A Guide to CTC Serial Data Communications

A comprehensive guide to the use of computer communications capabilities in Series 2200/2400/2800 Controllers

Table of Contents

The Role of Data Communications in the Factory

The increasing levels of automation being used in factories, coupled with a greater degree of sophistication required in the functions of Production Control, Quality Assurance and Maintenance, have brought about the need for unprecedented levels of data communications within the factory. Some of the goals for these communications efforts include:

- Providing design and/or process data (i.e.; dimensions, times/temperatures, product types, etc.) from a central engineering data base to a specific machine which must implement that design.

- Providing production data, including batch quantities desired, to a specific machine and, perhaps, coordinating the efforts of that machine with others.

- Gathering production data, including cumulative production quantities and relative efficiencies, from a number of machines.

- Tracking quality levels on machines capable of making qualitative measurements, monitoring reject rates or quality trends to allow timely action to be taken.

- Monitoring machine performance (cycle times, temperatures, etc.) to detect imminent failures and/or inefficiencies and aid in preventative maintenance.

- Monitoring machines and/or processes to detect faults, jams, etc., and provide timely information to Maintenance to effect a repair and minimize downtime.

As you can see, the role of data communications is quite broad, and will only increase in importance in the future. CTC has addressed the need for effective data communications with an approach which provides a great amount of flexibility to the designer, while allowing the machine control program to be developed independently of any computer-based monitoring or control program.

This booklet describes the three CTC Protocols for data communications which are an integral part of each CTC Automation Controller:

The CTC ASCII Computer Protocol

The CTC ASCII Terminal Protocol

The CTC Binary Protocol

A selection guide and overview description of these protocols are provided in section 3, and detailed information, as well as applications examples, are provided in subsequent sections.

A Practical Communications Hierarchy

In many instances, a computer is simply connected to a single controller and used to perform some data processing, operator interface or reporting task beyond the normal capabilities of the controller. Often, however, a controller's communications capabilities are used to create an information network, with one computer connected to a number of controllers. The goal in such a situation is usually more aggressive, with the computer being used to gather production information, provide manufacturing data, coordinate a number of different operations, etc.

In the past, many configurations have been tried in establishing communications among automated machines. All extremes from fully independent, unconnected machine controllers to total centralized control by a mainframe computer have been explored by those attempting to find the optimum approach to handling data communications in the plant environment.

Prior to establishing such an information network, two fundamental issues should be explored to insure that the installation will meet the long-term needs of all those concerned:

1. What are the **primary *initial* goals** of the comunications network? Is a computer to be used to monitor one or more machines, gather production data, provide parametric information (dimensions, temperatures, etc.), coordinate operations of different machines?

2. What **ultimate goals** are planned for the network? Will the installation eventually lead to a plant-wide information network to communicate engineering data, production and materials requirements data and plant status information?

These needs often point to the use of a "work-cell" hierarchy, shown in the accompanying diagram.

The Organization of a Workcell

A workcell hierarchy typically consists of three levels of control:

1. At the **local machine level**, an **automation controller** is used to control all "real time" functions of each machine. This controller contains a program for operating the machine and, typically, is capable of completely independent operation, allowing the machine to be run even in the event of a computer or network malfunction.

2. Machines performing related functions are then grouped into a **"workcell"**; for example, an injection molding machine, robotic transfer arm, deflashing machine and a milling machine used for secondary machining might constitute a workcell. The automation controllers for all machines in the workcell are connected via a common data network (called a **"local area network"**) to a **"workcell controller"**, which might consist of a Personal Computer (perhaps an industrial version).

The function of the workcell controller is to coordinate the operations of the various machines, as well as to perform such additional functions as data gathering, fault monitoring and local storage and dissemination of design (parametric) data.

3. Although such workcells are often initially set up as independent "islands" (and may exist in that state for years), the ultimate step in

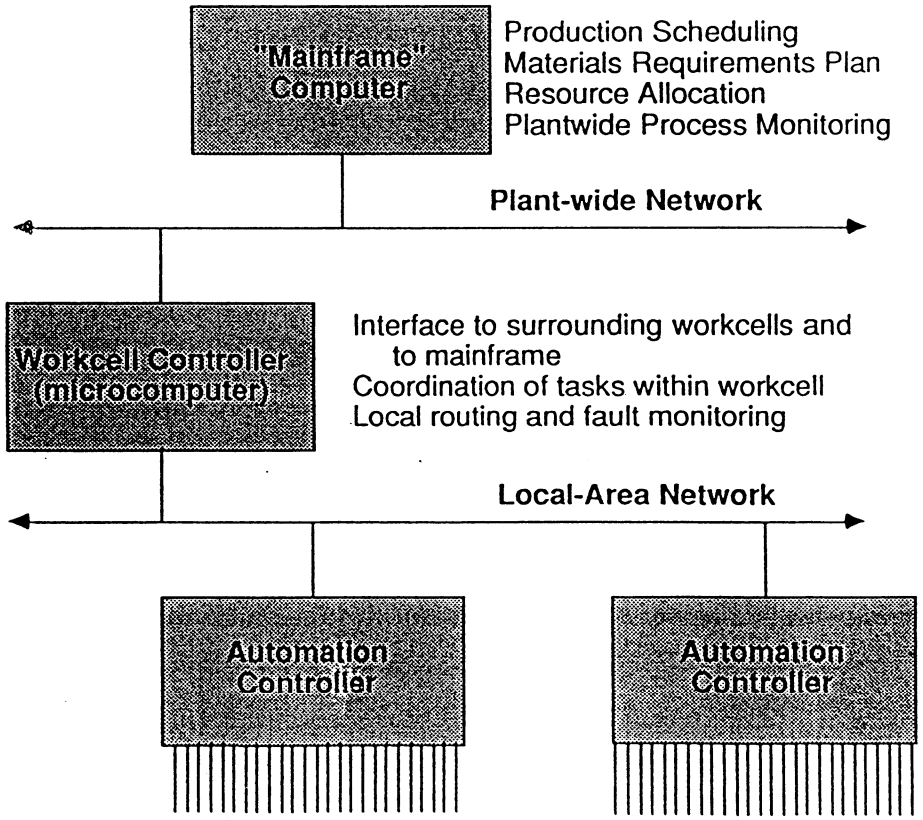
integrated communications is to link each workcell controller into a **plant-wide data network**. This is perhaps the most critical step; especially if the network is to play a major role in determining workflow and resource allocation on a fully-automated basis. Tasks which were formerly dependent on human judgement are suddenly at the mercy of a **central computer** system; this can result either in greatly increased efficiencies, or a nightmare of unanticipated problems, depending on the care exercised in the original planning of the system.

Of course, if the plant-wide network is to be used only for monitoring and information-gathering purposes, such dangers are not present. Often, a hybrid approach is advisable, where information is gathered by the network, analyzed by Production Control personnel, and used by them to command workflows via the network.

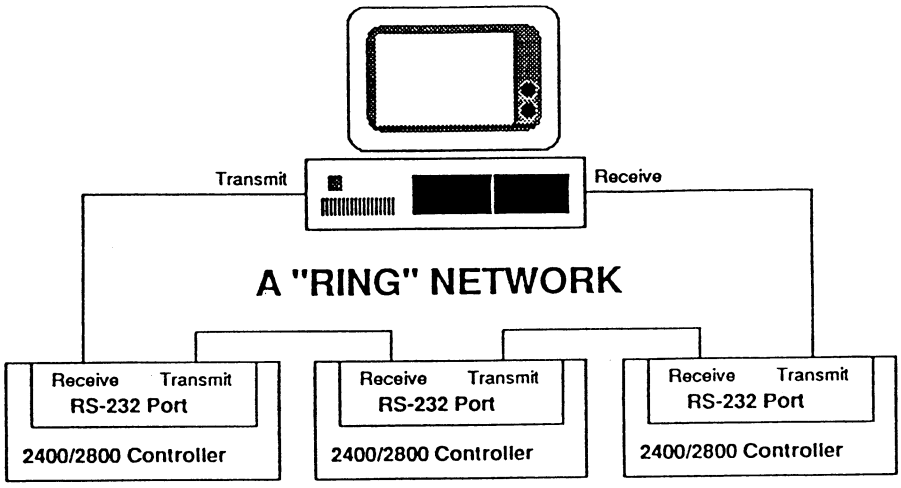
Implementing Workcells with Series 2400iE/2800iE Controllers

CTC controllers with the "E"-type (80186-based) CPU board are equipped with two integral serial communications channels. One of these channels (**channel B**) is fixed in an **RS-232** configuration and may be used for programming and for computer communications using a point-to-point (2-party) communications link. It is also possible to establish a **"ring network"** using this channel (refer to section 9).

A ring network allows information to be transmitted from system to system around a network connected as a "ring" until the information reaches the intended destination. The CTC Protocols support the use of the RS-232 ports in a ring network, and this represents an inexpen-



Control over individual actuators on machine tools, automated transfer stations, inspection stations, assembly stations, etc.



sive means of configuring a local area network.

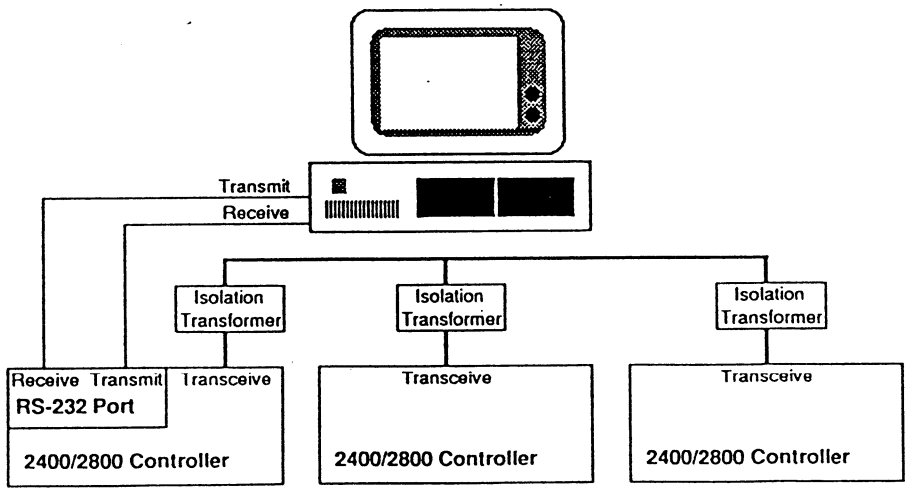
The other communications channel (**channel A**) is available as a second RS-232 port, identical in function to channel B, or, optionally, as a connection point for a **"multi-drop" local area network.** CTC will soon support an SDLC multi-drop network, based on technology developed by CTC for several custom networks, where several advantages over a ring network are present:

1. Transmission/response times are faster, due to the fact that repetitive receive/transmit cycles are not necessary to complete a transmission, as they are with a ring network.

2. The network is more hardy; if a given controller is powered-down, it will not affect the ability of other controllers to communicate.

3. Wiring is simpler, as is the addition of more controllers to the network as required.

Note that in the ring network configuration, the computer is connected directly into the network as an additional node. In the multi-drop configuration, however, the computer is typically not outfitted with the



A "Multi-Drop" Network

The RS-232 communications channel of the first controller may be used as a "gateway" for an external computer into the multi-drop network

necessary specialized hardware to connect directly to the network. The CTC Protocols will allow the "channel B" (RS-232) port of any of the controllers in the network to serve as a "gateway" into the network for a computer.

Thus, the computer may transmit commands to the RS-232 port of one controller, and that controller will retransmit each command over the multi-drop network, at which point the addressed controller will respond to the command. This response will also be reflected back to the computer by the controller acting as a gateway.

Once a local area network is established, using either methodology, the workcell controller (computer) may then be connected to a plant-wide network, if desired. This may be accomplished using MAP interface boards available for many microcomputers, or through an alternative interface (Ethernet, etc.) as appropriate. The computer is then programmed to act as a data accumulator/translator, as well as, perhaps, a local operator's control station, for the workcell.

Handling Information within the Hierarchy

The word "hierarchy" implies a division of the tasks to be performed among the various layers of a plant communications network, such that each successive layer is handling tasks at a higher level. One way of thinking about this is to imagine the Automation Controllers handling detailed tasks (i.e.; controlling a machine's actuators) on a second-to-second basis, while the Workcell Controller is performing supervisory functions on a minute-to-minute or hour-to-hour basis, while the Central Computer is coordinating plant operations on a day-to-day, week-to-week or even month-to-month basis.

With the increasing levels of power and capability which are available at the Automation Controller level, it is often practical (and usually advisable) to perform all real-time machine control functions at this level (unless the additional computational or storage capabilities of the Workcell Controller are required for these functions). This distribution of control down to the machine level allows the machine to be designed and to function on a stand-alone basis, resulting in greater simplicity and modularity.

With much of the machine control responsibility being carried by the Automation Controller, the role of the Workcell Controller can become relatively minor. The Workcell Controller may be used to coordinate operations of the various machines in the workcell (e.g.; telling the robot controller when a new workpiece is needed in the assembly machine, etc.), to monitor the various controllers in the workcell for detected fault conditions and to receive production data from the Central Computer and "translate" this data to the Automation Controllers. In addition, the Workcell Controller is often programmed to provide status information at the "local" level for Production Supervisors on the plant floor.

The plant's Central Computer will be mainly concerned with Production Control, product configuration and fault management matters. It is usually a serious mistake to have moment-by-moment manufacturing operations controlled directly from the Central Computer, due to the resultant absolute reliance on that system's functionality. By distributing control to a number of independent systems, the risk of downtime is typically also distributed (there are, of course, exceptions where a single, centralized system is indicated; but, if you put all of your eggs in one basket, *watch that basket!*).

Selection and Use of the CTC Protocols

The serial communications channels of the models 2400iE and 2800iE are supplied with a built-in protocol for the transfer of information into and out of the controller. This protocol functions in the background of the machine control program running in the controller, and no additional provision is required within the machine control program to accomplish serial communications.

The communications protocol is driven entirely from commands sent by an external computer (or other intelligent device) to the model 2400iE/2800iE controller. The controller responds to these commands by effecting a data transfer to or from the controller.

There are actually three protocols which may be used for communications with these controllers:

1. The CTC ASCII Computer Protocol

This protocol is most frequently used in instances where a computer program must interact with the operation of the controller, either providing parametric information for the controller's program to use, or requesting data (production quantities, machine status, etc.) from the controller. This protocol uses ASCII characters to carry information between the computer and controller, and terminates each message with a "carriage return" character (ASCII 13).

2. The CTC ASCII Terminal Protocol

Often, it is desirable to use a portable "dumb terminal" as a troubleshooting/diagnostic aid in setting up a new system. A battery-operated "lap-top" computer (e.g.; Radio Shack model 100, etc.) in terminal emulation mode is often used for this purpose. The CTC ASCII Terminal Protocol accomodates the use of a terminal by responding to messages with a "line feed" character (ASCII 10), avoiding the problem of characters piling on top of one another on the screen of the terminal. This protocol is otherwise virtually identical to the Computer Protocol above.

3. The CTC Binary Protocol

When data integrity, response time and processing time are major criteria, the CTC Binary Protocol supports the transfer of data packets in binary form, with checksumming and error reporting. This allows each transmission to be checked for data errors, and also eliminates the processor time required to perform a binary-to-ASCII conversion on the transmitted data. Recommended for more experienced programmers, this protocol is somewhat more complex to use.

This section will deal solely with the protocol (informational) aspects of communications with CTC controllers. For information relating to the physical connection of the external host to the controller's serial communications port, please refer to the controller's installation guide.

Communicating with the DSP™ Program

Although the communications protocol allows the direct reading of the controller's inputs, as well as the reading and forcing of the controller's outputs, a far more common usage of the serial communications capabilities is to transfer information to and from the controller's Numeric

Registers. This allows a computer to provide numeric parameters for use by the controller's program, or to monitor numeric parameters to which the controller has access.

Some of the applications for these capabilities include:

- Computer determination of motor position coordinates.

- Computer control of motor motion parameters (speed, accel/decel rates, servo filter parameters, etc.).

- Remote reading of production data (batch counts, good part/bad part counts, parametric deviation).

- Monitoring of process variables (temperature, pressure, position, level, etc.).

The controller must simply be programmed to store the required data (or to derive the incoming data) to or from one or more of its Numeric Registers. The computer then communicates, in effect, directly with these registers.

Data Table Transfer

Frequently, DSP™ programs for machine control are written to derive numeric parameters from a Data Table, stored in non-volatile memory along with the controller's program. This Data Table may represent information such as motor coord-inates or temperature or other process setpoints. The serial communications protocol allows these Data Tables to be transferred to or from a remote computer, allowing the computer to "configure" the machine according to immediate production requirements.

Initializing the Computer's Serial Port

Before any communications may take place, however, the serial port of the computer being used must be initialized to the serial data format used by the controller. This is typically accomplished by a statement in the language being used (BASIC, etc.), setting the parameters governing the operation of the port. Although the specific command required varies among different computers, the critical parameters required are given below:

Baud Rate: 9600 (may be set to an alternate baud rate, except on model 2200)
Parity: None
Character width: 8 bits
Number of Stop Bits: 1

For example, to initialize the first serial port on an IBM-PC computer to the above requirements, the following statement would be executed:

OPEN "COM1:9600,N,8,1,CS,DS" AS #1

Setting the Controller's Protocol

Series 2400/2800 Controllers are initialized into the CTC ASCII Terminal Protocol upon power-up. To change to the CTC ASCII Computer Protocol, a command is send to the controller's serial port establishing the new protocol:

P C <carriage return>

The "P" command sets the protocol, where "C" establishes the Computer Protocol and "T" establishes the Terminal Protocol. The carriage return which follows signals the end of the command. The controller responds to the above command with the message:

P C Ø <carriage return>

This acknowledges the change to the Computer Protocol. Note that the response is consistent with the Computer Protocol, in that is it terminated with a carriage return.

To return to the CTC ASCII Terminal Protocol, the following command is sent to the controller:

P T <carriage return>

The controller will then respond with the following acknowledgement:

<line feed> P T <carriage return> <line feed>

This response is consistent with the Terminal Protocol in that the command is immediately acknowledged with a line feed, and the response is terminated with both a carriage return and a line feed, creating a readable display on the terminal.

The CTC ASCII Computer Protocol

As mentioned earlier, serial communications is accomplished by using an external computer to send commands to the controller. These commands are in the form of simple "ASCII" messages (ASCII is a commonly-used form of coding for transmitting text information between computers). Most computer languages have provision for easily sending such ASCII messages to a serial communications port.

For example, to force a number into one of the controller's Numeric Registers (in this example, forcing "1200" into Register #10), the command would read "R10=1200". This command must be concluded with the code for a "carriage return" command (ASCII 13), signalling to the controller that the command is complete. The following "BASIC"-language statement would accomplish this transmission:

PRINT #1, "R10=1200"

This assumes that a serial communications channel on the computer had been previously "opened" and defined as output port #1 (computers and versions of BASIC vary as to how this is accomplished; refer to manufacturer's published data). Most versions of BASIC will automatically add the required carriage return at the end of the transmission.

When operating in the CTC ASCII Computer Protocol, the controller will respond with a "carriage return" command, acknowledging the receipt of the message. This should be received and tested by the computer, because if a transmission error occured, the controller will instead respond with an error message. This "test" may be accomplished with the following statements:

LINE INPUT #1,R$

IF R$<>"" THEN GOTO 100

The first statement will receive the controller's response (assuming the same serial port had been previously defined as input port #1), and assign the response to character string "R$". In most versions of BASIC, a response consisting of only a carriage return (with no characters preceding it) will be received as a "null string" (i.e.; an empty message). The second statement then tests the response; if the controller's response is not equal to a null string (<>""), then a transmission error has occured, and the BASIC language program will jump to line #100 to react to that fact.

It is important that the controller's response is "taken in" by the computer's program; otherwise, it will remain in the computer's communications buffer and effect the ability to receive future messages.

To meet the special requirements of certain languages and computer systems, the ASCII protocol may be modified to transmit a line feed automatically after the carriage return in the controller's response.

To select this option, the protocol selection command would read:

P C L <carriage return>

The controller will respond with:

P C L <carriage return> <line feed>

The CTC ASCII Terminal Protocol

Sometimes it is desirable to use a "dumb" terminal or a computer running a terminal emulation program to communicate with a controller. For example, a portable terminal may be used for diagnostic or debugging purposes, forcing outputs on or off, reading or forcing numeric registers, etc. (Of course, Quickstep™ or a model 2000A Programming Terminal may be used for the same purpose.)

The CTC ASCII Computer Protocol, however, is not ideally suited to this task. This protocol has been optimized for use in communicating with a running computer program; each response is terminated in a carriage return, signalling the completion of the message.

When using a terminal to directly view the response of the controller, the carriage return will return the terminal's cursor to the beginning of the same line; subsequent activity will just overwrite previous messages and responses.

The CTC ASCII Terminal Protocol solves this problem by responding to commands from a terminal (or computer) with an instantaneous "line feed", moving the terminal to the next line on its screen. The controller then transmits its response, if any, followed by a carriage return *and* a line feed. The resultant exchange is then recorded on the terminal's screen on successive lines.

The CTC ASCII Terminal Protocol is selected using the "P" (protocol) command:

P T <carriage return>

(refer to "Selection and Use of the CTC Protocols"). Of course, the terminal's transmission parameters must be set to agree with the controller's data format (i.e.; 9600 baud, no parity, 8 bit character width, 1 stop bit). Refer to the terminal's instructions to accomplish this; some terminals use DIP switches to establish these parameters, others are determined in software.

Aside from the use of line feeds, the Terminal Protocol is otherwise identical to the Computer Protocol. The section "Commands for the ASCII Protocols" illustrates the available commands and the responses provided in each of these two protocols.

Commands for the ASCII Protocols

The terminal or computer can access any of the following data within the controller: digital inputs and outputs, analog inputs and outputs, registers, counters, data table and flags. In addition, the controller can be commanded to START, STOP or RESET. In the following command descriptions, **<cr>** stands for carriage return (ASCII 13) and **<lf>** stands for line feed (ASCII 10). Digital input and output values use the number 0 for off and 1 for on. Flag values use the number 0 for clear and 1 for set. The responses are shown first for computer mode protocol and then for terminal mode protocol.

To examine a digital output:

command: **O** (letter "O", not zero) ***<output number>*** ***<cr>***

response: computer mode: ***<output value>*** ***<cr>***

terminal mode: ***<lf>*** ***<output value>*** ***<cr>*** ***<lf>***

To change a digital output:

command: **O** ***<output number>*** = ***<new value>*** ***<cr>***

response: computer mode: ***<cr>***

terminal mode: ***<lf>***

To examine a digital input:

command: **I** ***<input number>*** ***<cr>***

response: computer mode: ***<input value>*** ***<cr>***

terminal mode: ***<lf>*** ***<input value>*** ***<cr>*** ***<lf>***

To examine an analog output:

command: **A O** ***<output number>*** ***<cr>***

response: computer mode: ***<output value>*** ***<cr>***

terminal mode: ***<lf>*** ***<output value>*** ***<cr>*** ***<lf>***

To change an analog output:

command: **A O** ***<output number>*** = ***<new value>*** ***<cr>***

response: computer mode: ***<cr>***

terminal mode: ***<lf>***

To examine an analog input:

command: **A I** ***<input number>*** ***<cr>***

response: computer mode: ***<input value>*** ***<cr>***

terminal mode: ***<lf>*** ***<input value>*** ***<cr>*** ***<lf>***

To examine a counter or register:

command: **R** ***<counter/register number>*** ***<cr>***

response: computer mode: ***<counter/register value>*** ***<cr>***

terminal mode: ***<lf>*** ***<counter/register value>*** ***<cr>*** ***<lf>***

To change a counter or register:

command: **R** *<counter/register number>* = *<new value>* *<cr>*

response: computer mode: *<cr>*

terminal mode: *<lf>*

To examine a flag:

command: **F** *<flag number>* *<cr>*

response: computer mode: *<flag value>* *<cr>*

terminal mode: *<lf>* *<flag value>* *<cr>* *<lf>*

To change a flag:

command: **F** *<flag number>* = *<new value>* *<cr>*

response: computer mode: *<cr>*

terminal mode: *<lf>*

To examine a data table location:

command: **D** *<row number>* , *<column number>* *<cr>*

response: computer mode: *<table value>* *<cr>*

terminal mode: *<lf>* *<table value>* *<cr>* *<lf>*

To change a data table entry:

command: **D** *<row number>* , *<column number>* = *<new value>* *<cr>*

response: computer mode: *<cr>*

terminal mode: *<lf>*

To START the controller:

command: + *<cr>*

response: computer mode: *<cr>*

terminal mode: *<lf>*

To STOP the controller:

command: - *<cr>*

response: computer mode: *<cr>*

terminal mode: *<lf>*

To RESET the controller:

command: * *<cr>*

response: computer mode: *<cr>*

terminal mode: *<lf>*

To examine the input and output complement of the control rack:

command: **C** *<cr>*

response: computer mode:

I=<*i*> O=<*o*> A=<*a*> M=<*m*> C=<*c*> T=<*t*> D=<*d*> P=<*p*> <*cr*>

terminal mode:

<lf> I=<i> O=<o> A=<a> M=<m> C=<c> T=<t> D=<d> P=<p> <cr> <lf>

where: **<i>** is the number of digital inputs

<o> is the number of digital outputs

<a> is the number of analog inputs and the number of analog outputs

<m> is the number of stepping motors

<c> is the number of 2811 communication boards

<t> is the number of thumbwheels

<d> is the number of displays

<p> is the number of prototype boards

Error Messages

When the controller receives an illegal command, it sends back an error message. The error message consists of a character to indicate the type of error, followed by a BELL character (ASCII 7). The types of errors are listed below. As in the command listing, the response is shown first for computer protocol, and then for terminal protocol:

Number too small:

If an input, output, register, or flag number is specified as zero, then the controller sends the following error message:

response: computer mode: **< <bell> <cr>**

terminal mode: **<lf> < <bell> <cr> <lf>**

Number too large:

If an input, output, register, or flag number is too large (output number greater than the number of outputs, flag number greater than 32, etc.) then the controller sends the following error message:

response: computer mode: **> <bell> <cr>**

terminal mode: **<lf> > <bell> <cr> <lf>**

Protocol error:

If a "P" command (protocol) is not in the correct format, then the controller sends the following error message:

response: computer mode: **P <bell> <cr>**

terminal mode: **<lf> P <bell> <cr> <lf>**

Syntax error:

If the controller cannot make any sense of the command, then it sends the following error message:

response: computer mode: **? <bell> <cr>**

terminal mode: **<lf> ? <bell> <cr> <lf>**

The CTC Binary Protocol

Although the standard CTC ASCII Protocol is most frequently used when communicating between a computer and a CTC controller, there is an additional binary communications protocol which may be used in more demanding applications.

This CTC Binary Protocol, although somewhat more difficult to use, can significantly reduce the time required to transfer large blocks of data between a computer and the controller. The reason for this efficiency is twofold:

1. 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 must be transmitted.

2. Perhaps more importantly, the data received by the controller does not first have to be converted from ASCII to binary before it may be used, resulting in much shorter execution times. In addition, there may be significant time savings in the execution of the computer program, as data need not be converted to ASCII prior to transmission (this time savings may vary among different languages).

Selecting the CTC Binary Protocol

As with the standard CTC ASCII Protocol, communications in the CTC Binary Protocol are initiated from a host system (i.e.; computer or other intelligent device) outside the controller, by sending a command to one of the controller's serial inputs. To select the CTC Binary Protocol, the first character of such a command must be binary 1 (Ø1H). The rest of the command will then be interpreted by the controller according to the CTC Binary Protocol.

General Format of the CTC Binary Protocol

Communications from a host system to the controller using this protocol will follow the general format shown below:

<Ø1H>	Signifies CTC Binary Protocol
<Ø2H> to <3FH>	Specifies packet length to follow (defined as n data bytes + 2)
<data (n bytes)>	Consists of function code(s) plus relevant data
<checksum>	Complement of the modulo-256 sum of data bytes
<FFH>	Last byte of packet must be ØFFH

The following example will help to clarify the usage of the above format. To set Flag #4 in a controller, the following packet may be sent:

Ø1H,Ø5H,13H,Ø3H,FFH,EAH,FFH

In the above packet, the first byte (Ø1H) identifies the packet as using the CTC Binary Protocol. The second byte, Ø5H, represents the length of the packet to follow, expressed in bytes (note that this length includes the checksum byte and the termination byte).

The third byte, 13H, is the code for a "Change Flag" command, as described below (see "Command for the Binary Protocol"). It is followed by the number of the flag to be affected, where flags 1 through 32 are

represented by ØØH through 1FH. Therefore, Flag #4 is represented by Ø3H. The data for this flag is carried by the fifth byte; flags may possess one of two possible states, "SET", represented by ØFFH, or "CLEAR", represented by ØØH.

The sixth byte of the packet is a checksum which, when added to the modulo-256 sum of the data packet bytes will equal ØFFH. In this instance, the data packet consists of the third byte through the fifth byte, and their sum is 15H; therefore the checksum will equal ØEAH. (Note that the checksum may be easily calcuated by adding the data packet bytes and complementing the resultant sum.)

The last byte of the packet must always be ØFFH. The controller, upon receiving the packet, will count out the number of bytes specified by the "packet length" byte and, if the last byte is not ØFFH, will return an error message.

Responses from the Controller

Communications back from the controller follow the same general format shown above, with one exception: The controller will not transmit a leading (Ø1H) byte, because the host is assumed to know that the original message to the controller was transmitted in the CTC Binary Protocol.

If the command to the controller does not require a data response (i.e.; register information is not being requested, etc.), the controller will respond with an acknowledgement message in the form shown below:

<Ø3H>	Packet length
<64H>	"Acknowledge" code (=decimal 100)
<9BH>	Checksum of above byte
<FFH>	Last byte of packet

If, however, the original packet is not received properly by the controller (for example, the checksum does not calculate correctly, or the last byte of the packet is not ØFFH), the controller will transmit a "NOT ACKNOWLEDGE" code, 65H, in place of 64H in the message above. The checksum will therefore be 9AH in the controller's response.

Other error codes are possible if the format of the message is correct, but the controller is otherwise unable to execute the command. This might occur, for example, if a register number is specified which is out of the range of existing registers within the controller. These error messages are explained in the "Commands" section, below.

Data Transmission Specifications

As with the CTC ASCII Protocol, the following specifications must be observed for the data transmissions to the controller:

Baud Rate: 9600 (may be changed to an alternative baud rate, except in model 2200)
Data byte length: 8 bits
Number of Stop Bits: 1
Parity: None

Most languages/systems have provision for setting these parameters.

Commands for the Binary Protocol

This section documents the commands available via the CTC binary protocol as of this printing. Note that, due to controller resource limitations, some of these commands are not supported by all CTC Automation Controllers (as noted below, and within the individual command specifications), and that older versions of controller firmware may not support all of the commands listed. Contact CTC if you have questions regarding command availability, or if you have difficulty implementing any specific command.

Compatibility information:
Those commands marked "All ex. 2200" above are not available in Series 2200 Controllers at present. The command marked "iEA only" above is only available in model 2400iEA and 2800iEA ("Expanded Architecture") Controllers.

Command 9: Read a Numeric Register

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø5H	Packet Length
Ø9H	"Read Register" Function Code
LSB **MSB**	Number of register to be read, ØØØ1H to ØFFFFH, specified with least significant byte first
checksum	Complement of modulo-256 sum of previous 3 bytes
FFH	Last byte of packet

Response from Controller:

Ø7H	Packet Length
ØAH	"Register Contents" Function Code
LSB **3SB** **2SB** **MSB**	Four-byte representation of register data, expressed in 2's complement binary, with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 5 bytes
FFH	Last byte of packet

Note convention for expressing register number:

ØØØ1H through ØFFFFH correspond to Registers #1 through #65535, therefore, Register #10 is expressed as ØØØAH, and so on.

Some of the registers in this range perform special functions, others do not exist on certain models and revision levels; consult your programming manual for specific information regarding register functions.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø5H	Packet Length
4DH	"Register Group (16) Request" Function Code
LSB	First register to be read (ØØØØH - Ø3D9H allowed)
MSB	
checksum	Complement of modulo-256 sum of previous 3 bytes
FFH	Last byte of packet

Response from Controller:

45H	Packet Length
4EH	"Register Group Values" Function Code
LSB	First register to follow (ØØØØH - Ø3D9H allowed)
MSB	
LSB	Value of first register in group (*e.g.*, reg#1)
3SB	
2SB	
MSB	
LSB	Value of second register in group (*e.g.*, reg#2)
3SB	
2SB	
MSB	
	(...*etc.* for 16 registers total...)
checksum	Complement of modulo-256 sum of previous 67 bytes
FFH	Last byte of packet

Request 50 Register Values

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
4BH	"Register Group Request" Function Code
bank	Bank of 50 registers to be read (ØØH - 13H allowed)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

CCH	Packet Length
4CH	"Register Group Values" Function Code
bank	Bank of 50 registers to follow (ØØH - 13H allowed)
LSB	Value of first register in group (*e.g.*, reg#1)
3SB	
2SB	
MSB	
LSB	Value of second register in group (*e.g.*, reg#2)
3SB	
2SB	
MSB	
	(...*etc.* for 50 registers total...)
checksum	Complement of modulo-256 sum of previous 202 bytes
FFH	Last byte of packet

Important Note: Model 2800iEA controllers allow access to all general purpose registers with this packet. For model 2800iE controllers, however, only banks 0 through 9 may be requested — for access to non-volatile registers (banks 10 through 19) with these controllers, use command 77 (4DH).

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø9H	Packet Length
ØBH	"Change Register" Function Code
LSB **MSB**	Number of register to be set, ØØØ1H to ØFFFFH, specified with least significant byte first
LSB **3SB** **2SB** **MSB**	Four-byte representation of register data, expressed in 2's complement binary, with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 7 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note convention for expressing register number:

ØØØ1H through ØFFFFH correspond to Registers #1 through #65535, therefore, Register #10 is expressed as ØØØAH, and so on.

Some of the registers in this range perform special functions, others do not exist on certain models and revision levels; consult your programming manual for specific information regarding register functions.

Command 17: Read a Flag

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
11H	"Read Flag" Function Code
flag number	Number of flag to be read, ØØH to 1FH
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø4H	Packet Length
12H	"Flag Contents" Function Code
ØØH or non-zero	Flag status, equal to ØØH if flag is clear, least significant bit = 1 indicates flag is set, other results indeterminant
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Note convention for expressing flag number:

ØØH through 1FH correspond to Flags #1 through #32, therefore, Flag #1 is expressed as ØØH, and so on.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø5H	Packet Length
13H	"Change Flag" Function Code
flag number	Number of flag to be changed, ØØH to 1FH
ØØH or FFH	Data for specified flag, must be ØØH to "CLEAR" flag, FFH to "SET" flag; other values are indeterminant
checksum	Complement of modulo-256 sum of previous 3 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
checksum	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note convention for expressing flag number:

ØØH through 1FH correspond to Flags #1 through #32, therefore, Flag #1 is expressed as ØØH, and so on.

Command 15: Read a Bank of 8 Inputs

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
ØFH	"Read Inputs" Function Code
input bank	Number of input bank, ØØH to 7FH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø4H	Packet Length
1ØH	"Input Data" Function Code
ØØH to FFH	Data for eight inputs, with the lowest input number represented by the least significant bit. A "1" represents an "on" (grounded) input
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Note convention for expressing input bank number:

Input bank ØØH is a representation of the first eight inputs in the controller (designated inputs #1 through #8 in the DSP™ program). Input bank Ø1H represents inputs #9 through #16.

Request 128 Input Values

Send to Controller:

ØlH	Signifies CTC Binary Protocol
Ø4H	Packet Length
4FH	"Input (128) Request" Function Code
bank	Input bank to be read (ØØH - Ø7H allowed)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

14H	Packet Length
50H	"Input Values" Function Code
bank	Input bank to follow (ØØH - Ø7H)
inps1-8	Data for first 8 inputs in bank, with the lowest number input represented by the least significant bit. A "1" represents an "on" (grounded) input.
inps9-16	
	(...*etc.* for 128 inputs total...)
checksum	Complement of modulo-256 sum of previous 18 bytes
FFH	Last byte of packet

Note: Non-existent inputs within bank will be reported as "off" (*i.e.*, value = 0).

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
15H	"Read Outputs" Function Code
output bank	Number of output bank to be read, ØØH to 7FH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø4H	Packet Length
16H	"Output Status" Function Code
ØØH to FFH	An eight-bit representation of the output states of the selected bank, with the least significant bit indicating the status of the lowest-numbered output ("1" = ON), etc.
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Note convention for expressing output bank number:

Output bank ØØH is a representation of the first eight outputs in the controller (designated outputs #1 through #8 by the DSP™ program). Output bank Ø1H represents outputs #9 through #16, etc.

Send to Controller:

ØlH	Signifies CTC Binary Protocol
Ø4H	Packet Length
51H	"Output (128) Request" Function Code
bank	Output bank to be read (ØØH - Ø7H allowed)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

14H	Packet Length
52H	"Output Values" Function Code
bank	Output bank to follow (ØØH - Ø7H)
outs1-8	Data for first 8 outputs in bank, with the lowest number output represented by the least significant bit. A "1" represents an "on" output.
outs9-16	
	(...*etc.* for 128 outputs total...)
checksum	Complement of modulo-256 sum of previous 18 bytes
FFH	Last byte of packet

Note: Non-existent outputs within bank will be reported as "off" (*i.e.*, value = 0).

Command 25: Selectively Modify first 128 Outputs

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
23H	Packet Length
19H	"Modify Outputs" Function Code
off-mask-Ø through off-mask-15	A series of 16 eight-bit masks used to selectively turn OFF any or all of the controller's 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" will turn the associated output OFF; a mask value of "1" will leave that output unaffected by this mask (it may still be affected by the "on-masks")
on-mask-Ø through on-mask-15	A series of 16 eight-bit masks used to selectively turn ON any or all of the controller's 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" will turn the associated output ON; a mask value of "Ø" will leave that output unaffected by this mask
checksum	Complement of modulo-256 sum of previous 33 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note:

Separate off-masks and on-masks are used in the above instruction to allow selected outputs to be affected, while leaving other outputs undisturbed (i.e.; in their previous state).

As an example of their use, an "off-mask-Ø" of Ø6H (ØØØØ Ø11Ø binary) would turn OFF outputs #1, and #4 through #8. Outputs #2 and #3 would remain in their previous state.

A subsequent "on-mask-Ø" of CØH (11ØØ ØØØØ binary) would turn ON outputs #7 and #8.

Command 40: Selectively Modify a Group of 128 Outputs

SUPPORTED BY -iEA VERSIONS ONLY!!

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
24H	Packet Length
28H	"Modify Outputs" Function Code
bank	Bank of 128 outputs to be modified (ØØH to Ø7H), where ØØH represents outputs 1 to 128, Ø1H represents outputs 129 to 256, etc.
off-mask-Ø through off-mask-15	A series of 16 eight-bit masks used to selectively turn OFF any or all of the bank's outputs. The masks are applied to successive groups of 8 outputs, with the least significant bit of the mask being applied to the lowest-numbered output in the group. A mask value of "Ø" will turn the associated output OFF; a mask value of "1" will leave that output unaffected by this mask (it may still be affected by the "on-masks")
on-mask-Ø through on-mask-15	A series of 16 eight-bit masks used to selectively turn ON any or all of the group's outputs. The masks are applied to successive groups of 8 outputs, with the least significant bit of the mask being applied to the lowest-numbered output in the group. A mask value of "1" will turn the associated output ON; a mask value of "Ø" will leave that output unaffected by this mask
checksum	Complement of modulo-256 sum of previous 34 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note:
Separate off-masks and on-masks are used in the above instruction to allow selected outputs to be affected, while leaving other outputs undisturbed (i.e.; in their previous state).

As an example of their use, an "off-mask-Ø" of Ø6H (ØØØØ Ø11Ø binary) would turn OFF outputs #1, and #4 through #8. Outputs #2 and #3 would remain in their previous state. A subsequent "on-mask-Ø" of CØH (11ØØ ØØØØ binary) would turn ON outputs #7 and #8.

Command 29: Read an Analog Input

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
1DH	"Read Analog Input" Function Code
analog input	Number of analog input to be read, ØØH to 3FH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø5H	Packet Length
1EH	"Analog Input Value" Function Code
LSB **MSB**	Two-byte representation of analog value, expressed as a number in the range 0 - 10,000 decimal (ØØØØH - 271ØH), with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 3 bytes
FFH	Last byte of packet

Note convention for expressing analog input number:

ØØH through 3FH correspond to Analog Inputs #1 through #64, therefore, Analog Input #1 is expressed as ØØH, and so on.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
1FH	"Read Analog Output" Function Code
analog output	Number of analog output to be read, ØØH to 3FH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø5H	Packet Length
20H	"Analog Output Value" Function Code
LSB **MSB**	Two-byte representation of analog value, expressed as a number in the range 0 - 10,000 decimal (ØØØØH - 271ØH), with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 3 bytes
FFH	Last byte of packet

Note convention for expressing analog output number:

ØØH through 3FH correspond to Analog Outputs #1 through #64, therefore, Analog Output #1 is expressed as ØØH, and so on.

IMPORTANT: This command is not currently supported by the model 2200.

Command 33: Change an Analog Output

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø6H	Packet Length
21H	"Change Analog Output" Function Code
analog output	Number of analog output to be changed, ØØH to 3FH (see note)
LSB **MSB**	Two-byte representation of analog value, expressed as a number in the range 0 - 10,000 decimal (ØØØØH - 271ØH), with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 4 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note convention for expressing analog output number:

ØØH through 3FH correspond to Analog Outputs #1 through #64, therefore, Analog Output #1 is expressed as ØØH, and so on.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
17H	"Read Servo Position" Function Code
servo number	Number of servo axis to be read, ØØH to ØFH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø7H	Packet Length
18H	"Servo Position" Function Code
LSB **3SB** **2SB** **MSB**	Four-byte representation of servo position, expressed in 2's complement binary, with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 5 bytes
FFH	Last byte of packet

Note convention for expressing servo axis number:

ØØH through ØFH correspond to Servos #1 through #16, therefore, Servo #1 is expressed as ØØH, and so on.

IMPORTANT: This command is not currently supported by the model 2200.

Command 47: Read Servo Error

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
2FH	"Read Servo Error" Function Code
servo number	Number of servo axis to be read, ØØH to ØFH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø7H	Packet Length
3ØH	"Servo Error" Function Code
LSB **3SB** **2SB** **MSB**	Four-byte representation of servo error, expressed in 2's complement binary, with the least significant byte transmitted first
checksum	Complement of modulo-256 sum of previous 5 bytes
FFH	Last byte of packet

Note convention for expressing servo axis number:

ØØH through ØFH correspond to Servos #1 through #16, therefore, Servo #1 is expressed as ØØH, and so on.

IMPORTANT: This command is not currently supported by the model 2200.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
1BH	"Read Servo Inputs" Function Code
servo number	Number of servo axis to be read, ØØH to ØFH (see note)
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø4H	Packet Length
1CH	"Servo Input Status" Function Code
status	A one-byte representation of the control inputs for the referenced servo axis; the bit representations are shown below
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Note convention for expressing servo axis number:

ØØH through ØFH correspond to Servos #1 through #16, therefore, Servo #1 is expressed as ØØH, and so on.

The servo input "status" byte represents the following information (bit Ø is lsb):

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

The associated bit is a "Ø" if an input is active (grounded).

IMPORTANT: This command is not currently supported by the model 2200.

Command 49: Read Data Table Dimensions

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø3H	Packet Length
31H	"Read Data Table Dimensions" Function Code
CEH	Checksum of above byte
FFH	Last byte of packet

Response from Controller:

Ø6H	Packet Length
32H	"Data Table Dimensions" Function Code
LSB **MSB**	Number of Data Table rows in current program
cols	Number of Data Table columns (ØØH - 2ØH)
checksum	Complement of modulo-256 sum of previous 4 bytes
FFH	Last byte of packet

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø6H	Packet Length
33H	"Change Data Table Dimensions" Function Code
LSB **MSB**	Desired number of Data Table rows
columns	Desired number of Data Table columns
checksum	Complement of modulo-256 sum of previous 4 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note: an error code will be returned by the controller in the following circumstances:

1. If the requested Data Table size is too large for the controller.

2. If the requested Data Table size will not fit in memory, in combination with the existing DSP™ program.

3. If a Data Table column count greater than 32 is requested.

IMPORTANT: This command is not currently supported by the model 2200.

Command 53: Read a Data Table Location

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø6H	Packet Length
35H	"Read a Data Table Location" Function Code
LSB **MSB**	Data Table element desired – row number
columns	Data Table element desired – column number
checksum	Complement of modulo-256 sum of previous 4 bytes
FFH	Last byte of packet

Response from Controller:

Ø5H	Packet Length
36H	"Data Table Data" Function Code
LSB **MSB**	Data from specified location, expressed as a positive integer, in the range 0 to 65,535 (decimal)
checksum	Complement of modulo-256 sum of previous 3 bytes
FFH	Last byte of packet

Note: an error code will be returned by the controller if a non-existent Data Table location is specified.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø8H	Packet Length
37H	"Change Data Table Location" Function Code
LSB **MSB**	Target Data Table location – row number
column	Target Data Table location – column number
LSB **MSB**	New value for specified Data Table location, expressed as a positive integer, range 0 to 65,535.
checksum	Complement of modulo-256 sum of previous 6 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note: an error code will be returned by the controller if the specified Data Table location does not exist.

Command 57: Read a Data Table Row

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø7H	Packet Length
39H	"Read a Data Table Row" Function Code
LSB **MSB**	Data Table row desired
first col	Data Table column at which to start reading
quantity	Number of columns to read ('n'); <= 27 columns
checksum	Complement of modulo-256 sum of previous 5 bytes
FFH	Last byte of packet

Response from Controller:

length	Packet Length = (n*2) + 4, where n = number of columns read
3AH	"Data Table Row Data" Function Code
quant	Number of columns read ('n'); <= 27 columns
	For each of 'n' locations:
LSB **MSB**	Data from specified location, expressed as a positive integer, in the range 0 to 65,535 (decimal) ***End of location data.***
checksum	Complement of modulo-256 sum of previous (n*2)+1 bytes
FFH	Last byte of packet

Note: an error code will be returned by the controller if a non-existent Data Table row is specified.

If the quantity of Data Table columns specified extends beyond the actual number of columns, the response will contain only existent data (i.e.; the response will be shorter than expected).

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
length	Packet Length = (n*2)+7, where n = number of columns to be changed
3AH	"Change a Data Table Row" Function Code
LSB **MSB**	Data Table row to be changed
first col	Data Table column at which to start changing
quantity	Number of columns to change ('n'); <= 27 columns
	For each of 'n' locations:
LSB **MSB**	Data for specified location, expressed as a positive integer, in the range 0 to 65,535 (decimal) ***End of location data.***
checksum	Complement of modulo-256 sum of previous (n*2)+5 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Note: an error code will be returned by the controller if a non-existent Data Table row is specified or if the quantity of Data Table columns specified extends beyond the actual number of columns.

Command 61: Read Controller Status Byte

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø3H	Packet Length
3DH	"Read Status Byte" Function Code
C2H	Checksum of above byte
FFH	Last byte of packet

Response from Controller:

Ø4H	Packet Length
3EH	"Status Byte" Function Code
status	Status byte, where: bit Ø = 'Ø' if running, '1' if stopped bit 1 = 'Ø' if normal mode, '1' if programming mode bit 2 = 'Ø' if status O.K., '1' if Software Fault bit 3 = 'Ø' if mid-program, '1' if fresh reset Note: bit Ø is least significant bit, bits 4 through 7 are undefined at present.
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

IMPORTANT: This command is not currently supported by the model 2200.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
3FH	"Change Controller Status" Function Code
status	New controller status, where: bit Ø = 'Ø' to start controller, '1' to stop controller bit 3 = '1' to reset controller, otherwise 'Ø' Note: bit Ø is least significant bit, and will always either start or stop the controller. All unspecified bits should be set to 'Ø'.
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

IMPORTANT: This command is not currently supported by the model 2200.

Command 65: Read System Configuration

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø3H	Packet Length
41H	"Read System Configuration" Function Code
BEH	Checksum of above byte
FFH	Last byte of packet

Response from Controller:

Ø4H	Packet Length
42H	"System Configuration" Function Code
config	System Configuration byte, where: bit Ø = '1' if using input #1 for START function bit 1 = '1' if using input #2 for STOP function bit 2 = '1' if using input #3 for RESET function bit 3 = '1' if using input #4 for STEP function Note: bit Ø is least significant bit, bits 4 through 7 are undefined at present.
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

IMPORTANT: This command is not currently supported by the model 2200.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
43H	"Change System Configuration" Function Code
config	New system configuration, where: bit Ø = '1' if using input #1 for START function bit 1 = '1' if using input #2 for STOP function bit 2 = '1' if using input #3 for RESET function bit 3 = '1' if using input #4 for STEP function Note: bit Ø is least significant bit, bits 4 through 7 are undefined at present.
checksum	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

Ø3H	Packet Length
64H	"Acknowledge" code (decimal 100)
9BH	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

IMPORTANT: This command is not currently supported by the model 2200.

Command 13: Read Counts of Inputs, Outputs, etc.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø3H	Packet Length
ØDH	"I/O Count Request" Function Code
F2H	Checksum of above byte
FFH	Last byte of packet

Response from Controller:

ØCH	Packet Length
ØEH	"I/O Count" Function Code
flags	Number of flags in controller (typically 2ØH)
inputs LSB	Number of inputs in controller (LSB: ØØH to F8H,
inputs MSB	MSB: ØØH to Ø4H)
outputs LSB	Number of outputs in controller (LSB: ØØH to F8H,
outputs MSB	MSB: ØØH to Ø4H)
stepping mtrs	Number of stepping motor axes in controller (ØØH to 1ØH)
servos	Number of servo axes in controller (ØØH to 1ØH)
analog inputs	Number of analog inputs in controller (ØØH to FFH)
analog outs	Number of analog outputs in controller (ØØH to FFH)
checksum	Complement of modulo-256 sum of previous 10 bytes
FFH	Last byte of packet

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø3H	Packet Length
45H	"Misc. I/O Count Request" Function Code
BAH	Checksum of above byte
FFH	Last byte of packet

Response from Controller:

Ø7H	Packet Length
46H	"I/O Count" Function Code
protos	Number of prototyping boards in controller
h.s.counters	Number of high-speed counting channels in controller
twhls	Number of thumbwheel arrays (4-digit) connected to controller
disps	Number of numeric displays (4-digit) connected to controller
checksum	Complement of modulo-256 sum of previous 5 bytes
FFH	Last byte of packet

IMPORTANT: This command is not currently supported by the model 2200.

Send to Controller:

Ø1H	Signifies CTC Binary Protocol
Ø4H	Packet Length
23H	"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	Complement of modulo-256 sum of previous 2 bytes
FFH	Last byte of packet

Response from Controller:

39H	Packet Length
24H - 27H	"Controller Status" Function Code, for task banks Ø to 3, respectively
stopped	True ("ØFFH") if controller is stopped, otherwise false ("ØØH")
fault type	Type code for Software Fault, if any are present (otherwise ØØH) - see chart below for fault codes
LSB **MSB**	Step number of Software Fault, if any, where ØØØØH = step #1, ØØØ1H = step #2, etc. (unspecified if no Software Fault is present)
LSB **3SB** **2SB** **MSB**	Data relating to Software Fault, if any (otherwise unspecified)

48 Bytes follow, providing the following data for each of the eight tasks being reported:

LSB **MSB**	Step number currently being executed by this task, where ØØØØH = step #1, ØØØ1H = step #2, etc.
LSB **3SB** **2SB** **MSB**	32-bit mask, indicating with a '1' or a '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.

(end of task data)

checksum	Complement of modulo-256 sum of previous 55 bytes
FFH	Last byte of packet

Usage Notes:
This is the command which Quickstep uses to gather step information for reporting "Program Status". When executed four times, once for each group of eight possible tasks, all of the information necessary to reconstruct the hierarchy and status of the controller's tasks is provided. In addition, if a software fault has halted execution of your program, the controller's response will indicate the nature of the fault, as well as the step where the fault occured and any relevant parametric data.

As each new task is started by your DSP™ program, the task is assigned a task number from 1 to 32. The main program (i.e.; the program being executed prior to the commencement of multi-tasking) is *always* assigned to task number 1.

Each of the 32 tasks, whether they are currently being used or not, will report back a step number, along with a 32-bit "mask" word. If the task is being used by your program, the mask will show whether the task is currently suspended, waiting for one or more subsidiary tasks to be "done". 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 instance, if the main program (task #1) called up three subsidiary tasks (tasks #2, #3 and #4), the mask word for task #1 would be as follows:

ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØ 1 1 1 Ø
MSB LSB

The mask word for tasks 2 through 4 would all be ØØØØØØØØH, indicating that these tasks have no subsidiary tasks being executed.

To extract the hierarchy of tasks being executed, therefore, start with task #1 and read its mask word to determine its subsidiary tasks, if any. Then, read the mask word of each subsidiary task; these will indicate if any tasks are being executed at the next level down in the hierarchy, and so on. 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 (remembering that the step numbers are offset by -1).

Do not assume that the task numbers will be allocated in the order of task hierarchy; the starting and stopping of various tasks 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!*

List of Software Fault Codes:

1	Illegal Function	14	No Such Data Table Column
2	Bad/Corrupt Program Data	15	No Such Data Table Row
3	Destination step is Empty	16	No Such Prototyping Board
4	Bad Thumbwheel Data	17	Illegal Sample Time
5	Step #1 is Empty Step	18	No Such Analog Input
6	Too Many Tasks	19	No Such Analog Output
7	No Such Stepping Motor	20	No Such Display Exists
8	Motor Not Ready	21	No Such Input Exists
9	Motor Unprofiled	22	No Such Output Exists
10	No Such Servo Exists	23	No Such Thumbwheel Exists
11	Servo Not Ready	24	Illegal Data Table Value
12	Servo Error	25	Message Transmitting Busy
13	No Such Register Exists	26	Divide-by-zero Error
		27	Data Out Of Range

In the event that the data transmission from the host computer cannot be executed by the controller, the controller will respond with an error code indicating the nature of the fault. The error code will be transmitted in the following format:

Ø3H	Packet Length
error code	Error code (see below)
checksum	Checksum (i.e.; complement) of above byte
FFH	Last byte of packet

Possible error codes:

64H	No error (acknowledgement of transmission)
65H	Checksum error, or end of packet <> FFH
66H	Illegal register number specified
68H	Value out of range (e.g.; input number not present in controller)

Network Communications and the CTC Protocols

When setting up a Local Area Network, one of two possible categories of interchange may exist within the network: "peer-to-peer" or "host/slave". At this time, all networking approaches supported by CTC are of the host/slave type; this configuration more closely reflects the hierarchy of a typical factory information management scheme. This section will describe both categories, however, for informational purposes.

Peer-to-Peer Networks

As you may guess, the peer-to-peer network treats each system connected to the network as an equal. This is typically accomplished in the format of a multi-drop network, where each system is connected to a single transmission line which constitutes the network. Networks of this type work in one of two ways:

In a "collision-detection" network, any system on the network is allowed to asynchronously transmit a message. If two systems happen to simultaneously transmit, causing the message to be garbled, this is detected and a retransmission is initiated some random amount of time later.

On the other hand, a "token-passing" network allows each member of the network an opportunity to transmit a message. This is accomplished by passing a "token" (the token is actually a message which constitutes permission to transmit) from one system to the next in some predefined order.

The trade-off between these two implementations is largely a matter of transmission time. The collision-detect network will typically have the fastest *average* transmission time, because any system on the network may instantaneously transmit a message (assuming the network is not currently busy; an important assumption!). However, the worst-case transmission time may be very long in a collision-detect network, due to the fact that some random chance exists that repeated transmission trys will be unsuccessful.

In a token-passing network, the maximum transmission time may be controlled and determined by establishing the number of systems in the network and the allowable packet length for each transmission. A certainty then exists that, within a given maximum period of time, the transmission will be allowed. The average time involved will be longer, because the network is additionally occupied by the token-passing activity, and each system must wait its turn to effect a transmission.

Host/Slave Networks

Host/slave networks, which may be implemented with either a multi-drop or a ring-network topology, involve the use of a "host", or master, computer, which controls all transactions on the network. The CTC Protocols described in this manual are all host/slave networks, with a computer acting as a host, issuing commands on the network which are responded to by the individual controllers.

CTC Series 2400/2800 controllers support host/slave networks in a ring configuration and, in the near future, a multi-drop configuration. The protocol implications of these networks are explored further below.

The Ring Network (host/slave)

As shown in section 2 ("A Practical Communications Hierarchy"), data in a ring network is retransmitted from system to system until it reaches the intended "target" recipient. Therefore, there must be some means of determining for which system the data is intended.

In the CTC Series 2400/2800 Controllers, this is accomplished through an addition to the CTC ASCII Protocols. When a command is transmitted from the host system (i.e.; personal computer, etc.), it is prefaced with the letter "N" (standing for a "Network" transaction), followed by a number indicating the controller for which the command is destined. For example, in a non-network connection, the following command would result in the number "1200" being stored in the controller's Register #10:

R10 = 1200 <cr>

If, however, the command is to be transmitted to a ring network, destined for the fourth controller around the ring, the following command would be transmitted instead:

N4R10 = 1200 <cr>

Note the prefix "N" in the above command, followed by the desired controller number. The response which will be transmitted back to the computer is identical to that specified for the CTC ASCII Protocols, except that the response will be prefaced with "NØ" (zero, not the letter O). Because the above command would normally generate a carriage return as the only response (in the Computer Protocol), the following response would be received from the ring network by the computer:

NØ <cr>

How the Ring Network Works

When the first controller in the ring network receives the command "N4R10 = 1200 <cr>", the "N" prefix indicates to the controller that the command is a network transaction, and the "4" indicates that the command is destined for a later controller. The first controller will *subtract one from the controller number*, and retransmit the command as:

N3R10 = 1200 <cr>

The second controller, upon receiving this command, will once again retransmit the command as "N2R10 = 1200 <cr>", and so on. When the fourth controller finally receives the command as "N1R10 = 1200 <cr>", the "N1" indicates that the command is intended for that controller; it processes the command, and sends its acknowledgement ("NØ <cr>") on around the ring, back to the computer. Each remaining controller, as it receives the response prefaced with "NØ", simply retransmits the response intact.

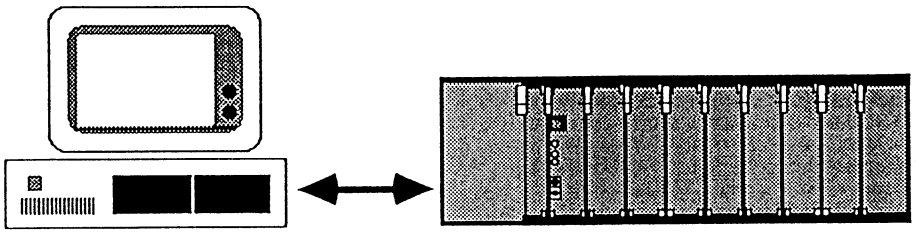
Note that the ring network configuration is sensitive to the order of connection of the various controllers. If this order is changed, the commands to be transmitted to affect specific controllers must be changed as well.

Communications Examples

Note: The program examples shown herein are for illustrative purposes only. In actual application, additional data checking and qualification may be indicated. Further, although these examples are given in BASIC, many versions of BASIC exist, with substantive differences in syntax and protocol.

Communications Example #1:

Computer Determines "Motor Position", Send Coordinates to Controller



In applications where extremely complex tasks are being performed (i.e.; vision processing, sophisticated process control functions, etc.), a computer may be needed as an active participant in the control task.

For example, if a vision system, connected to the computer, is able to determine the position and orientation of a workpiece, the computer may then be required to send that data to the controller. The controller is then able to position an actuator to grasp the part.

This can be accomplished with the CTC ASCII Computer Protocol by programming the compu-ter to force the position data into the controller's Numeric Registers. The motor commands to be executed by the controller will then derive their position data from these registers.

A Flag within the controller may be used for "hand-shaking", to tell the controller that a new set of position coordinates have been loaded, and to tell the computer when those coordinates have been used by the controller.

Considerations for Programming the Controller

The impact on the controller's DSP™ program will be minimal. First, the "hand-shaking" flag being used (Flag #1) must be initially CLEARed, preferably in Step #1 of the program. When the computer sees that this flag is clear, it will load coordinate information into Numeric Registers #10 and #11, and then SET the same flag, indicating that data is ready for use by the controller.

Just prior to the step of the DSP program where the controller will make use of the data, the instruction "MONITOR FLAG 1 SET GO NEXT" should be programmed to insure that new data has been loaded by the computer.

Then, the following instructions may be programmed at the next step:

turn motor#1 to reg#10
turn motor#2 to reg#11
clear flag#1
monitor (and motor#1:stopped motor#2:stopped) goto next

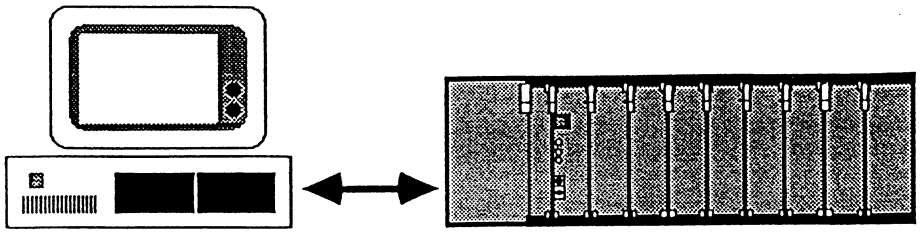
These instructions make use of the computer-loaded data as motor coordinates, clear the hand-shaking flag to signal the computer that the data has been used, and wait for the motor motions to finish before proceeding with the program.

Note that, if desired, math instructions could be inserted before the TURN MOTOR instructions to scale the position data prior to use. This would allow the transmitted data to be in the form of meaningful (i.e.; engineering units) information.

```
1Ø OPEN "COM1:9600,N,8,1,CS,DS" AS #1   :REM - initializes com. port on computer
2Ø PRINT #1,"PC"                        :REM - sets 2800 to "Computer Protocol"
3Ø LINE INPUT #1,R$                     :REM - get controller's response
4Ø IF R$ <> "PCØ" GOTO 2ØØ              :REM - if comm. not successful, jump out
5Ø X = 15ØØØ                            :REM - motor #1 position
6Ø Y = 215Ø                             :REM - motor #2 position
7Ø PRINT #1,"F1"                        :REM - request status of flag #1 from 2800
8Ø LINE INPUT #1,R$                     :REM - reads controller's response
9Ø IF R$ <> "Ø" GOTO 7Ø                 :REM - if 2800 not ready, try again
1ØØ PRINT #1,"R1Ø=";X                   :REM - transmit motor #1 pos. to REG-10
11Ø LINE INPUT #1,R$                    :REM - accept controller's response
12Ø IF R$ <> "" GOTO 2ØØ                :REM - if an error message, jump out
13Ø PRINT #1,"R11=";Y                   :REM - transmit motor #2 pos. to REG-11
14Ø LINE INPUT #1,R$                    :REM - accept controller's response
15Ø IF R$ <> "" GOTO 2ØØ                :REM - if an error message, jump out
16Ø PRINT #1,"F1=1"                     :REM - signal 2800 that data is ready
17Ø LINE INPUT #1,R$                    :REM - accept controller's response
18Ø IF R$ <> "" GOTO 2ØØ                :REM - if an error message, jump out
19Ø GOTO 5Ø                             :REM - do again
2ØØ PRINT "Communications Error":GOTO 2ØØ  :REM - error trap
```

Communications Example #2:

Computer Stores Parameters for Many Products, Downloads Appropriate Data Prior to Producing Batch



An increasingly-used technique for creating "flexible" machines is to store relevant production data (dimensions, time durations, etc.) for a number of products to be produced in a central computer system. Here the data may be modified, reviewed, etc., and the machine time may be scheduled by Production Control and coordinated with sales-driven requirements.

Then, when production is required for a given product, the data for that product is downloaded to the machine's controller (perhaps along with the desired production quantity).

Although the controller's Numeric Registers may certainly be used for this purpose, often the amount of data involved (along with a requirement for organizing the data in columns and rows) points to the use of the controller's Data Table. In addition to modification from your DSP™ program or by using Quickstep™, this Data Table may be accessed directly through the CTC ASCII Computer Protocol.

Along with this Data Table information, a desired production quantity may also be transferred into one of the controller's registers and, if the machine is to be completely automatic and unmanned (and suitable safety precautions have been taken), a flag may even be used to start the machine automatically. This flag may later be used by the controller to signal the completion of the batch.

Considerations for Programming the Controller

The program within the controller is written normally, with any variable production data (motor coordinates which establish dimensions, critical time delays, etc.) drawn from the controller's Data Table (see instructions for related programming information). In this instance, however, the Data Table will be down-loaded from a computer in each instance.

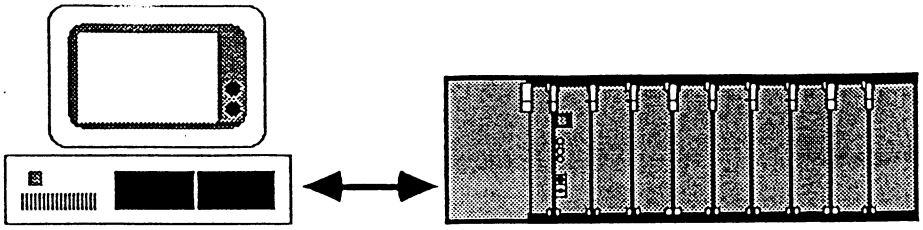
If the batch quantity is also to be down-loaded into one of the controller's registers, this register may then be decremented after each machine cycle with the instruction "***store reg#10 - 1 to reg#10***", and tested for completion with the instruction "***if reg#10 <= 0, goto [1]***". (This assumes that step #1 is an initialization step which will result in stopping the machine.)

If a Flag is to be used to automatically start the machine, this Flag may be tested with the instruction "***monitor flag#1:set, goto next***", prior to the beginning of the machine's cycle. After the production batch is complete, the program should clear the flag ("***clear flag#1***").

```
1Ø DIM N(6,4)                    :REM - establish array of parameters
2Ø DATA 1ØØØ,15ØØ, 382,  12
3Ø DATA 185Ø, 3ØØ,12ØØ,155Ø
4Ø DATA  128, 55Ø,21ØØ, 95Ø
5Ø DATA  999,125Ø,1ØØØ,  48
6Ø DATA 19ØØ,1ØØØ,15ØØ, 9ØØ
7Ø DATA 11ØØ, 4ØØ,825Ø,  5Ø
8Ø FOR R=1 TO 6                  :REM - read data into array
9Ø FOR C=1 TO 4
1ØØ READ N(R,C)
11Ø NEXT C
12Ø NEXT R
13Ø Q=15ØØ                       :REM - this represents desired quantity
14Ø OPEN "COM1:96ØØ,N,8,1,CS,DS" AS #1   :REM - initializes com. port on computer
15Ø PRINT #1,"PC"                :REM - sets 2800 to "Computer Protocol"
16Ø LINE INPUT #1,R$             :REM - gets controller's response
17Ø IF R$ <> "PCØ" GOTO 35Ø      :REM - if com. not successful, jump out
18Ø FOR R=1 TO 6                 :REM - row by row. . .
19Ø FOR C=1 TO 4                 :REM -      column by column. . .
2ØØ PRINT #1,"D";R;",";C;"=";N(R,C) :REM - send data to 2800's Data Table
21Ø LINE INPUT #1,R$             :REM - accept controller's response
22Ø IF R$ <> "" GOTO 35Ø         :REM - check for error
23Ø NEXT C
24Ø NEXT R
25Ø Q=STR$(Q)                    :REM - convert quantity to char. string
26Ø IF MID$(Q$,1,1)=" " THEN Q$=MID$(Q$,2,1Ø)
27Ø REM - this BASIC adds a leading space during 'STR$' if number is positive
28Ø PRINT #1,"R1Ø=";Q$           :REM - send desired production quantity
29Ø LINE INPUT #1,R$             :REM - accept controller's response
3ØØ IF R$ <> "" GOTO 35Ø         :REM - check for error
31Ø PRINT #1,"F1=1"              :REM - set flag in controller
32Ø LINE INPUT #1,R$             :REM - accept controller's response
33Ø IF R$ <> "" GOTO 35Ø         :REM - check for error
34Ø END
35Ø PRINT R$,"- Communications Error":GOTO 35Ø
```

Communications Example #3:

Computer "Monitors" Production Data
(Batch Counts, AQL info, etc.)



It is unfortunate that Production Control and Quality Assurance functions must often be performed on an "historical" basis, reacting to problems long after they have surfaced. The effective use of the CTC Protocols for data communications can provide current information to these functions, especially if their use is well integrated into the initial design of an automated machine.

The most basic of information which may be transferred is a cumulative production count (for the week, day, shift, etc.) which, if maintained in a non-volatile register within the controller (refer to programming information for the controller being used), will not be lost if power is removed from the machine.

If the machine has defect detection (and perhaps automatic bad-part rejection), separate "good-part / bad-part" counts may be kept by the controller, and an attached computer may then be used to track long-term trends in defect ratios.

Better still, if the machine has the ability to make qualitative measurements, either of the workpiece in process or of the machine's own performance (actuator reaction times, cycle times, critical temperatures, etc.), trends may be spotted by computer analysis of the resultant data, often long before defects start occuring.

The result can be greater uptime, better use of scheduled maintenance efforts and, in many instances, higher average product quality and improved rejection rates.

Considerations for Programming the Controller

Typically, production counts are kept in one of the controller's Numeric Registers. At the end of each machine cycle, an instruction such as "***store reg#10 + 1 to reg#10***" is used to increment the register. Additional registers may be used in a like fashion to maintain counts of bad workpieces detected by the machine.

For most applications, the coordination of the reading of production data by the computer is not a major issue. This is particularly true if the computer is passively reading a cumulative count.

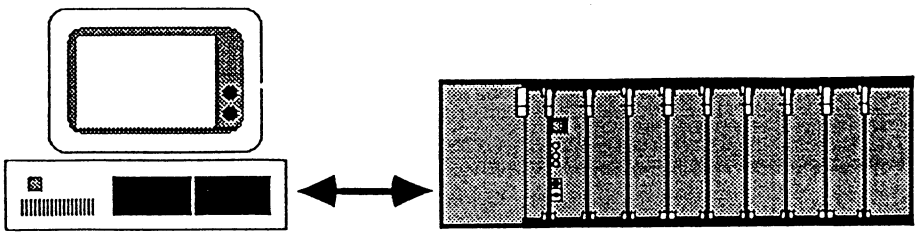
In instances where qualitative data is being periodically read, it may be desirable to have the computer reset the controller's accumulated counts to zero after having read the data. If the information is critical in nature, it is necessary to insure that the controller will not try to change any of the counts between the time the computer reads the data in the registers and the time the computer resets the registers (otherwise the count(s) added by the controller will be lost).

A Flag may be used for this purpose, by having the computer set the Flag prior to reading information, and then clear the Flag only after the registers have been reset to zero. The controller's program should be written to check this Flag prior to modifying the registers, proceeding only if the Flag is clear ("***monitor flag#1:clear, goto next***").

```
1Ø OPEN "COM1:96ØØ,N,8,1,CS,DS" AS #1    :REM - initializes com. port on computer
2Ø PRINT #1,"PC"                         :REM - sets 2800 to "Computer Protocol"
3Ø LINE INPUT #1,R$                      :REM - gets controller's response
4Ø IF R$ <> "PCØ" GOTO 21Ø               :REM - if com. not successful, jump out
5Ø CLS                                   :REM - clear the CRT screen
6Ø LOCATE 1Ø,5                           :REM - position the cursor for 1st message
7Ø PRINT "The current count of good parts is"
8Ø LOCATE 14,5                           :REM - position the cursor for 2nd message
9Ø PRINT "The current count of bad parts is"
1ØØ PRINT #1,"R1Ø"                       :REM - request 1st value from controller
11Ø LINE INPUT #1,A$                     :REM - get controller's response
12Ø IF MID$(A$,2,1)=CHR$(7) GOTO 21Ø     :REM - check response for error (ASCII bell)
13Ø PRINT #1,"R11"                       :REM - request 2nd value from controller
14Ø LINE INPUT #1,B$                     :REM - get controller's response
15Ø IF MID$(B$,2,1)=CHR$(7) GOTO 21Ø     :REM - check response for error
16Ø LOCATE 1Ø,4Ø                         :REM - position cursor after 1st message
17Ø PRINT A$;"          "
175 REM - Print 1st value, plus ten spaces to erase any previous, longer response
18Ø LOCATE 14,4Ø
19Ø PRINT B$;"          "                :REM - print 2nd value
2ØØ GOTO 1ØØ                             :REM - go back, get another update
21Ø PRINT "Communications Error":GOTO 21Ø :REM - error trap
```

Communications Example #4:

Computer Monitors for Fault Condition, Signals Operator if Present



The fact that serial communications, in CTC Controllers, is completely asynchronous to the operation of the controller's machine control program (written in DSP™), allows machine or process monitoring to be easily implemented.

The CTC Protocols allow rapid access to any of a controller's Numeric Registers, Inputs, Outputs, Analog I/O and Flags. If you write the controller's DSP program to insure that continuously-updated information is present in one of these resources, an attached computer is then free to continuously monitor and report on the status of that information.

Once this has been accomplished, any of the reporting resources available to the computer (which may include its CRT, printer, modem, etc.) may be used for alarm, logging or monitoring purposes.

Considerations for Programming the Controller

Several factors must be considered in properly writing a program to monitor a machine's operation. Perhaps the most important of these is to insure that the resource being monitored always has current information.

For example, if an operating pressure is being monitored by an analog input, and this data is mathematically converted by the controller to units of PSIG, with the resultant pressure stored in reg# 10, the information in reg# 10 will only be as current as the last time the math operations were performed. A separate, continuously-running task may be used to constantly update this information if desired. (An alternative would be to have the computer read the analog input directly and independently convert the data to PSIG.)

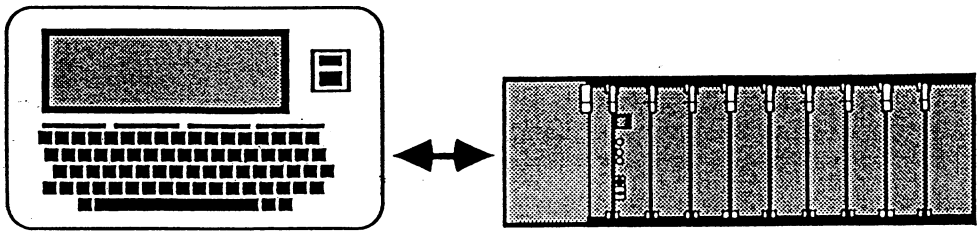
Another consideration in using data which the controller first manipulates is the impact of the controller's dedicated STOP and RESET functions, and the "***cancel other tasks***" instruction. Remember that these functions affect ALL of the controller's tasks, including any tasks which have been set up to convert data.

As in any instance where human safety is at stake, proper design practices point to the use of independent systems to detect critical conditions and effect emergency shutdowns.

```
1Ø OPEN "COM1:96ØØ,N,8,1,CS,DS" AS #1    :REM - initializes com port on computer
2Ø PRINT #1,"PC"                         :REM - sets 2800 to "Computer Protocol"
3Ø LINE INPUT #1,R$                      :REM - gets controller's response
4Ø IF R$ <> "PCØ" GOTO 21Ø               :REM - if com. not successful, jump out
5Ø CLS                                   :REM - clear the CRT screen
6Ø PRINT #1,"AI3"                        :REM - request data from Analog Input #3
7Ø LINE INPUT #1,R$                      :REM - accept controller's response
8Ø IF MID$(R$,2,1)=CHR$(7) GOTO 21Ø      :REM - check for error (ASCII bell)
9Ø IF VAL(R$) < 35ØØ GOTO 6Ø :REM - is analog value below limit?
1ØØ  LOCATE 1Ø,2Ø                        :REM - position cursor for warning message
11Ø  PRINT "WARNING - Temperature Limit exceeded!!":BEEP    :REM - print warning and
                                                              beep
12Ø  END
21Ø  PRINT "Communications Error":BEEP:GOTO 21Ø    :REM - error trap
```

Communications Example #5:

Using a Portable "Lap-Top" Computer for Start-up and Diagnostics



The widespread availability of small, inexpensive, battery-operated computers has made available a potentially valuable tool to the Machine Designer or Maintenance Technician. Computers such as the Radio Shack Series 100, when used in conjunction with the CTC Protocols, allow instantaneous access to virtually all of the controller's resources. This provides an important source of information for both initial setup of a machine and for diagnostic/troubleshooting purposes. A portable computer may also be used for data gathering, in instances where a permanent connection between a computer and the controller may be impractical.

There are two possible approaches to the use of portable computers with the CTC Protocols:

1. For the initial setup of a machine, the computer may be used as a "dumb terminal", allowing the Machine Designer to communicate directly with the controller's Numeric Registers, inputs, outputs, analog I/O, etc. In this manner, parameters determining stepping motor or servo characteristics may be quickly tuned, time durations may be varied to determine optimum performance, etc.

2. For maintenance or data gathering purposes, specific application programs may be written (typically in BASIC) for the computer. This allows an extremely "friendly" user interface to be created, with menus, prompting and on-screen identification of parameters.

"Dumb Terminal" Operation

The model 100 computer is supplied with a telecommunications program (called "TELCOM") which allows it to act like a terminal. When this program is started, it enters a command mode, at which time you must insure that the proper communications parameters have been set.

Set-up with the Radio Shack model 100

TELCOM will display its existing communications parameters when it is first started, in the form of a 5-character code (for example, "M7I1E"). This must be changed to agree with the requirements of the CTC Protocols, which require the code (for the model 100 only!) to be "88N1D" (this will set the baud rate to 9600, the word length to 8 bits, no parity, 1 stop bit and the line status to disable). The significance of each of these parameters is described more fully in the computer's operating manual.

The parameter code may be changed by pressing the "F3" function key (labelled "STAT"), followed by the characters "88N1D" (without the quotation marks and using the number "1", *not* the letter "I", in the four position!). Then press the "ENTER" key. The new parameters may be

confirmed by pressing the "F3" key again and, without entering new parameters, pressing the "ENTER" key. The computer will respond by displaying the currently-active communications parameters.

At this point you may enter the "terminal" mode, by pressing the "F4" function key (labelled "TERM"). Once in the terminal mode, the "F4" function key is used to toggle the terminal between the "half-duplex" and "full-duplex" modes of operation (the key will alternately be labelled "HALF" or "FULL"). The label for this key should read "HALF"; if it reads "FULL", press the "F4" key and it will change to "HALF". This will cause commands that you enter on the keyboard to be "echoed" (displayed) on the computer's screen.

Entering the CTC ASCII Terminal Protocol

Once the computer is properly initialized as a terminal (and assuming it is properly connected to the controller!), communications with the controller may begin. Start by setting the controller's communication protocol: type the characters "PT", followed by the "ENTER" key. The controller should respond with a line feed, the characters "PT", followed by another line feed. If this does not occur, a wiring problem is likely and the controller's Installation Guide should be consulted.

Assuming the protocol has been set properly, you may now enter commands into the computer, according to the ASCII Protocol described in this booklet. For example, entering the characters "R10" will cause the controller to respond with the current value stored in Numeric Register #10; entering the characters "R10=1000" will cause the controller to force the number "1000" into Numeric Register #10.

Writing an Applications Program

Just as an applications program may be written as part of a permanent installation, this technique may also be used with a portable computer. The model 100, along with most other portables, comes supplied with a built-in BASIC interpreter. This allows programs to be written and stored in the computer for execution.

The previous examples illustrate some of the techniques for accomplishing this, although minor differences in the versions of BASIC may require program modifications for proper execution. One area in which this is particularly true is in the initialization of the communications port; the model 100 will require the commands OPEN "COM:88N1D" FOR OUTPUT AS #1 and OPEN "COM:88N1D" FOR INPUT AS #2 to initialize the comm port.

Further differences in the versions of BASIC may be discovered upon careful reading of the manuals supplied with the specific computer you are using. The example below, written for the model 100, illustrates the use of Radio Shack's version.

```
1   REM - This program displays the status of 16 inputs, 16 outputs and two registers
5   CLS
10  MAXFILES=2                  :REM - sets maximum num. of files on comp.
20  OPEN "COM:88N1D" FOR OUTPUT AS #1  :REM - comm. port is opened and initialized
30  OPEN "COM:88N1D" FOR INPUT AS #2
40  PRINT #1,"PCL"              :REM - CTC Computer protocol is used,
50  INPUT #2,R$                 :REM -     with "line feed" option
60  IF R$<>"PCØL" THEN PRINT "COMM. ERROR":GOTO 40
70  PRINT @ 10,"CTC Diagnostics Demo"   :REM - Note screen formatting "@ XXX"
80  PRINT @ 80,"Inputs 1-16:"
90  C=92                        :REM - Cursor position marker
120 FOR X = 1 TO 16             :REM - Loop to get and display 16 inputs
130 PRINT #1,"I";X              :REM - Get an input status
140 INPUT #2,R$
150 IF R$="1" THEN PRINT @ (C+X),"X" ELSE PRINT @ (C+X),"–"
151 REM - Above line prints an "X" if input is active, otherwise prints a "–"
155 IF X/4=INT(X/4) THEN C=C+1
156 REM - Above line will skip a space every fourth input for a clearer display
160 NEXT X
165 PRINT @ 160,"Outputs 1-16:"
170 C=172:REM - Cursor position marker
175 FOR X=1 TO 16               :REM - Loop to get and display 16 outputs
180 PRINT #1,"O";X
190 INPUT #2,R$
191 IF R$="1" THEN PRINT @ (C+X),"X" ELSE PRINT @ (C+X),"–"
200 IF X/4=INT(X/4) THEN C=C+1
210 NEXT X
220 PRINT #1,"R10"              :REM - Get value of Register #10
230 INPUT #2,A$
233 PRINT #1,"R11"              :REM - Get value of Register #11
236 INPUT #2,B$
240 PRINT @ 240,"Registers 10,11:"  :REM - Display register values
250 PRINT @ 257,A$;", ";B$
260 GOTO 90                     :REM - Go back for another update
```

Glossary of Terms

ASCII - An industry-standard binary code for representing alphabetic and numeric characters, where a 7-bit binary code is assigned to each of the letters A to Z, the numerals 0 to 9, as well as a number of special control characters (carriage return, line feed, bell, etc.). The ASCII code is a common method of interchanging data between dissimilar systems.

Hierarchy - A structured array of systems (or of information) where systems at the lower level handle lower-level, immediate transactions, while systems at higher levels handle supervisory or higher-level functions.

LAN (Local Area Network) - Although "local" is a relative term, a Local Area Network is typically used to link together systems performing related functions which are in close proximity to one another (on the scale of several hundred feet).

MAP (Manufacturing Automation Protocol) - A standard proposed by General Motors for the communication of manufacturing data plantwide, MAP encompasses definitions of both electrical signal characteristics and informational content. MAP may possibly become a widely used standard for plant-wide networks.

Multi-Drop - A type of data network where a common communications link is used for all systems. Systems connected to a multi-drop network are typically coordinated so that only one system will be transmitting on the link at any given time, to avoid "contention" (two transmitters "fighting" each other for control over the line). The primary benefits of a multi-drop network configuration are that the loss of any one system typically will not disturb the network, and that communications speeds are often higher due to the use of a direct data route (without retransmissions).

Parallel - A method of data transmission employing a number of data lines (typically eight), whereby a full 8-bit byte of data is presented on the data lines and a separate control line is strobed to allow a receiving system to latch the data. Although this method is typically faster than serial communications, it is seldom employed for tranmissions over any great distance, due to the number of conductors necessary in the transmission cable.

Protocol - A definition of the data format and interchange necessary to complete a communication with a given system.

Ring Network - A type of network where a number of systems are interconnected in a "ring" configuration, where the "transmit" line from one system is connected to the receive line of the next, whose transmit line is, in turn, connected to the receive line of the next, etc. During a data transmission, each successive system receives the data, determines if the data is destined for that system and, if not, re-transmits the data to the next system. Responses, if required, are transmitted from the target system on around the ring, back to the originating system. Although a convenient and inexpensive means of creating a Local Area Network, ring networks suffer from the disadvantage of requiring all systems in the network to be powered and functioning to complete a transmission.

RS-232 - An electrical standard which defines the signal levels and

characteristics for data transmission. Note that "RS-232" does not in any way define the "protocol" or informational content of a transmission and that, therefore, "RS-232-compatible" means little in ascertaining system compatibilities.

Serial - A method of data transmission employing, typically, one transmit and one receive line, where all data is converted to a series of pulses transmitted serially. This is a commonly-used means of transmitting digital information, due to the fact that data may be transmitted with minimal cabling and transmission hardware.

Workcell - In an automated factory, a group of machines performing related functions, typically linked by a local area network to a common "Workcell Controller". For example, a workcell may consist of a milling machine, assembly station, video inspection station and a robotic arm to transfer workpieces among the stations.