and continue as shown:



Select Components

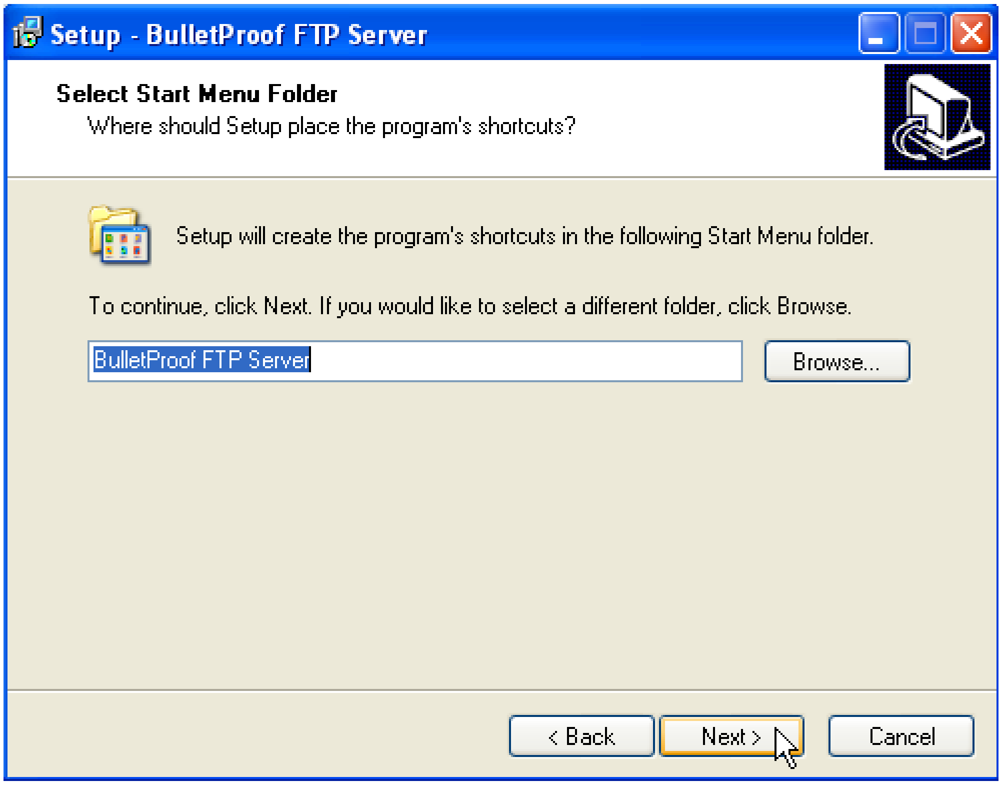
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑ BulletProof FTP Server	838 KB
☑ BulletProof IP Watch	347 KB
☑ Service installer for Windows NT/2000/2003/XP only	23 KB
☑ Help files	791 KB

Current selection requires at least 2.1 MB of disk space.



Setup - BulletProof FTP Server

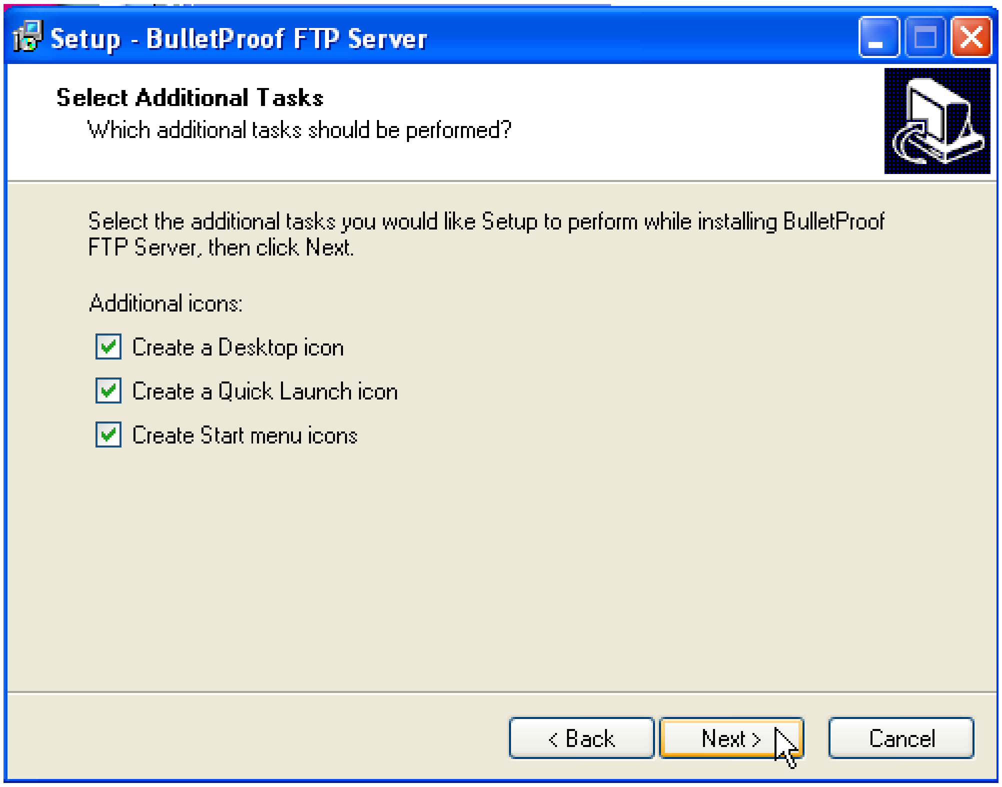
Select Start Menu Folder

Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

BulletProof FTP Server



Setup - BulletProof FTP Server

Select Additional Tasks

Which additional tasks should be performed?

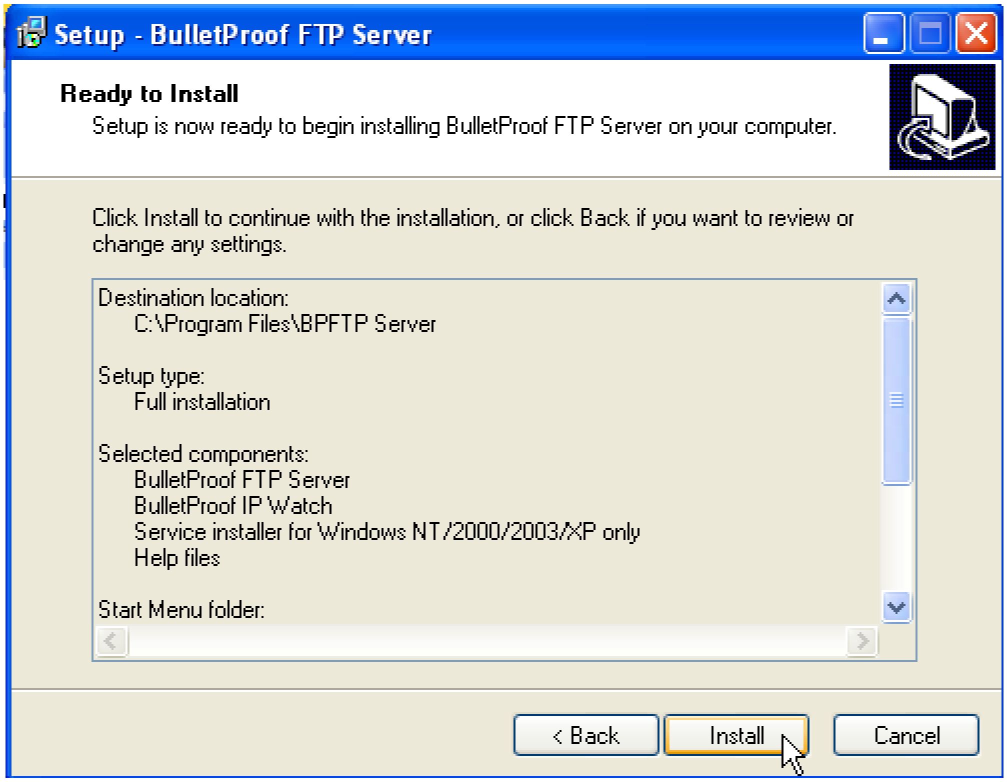
Select the additional tasks you would like Setup to perform while installing BulletProof FTP Server, then click Next.

Additional icons:

☑ Create a Desktop icon

☑ Create a Quick Launch icon

☑ Create Start menu icons



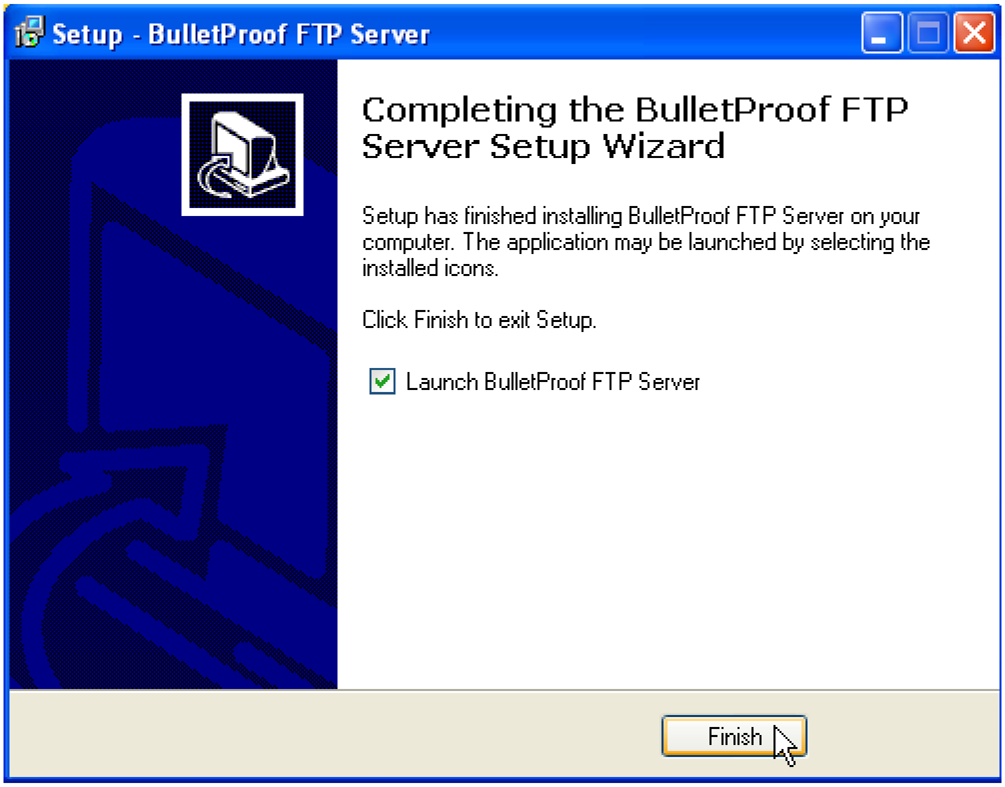
Ready to Install

Setup is now ready to begin installing BulletProof FTP Server on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
      C:\Program Files\BPFTP Server

Setup type:
      Full installation

Selected components:
      BulletProof FTP Server
      BulletProof IP Watch
      Service installer for Windows NT/2000/2003/XP only
      Help files

Start Menu folder:
```

7. Upon completion the final screen will appear, click Finish and the program will automatically be invoked:



Completing the BulletProof FTP Server Setup Wizard

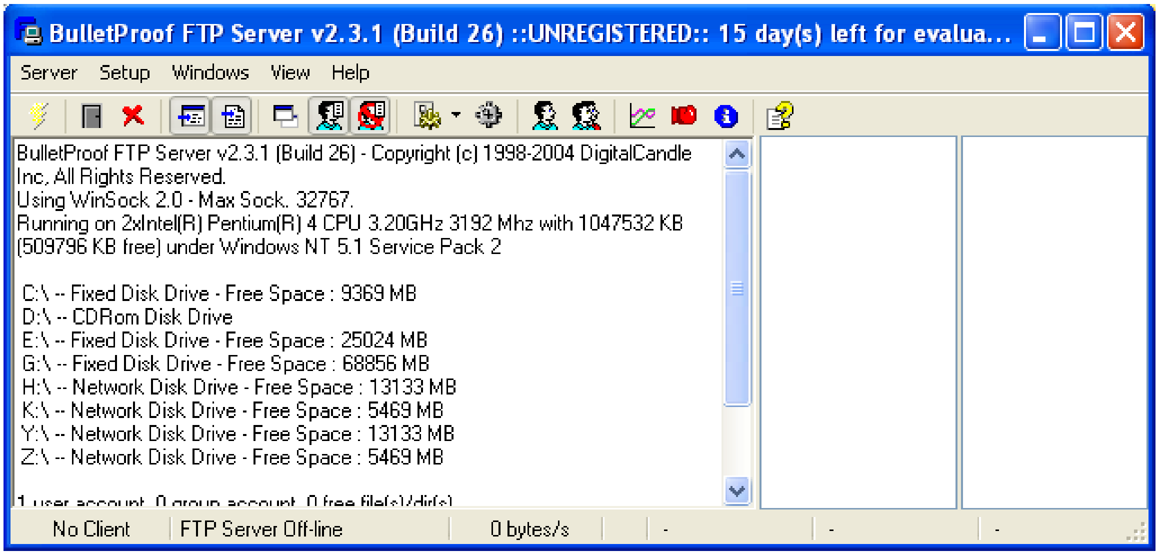
Setup has finished installing BulletProof FTP Server on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

☑ Launch BulletProof FTP Server

Operation

Upon initial install and execution the following will appear:

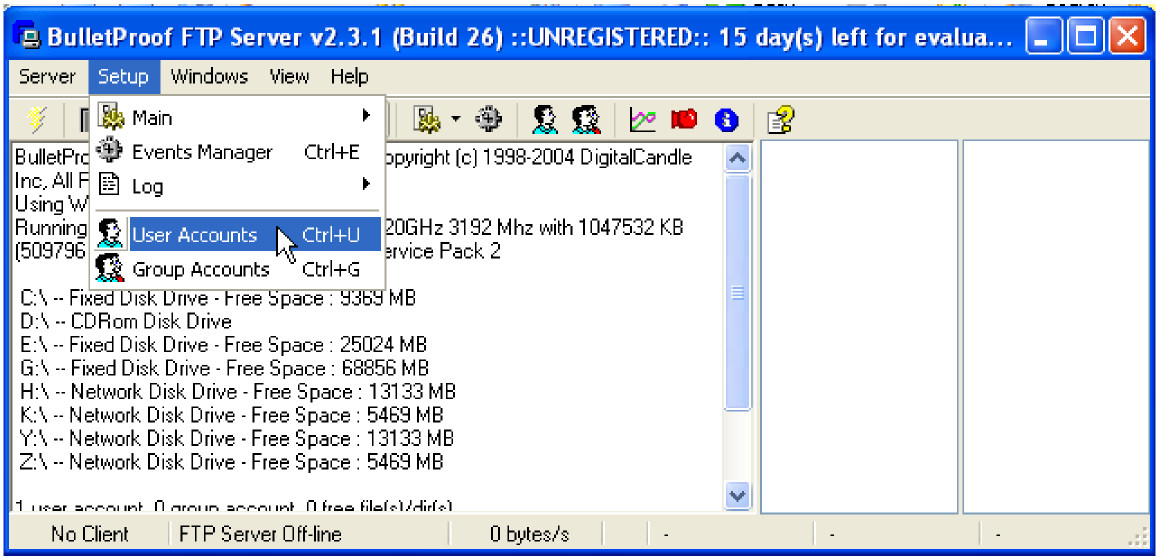


BulletProof FTP Server v2.3.1 (Build 26) ::UNREGISTERED:: 15 day(s) left for evalua...
Server
Setup
Windows
View
Help

```
BulletProof FTP Server v2.3.1 (Build 26) - Copyright (c) 1998-2004 DigitalCandle
Inc, All Rights Reserved.
Using WinSock 2.0 - Max Sock. 32767.
Running on 2xIntel(R) Pentium(R) 4 CPU 3.20GHz 3192 Mhz with 1047532 KB
(509796 KB free) under Windows NT 5.1 Service Pack 2

 C:\ -- Fixed Disk Drive - Free Space : 9369 MB
 D:\ -- CDRom Disk Drive
 E:\ -- Fixed Disk Drive - Free Space : 25024 MB
 G:\ -- Fixed Disk Drive - Free Space : 68856 MB
 H:\ -- Network Disk Drive - Free Space : 13133 MB
 K:\ -- Network Disk Drive - Free Space : 5469 MB
 Y:\ -- Network Disk Drive - Free Space : 13133 MB
 Z:\ -- Network Disk Drive - Free Space : 5469 MB

1 user account, 0 group account, 0 free file(s)/dir(s)
```

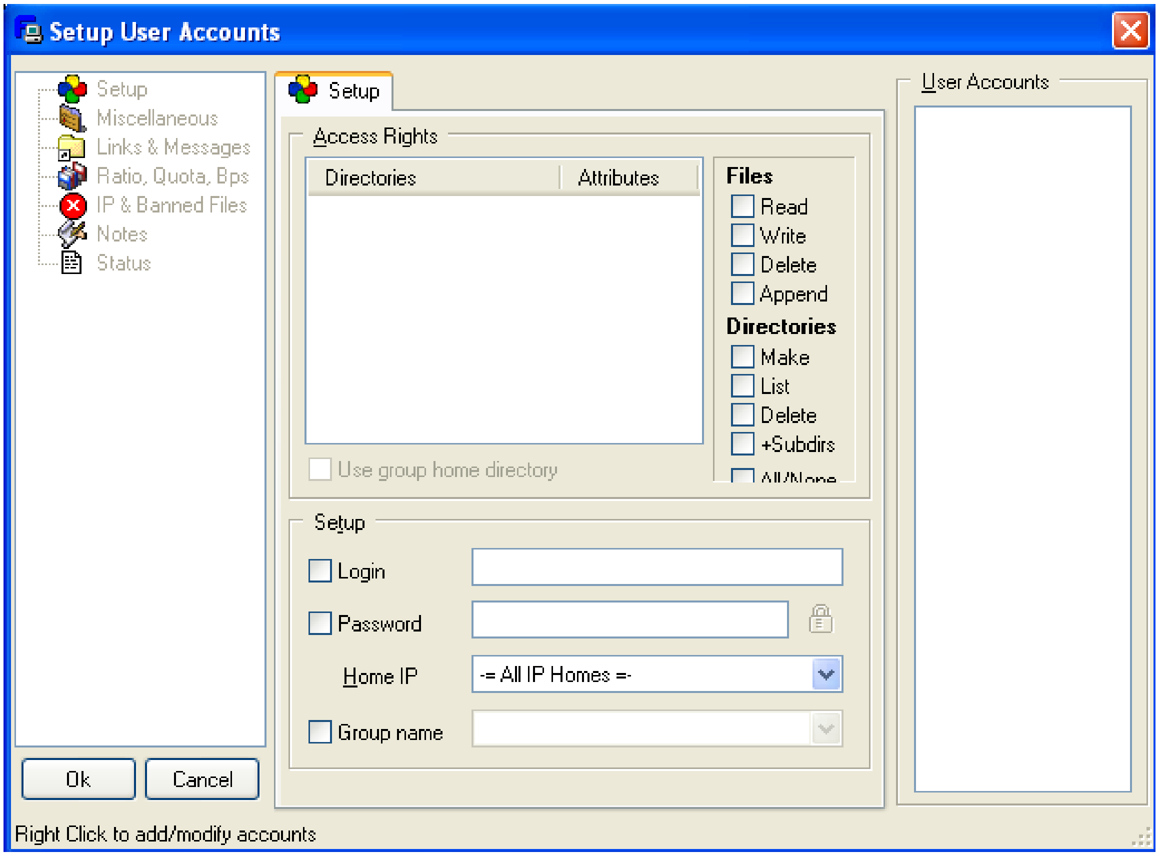
No Client FTP Server Off-line 0 bytes/s - - -

An initial user account must be added. This is the user name and password, along with access rights you will grant this person and/or controller. Click the Setup->User Accounts menu item:



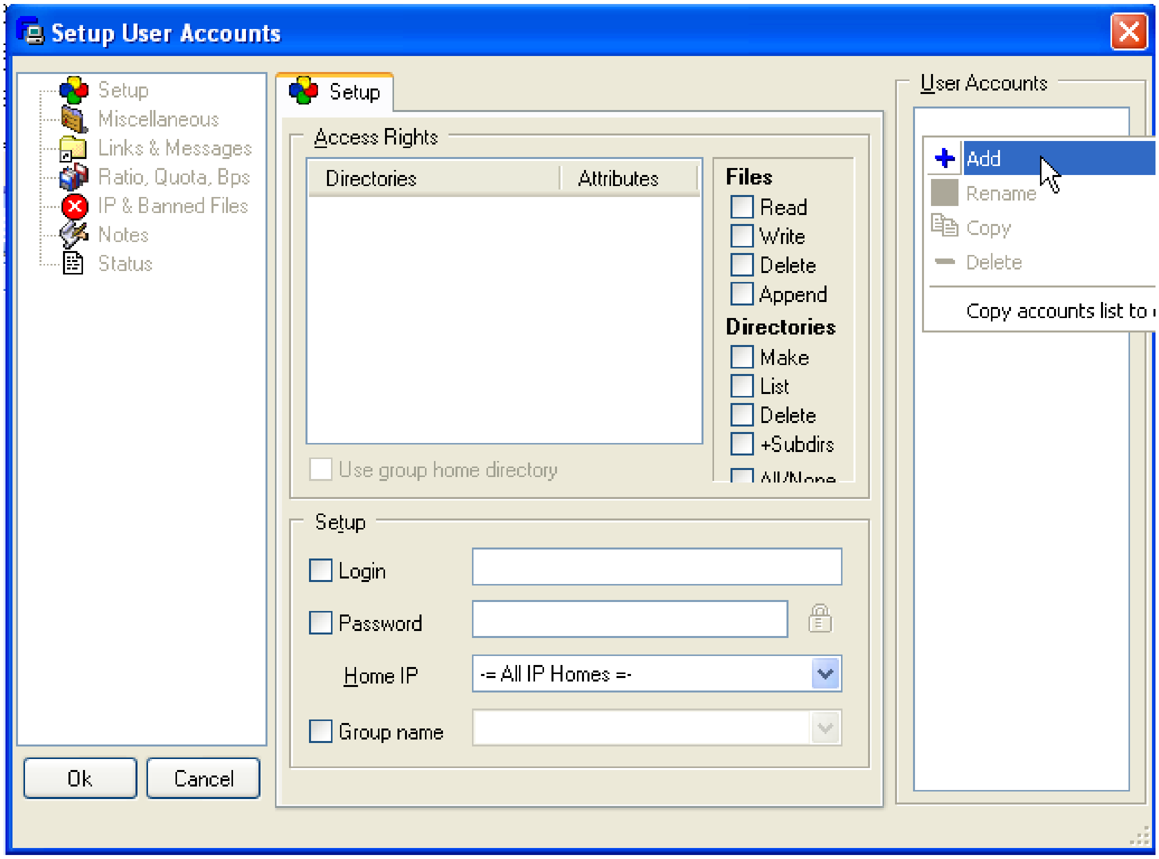
```
BulletPro                                   opyright (c) 1998-2004 DigitalCandle
Inc, All F
Using W
Running                                     20GHz 3192 Mhz with 1047532 KB
(509796                                     ervice Pack 2

C:\ -- Fixed Disk Drive - Free Space : 9369 MB
D:\ -- CDRom Disk Drive
E:\ -- Fixed Disk Drive - Free Space : 25024 MB
G:\ -- Fixed Disk Drive - Free Space : 68856 MB
H:\ -- Network Disk Drive - Free Space : 13133 MB
K:\ -- Network Disk Drive - Free Space : 5469 MB
Y:\ -- Network Disk Drive - Free Space : 13133 MB
Z:\ -- Network Disk Drive - Free Space : 5469 MB

1 user account, 0 group account, 0 free file(s)/dir(s)
```

The following Setup User Accounts screen will appear:



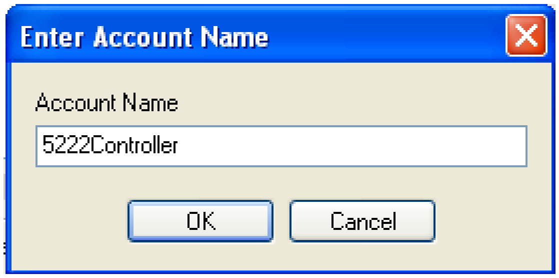
Right Click to add/modify accounts

To add an account, position the mouse over the User Accounts window and right click the mouse. A menu will appear; select Add:

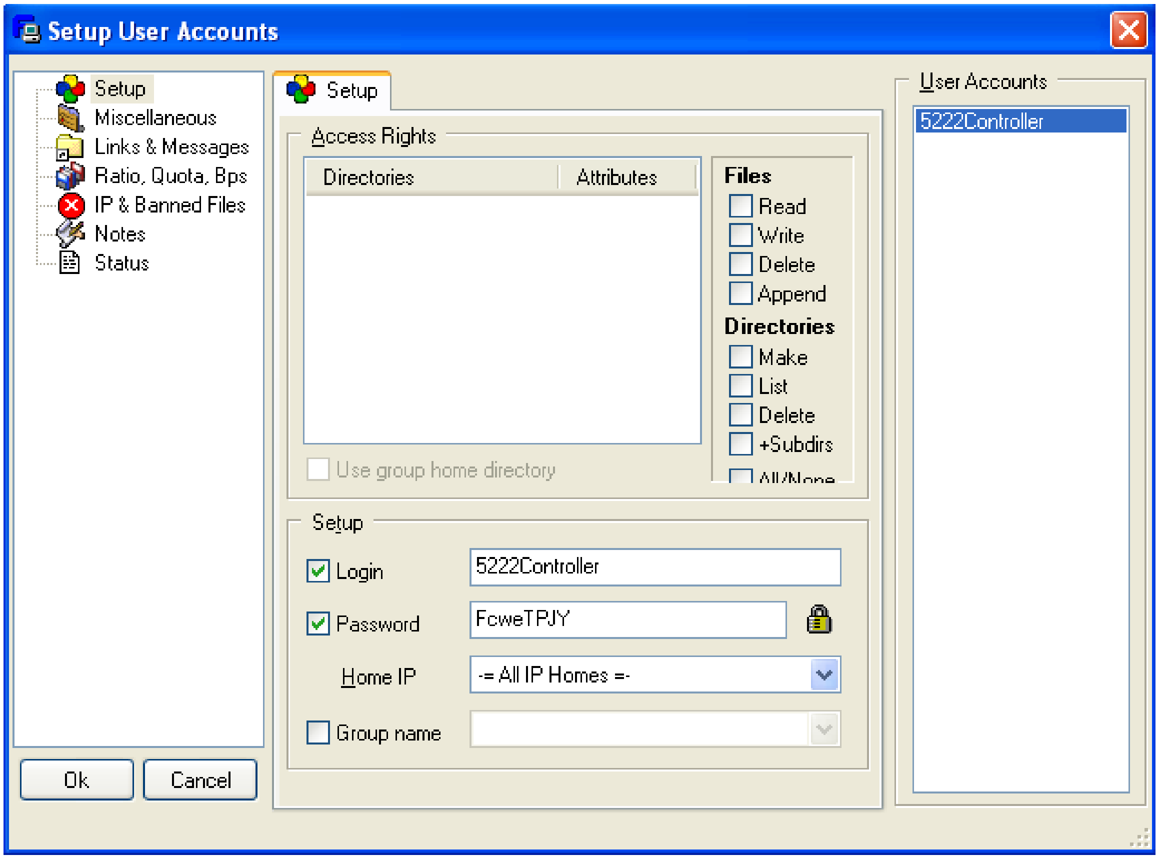


Home IP: -= All IP Homes =-

Enter the new account name, 5222Controller is shown in the example, click OK:



Account Name: 5222Controller



Setup

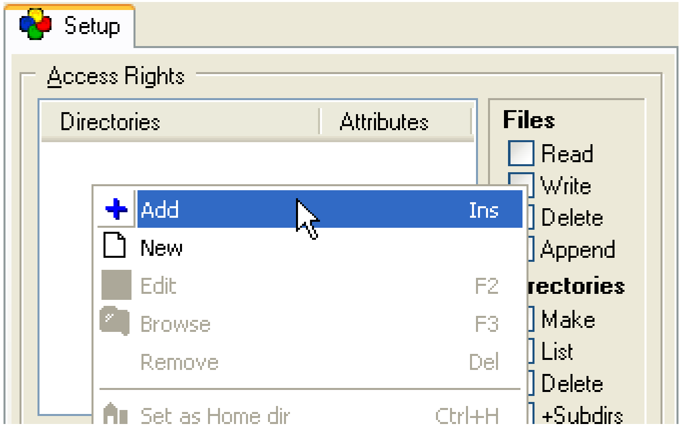
☑ Login: 5222Controller

☑ Password: FcweTPJY

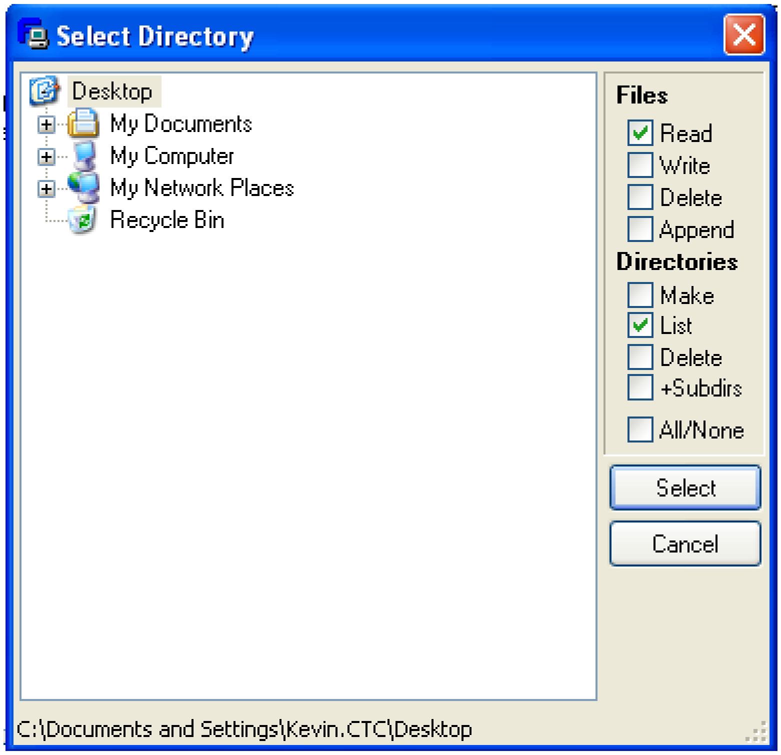
Home IP: -= All IP Homes =-

□ Group name:

The account has now been created. A default password of FcweTPJY is shown. This should be changed to anything desired. By default no directory or file access is granted, only the account created. In order to grant access a directory must be referenced and access privileges granted. To add a directory, right click the mouse in the Access Rights area of the screen and select Add:



A window will appear allowing you to select the desired directory and access permissions:



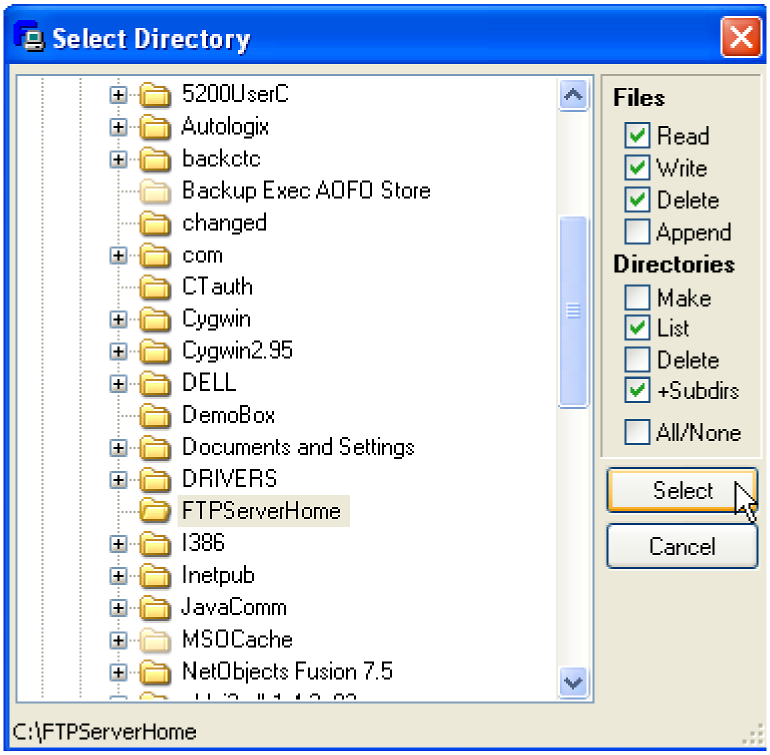
Files
☑ Read
☐ Write
☐ Delete
☐ Append

Directories
☐ Make
☑ List
☐ Delete
☐ +Subdirs

☐ All/None

C:\Documents and Settings\Kevin.CTC\Desktop

If subdirectories are to be allowed be sure to select the +Subdirs check box. Below shows a directory on the C: drive being added called `FTPServerHome`. File access will allow upload and download as well as access to subdirectories.

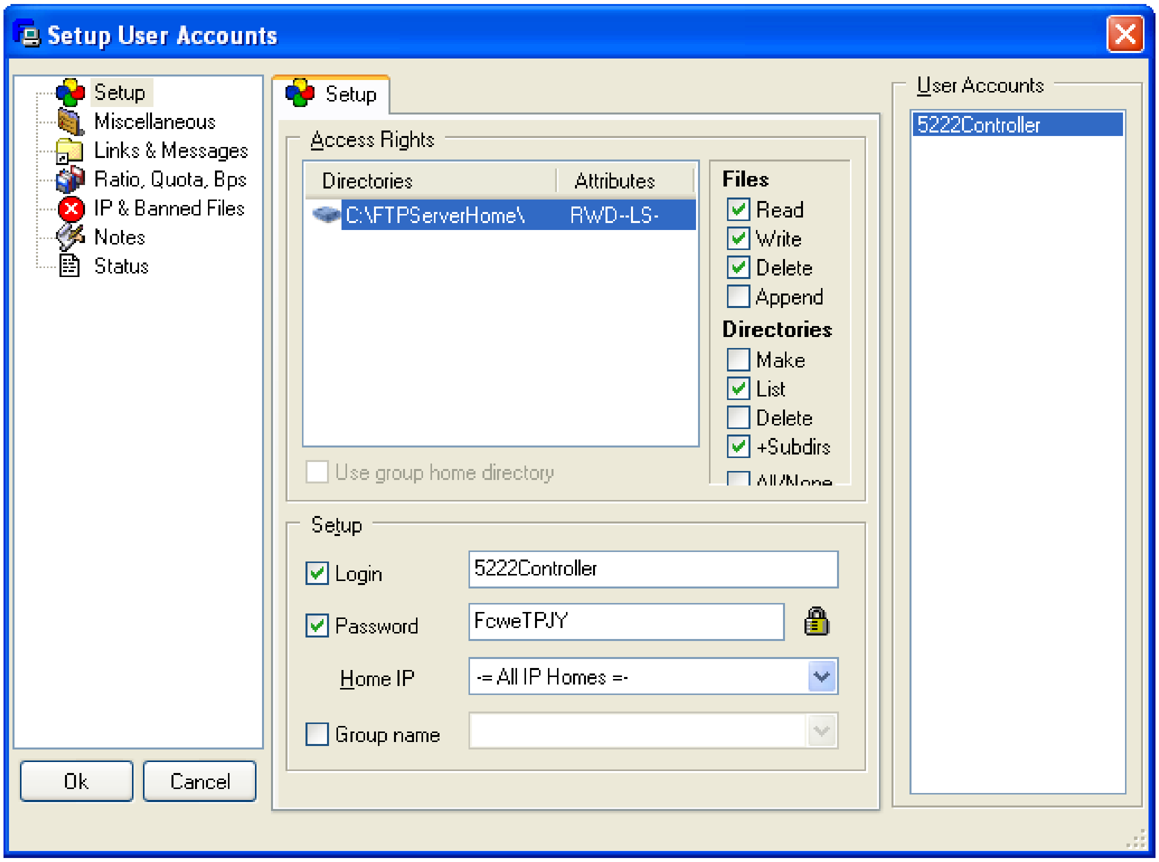


Files
☑ Read
☑ Write
☑ Delete
☐ Append

Directories
☐ Make
☑ List
☐ Delete
☑ +Subdirs

☐ All/None

C:\FTPServerHome



User Accounts: 5222Controller

Directories	Attributes
C:\FTPServerHome\	RWD--LS-

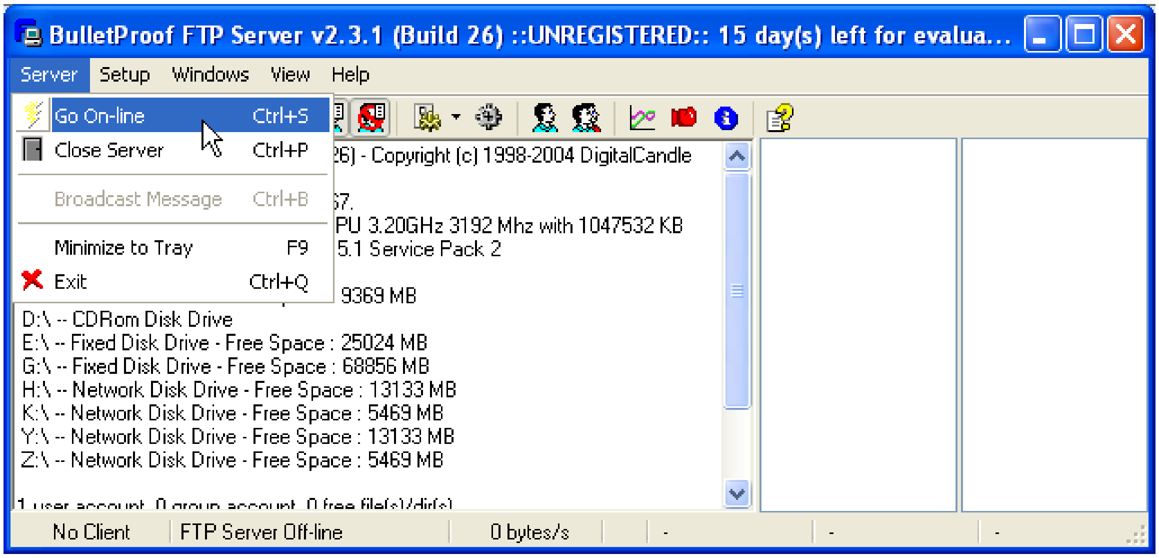
Files
☑ Read
☑ Write
☑ Delete
□ Append

Directories
□ Make
☑ List
□ Delete
☑ +Subdirs
□ All/None

□ Use group home directory

Setup
☑ Login: 5222Controller
☑ Password: FcweTPJY
Home IP: -= All IP Homes =-
□ Group name:

To activate the server you must select OK, then Server->Go On-line at the main menu:



```
26) - Copyright (c) 1998-2004 DigitalCandle

57.
PU 3.20GHz 3192 Mhz with 1047532 KB
5.1 Service Pack 2

9369 MB
D:\ -- CDRom Disk Drive
E:\ -- Fixed Disk Drive - Free Space : 25024 MB
G:\ -- Fixed Disk Drive - Free Space : 68856 MB
H:\ -- Network Disk Drive - Free Space : 13133 MB
K:\ -- Network Disk Drive - Free Space : 5469 MB
Y:\ -- Network Disk Drive - Free Space : 13133 MB
Z:\ -- Network Disk Drive - Free Space : 5469 MB

1 user account, 0 group account, 0 free file(s)/dir(s)
```

No Client FTP Server Off-line 0 bytes/s - - -

There are numerous other features available within the BulletProof software package. It is left to the user to read their documentation available on their web site.

http://www.bpftpserver.com/help/bpftpserver.com/manual_en/

Example: Access via the telnet or script command line to this account would be:

```
ftpconnect 12.40.53.52 5222Controller FcweTPJY
```

APPENDIX

B

[B] Network Port Usage

The Model 5300 uses a number of TCP and UDP data ports for communications. This section documents their usage.

Port Numbers

An IT professional can use the following port number list for configuration of corporate firewalls, VPN or NAT. Below are the common ports used. In many circumstances, especially when the Model 5300 is the client, ports are determined by the user.

Port	Function	Direction
21	FTP Server	Inbound, TCP
23	Telnet Administrative Interface	Inbound, TCP
501	Modbus TCP	Inbound, TCP
3000	CTC Binary Protocol	Inbound, UDP
6000	CTC Binary Protocol	Inbound, TCP
21896	Controller Discovery	Inbound and Outbound , UDP
40000 to 40640	Ports used for random binding, both UDP and TCP.	Inbound and Outbound
	BELOW FOR iPANEL, REF ONLY	
21891	Multicast [239.11.90.201] Used to discover **iPanel** devices and computers running **CT HMI Workstation**.	Outbound from the **Control** utility. Inbound to **iPanel** devices and computers running **CT HMI Workstation**.

21892	Multicast [239.11.90.201] Used to discover **iPanel** devices and computers running **CT HMI Workstation**.	Outbound from **iPanel** devices and computers running **CT HMI Workstation**. Inbound to the **Control** utility.
21893	Multicast [239.11.90.201] Used to communicate between **iPanels**, all **CT HMI** products (including Builder and Workstation) and **CTServer**. Used primarily for service discovery.	Inbound and Outbound.
9190	Used to communicate between the **Control** utility and an **iPanel** device or a computer running **CT HMI Workstation**.	Outbound from the **Control** utility. Inbound to **iPanel** devices and computers running **CT HMI Workstation**.
9191	Used to communicate between the **Remote Workstation** program and an **iPanel** device or a computer running **CT HMI Workstation**.	Outbound from the **Remote Workstation** program.
8194	Web server (HTTP) port for viewing logged historical data. This port is *only* used when developing a project using **CT HMI Builder**.	Inbound to **CT HMI Builder** from client browsers.
8195	Web server (HTTP) port for viewing logged historical data. This port is used by run-time instances of **CT HMI SA** (stand-alone, running on an iPanel) or **CT HMI WS** (workstation).	Inbound to run-time instances of **CT HMI**.
8192	Virtual services *class-server*, *registry and local-services* port used by CT HMI applications to communicate to CTServer instances.	Inbound to virtual services.
41001 through 41099	Used to communicate between **iPanels**, all **CT HMI** products (including Builder and Workstation) and **CTServer**.	Inbound to remote object servers in the listed products.