CYRANO COMMAND REFERENCE

Form 703-990421 — April, 1999

OPTO 22

43044 Business Park Drive, Temecula, CA 92590-3614
Phone: 800-321-OPTO (6786) or 951-695-3000
Fax: 800-832-OPTO (6786) or 951-695-2712
www.opto22.com

Product Support Services:
800-TEK-OPTO (835-6786) or 951-695-3080
Fax: 951-695-3017
E-mail: support@opto22.com
Web: support.opto22.com

Cyrano Command Reference
Form 703-990421 — April, 1999

TABLE OF CONTENTS

WELCOME

WHAT IS CYRANO?

The Cyrano 200 Visual Control Language ("Cyrano," for short) is a powerful, easy-to-use program that enables you to develop control applications for Opto 22's Mistic systems right from your PC. These applications are based on simple flowcharts familiar to anyone involved in process control design. Because these flowchart concepts are fundamental, and because the terminology used to program Cyrano is plain English rather than techno-jargon, you will find Cyrano easy to learn and intuitive, whether or not you have any previous programming experience.

But don't be fooled by Cyrano's ease of use. Features such as multitasking, full debugging capabilities, and an extensive set of built-in advanced tools combine to make Cyrano the most powerful and versatile control design program you will ever need.

WHAT IS CYRANO USED WITH?

An inexpensive and readily available IBM-compatible PC workstation, equipped with color graphics and a mouse, is all that you need to run Cyrano. By making selections from Cyrano's color graphic menus on your PC workstation and using your mouse to draw interconnections, you can create a control chart that defines how you want your application to work. Cyrano then completes the rest of the work for you by creating a computer program that runs your application on the Opto 22 Controller.

Once you have developed and debugged a control application using Cyrano, you can download it directly to an Opto 22 controller. At this point your program becomes a stand-alone application running on the controller, and the PC is no longer required.

Hardware

Applications developed in Cyrano will run on control systems with the following Opto 22 equipment:

- Opto 22 Controller (with Flash EEPROM or EPROM)
- Digital and Analog I/O Units
- G4 Type I/O Modules, as required by your application

For multidrop applications that require several controllers to be connected to digital and analog I/O bricks, the following hardware is required:

- An IBM-compatible PC workstation
- An Opto 22 AC24, AC422 RS-485/422, or AC37 (115-KBd) Adapter Card, or an SMC PC-130 ARCNET card

- An Opto 22 system with:
  – Opto 22 200 Controllers, as required by your application
  – Opto 22 Digital and Analog I/O Units, as required by your application
  – Opto 22 Digital and Analog I/O Modules (G4 type), as required by your application
- Serial cables, or coaxial cables and hubs, to multidrop-connect the PC to Opto 22 controllers

FIRMWARE

Cyrano requires compatible firmware to be installed on various hardware components, as detailed below:

	Current Version	Minimum Version Required
Analog Brick		
Single-Point Local (G4A8L)	R3.0a	LA 117
Single-Point Remote (G4A8R)	R3.0a	RA 117
HRD High-Density Local (G4HDAL)	R3.0a	LAM 105
HRD High-Density Remote (G4HDAR)	R3.0a	RAM 105
Digital Brick		
Local Multifunction Digital (G4D16L)	R3.0a	LD 109
Remote Multifunction Digital (G4D16R)	R3.0a	RD 109
Local Simple Digital (G4D16LS)	LS 101	LS 101
Remote Simple Digital (G4D32RS)	R3.0a	R3.0a
Mistic Controllers		
G4LC32ISA	R3.1h	R3.0a
G4LC32ISA-LT	R3.1h	R3.0a
G4LC32SX	R3.1h	R3.0a
G4LC32	R3.1h	R3.0a
M4	R3.1h	R3.0a
M4 I/O	R3.1h	R3.0a
M4RTU/DAS	R3.1h	R3.0a

ABOUT THIS MANUAL

The *Cyrano Command Reference* is the second of three volumes in the Cyrano documentation set. This reference manual provides complete descriptions of all Cyrano commands, both operations (which execute something) and conditions (which evaluate something). It also includes detailed overview information on various command groups.

The other two Cyrano manuals are:

- *Cyrano User's Guide* (Opto 22 form 702) — general information on installing and using Cyrano plus a description of all tools, menus, and dialog box options
- *Cyrano Tutorial* (Opto 22 form 704) — a step-by-step introduction to Cyrano application development

This manual is organized as follows:

- **Chapter 1: Overviews** — general information, tips, and usage examples for various command groups
- **Chapter 2: Operations** — complete descriptions of all Cyrano operation commands, organized alphabetically within command groups. Includes an alphabetical index of all operation commands at the beginning of the chapter.
- **Chapter 3: Conditions** — complete descriptions of all Cyrano condition commands, organized alphabetically within command groups. Includes an alphabetical index of all condition commands at the beginning of the chapter.
- **Chapter 4: Error Codes** — descriptions and possible causes of all Cyrano errors
- **Appendix A: Product Support** — how to reach Opto 22

Command descriptions include the following information:

- **Function** — a general description of the command's purpose
- **Typical Use** — a description of one or more common uses
- **Details** — specific information on how and when to use the command and what to know when using it
- **Arguments** — the number and type of command parameters required
- **Example** — a usage example, including sample arguments (if any)
- **Notes** — tips and special information (if any)
- **Dependencies** — special conditions to be met before using the command (if any)
- **Error Codes** — descriptions of all errors that could occur with the command (if any)
- **See Also** — related commands (if any)

DOCUMENT CONVENTIONS

- **Bold** typeface indicates text to be typed. Unless otherwise noted, such text may be entered in upper or lower case. (Example: “At the DOS prompt, type **cd \windows**.”)
- *Italic* typeface indicates emphasis and is used for book titles. (Example: “See the *Cyrano User’s Guide* for details.”)
- Names of menus, commands, dialog boxes, fields, and buttons are capitalized as they appear in the product. (Example: “From the File menu, select Print to bring up the PRINT TOPIC dialog box.”)
- File names appear in all capital letters. (Example: “Open the file TEST1.TXT.”)
- Key names appear in small capital letters. (Example: “Press SHIFT.”)
- Key press combinations are indicated by hyphens between two or more key names. For example, SHIFT-F1 is the result of holding down the SHIFT key, then pressing and releasing the F1 key. Similarly, CTRL-ALT-DELETE is the result of pressing and holding the CTRL and ALT keys, then pressing and releasing the DELETE key.
- “Press” (or “click”) means press and release when used in reference to a mouse button.
- Menu commands are sometimes referred to with the Menu➠Command convention. For example, “Select File➠Run” means to select the Run command from the File menu.
- Numbered lists indicate procedures to be followed sequentially. Bulleted lists (such as this one) provide general information.

ABOUT OPTO 22

Opto 22’s goal to deliver total control to industrial automation customers dates back to its beginnings in 1974 with the introduction of optically-isolated solid-state relays. Today, Opto 22 is the number one provider of I/O systems, with more than 80 million points of I/O working reliably worldwide. After earning a reputation for consistent innovation and leadership in automation hardware, Opto 22 realized it was time to take a new approach to control software. In 1988, Opto 22 introduced the first flowchart-based control programming language. Opto 22 continues to deliver successively more advanced generations of hardware and software.

All Opto 22 products are manufactured in the U.S. at the company’s headquarters in Temecula, California, and are sold through a global network of distributors, system integrators, and OEMs. Sales offices are located throughout the United States. For more information, contact Opto 22, 43044 Business Park Drive, Temecula, CA 92590-3614. Phone Opto 22 Inside Sales at 1-800-452-OPTO or Opto 22 headquarters at 951-695-3000. Fax us at 951-695-3095.

You can also visit our Web site at **www.opto22.com**.

CHAPTER 1

OVERVIEWS

This chapter provides general information on fundamental terms and concepts you will find valuable when using various Cyrano commands. Use this information as a reference for learning the function of several command groups within the Cyrano language.

CHART OVERVIEW

WHAT IS A CHART?

The term "chart" refers to a flowchart, also known as a "task." The maximum number of tasks that can run concurrently is 32. Since the HOST task and the INTERRUPT chart are included by default in the 32-task queue, this means that up to 30 user-configurable charts can be run concurrently. It should be noted that the total number of charts in a program is *not* limited to 32. Provided enough memory is available, a total of 1,295 charts can exist per program; however, only 32 can be running *at any one time.*

WHAT IS THE HOST TASK?

The HOST task is an invisible "chart" that always exists and is always part of the 32-task queue. Its purpose is to respond to master/slave communications from a Cyrano Debugger, MMI, or other device using Mistic HOST protocol. The HOST task functions as the slave, which means that it never originates a message, it only responds to inquiries or commands.

There are two types of HOST task: the default HOST task and additional HOST tasks.

What is the default HOST task?

- This task runs by default.
- The default HOST task is specified during configuration of the Mistic controller and the Cyrano software, typically on COM0 or COM4 (ARCNET).
- This task must be used to download a new kernel to the Mistic controller.
- Opto 22 binary communication mode ("binary mode") is the default (for use with ARCNET and direct serial connections).
- Opto 22 ASCII communication mode ("ASCII mode") must be selected when using modems.

Since most modems and radio modems do not support the Mistic controller's binary mode, ASCII mode must be selected when using modems. See the processor manual for details on how to change communication modes. Note that COM4 (ARCNET) always runs in binary mode, even if ASCII mode is selected.

WHAT ARE ADDITIONAL HOST TASKS?

- One or more additional HOST tasks can be started or stopped under program control at any time.
- Each task started will take up one task slot in the 32-task queue.
- Additional HOST tasks can be assigned to COM0 through COM4 (ARCNET).
- Either binary or ASCII communication mode can be specified for COM0 through COM3.
- Additional HOST tasks *cannot* be used to download a new kernel to the Mistic controller.

USES FOR ADDITIONAL HOST TASKS

- Remote debugging via modem
- Remote MMI connections via modem
- Supporting a Debugger on one port, an MMI on another
- Supporting a local Debugger on ARCNET, a remote Debugger via modem

The binary mode of the Mistic controller has a very efficient 11-bit frame tailored especially for addressable communications. The parity bit is used to identify an address byte, not to carry parity information. All serial ports support this mode. However, most modems and radio modems do not support any other use of the parity bit. If binary mode doesn't work, use ASCII mode.

The ASCII mode of the Mistic controller converts *all* bytes to two ASCII hex characters (00 to FF). ASCII mode is required for use with modems and radio modems. Defaults are no parity, eight data bits, and one stop bit. ASCII mode may be desired for use with some Windows applications that do not work well in binary mode.

WHAT IS THE INTERRUPT CHART?

- The INTERRUPT chart is automatically created by the Configurator and cannot be deleted.
- The purpose of this chart is to service interrupts from I/O units that have interrupt-generating event/reactions configured and have interrupt wiring connected to the Mistic controller.
- The INTERRUPT chart is suspended by default. It runs automatically when an interrupt is generated by an I/O unit.
- The INTERRUPT chart does not use CPU time while suspended, but it does take up one of the tasks in the 32-task queue.
- Using STOP CHART to stop the INTERRUPT chart will take it out of the 32-task queue and prevent it from running when an interrupt occurs.
- Using START CHART to restart the INTERRUPT chart will put it back into the 32-task queue (if a time slot is available) and leave it suspended at BLOCK-0, ready to process an interrupt.

WHAT IS THE 32-TASK QUEUE?

- The queue is a list of the tasks and charts that are to run concurrently.

- Every task on the list is executed one at a time over and over.
- The order in which the tasks appear on the list is subject to change frequently, since tasks can come and go from the list as they are started and stopped.
- Any chart or task that is running or suspended is on the task list.

WHAT IS A TIME SLICE?

- A time slice is a fixed unit of CPU time. This unit is currently set at 500 microseconds (one-half millisecond).
- Each task in the 32-task queue is allocated one time slice by default. This results in the smoothest task switching operation.
- The maximum number of time slices is 8,160 (32 tasks x 255 time slices).

WHAT IS PRIORITY?

- Priority is the number of consecutive time slices a task can use.
- All tasks have a priority of 1 by default.
- The HOST task priority can be changed using SET HOST PRIORITY.
- The priority for other charts can be changed using SET PRIORITY.
- The valid priority range is 1 to 255.

HOW MUCH CPU TIME CAN A TASK USE?

Up to 100%. Suppose there are three tasks in the 32-task queue, each with one time slice (a priority of 1). In this case each task will use 33.33% of CPU time. If the third task is given two time slices (i.e., its priority is changed to 2), the first two tasks will each use 25% of CPU time while the third task will use 50% of CPU time (two consecutive 25% time slices).

The number of consecutive time slices allowed for each task ranges from 1 to 255 and can be changed on the fly under program control. See SET PRIORITY and SET HOST PRIORITY.

WHAT ABOUT SUBROUTINES?

Whenever a chart calls a subroutine, the subroutine temporarily inherits the task in use by the calling chart along with its priority.

DOES A TASK ALWAYS USE ALL OF ITS ALLOCATED TIME?

Not always. If a chart or subroutine runs in a loop, all allocated time will be used. If a chart or subroutine does not need all of its allocated time to complete its job, all remaining time (including portions of a time slice) is given up.

The following conditions will cause a chart to use less than a full time slice:

- The chart or subroutine stops.

- The chart or subroutine is suspended.
- The DELAY command is used.

DELAYing 1 millisecond is a handy way to give up the time slice while waiting for an event such as CHARACTERS WAITING? to occur.

WHEN WILL THE REQUESTED CHANGE TO A CHART OR TASK STATUS TAKE EFFECT?

Not immediately. In any multitasking system, timing and synchronization issues are always a concern. The time required for a particular request to be implemented depends on the number of tasks currently running, the priority of each, and the specified chart's location in the 32-task queue. In other words, it's hard to say. However, the worst-case delay can be calculated. For example, if four charts and one HOST task are running, each with a priority of 2 (two time slices each), the worst case delay would be 5 x 2 x 500 microseconds = 5 milliseconds.

HOW MANY CHARTS SHOULD I HAVE RUNNING CONCURRENTLY?

As few as possible. This leaves options as the program grows. Get in the habit of running only a few charts concurrently. Set up "chains of charts" where each chart in the chain starts the next chart as its last command. This way, all charts in the chain use only one task in the 32-task queue.

If two charts are running, both with a priority of 1, each will have equal execution time. This can be seen by examining the first eight time slices, as shown below:

Table 1-1: HOST Task and a Chart Both Running with a Priority of 1

Slice 1	Slice 2	Slice 3	Slice 4	Slice 5	Slice 6	Slice 7	Slice 8
HOST Task	CHART_A	HOST Task	CHART_A	HOST Task	CHART_A	HOST Task	CHART_A

If the HOST task priority is changed to 3, the following will occur:

Table 1-2: HOST Task with a Priority of 3 Running with a Chart with a Priority of 1

Slice 1	Slice 2	Slice 3	Slice 4	Slice 5	Slice 6	Slice 7	Slice 8
HOST Task	HOST Task	HOST Task	CHART_A	HOST Task	HOST Task	HOST Task	CHART_A

COMMUNICATION OVERVIEW

WHAT ARE THE MISTIC PORT ASSIGNMENTS?

- Ports 0-3 (COM 0-3) are serial — a variable mix of RS-232 and RS-422/RS-485 2-wire and 4-wire.
- Port 4 is ARCNET.
- Port 5 is the front panel keypad and LCD display of the G4LC32.
- Port 6 is parallel (16 wide) for local I/O.
- Port 7 is ARCNET peer, a virtual port. It uses the same connector as port 4 (ARCNET) for external connections.

WHAT IS A HOST PORT?

Any port that supports the Mistic HOST protocol (where the Mistic controller is always a slave). Ports 0-4 are eligible. The HOST port is always used by the Debugger and MMI.

There are two types of HOST ports: the default HOST port and additional HOST ports. Additional HOST ports differ only in that they do not support Mistic kernel downloads. A Mistic controller always has a default HOST port, usually port 0 or port 4. Many additional HOST ports can be defined under program control. See the Chart Overview for details.

WHAT COMMUNICATION MODES ARE AVAILABLE?

All HOST ports support either Opto 22 Mistic controller binary communication mode (the default) or Opto 22 Mistic controller ASCII communication mode. Binary mode uses an 11-bit frame (1 start, 8 data, 1 stop, 1 parity) with the parity bit used to indicate that the current byte is an address byte. Since most modems do not support this use of the parity bit, binary mode cannot be used with most modems. For this reason, ASCII mode is also available. This mode uses a 10-bit frame (1 start, 8 data, 1 stop, no parity) with all characters being printable ASCII 0–127. In this mode, any eight-bit binary data is sent as two ASCII hex characters.

Any modem will work with ASCII mode. However, be sure to select ENABLED for CTS under PC COM Port Configuration in Cyrano. Also be sure to connect CTS from the modem to the PC (a standard PC-to-modem cable does this automatically).

HOW IS ASCII MODE SELECTED FOR A HOST PORT?

For the default HOST port, it depends on which Mistic controller is used. Current methods are via front panel, jumper, and EEPROM. See your processor’s user guide for specific details on how to select the communication mode for your particular processor.

For additional HOST ports, use START HOST TASK (ASCII) in the POWERUP chart.

WHAT MODES CAN SERIAL PORTS BE IN?

- Opto 22 Mistic controller binary mode — This is the default mode for talking to remote I/O units. Special drivers are available (and required) to talk to remote I/O units in ASCII mode via modem.
- Opto 22 Mistic controller ASCII mode — This mode is used for talking via modem on a HOST port.
- Standard mode — This is the default mode for all serial ports that are not talking to remote I/O units and are not configured as a HOST port. Default is a 10-bit frame (1 start, 8 data, 1 stop, no parity). These parameters can be changed under Cyrano program control using the CONFIGURE PORT command.

WHAT MODES CAN ARCNET PORTS BE IN?

Binary mode only.

WHAT IS PEER-TO-PEER COMMUNICATION?

A fast method for two or more Mistic controllers to communicate with each other via ARCNET. All communication via ARCNET is CRC error-checked by the ARCNET protocol. The MMI and the Debugger can use the ARCNET at the same time it's being used for peer-to-peer communication.

Peer-to-peer communication uses port 7 (a virtual port within the Mistic controller) and the ARCNET port for external connections.

Certain commands must be used to send data to port 7, such as SET PEER DESTINATION ADDRESS and PRINT NEW LINE (PORT) W/TIMEOUT. See example peer applications included with the Cyrano distribution files or on the Opto 22 BBS.

WHAT IS AN "OPEN" COMMUNICATION PORT?

One that is in use or "locked" by a chart or subroutine. An open port is not available to any other charts as long as it remains open.

A port is opened by using REQUEST PORT and closed by using RELEASE PORT. Valid commands for open ports always include the word PORT in parentheses, e.g., (PORT).

WHAT IS A "CLOSED" COMMUNICATION PORT?

One that is available for general use. Valid commands for closed ports always require the port number to be specified as part of the command.

HOW MANY PORTS CAN AN INDIVIDUAL CHART HAVE OPEN AT ONCE?

Only one. If a chart requires multiple ports, only one can be open at a time.

CAN TWO CHARTS HAVE THE SAME PORT OPEN AT THE SAME TIME?

No, not if the REQUEST PORT command has been used to open the port. For this reason you should check the status returned by this command to verify that the port was available (-1 indicates success).

WHAT IS A RECEIVE BUFFER?

Each port has a separate location in memory known as its receive buffer. Messages sent to the controller automatically go in this buffer for later retrieval by the program. The typical size of a receive buffer is 253 characters, although port 5's receive buffer holds only one character.

WHAT IS A TRANSMIT BUFFER?

Ports 4, 6, and 7 have separate locations in memory known as transmit buffers. Characters sent to these ports do not get transmitted right away. A command such as PRINT NEW LINE TO PORT must be used to transfer the contents of the transmit buffer to the port. The typical size of a transmit buffer is 250 characters.

HOW MANY MESSAGES CAN THESE BUFFERS HOLD?

For ports 0–3, as many as will fit.

For ports 4, 6, and 7, the receive buffer can hold only one message, regardless of length. This message consists of all characters that were in the transmit buffer of the sender when the message was sent (using PRINT NEW LINE TO PORT if the message came from another Mistic controller).

WHAT TYPE OF FLOW CONTROL IS SUPPORTED ON SERIAL PORTS?

Hardware only: RTS/CTS. RTS stands for Request To Send. The RTS output is on when characters are being sent. CTS stands for Clear To Send. The CTS input is on by default on most Mistic controllers (the known exception is port 0 on the M4RTU). CTS must be on to send. It is used to externally stop the sending of characters.

WHERE CAN BAUD RATE, # DATA BITS, ETC., BE CHANGED?

- Under program control using CONFIGURE PORT. The changes take effect immediately and override all other means used to set port baud rates. **Tip:** Set all serial port parameters in the POWERUP chart to ensure they are correct.
- For selected ports, the baud rate can be changed from the Mistic front panel, switches, or jumpers. Any changes made this way *do not take effect* until power is cycled and can be overridden under program control using CONFIGURE PORT.
- The Cyrano Configurator can be used to set baud rates. Such a change takes effect only after a download and can be overridden under program control using CONFIGURE PORT.

HOW DO YOU TROUBLESHOOT FAILED COMMUNICATIONS?

Serial

- Check baud rate, # data bits, # stop bits, parity, communication mode (binary vs. ASCII), address, etc.
- Connect RTS to CTS on the Mistic serial port.
- Cycle power to the Mistic controller and try again.

ARCNET

- Make sure the ARCNET card in the PC has a unique ARCNET address (usually set at 1 from the factory). You are advised to use address 128.
- Cycle power to the Mistic controller and try again.

DIGITAL POINT OVERVIEW

WHAT ARE XVAL AND IVAL?

All I/O points have two associated values: XVAL and IVAL. Unless you are using the Debugger to manipulate I/O values or to disable an I/O point or I/O unit, you do not need to be concerned with these values.

The external value, or XVAL, is the "real" (hardware) value as seen by the I/O unit. This value is external to the Mistic controller.

The internal value, or IVAL, is a logical or software variable copy of the XVAL that resides within the Mistic controller. The IVAL *may or may not be current,* since it is updated to match the XVAL only when a read or write is done to an enabled I/O point by the program in the Mistic controller.

Do not be concerned when the IVAL does not match the XVAL, since this means only that the program is not reading from or writing to the I/O point in question.

SIMULATION AND TEST: THE "REAL" USE FOR XVAL AND IVAL

To force an XVAL for a specific output to a particular value in order to test output performance, you do not necessarily have to disable the output. If the program is actively writing to the output, you will need to disable it. On the other hand, if the program is stopped, there is no need to disable any output.

To force an IVAL for a specific input to a particular value in order to test program logic, you must disable the input first.

Disabling can be performed under program control by using DISABLE DIGITAL POINT, DISABLE ANALOG POINT, DISABLE I/O UNIT, etc. However, disabling is usually handled via the Debugger by selecting "I/O" to view the "DEBUG POINT DISPLAY SCREEN."

DIGITAL COUNTERS

Before using a counter, it must be activated using START COUNTER for single inputs or START QUADRATURE COUNTER for quadrature inputs. This is normally done in the POWERUP chart.

To keep a counter active after a power failure at the I/O unit, use the Debugger to write or "burn" the current I/O unit configuration to EEPROM after the counter is started.

ADDITIONAL COMMANDS

Although not listed under Digital Point operations or conditions, several I/O Unit and Logical commands can be used for digital operations:

- MOVE can be used to cause an output on one I/O unit to assume the state of an input or output on another I/O unit. A digital input or output that is on will return a True (-1). A True (non-zero) sent to a digital output will turn it on.
- NOT can be used to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Event/reactions can be used to cause an output to track an input on the same digital multifunction I/O unit.
- DO BINARY READ can be used to get the state of all 16 channels at once. BIT TEST can then be used to determine the state of individual channels.
- DO BINARY WRITE, DO BINARY ACTIVATE, or DO BINARY DEACTIVATE can be used to control all 16 outputs at once.

EVENT/REACTION OVERVIEW

WHAT IS AN EVENT/REACTION?

An event/reaction is a powerful and unique feature of the Mistic system that allows users to "off load" or distribute control logic to an I/O unit. That is, some of the logic in a control strategy can be run on the I/O unit independently of the Mistic controller.

As the name suggests, an event/reaction consists of an event and a corresponding reaction. Each time an event becomes true, its corresponding reaction is executed once. The event is a user-defined state that the I/O unit can recognize. The defined state can be a combination of values, inputs, and outputs.

On a digital multifunction I/O unit, for example, any pattern of input and output states (on and off) can constitute an event. On an analog I/O unit, an event could occur when an input channel attains a reading greater than a preset value. Examples of reactions include turning on or off a set of outputs, ramping an analog output, and enabling or disabling other event/reactions.

The predefined communications watchdog timer at the I/O unit is another example of the event/reaction concept. When active, this built-in event/reaction will change the state of an output channel after communication with the controller fails.

Event/reactions are stored in each I/O unit. They are scanned continuously in alphanumeric order (just as they appear in the Configurator) as soon as power is applied to the I/O unit. Since each I/O unit can be configured with up to 256 event/reactions, complex tasks and sequences can be performed.

WHY USE EVENT/REACTIONS?

- To reduce communication overhead between the I/O unit and the Mistic controller.
- To distribute control logic sequences to the I/O unit rather than concentrating them in the Mistic controller.
- To handle high-speed logic functions local to an I/O unit.
- To increase the execution speed of a program in the Mistic controller.
- To simplify overall control strategy.

TYPICAL APPLICATIONS FOR EVENT/REACTIONS:

- Motor-starting logic
- Drum sequencers
- Alarm enunciation
- Analog biasing
- Power-up sequencing
- Monitoring emergency stop buttons (notifying the Mistic controller when pressed)
- Monitoring analog inputs (notifying the Mistic controller if inputs fall outside acceptable limits)
- Auto seeking of backup communication paths

WHAT CAN BE CONFIGURED AS AN EVENT?

Digital Multifunction I/O Unit Events

Communication Watchdog Timeout
Counter >= Value
Counter <= Value
Quadrature >= Value
Quadrature <= Value
Frequency >= Value
Frequency <= Value
Totalize ON >= Value
Totalize OFF >= Value
ON Pulse >= Value
OFF Pulse >= Value
Period >= Value
MOMO Match

Analog I/O Unit Events

Communication Watchdog Timeout
Analog Input >= Value
Analog Input <= Value
Analog Output >= Value
Analog Output <= Value

In these events, VALUE refers to a setpoint supplied by the user. Analog inputs and outputs can compare the current reading as well as the average, peak, lowest, or totalized readings against a value.

Both digital and analog I/O units have a watchdog timeout event. A watchdog timeout value can be set such that, if communication is lost for a time greater than this value, a corresponding reaction will be executed. This can be useful in situations requiring an orderly shutdown of equipment should communication between the Mistic controller and the I/O unit be lost.

The MOMO (Must On, Must Off) MATCH event in the digital multifunction I/O unit is used to define an event based on a specified input and/or output pattern. MOMO MATCH compares the inputs and outputs on the I/O unit to a pattern entered by the user. As soon as the pattern matches the current state of the I/O unit channels, the event becomes true. The MOMO MATCH event allows the user to specify On, Off, or DON'T CARE for each input or output channel. This event is very useful for identifying emergency conditions or implementing basic combinational logic, such as an AND gate.

WHAT CAN BE CONFIGURED AS A REACTION?

After an event has occurred, a reaction is executed once. The following reactions can be performed in response to an event:

Digital Multifunction I/O Unit Reactions

None
Enable Scan for Events
Disable Scan for Events
Disable Scan for All Events
Set MOMO Outputs
Start ON Pulse
Start OFF Pulse
Start Counter
Stop Counter
Clear Counter/Timer
Clear Quadrature Counter
Read and Hold Counter Value
Read and Hold Quadrature Value
Read and Hold Totalize ON Value
Read and Hold Totalize OFF Value
Read and Hold ON Pulse Value
Read and Hold OFF Pulse Value
Read and Hold Period Value
Read and Hold Frequency Value

Analog I/O Unit Reactions

None
Enable Scan for Event
Disable Scan for Event
Disable Scan for All Events
Read and Hold Analog Input Data
Read and Hold Analog Output Data
Activate PID Loop
Deactivate PID Loop
Set PID Setpoint
Set Analog Output
Ramp Analog Output to Endpoint

One of the most powerful features of event/reactions is their ability to start and stop one another. This feature allows dynamic restructuring of the control logic running at the I/O unit. This is accomplished using the reactions ENABLE SCAN FOR EVENT, DISABLE SCAN FOR EVENT, and DISABLE SCAN FOR ALL EVENTS.

Both analog and digital I/O units have read-and-hold reactions. These reactions are used to capture a count, period, analog value, or frequency at the moment the event occurs and store it in a hold buffer for later retrieval by the Mistic controller. Each event/reaction has a dedicated hold buffer.

SIMPLE EVENT/REACTION EXAMPLE

As an example of an event/reaction, let's build a motor controller. It consists of two inputs and one output on the same digital multifunction I/O unit. The two inputs are wired to two momentary push buttons that are normally open: START MOTOR and STOP MOTOR. The output, called MOTOR RUN, is connected to a motor starter.

The operation of the motor starter is simple. When the START MOTOR button is pressed, the motor starts and remains on until the STOP MOTOR button is pressed.

To build the logic for this example, two event/reactions are required. The first watches the START MOTOR button (event) and turns on the MOTOR RUN output if the button is pressed (reaction). The second watches the STOP MOTOR button and turns off the MOTOR RUN output if the button is pressed.

Event	**Reaction**
1. START MOTOR (input changes from off to on)	Turn on MOTOR RUN output
2. STOP MOTOR (input changes from off to on)	Turn off MOTOR RUN output

Follow these steps to actually create this example event/reaction:

1. Launch the Cyrano Configurator.
2. From the Configure menu, select I/O Point. Create two digital inputs (START MOTOR and STOP MOTOR) and one digital output (MOTOR RUN) on a digital multifunction I/O unit (assuming one has already been configured).
3. From the Configure menu, select Event/Reaction.
4. From the Select I/O Unit Type dialog box, select DIGITAL MF.
5. Select the digital multifunction I/O unit you are using. If there are no event/reactions on this I/O unit, the message "No Event/Reaction Defined" will pop up. Ignore it.
6. To add the first event/reaction, select ADD in the EVENT/REACTION FOR I/O UNIT dialog box and do the following:

   a. Enter START MOTOR as the name of the event/reaction and press ENTER.

   b. Cursor down to the EVENT TYPE field, leaving intervening fields at their defaults.

   c. Under EVENT TYPE, cursor left to MOMO MATCH and press ENTER.

   d. Cursor down to the START MOTOR input. Cursor left to ON and press ENTER.

   e. Cursor down to ACCEPT and press ENTER.

   f. In the REACTION TYPE field, cursor right four times to SET MOMO OUTPUTS and press ENTER.

   g. Cursor down to the MOTOR RUN output. Cursor left to ON and press ENTER.

   h. Cursor down to ACCEPT and press ENTER.

   i. Press ENTER again to accept this event/reaction.

7. To add the second event/reaction, select ADD in the EVENT/REACTION FOR I/O UNIT dialog box and do the following:

   a. Enter STOP MOTOR as the name of the event/reaction and press ENTER.

   b. Cursor down to the EVENT TYPE field, leaving intervening fields at their defaults.

   c. Under EVENT TYPE, cursor left to MOMO MATCH and press ENTER.

   d. Cursor down to the STOP MOTOR input. Cursor left to ON and press ENTER.

   e. Cursor down to ACCEPT and press ENTER.

   f. In the REACTION TYPE field, cursor right four times to SET MOMO OUTPUTS and press ENTER.

   g. Cursor down to the MOTOR RUN output. Cursor left to OFF and press ENTER.

   h. Cursor down to ACCEPT and press ENTER.

   i. Press ENTER again to accept this event/reaction.

8. Use the Cyrano Debugger to download and run this strategy to activate these event/reactions.

ENHANCEMENTS

You can enhance the event/reactions in the previous example by:

- requiring the opposite input to be off;
- requiring the MOTOR RUN output to be in the opposite state.

QUESTIONS AND ANSWERS

Q. Where are event/reactions defined?

A. In the Cyrano Configurator.

Q. How do event/reactions get sent to the I/O unit?

A. They are automatically sent to the Mistic controller by the Debugger during download.

After selecting RUN, all event/reactions are forwarded to their respective I/O units during I/O unit initialization.

Q. How fast do event/reactions execute?

A. That depends on how many there are and how often the I/O unit is polled for data.

A typical answer is 0.5 milliseconds. The maximum possible delay would be 5 milliseconds if all 256 event/reactions were in use.

Rapid polling of the I/O unit will significantly increase these times.

Q. Can an I/O unit notify the Mistic controller when certain events occur?

A. Yes. An event/reaction can also trigger an interrupt wired to the Mistic controller.

Q. Can *each* event/reaction be configured to notify the Mistic controller of an event?

A. Yes.

Q. Does the reaction keep occurring as long as the associated event is True?

A. No. The reaction occurs once each time the specified event status changes from False to True.

Q. Does the order of event/reaction execution matter?

A. Not usually. Only if subsequent event/reactions rely on the results of previous event/reactions.

Keep in mind that event/reactions execute in alphanumeric order (just as they appear in the Configurator).

Q. Are event/reactions permanently stored at the I/O unit?

A. Not automatically. Unless they are stored in Flash EEPROM, event/reaction definitions will be lost if the I/O unit loses power. See "How to Store Event/Reactions in Flash EEPROM at the I/O Unit" on the following page for more information.

Q. Can event/reactions be individually started and stopped by the program, by the Debugger, and by each other?

A. Yes.

Q. Can the event criteria be changed on the fly by the program?

A. Yes.

Q. Are all reactions latched?

A. Yes. Every reaction is maintained regardless of the state of the event.

Q. Can the same event be used in multiple event/reactions?

A. Yes, unless the event references a counter. In these cases, use the reaction as the event for subsequent related event/reactions to guarantee reliable performance.

Q. Can counts, analog values, etc., be captured as the reaction?

A. Yes. There is a hold buffer for each event/reaction specifically for this purpose.

Q. Which is more important on the Cyrano Debugger event/reaction screen, the XVAL or the IVAL?

A. Since event/reactions occur entirely at the I/O unit, the XVAL is the most important because it shows the current status of the event/reaction. The IVAL *may or may not be current,* since it is updated to match the XVAL only when a read or write is done to an enabled I/O point by the program in the Mistic controller. Do not be concerned when the IVAL does not match the XVAL, since this means only that the program is not reading from or writing to the I/O point in question.

HOW TO USE THE INTERRUPT CHART TO HANDLE REACTIONS THAT GENERATE AN INTERRUPT

WHY USE THE INTERRUPT CHART?

- To be promptly notified of critical events that occur at the I/O unit.
- To allow an event on one I/O unit to quickly cause a reaction on another I/O unit using logic in the INTERRUPT chart as the gateway. (Note that reactions should be configured to occur at the I/O unit whenever possible for maximum speed and efficiency.)

FOLLOW THIS PROCEDURE:

- Be sure to wire the interrupt lines from remote I/O units to the Mistic controller.
- Use GENERATING INTERRUPT? to determine which I/O units are generating an interrupt.

FOR EACH I/O UNIT THAT IS GENERATING AN INTERRUPT, SEQUENTIALLY PERFORM THE FOLLOWING:

1. Use CLEAR I/O UNIT INTERRUPT.
2. Use HAS EVENT OCCURRED? to determine which event/reaction(s) caused the interrupt.
3. For each event that occurred, use CLEAR EVENT LATCH.
4. React to each event as desired.
5. IMPORTANT! Be sure to check *every* event that may have caused the interrupt. There may have been multiple events.

HOW TO STORE EVENT/REACTIONS IN FLASH EEPROM AT THE I/O UNIT

Event/reactions are *not* automatically stored at the I/O unit. This means that after a power failure at the I/O unit, all event/reactions will be lost unless they were written to Flash EEPROM via the Debugger or by program command.

To store event/reactions in Flash EEPROM, do the following:

1. From the Debugger, download and run your strategy.
2. From the Controller menu, select Clear/Store I/O Unit Cfg, select the I/O unit, and click STORE. A confirmation message will appear; click YES to confirm the action.

This will store the first 32 event/reactions in Flash EEPROM. More event/reactions can be stored by replacing the socketed Flash EEPROM with a higher-capacity chip.

HOW TO REMOVE EVENT/REACTIONS PREVIOUSLY WRITTEN TO FLASH EEPROM AT THE I/O UNIT

- Using the Debugger, download a program to the Mistic controller (but don't run it!) with the specified I/O unit configured but with *no* event/reactions defined.
- From the Controller menu, select Turn Reset I/O On.
- *Now* click RUN. This will clear the RAM at the I/O unit.
- From the Controller menu, select Clear/Store I/O Unit Cfg, select the I/O unit, and click STORE. A confirmation message will appear; click YES to clear the EEPROM.

HOW TO CHANGE EVENT CRITERIA ON THE FLY FROM THE MISTIC CONTROLLER

Use SEND/RECEIVE PORT W/CRC to send special commands to Mistic I/O units, as detailed in the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270). Many additional event/reaction control features are also available, such as CHANGE ANALOG >= EVENT LIMIT. Consult the Opto 22 BBS for information on these advanced control features.

Tip: You must use ENABLE INTERRUPT ON EVENT (if using interrupts) followed by ENABLE SCAN FOR EVENT immediately after any change to event criteria.

LOGICAL OVERVIEW

WHAT IS LOGICAL TRUE?

True is represented by any non-zero value. Cyrano always uses -1 (all 32 bits on) to indicate True in an integer variable.

A digital input or output that is on will return a True (-1). A True or any non-zero value sent to a digital output will turn it on. Any value other than zero can be used to indicate True.

For individual bits within an integer variable, bits that are set (1) indicate True values. Bits that are cleared (0) indicate False values.

For condition blocks, if *all* the conditions within the block are True, the T exit will be taken, otherwise the F exit will be taken.

Tip: Some programs that communicate with the Mistic controller use 1 rather than -1 for logical True. This is only a problem when such programs read Boolean values from the Mistic controller. An easy way to convert a -1 Boolean result to a 1 is to use TAKE ABSOLUTE VALUE OF.

What is logical False?

False is represented by a value of zero.

For individual bits within an integer variable, bits that are cleared (0) indicate False values.

For condition blocks, a False result means the F exit will be taken, otherwise the T exit will be taken.

WHAT TYPES OF VALUES DO LOGICAL OPERATIONS AND CONDITIONS WORK WITH?

Logical operations and conditions work with integers, individual bits within an integer, a single digital I/O channel, or a group of digital I/O channels (a digital I/O unit). All values are treated as Boolean; that is, they are always either True or False, on or off.

CAN FLOATS BE USED IN LOGIC?

Yes. However, integers are strongly recommended whenever any bits are referenced. Since Cyrano does not permit bits in a float value to be altered, float values must be converted to integers before evaluation. The result of a Boolean operation is always an integer value of either 0 or -1.

See the Mathematical Overview for further information on integers and floats.

WHAT IS A MASK?

A mask is an integer variable or constant with one or more specific bits set. These bits define a set of bits for other operations to work on.

For example, a mask of 255 (the eight least significant bits set) is used with BIT AND either to keep the value in the least significant byte of an integer variable, or to force the 24 most significant bits to zero.

HOW ARE MULTIPLE ENTRIES IN CONDITION BLOCKS EVALUATED?

All the conditions in the block are evaluated and ANDed together. In other words, they all must evaluate to True for the result of the condition block to be True. All entries are evaluated, even if the first entry is False.

MATHEMATICAL OVERVIEW

WHAT IS AN INTEGER?

In Cyrano an integer is a 32-bit signed number ranging from -2,147,483,648 to 2,147,483,647 (roughly ±2 billion). An integer can only be a whole number (-1, 0, 1, 2, 3, etc.). In other words, integers do not include a decimal component.

A special feature of integers is that when all 32 bits are on, the value is -1, since negative numbers are represented in twos complement form. When all 32 bits are off, the value is 0.

A digital I/O unit is considered a 16-bit unsigned integer. For example, if the status of all 16 channels of a digital I/O unit is copied to an integer using DO BINARY READ, the lower 16 bits (bits 15 to 0) will contain an exact image of the status of all 16 channels whether they are inputs or outputs. The upper 16 bits of the target integer are not used.

HOW ARE INTEGER BITS NUMBERED?

The 32 bits are numbered left to right, 31 to 0, as follows:

Byte 3	Byte 2	Byte 1	Byte 0
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00

WHAT IS A FLOAT?

In Cyrano a floating point number (or simply “float”) is a 32-bit IEEE single-precision number ranging from $\pm 3.402824 \times 10^{-38}$ to $\pm 3.402824 \times 10^{38}$.

Note that this format guarantees only about six and a half digits of significance in the mantissa. Because of this, mathematical operations involving floats with seven or more significant digits may incur errors after the sixth significant digit. For example, a float-to-integer conversion of 555444333.0 yields 555444416 (note the error in the last three digits).

All analog values read from an I/O unit are floats.

CAN INTEGERS AND FLOATS BE MIXED IN THE SAME COMMAND, AND CAN THEY BE CONVERTED FROM ONE TO THE OTHER?

Yes. Type conversion is automatic. An analog value read from an I/O unit and put into an integer is converted from float to integer automatically.

However, to maintain the integrity and accuracy of a numeric type (float or integer), keep all item types the same. For example, use MOVE to copy an integer value to a variable float when exclusively float calculations are desired.

CAN ROUNDING BE CONTROLLED?

Yes. To round the result of a float calculation, use MOVE to copy the float result to a variable integer. Note that 1.5 rounds up to 2, 1.49 rounds down to 1.

To round down only, divide an integer by an integer (5/3 = 1).

WHAT IS A RADIAN?

A radian is a natural unit of angular measurement equal to 57.29578 degrees of arc. Note that 2p radians = 360 degrees and 2p f = angular frequency in radians per second (represented by the Greek letter omega, ω), where f = frequency in Hz.

PID OVERVIEW

The Opto 22 PID algorithm is an *interacting* type with a *reverse* output. This means that

1. The gain, integral, and derivative *all* interact, and
2. The output increases as the input decreases (*reverse* output).

The *reverse output* mode is used for "pump-up" control, such as maintaining level, pressure, and flow as well as heating. For cooling or "pump-down" control, *direct* output is required. To switch to *direct,* simply reverse the sign of the gain (e.g., a gain of 1.28 would become -1.28). Note that this is *not* negative gain. The minus sign only serves to change the type of PID output from *reverse* to *direct*.

THEORY OF OPERATION

Gain (P) — For those familiar with the term "proportional band," gain is simply the inverse. Gain acts directly on the *change in error* since the last scan (error is the setpoint minus the input value in engineering units). Therefore, in the case of steady-state error (i.e., change in error = 0), gain alone has no effect on the output. For this reason, gain cannot be used alone. Gain is also used as a multiplier on the integral and derivative.

Opto 22 uses gain much as it is used in the Honeywell "type A" PID and the Bailey "error input" type PID. Higher gain results in increased output change. Too much gain results in output oscillation. Too little gain results in very slow performance.

Keep in mind that *a gain value other than zero is required.*

Integral (I) — This term acts only on the current error. It is used to reduce the current error to zero. Note that during steady-state conditions, integral multiplied by current error multiplied by gain is the only thing affecting the output. The larger the integral value, the larger the output change.

Keep in mind that *a positive integral value is required*.

Derivative (D) — This term acts only on the change in slope of the input signal. Its purpose is to anticipate where the input will be on the next scan based on a change in the rate of change of the input value. In other words, it changes the output as the input gets near the setpoint to prevent overshooting or undershooting.

Derivative is used in "feed forward" applications and in systems that have a lot of dead time. Its action type is unlimited (i.e., it has no filtering). If the input signal is noisy and the derivative value is greater than zero, the input value must be filtered. (See step 4 on page 1-22 for details.) If the slope of the input signal has remained unchanged for the last two scans, the derivative will have no effect.

Judging by the change in direction of the input, the derivative contributes an appropriate value to the output that is consistent with where the input will be at the next scan if it continues at its current rate of change.

Integral-Derivative Interaction — Integral and derivative can be trying to move the output in opposite directions. When this is the case, the derivative should be large enough to overcome the integral (since the derivative is "looking ahead" based on the change in slope, it has a bigger picture than the integral does).

This can be observed when the input is below the setpoint and is rising fast. The integral tries to increase the output (which will only make things worse), while the derivative tries to decrease the output because, at the current rate of change of the input, there will be an input overshoot if the output is increased. Therefore, the derivative needs to be large enough to counteract the integral when necessary.

SUGGESTED TUNING METHOD

1. Determine the scan rate in seconds.

This is the most important step. If the scan rate is not matched to the system that the PID is to control, tuning the PID will be difficult if not impossible. The scan rate *must* be greater than the loop dead time. The dead time is the time it takes the input to begin changing in response to an output step change.

To determine the dead time, put the PID output in manual mode, then set the output somewhere around midrange. After the loop has achieved a steady state, change the output by at least 10% of its span. Measure the time (in seconds) that it takes the input to *start* responding to the change. This is the dead time.

For conservative tuning, set the scan rate greater than or equal to the dead time. A quick value to use for the scan rate is 1.5 times the dead time. For aggressive tuning, set the scan rate to one-third of the dead time.

Tip: If the scan rate is too short, it will be impossible to tune the loop because the PID calculation is using an input value that has not had a chance to change yet in response to the last output change.

Note: None of the following applies to aggressive tuning!

2. Determine the gain.

The recommended initial gain is the percent of output span change divided by the resulting percent of input span change.

Let's say that the output, which is scaled 0–100, was changed from 50 to 60, a 10% change. As a result, the input, which is scaled 500–1500, changed from 600 to 678, a 7.8% change. We can calculate the gain as 10/7.8 = 1.28. This gain causes a 7.8% input change to result in a 10% output change, which represents an equivalent gain of 1.0. In this example we would set the gain greater than or equal to 1.28.

Tip: Remember, to reverse the action of the PID output, make the gain negative.

Tip: Limit changes to the initial gain value to between +20% and -0% while fine tuning.

Tip: If you wish to change just the gain and leave the integral and derivative as they are, change their values proportionally to the change in gain. For example, if you double the gain, cut both the integral and derivative values in half.

3. Determine the integral.

The integral is required and must be greater than zero. The recommended initial integral setting is 60 divided by the scan rate determined above. For example, if the scan rate is 5 seconds, the integral would be 60/5 = 12. In this example we would set the integral less than or equal to 12 but greater than 0.

Tip: Limit changes to the calculated integral value to between +0% and -50% while fine tuning.

Tip: Reduce the integral value if the output is overshooting during steady-state error conditions.

4. Determine the derivative.

The derivative is very useful in loops with a long dead time and long time constants. Set it to zero to disable it. The recommended initial derivative setting is 1/integral, giving it equal ability to the integral in affecting output. For example, if the integral is 12, the derivative would be 1/12 = 0.0833. In this example we would set the derivative to 0.0833.

Tip: Limit changes to the calculated derivative value to ±50% while fine tuning.

Tip: Increase derivative value to improve "look-ahead" performance.

Tip: Activate input filtering if the input signal is the least bit noisy. See the following page for details.

5. Set the output lower and upper clamps.

This is *particularly important* if the device controlled by the output signal has "dead areas" at either end. For example, say the output is scaled 0–10. It is connected to a valve that begins to open at 1.25 and is "effectively" fully open at 5.75 (even though it may only be 70% open). Set LOWER CLAMP to 1.2 (valve closed) and UPPER CLAMP to 5.75 (valve effectively fully open). This prevents reset windup, potentially resulting in dramatically improved control when the output value has reached either limit and has to suddenly reverse direction.

6. Set the maximum change rate of the output.

The MAX CHANGE RATE can be ignored, since it defaults to 100% per scan. If you wish to limit the output rate of change, set MAX CHANGE RATE to 10% or so to start. This setting would limit the output rate of change to 100% in 10 scan rate periods.

Tip: The output can be preset or changed at any time by the user or by the user program. For example, if you have determined that the output should start at 40% whenever the system is activated, simply set the PID output (or the analog channel output) to this value under program control.

Tip: The factory default causes the setpoint to track the input when the PID is in manual mode. This means that the setpoint will be altered. If this is undesirable, disable setpoint tracking so that when the PID is in manual mode, the setpoint will not be altered. Use DISABLE PID MAN. SETP. TRACK:, specifying the appropriate PID. (Note: As of February 1995, DISABLE PID MAN. SETP. TRACK: is an "external" command that requires library support. Consult the Opto 22 BBS for details.)

INPUT FILTERING

If the input signal is noisy, you may want to activate input filtering. Follow the procedures below:

1. Use SET ANALOG FILTER WEIGHT, specifying the appropriate analog input channel. Use a filter weight value of less than 10 times the scan rate. Otherwise, the loop cannot be tuned.
2. Use START PID AVERAGE: to tell the PID to switch to the filtered input value.
3. Use WRITE TO EEPROM: to save the filter weight and the input type (current or average) to EEPROM at the analog brick. This ensures that the PID will automatically use these values.

Note: As of February 1995, START PID AVERAGE: and WRITE TO EEPROM: are "external" commands that require library support. Consult the Opto 22 BBS for details.

OPTO 22'S PID FORMULA

Change in output =	Gain *	
	((Error - Last Error) +	(P)
	(Integral * Time * Error) +	(I)
	((Derivative/Time) * (Error - (2 * Last Error) + Oldest Error)))	(D)
where	Error is (Setpoint - Input) in engineering units	
	Time is (Scan Rate/60), which results in time in minutes	

All values are in engineering units.

The change in output calculated above is added to the existing PID output. If the input span and the output span are different, the change is normalized and then added to the output. This is accomplished by converting the change to a percentage of input span. The same percentage of output span is then added to the output.

Tip: If the input engineering units are negative, the output may move in the direction opposite that desired. If this happens, reverse the sign of the gain.

Tip: The input *must not be bipolar*. An input range of -10 to +10, for example, will not work. Values such as -300 to -100, -100 to 0, and 0 to 100 are acceptable. If an application has a bipolar input range, it will have to be rescaled to a generic range, such as 0 to 100. The setpoint range will then have to match this generic input range.

STRING OVERVIEW

All numbers are decimal unless otherwise stated.

WHAT IS A STRING?

A Cyrano string can be likened to a string of beads. Each bead represents a single character (such as "A" or "1"). The string to which the beads are attached represents the place where the characters are "strung together." Each string has a user-defined name and width and is called either a string variable or a string table.

Characteristics of strings include the following:

- Strings are always referred to by name (and, if in a table, by index).
- Each character is represented by one byte.
- Each character is represented by its ASCII code (0 to 255).
- The relationships of characters or the meaning of a particular character is always defined by the user.
- Although a string may appear to contain numeric values, it does not. Digits "0" through "9" are characters just as much as "A" through "Z"; they do not represent numeric values.

- A string containing no characters is sometimes referred to as an "empty string."
- Strings are frequently used in serial communication as a container for moving numeric characters from one location to another.

To illustrate, let's look at the number 22. This is a decimal number representing a quantity of 22. The number 22 can be represented in a string in several ways:

- As "22": two character 50's (the ASCII code for "2" is 50). Use APPEND STRING to append "22" or use APPEND CHARACTER to append character 50 twice.
- As "16": a character 49 ("1") and a character 54 ("6") (16 represents the hex value of 22). Use APPEND STRING to append "16" or use APPEND CHARACTER to append characters 49 and 54.
- As "■": a character 22. Use APPEND CHARACTER to append character 22.

Note that the string representation of the number 22 is no longer a number. It is simply one or two ASCII characters. *The string representation of a number must be converted to a numeric value if it is to be used in calculations.* Several CONVERT commands are available for this purpose.

WHAT IS THE DIFFERENCE BETWEEN STRING LENGTH AND WIDTH?

Length is not the same as width. Width is the *maximum length* a string can be; length is the *actual number of characters* contained in the string. A string with a width of 100 may currently be empty, which means its length is actually zero. A string with a width of 10 containing the characters "Hello " has a length of six (five for "Hello" and one for the space after the "o"). Although a string's length may change dynamically as the string is modified by the program, its width remains constant.

For applications requiring strings wider than the width supported by Cyrano (127), there are several options:

- Use several strings to hold the data.
- Use string tables, which are arrays of strings.
- Use numeric tables, described below.

CAN NUMERIC TABLES BE USED AS AN ALTERNATIVE TO STRINGS?

Yes. Since a string is nothing more than a sequence of characters, you can store a "string" in a numeric table, with each table element holding a character.

The advantages of using numeric tables for strings are:

- A numeric table can store strings of any size.
- A numeric table can access binary data easier in the Debugger.

The disadvantages are:

- Memory usage is three times greater.
- No string conversion functions are available for numeric tables. An intermediate temporary string would be required to utilize string commands for these tables.

HOW ARE STRINGS HANDLED DURING MULTITASKING?

Although string commands are completed before the current task loses its time slice, it is important to note that a string may require more than one time slice to be constructed if multiple steps are used to construct it.

For example, if a string is being constructed with two steps (such as MOVE STRING "Hello" and APPEND STRING " World"), after the first step a task switch *could occur* such that another chart looking at the resulting string might see "Hello" rather than "Hello World."

If another chart is relying on a completed string, an integer should be used as a flag to indicate whether the string is completely built. This is illustrated in the following example:

1. The variable string MSG$ is empty and is about to be built by the chart BUILD STRING. MSG$ is intended to be printed by chart USE STRING. In this example, BUILD STRING builds "The temperature is 56.77."
2. Chart BUILD STRING uses MOVE STRING to put "The temperature is " into MSG$.
3. A task switch now occurs, and the chart USE STRING gets control. If this chart looks at MSG$, it would see a partially constructed string containing "The temperature is " but not the actual temperature.
4. Chart BUILD STRING gets control again after a task-switch and completes its work on MSG$ by adding the actual temperature. It then sets an integer variable called STRING COMPLETE to True.
5. Chart USE STRING sees a non-zero value in STRING COMPLETE and now prints the completed string MSG$. It then clears the flag by resetting STRING COMPLETE to False, thus signalling the BUILD STRING chart that it can begin building another string.

HOW CAN BINARY BYTES BE VIEWED IN THE CYRANO DEBUGGER?

Some strings may contain non-printable ASCII characters, such as an STX or a carriage return. To see these characters in hex when using the Debugger, use the CTRL-C key (which is normally used to view the communications buffer). In this "CTRL-C" window, the field RBUF shows the string contents. The first bytes are communication header bytes and may be ignored (the first byte is packet length, the second is an error byte where 0 means no error, and the third is an ID byte used only when ARCNET is in use). For example, the RBUF for the string "Hello" would contain the following: 06 00 nn 48 65 6C 6C 6F, where the first three bytes contain communication info and the "48" is the first letter of "Hello."

SHOULD QUOTES BE USED WITHIN STRINGS?

Double quotes cannot be entered for strings in the Cyrano Configurator. For example, if using MOVE STRING to put "Hello" into variable MY STRING, do not type in the beginning or ending double quotes. You may use single quotes around text, although they are not required.

HOW CAN A CONTROL CHARACTER BE ADDED TO A STRING?

You can input control characters by turning on the num lock on your keyboard and inputting the appropriate control codes through the numeric keypad (at the far right of most keyboards).

For example, to add a CR, press and hold ALT as you type 16 on the keypad, then press and hold ALT as you type 13 on the keypad. To add a DLE, press and hold ALT as you type 16 on the keypad, then press and hold ALT as you type 13 on the keypad.

SAMPLE STRING VARIABLE

- Declared Name: STRING 1
- Declared Width: 22
- Maximum Possible Width: 127 (may increase in future versions of Cyrano)
- Bytes of Memory Required: Declared Width + Name Length + 46 = 22 + 8 + 46 = 76

Width is 22

STRING 1																						

Length is 0

Strings are referred to by their name. The above string is empty, giving it a length of zero.

Later, during program execution, seven characters were added to STRING 1, increasing its length to 7:

Width is 22

STRING 1	O	P	T	O		2	2															

Length is 7

SAMPLE STRING TABLE

- Declared Name: PROMO MESSAGES
- Declared Width: 26
- Maximum Possible Width: 127 (may increase in future versions of Cyrano)
- Declared Length (Size): 5
- Maximum Possible Length (Size): 65,535
- Bytes of Memory Required: (Declared Width + 12) x (Declared Length (Size) + 1) + Name Length + 80 = (26 + 12) x (5 + 1) + 14 + 80 = 322

Width is 26

Index	O	P	T	O		2	2																			
Index 1	M	I	S	T	I	C		C	O	N	T	R	O	L	L	E	R	S								
Index 2	L	e	a	d	i	n	g		t	h	e		w	a	y	!										
Index 3	T	i	g	h	t	l	y		I	n	t	e	g	r	a	t	e	d		M	M	I				
Index 4	T	o	p	-	N	o	t	c	h		T	e	c	h		S	u	p	p	o	r	t				
Index 5	P	r	o	d	u	c	t		o	f		t	h	e		Y	e	a	r		A	w	a	r	d	!

A string table is a collection of strings. Each string is referred to by the name of the table it is in and the index at which it can be found.

The *width* of each string in the table is the same. The *length* of each string can vary from 0 to the width.

The length (size) of a string table is the number of strings it can hold + 1.

STRING DATA EXTRACTION EXAMPLES

To extract various pieces of information from a string, use GET SUBSTRING. Consider the following example:

	1	2	3	4	5	6	7														
STRING 1	O	P	T	O		2	2														

One way to get two separate pieces of information from this string is to get characters 1–4 and then get characters 6 and 7, as shown in the following examples:

GET SUBSTRING

	STRING 1	*variable string*
Start At	1	*constant integer*
Number Of	4	*constant integer*
Move To	SUB$ 1	*variable string (width = 5)*

Results in:

	1	2	3	4	5
SUB$ 1	O	P	T	O	

GET SUBSTRING

	STRING 1	*variable string*
Start At	6	*constant integer*
Number Of	2	*constant integer*
Move To	SUB$ 2	*variable string (width = 5)*

Results in:

	1	2	3	4	5
SUB$ 2	2	2			

STRING BUILDING EXAMPLE

Strings are assembled using MOVE STRING, APPEND CHARACTER, and APPEND STRING. Consider the following original string and the examples that follow:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
STRING 1	M	I	S	T	I	C		2	0	0		P	R	O	C	E	S	S	O	R	

MOVE STRING

From	“Opto”	*constant string*
To	STRING 1	*variable string*

Results in: *(note that MOVE STRING erased the previous contents of the string)*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
STRING 1	O	P	T	O																	

|Length is 4|

APPEND CHARACTER

From	32	*constant integer (represents a space)*
To	STRING 1	*variable string*

Results in: *(note the space character in position 5)*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
STRING 1	O	P	T	O																	

|– Length is 5 –|

APPEND STRING

From	“22”	*constant string*
To	STRING 1	*variable string*

Results in:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
STRING 1	O	P	T	O		2	2														

|——— Length is 7 ——|

APPEND CHARACTER

From	13	*constant integer (carriage return)*
To	STRING 1	*variable string*

Results in:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
STRING 1	O	P	T	O		2	2	¶													

|——— Length is 8 ———|

Comparison to Visual Basic and C

Table 1-3 lists Cyrano string commands and their equivalents in Visual Basic and C.

Table 1-3: Visual Basic and C Equivalents of Cyrano Commands

Cyrano Command	Visual Basic	C

APPEND CHARACTER	S$ = S$ + Chr$(MyChar%)	i = strlen(str);
		str[i] = 0;
		str[i] = 0;

APPEND STRING	S$ = S$ + "Hello"	strcat(str, "Hello");
CONVERT HEX STRING TO NUMBER	1% = "&H" + S$	sscanf(str,"%x",&iNum);
CONVERT NUMBER TO HEX STRING	S$ = Hex$(1%)	sprintf(str,"%x",iNum);
CONVERT NUMBER TO STRING	S$ = CStr(1%)	sprintf(str,"%d",iNum); sprintf(str,"%f",fNum):
CONVERT STRING TO NUMBER	1% = CInt(S$)	sscanf(str,"%d",&iNum); iNum = atoi(str);
GET NTH CHARACTER	MyByte% = ASC (MID$(Str$,n%,1))	MyByte = str[n];
GET STRING LENGTH	MyLENGTH% = LEN(Str$)	iLEN = strlen(str);
GET SUBSTRING	SubStr$ = MID$(Str$,i,n)	strncpy(subStr,&str[i]); subStr[n] = "\0";
MOVE STRING	STR$ = "Hello"	strcpy(strDest,"Hello");
TEST EQUAL STRINGS	Equal% = (STR$ = "Hello")	i = strcmp(str1,"Hello");
STRING EQUAL	if STR$ = "Hi" then...	if(!strcmp(str1,"Hi"))
EQUAL TO STRING	if STR$(n%) = "Hi" then...	if(!strcmp(str1[n],"Hi")

CONVERT-TO-STRING EXAMPLES

Table 1-4 compares the five "convert-to-string" commands. These commands are typically used when printing a number to a port. This table shows examples of how various parameters affect the string. Note the following:

- "Value" indicates the numeric value to be converted.
- "Dec" indicates the number of digits to the right of the decimal point (not applicable to hex conversions).
- The "*" in strings indicates an overflow where the whole-number portion of the resulting string is longer than its allocated space.

- Some commands add leading spaces to achieve the specified length. These spaces are indicated with underline ("_") characters.
- Floats (if used) are automatically rounded to integers before conversion *except* when using CONV. FORMATTED # TO HEX STR.

Table 1-4: Convert-to-String Commands

Command Parameters			"Convert-to-String" Commands				
Value	Dec	Len	CONV. FORMATTED # TO HEX STR.	CONV. FLOATING POINT # TO STR.	CONVERT NUMBER TO HEX STRING	CONVERT NUMBER TO STR. FIELD	CONVERT NUMBER TO STRING
Float 16.0	1	4	417FFFFF (Len 8 req'd for floats)	16.0	10	1.6e+01	1.6e+01
Float 16.0	2	4	417FFFFF (Len 8 req'd for floats)	**** (too big)	10	1.6e+01	1.6e+01
Float -16.0	1	4	C17FFFFF (Len 8 req'd for floats)	**** (too big)	FFFFFFF0	-1.6e+01	-1.6e+01
Float 1.23	1	4	3F9D70A4 (Len 8 req'd for floats)	_1.2	1	1.23	1.23
Float 12.3	1	4	4144CCCD (Len 8 req'd for floats)	12.3	C	1.23e+01	1.23e+01
Float 0.0	1	4	00000000 (Len 8 req'd for floats)	_0.0	0	_ _ _0	0
Int 16	1	4	0010	16.0	10	_ _16	16
Int 16	2	4	0010	**** (too big)	10	_ _16	16
Int -16	1	4	FFF0	**** (too big)	FFFFFFF0	_-16	-16
Int 0	1	4	0000	s0.0	0	_ _ _0	0
Int 1000	1	2	E8	** (too big)	3E8	1000	1000

ASCII TABLE

Table 1-5 displays ASCII characters with their decimal and hex values. For characters 0–31, equivalent control codes are also listed; for example, a carriage return (character 13) is equivalent to a CTRL-M (^M). Printable ASCII characters are shown in **bold**. Hex values consist of the column heading appended with the row heading. For example, a "**P**" is character 80 (hex 50 since it's in column 5, row 0), a "**p**" is character 112 (hex 70).

Table 1-5: ASCII Characters with Decimal and Hex Values

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 (1) ^@ NUL	16 ► ^P DLE	32 (1) **space**	48 **0**	64 **@**	80 **P**	96 **`**	112 **p**	128 Ç	144 É	160 á	176 ░	192 └	208 ╨	224 α	240 ≡
1	1 ☺ ^A SOH	17 ◄ ^Q DC1	33 **!**	49 **1**	65 **A**	81 **Q**	97 **a**	113 **q**	129 ü	145 æ	161 í	177 ▒	193 ┴	209 ╤	225 ß	241 ±
2	2 ☻ ^B STX	18 ↕ ^R DC2	34 **"**	50 **2**	66 **B**	82 **R**	98 **b**	114 **r**	130 é	146 Æ	162 ó	178 ▓	194 ┬	210 ╥	226 Γ	242 ≥
3	3 ♥ ^C ETX	19 ‼ ^S DC3	35 **#**	51 **3**	67 **C**	83 **S**	99 **c**	115 **s**	131 â	147 ô	163 ú	179 │	195 ├	211 ╙	227 π	243 ≤
4	4 ♦ ^D EOT	20 π ^T DC4	36 **$**	52 **4**	68 **D**	84 **T**	100 **d**	116 **t**	132 ä	148 ö	164 ñ	180 ┤	196 ─	212 ╘	228 Σ	244 ⌠
5	5 ♣ ^E ENQ	21 § ^U NAK	37 **%**	53 **5**	69 **E**	85 **U**	101 **e**	117 **u**	133 à	149 ò	165 Ñ	181 ╡	197 ┼	213 ╒	229 σ	245 ⌡
6	6 ♠ ^F ACK	22 ▬ ^V SYN	38 **&**	54 **6**	70 **F**	86 **V**	102 **f**	118 **v**	134 å	150 û	166 ª	182 ╢	198 ╞	214 ╓	230 µ	246 ÷
7	7 • ^G BEL	23 ↨ ^W ETB	39 **'**	55 **7**	71 **G**	87 **W**	103 **g**	119 **w**	135 ç	151 ù	167 º	183 ╖	199 ╟	215 ╫	231 τ	247 ≈
8	8 ◘ ^H BS	24 ↑ ^X CAN	40 **(**	56 **8**	72 **H**	88 **X**	104 **h**	120 **x**	136 ê	152 ÿ	168 ¿	184 ╕	200 ╚	216 ╪	232 Φ	248 °
9	9 ○ ^I HT	25 ↓ ^Y EM	41 **)**	57 **9**	73 **I**	89 **Y**	105 **i**	121 **y**	137 ë	153 Ö	169 ⌐	185 ╣	201 ╔	217 ┘	233 Θ	249 •
A	10 ◙ ^J LF	26 → ^Z SUB	42 *****	58 **:**	74 **J**	90 **Z**	106 **j**	122 **z**	138 è	154 Ü	170 ¬	186 ║	202 ╩	218 ┌	234 Ω	250 ·
B	11 ♂ ^K VT	27 ← ^[ESC	43 **+**	59 **;**	75 **K**	91 **[**	107 **k**	123 **{**	139 ï	155 ¢	171 ½	187 ╗	203 ╦	219 █	235 δ	251 √
C	12 ♀ ^L FF	28 ∟ ^/ FS	44 **,**	60 **<**	76 **L**	92 ****	108 **l**	124 **¦**	140 î	156 £	172 ¼	188 ╝	204 ╠	220 ▄	236 ∞	252 ^{n}
D	13 ¶ ^M CR	29 ↔ ^] GS	45 **-**	61 **=**	77 **M**	93 **]**	109 **m**	125 **}**	141 ì	157 ¥	173 ¡	189 ╜	205 ═	221 ▌	237 ∅	253 ^{2}
E	14 ^N SO	30 ▲ ^^ RS	46 **.**	62 **>**	78 **N**	94 **^**	110 **n**	126 **~**	142 Ä	158 ₧	174 «	190 ╛	206 ╬	222 ▐	238 ε	254 ■
F	15 ¤ ^O SI	31 ▼ ^ US	47 **/**	63 **?**	79 **O**	95 **—**	111 **o**	127 ⌂	143 Å	159 ƒ	175 »	191 ┐	207 ╧	223 ▀	239 ∩	255 (1)

(1) = No symbol defined

TIMERS OVERVIEW

Timers are a special type of numeric variable. To create a timer in OptoControl, configure a numeric variable and select the type called "Timer". An OptoControl timer stores elapsed time in units of seconds with resolution of milliseconds. Timers in OptoControl continuously count down to zero.
Up timers are not currently supported, and there are no commands to suspend or continue timers.

PROPERTIES

Type:	Continuous countdown to zero.
Units:	Seconds.
Resolution:	0.001 second.
Range:	0.001 to 4.611686 x 10^{15} seconds.
Numeric Type:	Numeric variable configured as a timer. Internally the timer is a 64-bit integer in units of 0.0005 seconds. For convenience, this value is presented to the user as a float with units in seconds.
Time Sync:	The timer is independent from the controller's clock. Over thousands of seconds, the timer and the controller's clock will not match.
Defaults:	Timer stopped with a value of zero.
Features:	Start and end.
Overhead:	None. The timers do not place any additional load on the CPU.

OPERATION

To start a timer, use the [MOVE] command to put a value greater than zero into the timer. The timer will begin counting down immediately. To force an end to the timing, use the [MOVE] command to put a value of zero into the timer.

The current value of a timer can be viewed at any time from the OptoControl Debug mode. Any OptoControl command that references a numeric variable can be used to access a timer from within an OptoControl strategy.

To determine if the timer is finished, use the condition <TIMER EXPIRED?>. This condition simply tests the timer to see if it is zero. This condition is much faster than using the condition <EQUAL?>
to compare the timer to a value of zero.

NOTES

1. The condition <TIMER EXPIRED?> will be true any time the timer has a value of zero. When using this condition statement, the resolution is 1 msec.
2. When program execution speed is a priority, use the [MOVE] command to put an integer value into the timer (rather than a float) to start the timer. This eliminates the float to integer conversion time.
3. The timing function internal to the controller maintains its accuracy regardless of the value of the timer. However, the resolution when viewing or evaluating the remaining time depends on the amount of time remaining. When viewing the remaining time in a timer from the OptoControl Debug mode or by using commands within an OptoControl strategy, the resolution will depend on the remaining time as follows:

Remaining Time (Seconds)	**Best Resolution** (Seconds)
0 - 9,999	0.001
10,000 - 99,999	0.01
100,000 - 999,999	0.1
1,000,000 - 9,999,999	1.0
values >= 10,000,000	seven digits plus exponent (9.999999 x 10^n).

ANALOG I/O OVERVIEW

Analog Mistic 200 I/O Units constantly update the status of their I/O. Input modules are read every 7 milliseconds and the data held in memory until requested by the host CPU. Output module data is held in memory and output to each module every 50 milliseconds.

OPERATIONS

CHAPTER 2

OVERVIEW

This appendix provides reference data on all Cyrano operation commands.

To locate a command, look it up in the index below or browse through the appropriate command group (Analog Point, Chart, etc.) in this chapter.

INDEX OF OPERATION COMMAND GROUPS

INDEX OF OPERATION COMMANDS

ANALOG POINT OPERATIONS

CALC & SET ANALOG GAIN — Analog Point

Function: To improve the accuracy of an analog input signal or to change its range.

Typical Uses:
- To improve calibration on a temperature input.
- To rescale an input from one range (say, 25–50%) to a range of 0–100%.

Details:
- Reads the current value of a specified analog input and interprets it as the maximum (100%, full-scale) value. Hence, the analog input should always be set to the full-scale value before this command is used.
- Calculates a gain based on the current value that will cause this value to read 100% (full scale).
- Stores the calculated gain in Argument 2 for subsequent use by SET ANALOG GAIN, if desired.
- The calculated gain will be used until power is removed from the I/O unit, or it will always be used if it is stored in permanent memory at the I/O unit.
- The default gain value is 1.0. The valid range for gain is 0.0003 to 16.0.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **CALC & SET ANALOG GAIN**

For	BOILER TEMPERATURE	*analog input*
Move To	GAIN COEFFICIENT	*variable float*

Notes:
- A Cyrano calibration chart could be created to prompt the user to input a “known good” high-scale value. After acknowledgment by the user, the CALC & SET ANALOG GAIN command could be executed. This procedure should only have to be performed once.
- To ensure that the calculated gain coefficient will always be used, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)

Dependencies:
- Always use CALC & SET ANALOG OFFSET before using this command.
- Always set the analog input to the full-scale (100%) value before using this command.
- This command is not supported by high-density analog input cards, such as the G4AIVA and G4AITM.

See Also: CALC & SET ANALOG OFFSET, SET ANALOG GAIN, SET ANALOG OFFSET

CALC & SET ANALOG OFFSET — Analog Point

Function: To improve accuracy of an analog input signal or to change its range.

Typical Uses:
- To improve calibration on a temperature input.
- To rescale an input from one range (say, 25–50%) to a range of 0–100%.

Details:
- Reads the current value of a specified analog input and interprets it as the minimum (0%, zero-scale) value. Hence, the analog input should always be set to the zero-scale value before this command is used. (Note that zero scale on a bipolar input module with a range of -10 VDC to +10 VDC is -10 VDC.)
- Calculates an offset based on the current input value that will cause this value to read 0% (zero scale).
- Stores the calculated offset in Argument 2 for subsequent use by SET ANALOG OFFSET, if desired.
- The calculated offset will be used until power is removed from the I/O unit, or it will always be used if it is stored in permanent memory at the I/O unit.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **CALC & SET ANALOG OFFSET**

For	BOILER TEMPERATURE	*analog input*
Move To	OFFSET	*variable integer*

Notes:
- This command is intended to be used in conjunction with CALC & SET ANALOG GAIN.
- A Cyrano calibration chart could be created to prompt the user to input a “known good” low-scale value. After acknowledgment by the user, the CALC & SET ANALOG OFFSET command could be executed. This procedure should only have to be performed once.
- To ensure that the calculated offset will always be used, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)

Dependencies:
- This command is not supported by high-density analog input cards, such as the G4AIVA and G4AITM.

See Also: CALC & SET ANALOG GAIN, SET ANALOG GAIN, SET ANALOG OFFSET

DISABLE ANALOG POINT — Analog Point

Function: To disable communication between the program in the Mistic controller and an individual analog channel.

Typical Use: To disconnect the program from a specified analog channel for simulation and program testing.

Arguments:

ARGUMENT 1
ANALOG IN
ANALOG OUT

Example: **DISABLE ANALOG POINT**

TANK LEVEL *analog point*

Details:

- All analog point communication is enabled by default.
- This command does not affect the analog channel in any way. It only disconnects the program in the Mistic controller from the analog channel.
- When communication to an analog channel is disabled, program actions have no effect.
- When a program reads the value of a disabled channel, the last value before the channel was disabled (IVAL) will be returned. Likewise, any attempts by the program to change the value of an output channel will affect only the IVAL, not the actual output channel (XVAL).
- Disabling an analog channel while a program is running has no effect on the program.

Notes:

- Disabling an analog channel is ideal for a start-up situation, since the program thinks it is reading an input or updating an output as it normally would be.
- Use the IVAL field in the Debugger to change the value of an analog input.
- Use the XVAL field in the Debugger to change the value of an analog output.

See Also: ENABLE ANALOG POINT

ENABLE ANALOG POINT — Analog Point

Function: To enable communication between the program in the Mistic controller and an individual analog channel.

Typical Use: To reconnect the program to a specified analog channel after simulation or program testing.

Details:
- All analog channel communication is enabled by default.
- This command does not affect the analog channel in any way. It only connects the program in the Mistic controller with the analog channel.
- When communication to an analog channel is enabled, program actions again take effect.
- When a program reads the value of an enabled input channel, the current value of the channel (XVAL) will be returned to the program (IVAL). Likewise, an enabled output channel will update when the program writes a value. The XVAL and IVAL will match at this time.

Arguments:

ARGUMENT 1
ANALOG IN ANALOG OUT

Example: **ENABLE ANALOG POINT**

TANK LEVEL	*analog point*

Notes:
- Use this command to enable an analog channel previously disabled by the DISABLE ANALOG POINT command.

See Also: DISABLE ANALOG POINT

RAMP TO POINT — Analog Point

Function: To change an analog output value to a new value at a constant rate.

Typical Use: To raise or lower oven temperature from point A to point B at a specified rate.

Details:

- When the I/O unit receives this command, it will assume control of the analog output channel.
- Ramping starts from the current output value and proceeds toward the specified endpoint value.
- The ramp rate is specified in engineering units per second. A rate of zero is illegal (returns a queue error 7).
- Updates to the current output value will be made at 50-millisecond intervals.
- If this command is executed while the output is ramping, the ramp rate will be changed.
- If this command is executed too frequently, the output will not get a chance to ramp at all.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	ANALOG OUT
CONSTANT INTEGER	CONSTANT INTEGER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	

Example: **RAMP TO POINT**

Endpoint	SOAK TEMP	*variable float (endpoint value)*
Units/Sec	RAMP RATE	*variable float (rate value)*
For	TEMP CONTROL	*analog output*

Notes:

- To stop the ramp at any time, use MOVE to send the desired “static” value to the analog output channel.
- Use this command only to *change* or *start* the ramp.
- Be sure the analog output value is at the desired starting point before using this command.
- If the output value must be changed, *wait at least 50 milliseconds* before using this command.

Error Codes: Queue error 7 = Value sent to I/O unit is out of range

READ & CLEAR ANALOG MAX VAL — Analog Point

Function: To retrieve the peak value of a specified analog input since its last reading, then reset it to the current value.

Typical Use: To capture the peak pressure over a given period of time.

Details:

- The current value for each channel is read and stored at the I/O unit every seven milliseconds. However, the response time of the input module may be much slower due to smoothing built in to the module. Check the specifications for the module to be used if high-speed readings are required.
- Min and max values are recorded at the I/O unit immediately after the current value is updated.
- Channels without a module installed or with a thermocouple module that has an open thermocouple will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ & CLEAR ANALOG MAX VAL**

From	PRES SENSOR	*analog input*
Move To	MAX KPA	*variable float (max value)*

Notes:

- Use this command to clear the analog max value before actual readings commence.

Dependencies:

- If digital filtering is active (see SET ANALOG FILTER WEIGHT), min and max value detection is derived from the filtered reading, which is only updated every 100 milliseconds. This could reduce the ability to capture min and max values by several orders of magnitude.

See Also: READ & CLEAR ANALOG MIN VALUE, READ ANALOG MIN VALUE, SET ANALOG FILTER WEIGHT

READ & CLEAR ANALOG MIN VAL — Analog Point

Function: To retrieve the lowest value of a specified analog input since its last reading, then reset it to the current value.

Typical Use: To capture the lowest pressure over a given period of time.

Details:

- The current value for each channel is read and stored at the I/O unit every seven milliseconds. However, the response time of the input module may be much slower due to smoothing built in to the module. Check the specifications for the module to be used if high-speed readings are required.
- Min and max values are recorded at the I/O unit immediately after the current value is updated.
- Channels without a module installed or with a thermocouple module that has an open thermocouple will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **READ & CLEAR ANALOG MIN VAL**

From	PRES SENSOR	*analog input*
Move To	MIN KPA	*variable float (min value)*

Notes:

- Use this command to clear the analog min value before actual readings commence.

Dependencies:

- If digital filtering is active (see SET ANALOG FILTER WEIGHT), min and max value detection is derived from the filtered reading, which is only updated every 100 milliseconds. This could reduce the ability to capture min and max values by several orders of magnitude.

See Also: READ & CLEAR ANALOG MAX VALUE, READ ANALOG MAX VALUE, SET ANALOG FILTER WEIGHT

READ & CLEAR ANALOG TOTAL VAL — Analog Point

Function: To read and clear the totalized (integrated) value of a specified analog input.

Typical Use: To capture a flow total that has been accumulating at the I/O unit before it reaches its maximum value.

Details:

- Totalizing is performed at the I/O unit. This command reads the current total, then clears it to zero.
- The value returned will be an integer from -32,768 to 32,767.
- Totalizing will be bidirectional if the input range is -10 to +10.
- Totalizing will stop when the total reaches either limit. Totalizing will resume after using READ & CLEAR ANALOG TOTAL VAL.
- Totalizing will stop when an input channel is too far under range (below -1.25% of span). Totalizing will resume when the input signal is back within range.
- Channels without a module installed will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **READ & CLEAR ANALOG TOTAL VAL**

From	FLOW RATE	*analog input*
Move To	TOTAL BARRELS	*variable float (total value)*

Notes:

- See Notes for SET ANALOG TOTALIZE RATE before using this command.
- Use this command to clear the total before actual readings commence.
- Use READ ANALOG TOTAL VALUE periodically to simply “watch” the total. When it exceeds 30,000, use READ & CLEAR ANALOG TOTAL VAL to capture the total to a float variable and reset it to zero.
- Do not use this command frequently when the total is a small value. Doing so may degrade the cumulative accuracy.

Dependencies:

- Before using this command, SET ANALOG TOTALIZE RATE must be executed. Otherwise, a value of -32,768 will be returned to indicate an error.

See Also: READ ANALOG TOTAL VALUE, SET ANALOG TOTALIZE RATE

READ & CLR ANALOG FILT VAL — Analog Point

Function: To read a digitally filtered input value from a specified analog channel, then set the filtered value to the current value.

Typical Use: To restart digital filtering using the current value as the default.

Details:
- Digital filtering must be activated before using this command by using SET ANALOG FILTER WEIGHT.
- Digital filtering, if activated, is performed at the I/O unit. Sample rate is 10 per second.
- The unfiltered analog input is still available using standard analog commands.
- Channels without a module installed or with a thermocouple module that has an open thermocouple will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ & CLR ANALOG FILT VAL**

From	TEMP SENSOR	*analog input*
Move To	FILTERED TEMP	*variable float (filtered value)*

Notes:
- Do not use this command for frequent reads (one per second or faster) since it continually resets the averaging. Use READ ANALOG FILT VALUE instead.
- To ensure that digital filtering will always be active, store changeable I/O unit values (such as filter weight) in permanent memory at the I/O unit. (You can do so through the Debugger.)

Dependencies:
- Before using this command, SET ANALOG FILTER WEIGHT must be executed. Otherwise, a value of -32,768 will be returned to indicate an error.

See Also: READ ANALOG FILT VALUE, SET ANALOG FILTER WEIGHT

READ ANALOG FILT VALUE — Analog Point

Function: To read the digitally filtered input value of a specified analog channel.

Typical Use: To smooth noisy or erratic signals.

Details:
- Digital filtering must be activated before using this command by using SET ANALOG FILTER WEIGHT.
- Digital filtering, if activated, is performed at the I/O unit. Sample rate is 10 per second.
- The unfiltered analog input is still available using standard analog commands.
- Channels without a module installed or with a thermocouple module that has an open thermocouple will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ ANALOG FILT VALUE**

From	TEMP SENSOR	*analog input*
Move To	FILTERED TEMP	*variable float (filtered value)*

Notes:
- Use SET ANALOG FILTER WEIGHT to restart filtering after a value of -32,768 is returned.
- To ensure that digital filtering will always be active, store changeable I/O unit values (such as filter weight) in permanent memory at the I/O unit. (You can do so through the Debugger.)

Dependencies:
- Before using this command, SET ANALOG FILTER WEIGHT must be issued. Otherwise, a value of -32,768 will be returned to indicate an error.

See Also: READ & CLR ANALOG FILT VAL, SET ANALOG FILTER WEIGHT

READ ANALOG MAX VALUE — Analog Point

Function: To retrieve the peak value of a specified analog input since its last reading.

Typical Use: To capture the peak pressure over a given period of time.

Details:
- The current value for each channel is read and stored at the I/O unit every seven milliseconds. However, the response time of the input module may be much slower due to smoothing built in to the module. Check the specifications for the module to be used if high-speed readings are required.
- Min and max values are recorded at the I/O unit immediately after the current value is updated.
- Channels without a module installed or with a thermocouple module that has an open thermocouple will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **READ ANALOG MAX VALUE**

From	PRES SENSOR	*analog input*
Move To	MAX KPA	*variable float (max value)*

Notes:
- Use READ & CLEAR ANALOG MAX VAL to clear the max value before actual readings commence.

Dependencies:
- If digital filtering is active (see SET ANALOG FILTER WEIGHT), min and max value detection is derived from the filtered reading, which is only updated every 100 milliseconds. This could reduce the ability to capture min and max values by several orders of magnitude.

See Also: READ & CLEAR ANALOG MAX VAL, READ & CLEAR ANALOG MIN VAL, READ ANALOG MIN VALUE

READ ANALOG MIN VALUE — Analog Point

Function: To retrieve the lowest value of a specified analog input since its last reading.

Typical Use: To capture the lowest pressure over a given period of time.

Details:

- The current value for each channel is read and stored at the I/O unit every seven milliseconds. However, the response time of the input module may be much slower due to smoothing built in to the module. Check the specifications for the module to be used if high-speed readings are required.
- Min and max values are recorded at the I/O unit immediately after the current value is updated.
- Channels without a module installed or with a thermocouple module that has an open thermocouple will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **READ ANALOG MIN VALUE**

From	PRES SENSOR	*analog input*
Move To	MIN KPA	*variable float (min value)*

Notes:

- Use READ & CLEAR ANALOG MIN VAL to clear the min value before actual readings commence.

Dependencies:

- If digital filtering is active (see SET ANALOG FILTER WEIGHT), min and max value detection is derived from the filtered reading, which is only updated every 100 milliseconds. This could reduce the ability to capture min and max values by several orders of magnitude.

See Also: READ & CLEAR ANALOG MIN VAL, READ & CLEAR ANALOG MAX VAL, READ ANALOG MAX VALUE

READ ANALOG SQRT FILT VALUE — Analog Point

Function: To read and linearize the digitally filtered input value of a flow signal from a differential pressure (DP) transmitter.

Typical Use: To smooth noisy or erratic signals from a DP transmitter connected to an orifice plate or venturi tube.

Details:

- Automatically linearizes flow values from DP transmitters (which require square root extraction) to engineering units.
- Digital filtering must be activated before using this command by using SET ANALOG FILTER WEIGHT.
- Digital filtering, if activated, is performed at the I/O unit. Sample rate is 10 per second.
- The unfiltered analog input is still available using standard analog commands.
- Channels without a module installed will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ ANALOG SQRT FILT VALUE**

From	DP FLOW XMTR	*analog input*
Move To	FILTERED FLOW	*variable float (filtered value)*

Notes:

- Use SET ANALOG FILTER WEIGHT to restart filtering after a value of -32,768 is returned.
- To ensure that filtering will always be active, store the filter value in permanent memory at the I/O unit. (You can do so through the Debugger.)
- Do not issue this command more than 10 times per second. Doing so will degrade the performance speed of the analog I/O unit.

Dependencies:

- Before using this command, SET ANALOG FILTER WEIGHT must be executed. Otherwise, a value of -32,768 will be returned to indicate an error.

See Also: READ ANALOG SQRT VALUE, SET ANALOG FILTER WEIGHT

READ ANALOG SQRT VALUE — Analog Point

Function: To read and linearize the analog input value of a flow signal from a differential pressure (DP) transmitter.

Typical Use: To linearize flow signals from a DP transmitter connected to an orifice plate or venturi tube.

Details:
- Automatically linearizes flow values from DP transmitters (which require square root extraction) to engineering units.
- Channels without a module installed will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **READ ANALOG SQRT VALUE**

From	DP FLOW XMTR	*analog input*
Move To	FLOW RATE	*variable float (flow value)*

Notes:
- Do not issue this command more than 10 times per second. Doing so will degrade the performance speed of the analog I/O unit.

See Also: READ ANALOG SQRT FILT VALUE

READ ANALOG TOTAL VALUE — Analog Point

Function: To read the totalized (integrated) value of a specified analog input.

Typical Use: To examine a flow total that has been accumulating at the I/O unit to determine when to clear it.

Details:
- Totalizing is performed at the I/O unit. This command reads the current total.
- The value returned will be an integer from -32,768 to 32,767.
- Totalizing will be bidirectional if the input range is -10 to +10, for example.
- Totalizing will stop when the total reaches either limit. Totalizing will resume after using READ & CLEAR ANALOG TOTAL VAL.
- Totalizing will stop when an input channel is too far under range (below -1.25% of span). Totalizing will resume when the input signal is back within range.
- Channels without a module installed will return a value of -32,768 to indicate an error.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ ANALOG TOTAL VALUE**

From	FLOWRATE	*analog input*
Move To	TOTAL BARRELS	*variable float (total value)*

Notes:
- See Notes for SET ANALOG TOTALIZE RATE before using this command.
- Use READ & CLEAR ANALOG TOTAL VAL to clear the total before actual readings commence.
- Use this command periodically to simply “watch” the total. When it exceeds 30,000, use READ & CLEAR ANALOG TOTAL VAL to capture the total to a float variable and reset it to zero.

Dependencies:
- Before using this command, SET ANALOG TOTALIZE RATE must be executed. Otherwise, a value of -32,768 will be returned to indicate an error.

See Also: READ & CLEAR ANALOG TOTAL VAL, SET ANALOG TOTALIZE RATE

SET ANALOG FILTER WEIGHT — Analog Point

Function: To activate digital filtering and set the amount of filtering to use on an analog input channel.

Typical Use: To smooth noisy or erratic input signals.

Details:

- When issued, this command copies the current input value to the filtered value to initialize it. Thereafter, a percentage of the difference between the current input value and the last filtered value is added to the last filtered value *at the rate of 10 times per second.*
- To read the filtered value, use READ ANALOG FILT VALUE, READ & CLR ANALOG FILT VAL, or READ ANALOG SQRT FILT VALUE. *All other commands will read the unfiltered value!*
- The digital filtering algorithm is an implementation of a first-order lag filter:
  New Filtered Value = ((Current Reading - Old Filter Value) / Filter Weight) +
  Old Filter Value
- To calculate the filter weight value that will result in a particular time constant value, use:
  Filter Weight = (Time Constant [in seconds] + 0.1) * 10
  A one-second time constant requires a filter weight of 11.
- To calculate the time constant that a particular filter weight will result in, use:
  Time Constant (in seconds) = (Filter Weight / 10) - 0.1
- With a filter weight of 11, an input value that suddenly changes from 0% to 100% (a 100% step change) will take over five seconds to be fully recognized. This is considered to be a time constant of one second (which is the time it takes for the input to reach 63.21% of its final value), as shown below:

100% STEP CHANGE, FILTER WEIGHT OF 11

INPUT VALUE	TIME IN SECONDS	VALUE READ
100%	0	0%
100%	1	63.21%
100%	2	86.47%
100%	3	95.02%
100%	4	98.17%
100%	5	99.33%

- A filter weight value of zero specifies digital filtering is to be discontinued.
- The filter weight will be used until power is removed from the I/O unit, or it will always be used if it is stored in permanent memory at the I/O unit.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	ANALOG IN
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

SET ANALOG FILTER WEIGHT *(CONTINUED)* Analog Point

Example: **SET ANALOG FILTER WEIGHT**

Samples	FILTER WEIGHT	*variable integer*
	TEMP IN1	*analog input*

Notes:

- Do not continually issue this command since it resets the filtered value to the current value.
- To ensure that digital filtering will always be active, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)

See Also: READ ANALOG FILT VALUE, READ & CLR ANALOG FILT VAL, READ ANALOG SQRT FILT VALUE

SET ANALOG GAIN — Analog Point

Function: To improve accuracy of an analog input signal or to change its range.

Typical Uses:
- To improve calibration on a temperature input.
- To rescale an input from one range (say, 25–50%) to a range of 0–100%.

Details:
- Always use SET ANALOG OFFSET before using this command.
- The default gain value is 1.0. The valid range for gain is 0.0003 to 16.0.
- A gain of 4.0 will cause a 25% input value to read 100% (full scale).
- The calculated gain will be used until power is removed from the I/O unit, or it will always be used if the gain is stored in permanent memory at the I/O unit.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	ANALOG IN
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET ANALOG GAIN**

Value	GAIN COEFFICIENT	*variable float (gain coefficient value)*
To	PRESS IN	*analog input*

Notes:
- This procedure should only have to be performed once.
- To ensure that the gain will always be used, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)

Dependencies:
- Must use SET ANALOG OFFSET first.

See Also: SET ANALOG OFFSET, CALC & SET ANALOG GAIN

SET ANALOG OFFSET — Analog Point

Function: To improve the accuracy of an analog input signal or to change its range.

Typical Uses:
- To improve calibration on a temperature input.
- To rescale an input from one range (say, 25–50%) to a range of 0–100%.

Details:
- Always use SET ANALOG GAIN after using this command.
- The default offset value is 0. The valid range for offset is -4,095 to 4,095 (integer values only).
- An offset of -1,024 will cause a 25% input value to read 0% (zero scale).
- The calculated offset will be used until power is removed from the I/O unit, or it will always be used if the offset is stored in permanent memory at the I/O unit.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	ANALOG IN
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET ANALOG OFFSET**

Value	OFFSET	*variable integer*
To	PRESS IN	*analog input*

Notes:
- This procedure should only have to be performed once.
- To ensure that the offset will always be used, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)

See Also: SET ANALOG GAIN, CALC & SET ANALOG OFFSET

SET ANALOG TOTALIZE RATE — Analog Point

Function: To start the totalizer and to establish the sampling rate.

Typical Use: To accumulate total flow based on a varying flow rate signal.

Details:
- The specified analog input channel is sampled at the end of each time interval.
- The sampled value is added to the previous accumulated total.
- Valid range for the sampling rate is 0.0 to 3276.7 seconds.
- Setting the sampling rate to 0.0 seconds will discontinue totalizing.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	ANALOG IN
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET ANALOG TOTALIZE RATE**

Seconds	TOTALIZE RATE	*variable float (number of seconds between samples)*
	FUEL FLOW	*analog input*

Notes:
- Set the sampling rate to 1.0 seconds.
- Use READ ANALOG TOTAL VALUE to "watch" the total accumulate. Wait for a reasonable value to accumulate (the greater the better but less than 32,767) before proceeding.
- Use READ & CLEAR ANALOG TOTAL VAL to move the accumulated total to a temporary float variable. Divide the temporary float variable by the appropriate divisor from the conversion table below, putting the result in the temporary float variable. Finally, add the temporary float variable to the cumulative total float variable.

FLOW RATE UNITS	DIVISOR (CONSTANT FLOAT)
per Second	1.0
per Minute	60.0
per Hour	3600.0
per Day	86400.0

- The following series of commands reads the accumulated total from the I/O unit, scales it, then adds the result to a float variable representing the total number of liters. The flow signal is scaled 0–1000 liters per minute.

READ & CLEAR ANALOG TOTAL VAL

Move To	TEMP FLOAT1	*variable float (temp value)*

DO DIVIDE TEMP FLOAT1

By	60.0	
Put Result in	TEMP FLOAT1	*variable float (temp value)*

DO ADD TEMP FLOAT1

Plus	LITERS	
Put Result in	LITERS	*variable float (total value)*

See Also: READ ANALOG TOTAL VALUE, READ & CLEAR ANALOG TOTAL VAL

SET ANALOG TPO PERIOD — Analog Point

Function: To set the time proportional output period of an analog channel where the analog TPO module is used.

Typical Use: To control the duty cycle of resistive heating elements used for temperature control.

Details:

- Analog channels will not function as TPOs until this command is issued.
- TPO periods are multiples of 2.048 seconds (i.e., 2.048, 4.096, 6.144, etc.) ranging from 2.048 to 522.2 seconds.
- If the value entered is not an exact multiple of 2.048 seconds, it will be rounded to the nearest period value.
- The time proportion period specifies the total time the output is varied over.
- Use MOVE to set the percent of on time by moving a value from 0–100 to the analog output channel.
- Always use 0–100 for the analog TPO scaling.
- PID outputs can be analog TPO channels.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	ANALOG OUT
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: This example sets the period for the TPO channel named TPO OUTPUT to 6.144 seconds (the value 6.0 is rounded automatically to the nearest period value, 6.144). If MOVE is used to set a 50% duty cycle (by MOVEing 50.0 to TPO OUTPUT), then the analog output will repeatedly cycle on for 3.072 seconds and off for 3.072 seconds.

SET ANALOG TPO PERIOD

Period	6.0	*constant float or variable float (period time in seconds)*
Point	TPO OUTPUT	*analog input (G4DA9)*

Notes:

- To ensure that the TPO period will always be correct, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)
- If the TPO period is not stored in permanent memory at the I/O unit, use SET ANALOG TPO PERIOD immediately before MOVEing a new value to the TPO every time. This ensures that the TPO period will be configured properly if the I/O unit has experienced loss of power. Do not, however, issue these commands more frequently than necessary since this can be counterproductive.

Dependencies:

- This command is valid only when used on a properly configured G4DA9 time proportional output module.

CHART OPERATIONS

CONTINUE CALLING CHART — Chart

Function: To continue the chart that started the current chart without having to know its name.

Typical Use: To use a chart as a form of subroutine, where this "subchart" may be called from many other charts to perform some common function.

Details:
- The only effect this command will have is to continue a suspended chart. If the calling chart is in any other state, the calling chart will be unaffected by this command.
- The calling chart will resume execution at its next scheduled time in the 32-task queue.
- The STATUS variable indicates success (-1) or failure (0). Since a failure would "break the chain" of execution, care must be taken to ensure success. In this example, it is possible for CHART_A to start SUB_CHART_A, then lose its time slice before it suspends itself, leaving it in the running state. Further, it is possible for SUB_CHART_A to complete execution in its allocated time slice(s) and issue the CONTINUE CALLING CHART command, which will *fail* because the calling chart is still in the running state. To prevent this situation, SUB_CHART_A should be modified to add the condition CALLING CHART SUSPENDED? just before the CONTINUE CALLING CHART operation. The True exit will lead directly to the CONTINUE CALLING CHART operation, but the False exit will loop back to the CALLING CHART SUSPENDED? condition itself to re-evaluate if the chart has been suspended. This ensures proper operation.
- For the same reason, the condition CHART STOPPED? should preface the START CHART "SUB_CHART_A" command.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **CONTINUE CALLING CHART**

Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- A safer method from a multitasking perspective is to utilize Cyrano's built-in subroutine feature.

See Also: CONTINUE CHART, START CHART, STOP CHART, SUSPEND CHART, CALLING CHART SUSPENDED?

CONTINUE CHART **Chart**

Function: To change the state of a specified chart from suspended to running.

Typical Use: In conjunction with SUSPEND CHART, to cause a specified chart to resume execution from where it left off.

Details:
- The only effect this command will have is to continue a suspended chart. If the specified chart is in any other state, it will be unaffected by this command.
- Upon success, the chart will resume execution at its next scheduled time in the 32-task queue at the point at which it was suspended.
- Suspended charts give up their time slice.
- The STATUS variable indicates success (-1) or failure (0).
- It is possible for CHART_A to complete execution of the commands between Suspending Chart B and Continuing Chart B in its allocated time slice(s). If this happens the CONTINUE CHART "CHART_B" command will fail, because the actual state of Chart B hasn't changed since it hasn't received a time slice yet.

Arguments:

ARGUMENT 1	ARGUMENT 2
CHART	VARIABLE FLOAT VARIABLE INTEGER

Example: **CONTINUE CHART**

	CHART_A	*chart name (chart of interest)*
Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Loop on CHART SUSPENDED? before this command if success is critical.

See Also: SUSPEND CHART, CHART SUSPENDED?, SET PRIORITY

GET CHART STATUS — Chart

Function: To determine the current status of a specified chart.

Typical Use: To determine in detail the current status of a chart.

Details:
- Status is returned as a 32-bit integer or float.
- Significant bits are 0–3:
  - Bit 0: Running Mode (0 = chart is stopped; 1 = chart is running)
  - Bit 1: Suspended Mode (0 = chart is not suspended; 1 = chart is suspended)
  - Bit 2: Step Mode (0 = chart is not being stepped through; 1 = chart is being stepped through)
  - Bit 3: Break Mode (0 = chart does not have break points defined; 1 = chart has break points defined)
- Bits 4–31 are reserved for Opto 22 use.
- Running Mode is on whenever a chart is running.
- Suspended Mode is on whenever a chart is suspended from Running Mode.
- Step Mode is on whenever a chart is being automatically or manually stepped through.
- Break Mode is on whenever a chart has a break point defined in one or more of its blocks.
- A chart that has never been started is considered stopped.
- A chart that is not suspended is either running or stopped.

Arguments:

ARGUMENT 1	ARGUMENT 2
CHART	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET CHART STATUS**

	CHART_A	*chart name (chart of interest)*
Put Status In	STATUS	*variable integer (status bits)*

Notes:
- Bit testing (rather than number testing) should be used to determine the current status, since a chart can simultaneously have multiple bits set at once. For example:
  - Break Mode Bit 3 = 1
  - Step Mode Bit 2 = 1
  - Running Mode Bit 0 = 1
  - Reserved Bits Bits 4–31 can have any value
- Avoid putting the returned status into a variable float, since the bits cannot be tested.

See Also: CHART SUSPENDED?, CHART STOPPED?, CHART RUNNING?, BIT TEST

SET HOST PRIORITY — Chart

Function: To increase the relative percentage of execution time for the HOST task.

Typical Use: To improve communication performance to anything connected to a HOST port.

Details:

- The new priority takes effect at the next scheduled time in the 32-task queue for the HOST task.
- Valid priority settings range from 1 to 255.
- Increasing the HOST task priority will give it more time to execute while giving all other charts less time to execute.
- Valid range for the *On Port* parameter (Argument 2) is 0 to 5. Use 5 for the ISA controller when its HOST port is configured as the "ISA Bus" port.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT INTEGER
CONSTANT INTEGER	VARIABLE INTEGER
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET HOST PRIORITY**

To	5	*constant integer (number of time slices)*
On Port	4	*constant integer (ARCNET port #)*

Notes:

- See the Chart Overview in Chapter 1 for important information.
- Increase the HOST task priority to 5 to improve communication performance to an MMI.
- *Warning:* Setting the HOST task priority too high will severely limit the capability of all other charts. It is advisable to use priority values of 10 or less.

See Also: SET PRIORITY

SET PRIORITY — Chart

Function: To increase the relative percentage of execution time for a chart.

Typical Use: To improve performance of the INTERRUPT chart or any time-sensitive task.

Details:
- The new priority takes effect immediately.
- Valid priority settings range from 1 to 255.
- The priority can be changed on the fly to instantly adjust allocated time to a specific portion of a chart.
- Increasing a chart's priority will give it more time to execute while giving all other charts less time to execute.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET PRIORITY**

To	PRIORITY	*variable integer (number of time slices)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Unless you have a specific timing problem to resolve, there is no benefit to changing the priority from its default value of 1.
- *Warning:* Setting the priority too high in a chart that runs in a loop will severely limit the capability of the HOST task to communicate with the MMI or Debugger. It is advisable to use priority values of 5 or less for charts that run continuously.
- INTERRUPT chart usage: Put in BLOCK-0 to give it increased priority (if needed) when it runs. The suggested value is 50.
- HOST task usage: See SET HOST PRIORITY.

See Also: SET HOST PRIORITY

START CHART — Chart

Function: To request that a chart leave the STOPPED or suspended state and begin executing at BLOCK-0.

Typical Use: In the POWERUP chart, to start all other charts that need to run. Also used by a main chart to start event-driven charts.

Details:
- This command is only a request.
- The STATUS variable indicates success (-1) or failure (0).
- If the chart is stopped or suspended and fewer than 32 tasks are running, this command will succeed. Otherwise, it has no effect.
- Upon success, the chart is put into the 32-task queue (if it wasn't there already) and will start at its next scheduled time.

Arguments:

ARGUMENT 1	ARGUMENT 2
CHART	VARIABLE FLOAT VARIABLE INTEGER

Example: **START CHART**

	CHART_B	*chart name (chart of interest)*
Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Normally the status does not need to be checked, since the command will succeed in most cases. If there are any doubt or concerns, check the STATUS variable.
- Use STOP CHART to stop the INTERRUPT chart (if it's not in use) to free up a task in the 32-task queue, if desired.

Dependencies:
- A task must be available in the 32-task queue.

See Also: CONTINUE CHART, STOP CHART, START DEFAULT HOST TASK

START DEFAULT HOST TASK — Chart

Function: To request that the default HOST task leave the STOPPED or suspended state and begin executing.

Typical Use: To resume use of the HOST task protocol on the default HOST port after the port was used for something else.

Details:
- This command is only a request.
- The STATUS variable indicates success (-1) or failure (0).
- If the task is stopped or suspended and fewer than 32 tasks are running, this command will succeed. Otherwise, it has no effect.
- Upon success, the task is put into the 32-task queue (if it wasn't there already) and will start at its next scheduled time.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **START DEFAULT HOST TASK**

Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Normally the status does not need to be checked, since the command will succeed in most cases.
- The default HOST task can be stopped so the port can be used for other purposes and protocols. While it is stopped, no debugging can be done unless another HOST task is running on another port.

Dependencies:
- A task must be available in the 32-task queue.

See Also: SUSPEND DEFAULT HOST TASK, STOP HOST TASK, START HOST TASK (ASCII), START HOST TASK (BINARY)

START HOST TASK (ASCII) Chart

Function: To request an *additional* HOST task on a port other than that of the default HOST task.

Typical Use: To connect a modem or radio to a HOST port for remote debugging or for use with the MMI.

Details:

- Starts an additional HOST task that uses ASCII mode rather than BINARY mode.
- This command is only a request.
- The STATUS variable indicates success (-1) or failure (0).
- If the task is stopped or suspended and fewer than 32 tasks are running, this command will succeed. Otherwise, it has no effect.
- Upon success, the HOST task is put into the 32-task queue and will start at its next scheduled time.
- The HOST task cannot be suspended; it can only be stopped using STOP HOST TASK.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **START HOST TASK (ASCII)**

On Port	1	*constant integer (communication port #)*
Put Status In	STATUS	*variable integer (success or failure code)*

Notes:

- See the Chart Overview in Chapter 1 for important information.
- Normally the status does not need to be checked, since the command will succeed in most cases. If there are any doubt or concerns, check the STATUS variable.
- If the Debugger or MMI is connected via modem or radio, it must also be in ASCII mode.

Dependencies:

- A task must be available in the 32-task queue.

See Also: START CHART, SET PRIORITY, STOP HOST TASK, START HOST TASK (BINARY)

START HOST TASK (BINARY) — Chart

Function: To request an *additional* HOST task on a port other than that of the default HOST task.

Typical Use: To connect a Debugger via a serial port while an MMI is connected via ARCNET.

Details:
- Starts an additional HOST task that uses BINARY mode rather than ASCII mode.
- This command is only a request.
- The STATUS variable indicates success (-1) or failure (0).
- If the task is stopped or suspended and fewer than 32 tasks are running, this command will succeed. Otherwise, it has no effect.
- Upon success, the task is put into the 32-task queue and will start at its next scheduled time.
- This task cannot be suspended; it can only be stopped using STOP HOST TASK.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **START HOST TASK (BINARY)**

On Port	1	*constant integer (communication port #)*
Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Normally the status does not need to be checked, since the command will succeed in most cases. If there are any doubt or concerns, check the STATUS variable.
- The Debugger must also be in BINARY mode.

Dependencies:
- A task must be available in the 32-task queue.

See Also: START CHART, SET PRIORITY, STOP HOST TASK, START HOST TASK (ASCII)

STOP CHART — Chart

Function: To stop a specified chart.

Typical Use: To stop another chart or the chart in which the command appears.

Details:
- Unconditionally stops any chart that is either running or suspended.
- Removes the stopped chart from the 32-task queue, making another task available.
- A chart can stop itself or any other chart.
- A chart that stops itself will immediately give up the remaining time allocated in its time slice(s).
- Stopping another chart won't take effect immediately but will take effect at the beginning of that chart's scheduled time in the queue.
- Charts that are stopped or suspended cannot start or continue themselves (nor can they do anything else).
- Stopped charts cannot be continued; they can only be started again (that is, their execution will begin again at BLOCK-0, not at the point at which they were stopped).

Arguments:

ARGUMENT 1
CHART

Example: **STOP CHART**

CHART_B	*chart name (chart of interest)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Use SUSPEND CHART if you want to continue a chart from where it left off.

See Also: START CHART, SUSPEND CHART, CHART STOPPED?

STOP CHART ON ERROR — Chart

Function: To stop the chart that caused the error at the top of the error queue.

Typical Use: To include in an error handler chart that runs with the other charts in a strategy. This chart monitors the error queue and takes appropriate action. Utilizing this command, the error handler chart can stop any chart that causes an error.

Details:

- Since Cyrano is a multitasking environment in the Mistic controller, an error handler chart cannot stop another chart instantaneously with this command (since the error handler chart itself only executes periodically). The actual time required depends on how many charts are running simultaneously as well as on the priority of each.
- The following errors can appear in the error queue:

CODE	ERRORS FROM I/O UNITS (BRICKS)	CODE	ERRORS FROM MISTIC CONTROLLER
1	Undefined command	31	Send timeout; Mistic couldn't send message
2	Bad CRC or checksum	32	Bad table index value
3	Buffer overrun	33	Arithmetic overflow
4	I/O unit has powered up since last access	35	Not a real number
5	Incorrect command length	36	Division by zero
6	Communication watchdog timeout	38	Processor failure or factory software fault
7	Specified data invalid	39	Port already in use
8	Busy error	40	E/R does not have a "read &hold" reaction
9	Command & channel configuration mismatch	41	Invalid E/R hold buffer at I/O unit (brick)
10	Invalid event type	42	ARCNET port busy
11	Invalid time for TPO, sq. wave or pulse	43	Host relock
29	I/O unit response timeout	44	Invalid board type
30	Invalid serial port number	45	String too short to hold data

Arguments: None.

Example: **STOP CHART ON ERROR**

Notes:

- See the Chart Overview in Chapter 1 for important information.
- To get to each error in the error queue, the top error must be discarded, bringing the next error to the top. Use POINT TO NEXT ERROR to do this.

See Also: POINT TO NEXT ERROR, GET ERROR COUNT, SUSPEND CHART ON ERROR

STOP HOST TASK — Chart

Function: To stop any *additional* HOST task or suspend the default HOST task.

Typical Use: To temporarily use the default HOST port to communicate with a non-HOST protocol device, such as a hand-held terminal.

Details:

- Unconditionally stops or suspends any HOST task that is either running or suspended.
- Does not take effect immediately, but takes effect at the beginning of the task's scheduled time in the queue.
- The default HOST task can only be suspended, not stopped, so it will never lose its place in the 32-task queue.
- An additional HOST task will be removed from the 32-task queue, making another task available.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **STOP HOST TASK**

On Port	4	*constant integer (communication port #)*

Notes:

- See the Chart Overview in Chapter 1 for important information.

See Also: STOP CHART, START DEFAULT HOST TASK, START HOST TASK (ASCII), START HOST TASK (BINARY)

SUSPEND CHART — Chart

Function: To suspend a specified chart.

Typical Use: To suspend another chart or the chart in which the command appears.

Details:
- Unconditionally suspends any chart that is running.
- Does not remove the suspended chart from the 32-task queue.
- A chart can suspend itself or any other chart.
- A chart that suspends itself will immediately give up the remaining time allocated in its time slice(s) and will no longer use a time slice.
- Suspending another chart won't take effect immediately but will take effect at the beginning of that chart's scheduled time in the queue.
- Charts that are suspended cannot start or continue themselves (nor can they do anything else).
- Suspended charts can be continued, started (execution begun at BLOCK-0), or stopped.

Arguments:

ARGUMENT 1	ARGUMENT 2
CHART	VARIABLE FLOAT VARIABLE INTEGER

Example: **SUSPEND CHART**

	CHART_B	*chart name (chart of interest)*
Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.

See Also: START CHART, CONTINUE CHART, CHART SUSPENDED?

SUSPEND CHART ON ERROR — Chart

Function: To suspend the chart that caused the error at the top of the error queue.

Typical Use: To include in an error handler chart that runs with the other charts in a strategy. This chart monitors the error queue and takes appropriate action. Utilizing this command, the error handler chart can suspend any chart that causes an error.

Details:

- Since Cyrano is a multitasking environment in the Mistic controller, an error handler chart cannot suspend another chart instantaneously with this command (since the error handler chart itself only executes periodically). The actual time required depends on how many charts are running simultaneously as well as on the priority of each.
- The following errors can appear in the error queue:

CODE	ERRORS FROM I/O UNITS (BRICKS)	CODE	ERRORS FROM MISTIC CONTROLLER
1	Undefined command	31	Send timeout; Mistic couldn't send message
2	Bad CRC or checksum	32	Bad table index value
3	Buffer overrun	33	Arithmetic overflow
4	I/O unit has powered up since last access	35	Not a real number
5	Incorrect command length	36	Division by zero
6	Communication watchdog timeout	38	Processor failure or factory software fault
7	Specified data invalid	39	Port already in use
8	Busy error	40	E/R does not have a "read &hold" reaction
9	Command & channel configuration mismatch	41	Invalid E/R hold buffer at I/O unit (brick)
10	Invalid event type	42	ARCNET port busy
11	Invalid time for TPO, sq. wave or pulse	43	Host relock
29	I/O unit response timeout	44	Invalid board type
30	Invalid serial port number	45	String too short to hold data

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **SUSPEND CHART ON ERROR**

Notes:

- See the Chart Overview in Chapter 1 for important information.
- To get to each error in the error queue, the top error must be discarded which brings the next error to the top. Use POINT TO NEXT ERROR to do this.

See Also: POINT TO NEXT ERROR, GET ERROR COUNT, STOP CHART ON ERROR

SUSPEND DEFAULT HOST TASK — Chart

Function: To suspend the default HOST task.

Typical Use: To temporarily use the default HOST port to communicate with a non-HOST protocol device, such as a hand-held terminal.

Details:
- Unconditionally suspends the default HOST task . This does not take effect immediately, but takes effect at the beginning of the task's scheduled time in the queue.
- The STATUS variable indicates success (-1) or failure (0).
- A failure indicates only that the default HOST task is already suspended.
- After this command has executed, the port that the default HOST task was using will become available for general use.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SUSPEND DEFAULT HOST TASK**

Put Status In	STATUS	*variable integer (success or failure code)*

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Normally the status does not need to be checked, since the command will succeed in most cases.
- If the port configuration (baud rate, etc.) is changed, be sure to return to the original configuration before executing the START DEFAULT HOST TASK command.

See Also: START DEFAULT HOST TASK, START HOST TASK (ASCII), START HOST TASK (BINARY)

COMMUNICATION OPERATIONS

OF CHARACTERS WAITING (PORT) — Communication

Function: To get the number of characters in the receive buffer of an open communication port and put it into a numeric variable.

Typical Use: To determine if there are any characters or a particular number of characters in the receive buffer before actually receiving them.

Details:
- A value of 0 means the receive buffer is empty.
- Each character counts as one regardless of what it is.
- As characters are received on ports 0–3, the count will increase.
- For ports 4 and 7 (ARCNET), any value greater than zero means that a complete message is waiting in the receive buffer.
- For ports 4 and 7 (ARCNET), only one message can be in the receive buffer.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **# OF CHARACTERS WAITING (PORT)**

Move To	CHAR COUNT	*variable integer (the count)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Use this command to determine if the number of characters expected equals the number of characters actually received in the buffer.

Dependencies:
- Must use REQUEST PORT first to open the port.

See Also: # OF CHARACTERS WAITING FROM PORT, CHARACTERS WAITING (PORT)?, CHARACTERS WAITING?

OF CHARACTERS WAITING FROM PORT — Communication

Function: To get the number of characters in the receive buffer of a closed communication port and put it into a numeric variable.

Typical Use: To determine if there are any characters or a particular number of characters in the receive buffer before actually receiving them.

Details:
- A value of 0 means the receive buffer is empty.
- Each character counts as one regardless of what it is.
- As characters are received on ports 0–3, the count will increase.
- For ports 4 and 7 (ARCNET), any value greater than zero means that a complete message is waiting in the receive buffer.
- For ports 4 and 7 (ARCNET), only one message can be in the receive buffer.
- A negative value indicates an error.
- For this command to be meaningful, the port should not be in use by any other chart.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **# OF CHARACTERS WAITING FROM PORT**

Port	1	*constant integer (port # to use)*
Move To	CHAR COUNT	*variable integer (the count)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Use to determine if the number of characters expected equals the number of characters actually received in the buffer.

Error Codes: -51 = Invalid port # — use port 0–7

See Also: # OF CHARACTERS WAITING FROM PORT, CHARACTERS WAITING (PORT)?, CHARACTERS WAITING?

CLEAR RECEIVE BUFFER (PORT) — Communication

Function: To empty the receive buffer of an open communication port.

Typical Use: To put the receive buffer in a known state (empty). To empty it of garbage characters or partial messages.

Details:
- All characters in the receive buffer will be deleted.

Arguments: None.

Example: **CLEAR RECEIVE BUFFER (PORT)**

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use once before starting communications.
- Always use just before sending a message that requires a response.
- Always use after communication errors to help recover.

Dependencies:
- Must use REQUEST PORT first to open the port.

See Also: CLEAR RECEIVE BUFFER

CLEAR RECEIVE BUFFER — Communication

Function: To empty the receive buffer of a closed communication port.

Typical Use: To put the receive buffer in a known state (empty). To empty it of garbage characters or partial messages.

Details:
- All characters in the receive buffer will be deleted, even if the port is in use by another chart.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER VARIABLE INTEGER	VARIABLE INTEGER

Example: **CLEAR RECEIVE BUFFER**

Port	MY PORT	*variable integer (port # to use)*
Put Result In	MY PORT STATUS	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use once before starting communications.
- Always use just before sending a message that requires a response.
- Always use after communication errors to help recover.

Error Codes:

0	=	Port is in use already
-1	=	OK
-51	=	Invalid port # — use port 0–7

See Also: CLEAR RECEIVE BUFFER (PORT)

CONFIGURE PORT — Communication

Function: To set serial port baud rate, parity, # data bits, # stop bits, and CTS on ports 0–3.

Typical Uses:
- To deviate from the factory defaults (no parity, 8 data bits, 1 stop bit, CTS disabled).
- To set the baud rate independently of either the Configurator settings or the front panel settings on the controller.
- To activate CTS control when sending to radios and modems.

Details:
- Parameters are case-insensitive.
- Works only on ports 0–3.
- Sets a default port timeout delay that is baud rate-dependent.
- Use COM0 for port 0, COM1 for port 1, COM2 for port 2, COM3 for port 3.
- Valid baud rates are 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 76800, and 115200.
- Valid parity choices are N (none), E (even), O (odd).
- Valid data bit choices are 5–8.
- Valid stop bit choices are 1–2.
- Valid CTS choices are "CTS" (enabled) or no entry (disabled).

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: **CONFIGURE PORT**

Use	COM1:38400,N,8,1,CTS	*constant string (the configuration)*
Put Status In	MY PORT STATUS	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Overrides all previous settings made by the Configurator or controller front panel.
- Use before SET PORT TIMEOUT DELAY, since this command will alter its value.
- Use the "CTS" parameter when communicating with radios and modems.

Error Codes:

0 = OK
-40 = Timeout — specified port is already in use
-50 = Improper configuration string syntax

GET ARCNET DEST. ADDR. — Communication

Function: To get the source address of the last ARCNET message received or the destination address of the next message to be sent.

Typical Use: To log ARCNET activity complete with source and destination addresses when ARCNET is not the HOST port.

Details:
- When used after receiving an ARCNET message, the source address of the message received is returned.
- When used after the command SET ARCNET DEST. ADDR., the destination address is returned.
- All references to ARCNET use port 4.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET ARCNET DEST. ADDR.**

Move To	ARCNET ADDR	*variable integer (the address)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Use before SET ARCNET DEST. ADDR., since this command will alter the value returned.

See Also: SET ARCNET DEST. ADDR

GET CHAR (PORT) Communication

Function: To get a single character from the receive buffer of an open communication port and move it to a numeric variable.

Typical Use: To get a message from another device one character at a time. Using APPEND CHARACTER, to append these characters (selectively if desired) to a string variable.

Details:
- Removes the oldest character from the receive buffer. Character values will be 0–255.
- If there are no characters in the receive buffer, this command will wait indefinitely until a character comes in.
- A character 0 (ASCII null) will have a value of zero; a character 48 (ASCII zero) will have a value of 48. These values will appear in the numeric variable. When appending a character 48 to a string variable, the number 0 will appear in the string.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **GET CHAR (PORT)**

Move To	CHAR	*variable integer (the character)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use the condition CHARACTERS WAITING (PORT)? before this command to avoid unnecessary delays.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.

See Also: REQUEST PORT, GET CHR FROM PORT, CONFIGURE PORT, APPEND CHARACTER

GET CHR FROM PORT — Communication

Function: To get a single character from the receive buffer of a closed communication port and move it to a numeric variable.

Typical Use: To get a message from another device one character at a time. Using APPEND CHARACTER, to append these characters (selectively if desired) to a string variable.

Details:
- Removes the oldest character from the receive buffer. Character values will be 0–255.
- If there are no characters in the receive buffer, a timeout error (-42) will eventually occur.
- A character 0 (ASCII null) will have a value of zero; a character 48 (ASCII zero) will have a value of 48. These values will appear in the numeric variable. When appending a character 48 to a string variable, the number 0 will appear in the string.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **GET CHR FROM PORT**

From Port	1	*constant integer (port # to use)*
Put Result In	CHAR	*variable integer (the character)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use the condition CHARACTERS WAITING? before this command to avoid unnecessary timeout errors.

Dependencies:
- Ports 0–3: baud rate, parity, # data bits, # stop bits.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-42	=	Timeout — probably didn't use CHARACTERS WAITING? before this command (see SET PORT TIMEOUT DELAY also)
-51	=	Invalid port # — use port 0–7

See Also: GET CHAR (PORT), CONFIGURE PORT, APPEND CHARACTER

GET PEER DESTINATION ADDRESS — Communication

Function: To get the source address of the last peer message received or the destination address of the next peer message to be sent.

Typical Use: To log peer activity complete with source and destination addresses.

Details:
- When used after receiving a peer message, the source address of the message received is returned.
- When used after the command SET PEER DESTINATION ADDRESS, the destination address is returned.
- All references to peer use port 7, which is a special gateway to the ARCNET cable.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET PEER DESTINATION ADDRESS**

Move To	PEER ADDR	*variable integer (the address)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Use before SET PEER DESTINATION ADDRESS, since this command will alter the value returned.

See Also: SET PEER DESTINATION ADDRESS

GET STRING (PORT) — Communication

Function: To get a message from the receive buffer of an open communication port and move it to a variable string.

Typical Use: To get ASCII messages from weigh scales, barcode readers, data entry terminals, and other Mistic controllers.

Details:
- The message is expected to end with a carriage return (character 13). This carriage return is deleted as the message is moved to the variable string.
- The variable string length must be at least two greater than the longest message.
- For ports 0–3, multiple messages can be in the receive buffer as long as each is delimited by a carriage return.
- For ports 4 and 7, only one message can be in the receive buffer. Until this message is removed from the receive buffer, all subsequent messages are discarded without error.
- If the first set of characters in the receive buffer that is equal to the length of the variable string does not contain a carriage return, these characters will be moved to the variable string without error and all remaining characters up to and including the first carriage return encountered (if any) will be deleted from the receive buffer.
- If the number of characters in the receive buffer is less than the length of the variable string, and if none of the characters is a carriage return, this command will wait indefinitely until at least one of these conditions is true.

Arguments:

ARGUMENT 1
VARIABLE STRING

Example: **GET STRING (PORT)**

Move To	RECEIVED MESSAGE	*variable string (the message)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER (PORT) once before using this command for the first time.
- Always use the condition CHARACTERS WAITING (PORT)? before this command to avoid unnecessary delays.
- When messages are terminated by a carriage return and a line feed (character 10), all messages received (starting with the second message) will have a line feed as the first character in the variable string. To remove it, get the first character of the variable string using GET NTH CHARACTER, where n = 1. If the nth character is equal to 10, use GET SUBSTRING with *Start At* set to 2 and *Number Of* set greater than or equal to the number of characters expected.
- Do not use this command for binary messages, since they may contain numerous carriage returns at unpredictable locations.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.
- Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.

See Also: REQUEST PORT, RECEIVE FROM PORT, GET CHR FROM PORT, CONFIGURE PORT

MOVE FROM TABLE TO (PORT) — Communication

Function: To send 32 numeric table values to an open communication port.

Typical Use: To share numeric table data with another controller. To send large amounts of numeric table data efficiently.

Details:
- Sends up to 32 table values directly from memory.
- If the table does not have at least 32 elements starting from the specified index, zeros will be sent for the missing elements.
- 128 bytes will be sent, four bytes per value. Since values are sent directly from memory, it doesn't matter if the data is integer or float.
- Valid table indices range from 0 to the declared table length.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on and leaves it on. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	FLOAT TABLE
VARIABLE INTEGER	INTEGER TABLE

Example: **MOVE FROM TABLE TO (PORT)**

Index	INDEX	*variable integer (table index to start at)*
From	MY TABLE	*integer or float table*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- Use MOVE TO TABLE FROM (PORT) to receive this data in the other controller.
- Always send the starting table index before sending the values so that the receiving controller will know where to put the data. If there is only one block of data that always has the same starting index, there is no need to send the starting index separately.
- If sending both integer and float values, be sure to send a type code first so that the receiving controller will know what type of table to store the values in. If the values are stored in the wrong type of table, their value will be interpreted incorrectly.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.
- Use error-checked communications or calculate and send a CRC first to ensure the integrity of the 128-byte packet. Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: MOVE FROM TABLE TO PORT, REQUEST PORT, CONFIGURE PORT

MOVE FROM TABLE TO PORT — Communication

Function: To send 32 numeric table values to a closed communication port.

Typical Use: To share numeric table data with another controller. To send large amounts of numeric table data efficiently.

Details:

- Sends up to 32 table values directly from memory.
- If the table does not have at least 32 elements starting from the specified index, zeros will be sent for the missing elements.
- 128 bytes will be sent, four bytes per value. Since the values are sent directly from memory, it doesn't matter if the data is integer or float.
- Valid table indices range from 0 to the declared table length.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will eventually timeout and return a -41 error. No message will be sent if CTS is off. A partial message may be sent if the timeout is too short.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT INTEGER	FLOAT TABLE	CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	INTEGER TABLE	VARIABLE INTEGER	VARIABLE INTEGER

Example: **MOVE FROM TABLE TO PORT**

Index	INDEX	*variable integer (table index to start at)*
From	MY TABLE	*integer or float table*
Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- Use MOVE ANL. I/O UNIT TO TABLE to read all 16 channels of an I/O unit and put the result in a float table.
- Use MOVE TO TABLE FROM PORT to receive this data in the other controller.
- Always send the starting table index before sending the values so that the receiving controller will know where to put the data. If there is only one block of data that always has the same starting index, there is no need to send the starting index separately.
- If sending both integer and float values, be sure to send a type code first so that the receiving controller will know what type of table to store the values in. If the values are stored in the wrong type of table, their value will be interpreted incorrectly.
- Use error-checked communications or calculate and send a CRC first to ensure the integrity of the 128-byte packet. Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.

MOVE FROM TABLE TO PORT *(continued)* Communication

Dependencies:
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE TO PORT to actually send the message.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-41	=	Send timeout — CTS is off or timeout is too short (see SET PORT TIMEOUT DELAY). For ports 4 and 7, this error indicates the transmit buffer is full.
-51	=	Invalid port # — use port 0, 1, 2, 3, 4, 6, or 7

See Also: MOVE FROM TABLE TO (PORT), CONFIGURE PORT

MOVE TO TABLE FROM (PORT) — Communication

Function: To get 32 numeric table values from an open communication port.

Typical Uses:
- To receive shared numeric table data from another controller.
- To get large amounts of numeric table data efficiently.

Details:
- Gets 128 bytes from the receive buffer and puts them directly in memory.
- If the table does not have at least 32 elements starting from the specified index, only a portion of the 128 bytes will be written to memory. Remaining bytes will be discarded.
- Valid table indices range from 0 to the declared table length.
- All remaining characters in the receive buffer will be discarded.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	FLOAT TABLE
VARIABLE INTEGER	INTEGER TABLE

Example: **MOVE TO TABLE FROM (PORT)**

Index	INDEX	*variable integer (table index)*
To	MY TABLE	*integer or float table*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use # OF CHARACTERS WAITING (PORT) to determine if the entire 128-byte packet is in the receive buffer. This number will be higher if an index or other data is sent as well. For example, if an index of 32 followed by a carriage return (character 13) was sent along with the 128 bytes, the total number of characters will be at least 131 (128+2+1).
- Do not use this command unless there are at least 128 bytes in the receive buffer, as the command will wait indefinitely until there are.
- If the data received must be put in the table at a different index each time, the index must be sent by the other controller before the data is sent. An easy way to do this is to send the index as an integer followed by a carriage return (character 13), then send the 128 bytes. Use GET STRING (PORT) to get the index. Then use CONVERT STRING TO NUMBER to put the index into a variable integer. Finally, get the table data.
- Be sure to put float data into a float table, integer data into an integer table. Otherwise, data values will be interpreted incorrectly.
- Use error-checked communications or calculate the CRC on the data to ensure the integrity of the 128-byte packet before putting it in the destination table. Since it must be received first, put it into a “holding table,” check the CRC, then copy it to the final destination table. Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.
- Use MOVE TABLE TO ANL I/O UNIT to write the float table data to all 16 channels of an I/O unit.
- Use MOVE FROM TABLE TO (PORT) in the other controller to send this data.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.

See Also: MOVE TO TABLE FROM PORT, REQUEST PORT, CONFIGURE PORT

MOVE TO TABLE FROM PORT — Communication

Function: To get 32 numeric table values from a closed communication port.

Typical Uses:
- To receive shared numeric table data from another controller.
- To get large amounts of numeric table data efficiently.

Details:
- Gets 128 bytes from the receive buffer and puts them directly in memory.
- If the table does not have at least 32 elements starting from the specified index, only a portion of the 128 bytes will be written to memory. Remaining bytes will be discarded.
- Valid table indices range from 0 to the declared table length.
- All remaining characters in the receive buffer will be discarded.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT INTEGER VARIABLE INTEGER	FLOAT TABLE INTEGER TABLE	CONSTANT INTEGER VARIABLE INTEGER	VARIABLE FLOAT VARIABLE INTEGER

Example: **MOVE TO TABLE FROM PORT**

Index	INDEX	*variable integer (table index to start putting data into)*
To	MY TABLE	*integer or float table*
Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use # OF CHARS WAITING FROM PORT to determine if the entire 128-byte packet is in the receive buffer. This number will be higher if an index or other data is sent as well. For example, if an index of 32 followed by a carriage return (character 13) was sent along with the 128 bytes, the total number of characters will be at least 131 (128+2+1).
- Do not use this command unless there are at least 128 bytes in the receive buffer, as the command will result in a timeout error (-42).
- If the data received must be put in the table at a different index each time, the index must be sent by the other controller before the data is sent. An easy way to do this is to send the index as an integer followed by a carriage return (character 13), then send the 128 bytes. Use RECEIVE FROM PORT to get the index. Then use CONVERT STRING TO NUMBER to put the index into a variable integer. Finally, get the table data.
- Be sure to put float data into a float table, integer data into an integer table. Otherwise, data values will be interpreted incorrectly.
- Use error-checked communications or calculate the CRC on the data to ensure the integrity of the 128-byte packet before putting it in the destination table. Since it must be received first, put it into a “holding table,” check the CRC, then copy it to the final destination table. Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.
- Use MOVE FROM TABLE TO PORT in the other controller to send this data.

Dependencies:
- Ports 0–3: baud rate, parity, # data bits, # stop bits.

MOVE TO TABLE FROM PORT *(continued)* Communication

Error Codes:

0 = No error
-40 = Timeout — specified port already in use
-42 = Timeout — probably didn't use CHARACTERS WAITING? before this command (see SET PORT TIMEOUT DELAY also)
-51 = Invalid port # — use port # 0, 1, 2, 3, 4, 6, or 7

See Also: MOVE TO TABLE FROM PORT, REQUEST PORT, CONFIGURE PORT

PRINT CHARACTER (PORT) — Communication

Function: To send a single character to an open communication port.

Typical Use: To send a message to another device one character at a time. Send a line feed (character 10) to a serial printer.

Details:

- Character values sent will be 0–255. Only the last eight bits are sent when the value is greater than 255.
- A value of 256 will be sent as a zero. A value of 257 will be sent as a 1.
- To send an ASCII null, use zero. To send an ASCII zero, use 48.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on and leaves it on. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), one character will be moved to the transmit buffer. When CTS turns on, the character will be sent. Sending more than one character with CTS off will cause this command to wait indefinitely.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **PRINT CHARACTER (PORT)**

From	10	*constant integer*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- Use PRINT STRING (PORT) instead when there are a lot of characters to send or when using radios that require RTS-CTS handshaking.
- If sending an eight-bit checksum, no need to BIT AND the checksum value with 255.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.
- Use SET LAST CHARACTER before this command to automatically turn RTS off after the character is sent.

Dependencies:

- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: REQUEST PORT, PRINT CHR TO PORT, CONFIGURE PORT, SET LAST CHARACTER

PRINT CHR TO PORT — Communication

Function: To send a single character to a closed communication port.

Typical Uses:
- To send a message to another device one character at a time.
- To send a line feed (character 10) to a serial printer.

Details:
- Character values sent will be 0–255. Only the last eight bits are sent when the value is greater than 255.
- A value of 256 will be sent as a zero. A value of 257 will be sent as a 1.
- To send an ASCII null, use zero. To send an ASCII zero, use 48.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on and leaves it on. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), one character will be moved to the transmit buffer. When CTS turns on, the character will be sent. Sending more than one character with CTS off will eventually result in a -41 error.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT INTEGER	VARIABLE FLOAT
CONSTANT INTEGER	VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE FLOAT		
VARIABLE INTEGER		

Example: **PRINT CHR TO PORT**

From	10	*constant integer*
To Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- Use PRINT TO PORT instead when there are a lot of characters to send or when using radios that require RTS-CTS handshaking.
- If sending an eight-bit checksum, no need to BIT AND the checksum value with 255.
- Use SET LAST CHARACTER before this command to automatically lower RTS after the character is sent.

Dependencies:
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE TO PORT to actually send the message.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-41	=	Send timeout — CTS is off or timeout is too short (see SET PORT TIMEOUT DELAY). For ports 4 and 7, this error indicates the transmit buffer is full.
-51	=	Invalid port # — use port 0–7

See Also: PRINT CHARACTER (PORT), CONFIGURE PORT

PRINT DATE (PORT) — Communication

Function: To send the date to an open communication port.

Typical Use: To print the date on a serial printer.

Details:
- Eight characters are sent. Format used is mm:dd:yy, where mm = month (01-12), dd = day (01-31), and yy = year (00–99).
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.

Arguments: None.

Example: **PRINT DATE (PORT)**

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: REQUEST PORT, CONFIGURE PORT

PRINT FORMATTED NUMBER (PORT) — Communication

Function: To send a number using a specified format to an open communication port.

Typical Uses:
- To print a number on a serial printer.
- To end a number with a fixed length to another device.

Details:
- The value printed will always have the length specified and the number of decimal digits specified.
- This command can be used to send integers. Set the number of decimal digits to zero. No decimal point will be sent.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT INTEGER	CONSTANT INTEGER
CONSTANT INTEGER	VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE FLOAT		
VARIABLE INTEGER		

Example: **PRINT FORMATTED NUMBER (PORT)**

From	TANK LEVEL	*variable float (the value)*
Length	5	*constant integer (total # of characters)*
Decimals	2	*constant integer (# of digits to the right of the decimal)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Remember to allow room for a minus sign if one is expected.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: REQUEST PORT, CONFIGURE PORT, PRINT NUMBER (PORT)

PRINT NEW LINE TO PORT — Communication

Function: This command has two context-sensitive functions:

- Ports 0–3: To send a carriage return (character 13) and a line feed (character 10) to a closed port.
- Ports 4, 6, and 7: To send the message in the transmit buffer of the *closed* ARCNET port (port 4), the *closed* local port (port 6), or the *closed* peer port (port 7). For ports 4 and 7, a carriage return (character 13) is appended to the message sent.

Typical Uses:

- To send a carriage return/line feed to a serial printer.
- To send anything to ports 4, 6, and 7.

Details:

- Ports 0–3: Sends two ASCII characters (13 and 10) to the specified port.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will eventually timeout and return a -41 error.
- Ports 4, 6, and 7: Must use this command to actually send what was "sent" by any other command. Anything "sent" to one of these ports is held in the transmit buffer of the port until this command is used. An acknowledgment is expected from the destination. For ports 4 and 7, this acknowledgment is an automatic feature of ARCNET. This command will wait up to the port timeout value for the acknowledgment. Retries will also be performed up to the retry limit. If no acknowledgment is received, this command will eventually timeout and return a -41 error.
- Ports 4 and 7: All communications are 16-bit CRC error checked.
- *Caution:* The message could be sent and acknowledged but discarded by the destination with no error if a message is already held in its receive buffer.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **PRINT NEW LINE TO PORT**

Port #	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- Ports 4 and 7: To be sure that a message sent was actually received, configure the destination device to reply with an "ACK" or an empty string immediately after receiving the message. Wait for this "ACK" for a second or so to verify receipt of the message.

Dependencies:

- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4 and 7: Must use SET ARCNET DEST. ADDR. for port 4 or SET PEER DESTINATION ADDRESS for port 7 before using this command.

PRINT NEW LINE TO PORT *(continued)* Communication

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-41	=	Send timeout — CTS is off (ports 0–3), timeout is too short (see SET PORT TIMEOUT DELAY), or there is no response from peer. For ports 4 and 7, this error indicates the transmit buffer is full.
-51	=	Invalid port # — use port 0–7

See Also: PRINT NEW LINE (PORT) W/TIMEOUT, PRINT NEW LINE (PORT), CONFIGURE PORT

PRINT NEW LINE (PORT) — Communication

Function: This command has two context-sensitive functions.
- Ports 0–3: To send a carriage return (character 13) and a line feed (character 10) to the open port.
- Port 6: To send the message in the transmit buffer of the open local port (port 6).

Typical Uses:
- To send a carriage return/line feed to a serial printer.
- To send anything to port 6 if it is open.

Details:
- Ports 0–3: Sends two ASCII characters (13 and 10) to the specified port.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.
- Port 6: Must use this command to actually send what was "sent" by any other command. Anything "sent" to this port is held in the transmit buffer until this command is used. All communications are 16-bit CRC error checked.

Arguments: None.

Example: **PRINT NEW LINE (PORT)**

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Do not use for peer-to-peer communication. Use PRINT NEW LINE (PORT) W/TIMEOUT or PRINT NEW LINE TO PORT instead.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.

See Also: PRINT NEW LINE (PORT) W/TIMEOUT, PRINT NEW LINE TO PORT, CONFIGURE PORT

PRINT NEW LINE (PORT) W/TIMEOUT — Communication

Function: To send the message in the transmit buffer of the open ARCNET port (port 4) or the open peer port (port 7).

Typical Use: To send anything to ports 4 and 7 if they are open.

Details:
- Must use this command to actually send what was “sent” by any other command. Anything “sent” to one of these ports is held in the transmit buffer of the port until this command is used. An acknowledgment is expected from the destination. This acknowledgment is an automatic feature of ARCNET. This command will wait up to the port timeout value for the acknowledgment. Retries will also be performed up to the retry limit. If an acknowledgment is not received, this command will eventually timeout and return a -41 error.
- All communications are 16-bit CRC error checked. A carriage return (character 13) is appended to the message sent.
- *Caution:* The message could be sent and acknowledged but discarded by the destination with no error if a message is already held in its receive buffer.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **PRINT NEW LINE (PORT) W/TIMEOUT**

Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use this command to send any ARCNET message to an open port.
- To be sure that a message sent was actually received, configure the destination device to reply with an “ACK” or an empty string immediately after receiving the message. Wait for this “ACK” for a second or so to verify receipt of message.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Must use SET ARCNET DEST. ADDR. for port 4 or SET PEER DESTINATION ADDRESS for port 7 before using this command.

Error Codes:

0	=	No error
-41	=	Send timeout — no acknowledgment was received. For ports 4 and 7, this error indicates the transmit buffer is full.

See Also: PRINT NEW LINE TO PORT

PRINT NUMBER (PORT) — Communication

Function: To send a number as is to an open communication port.

Typical Uses:
- To print a number on a serial printer.
- To send a number to another device.

Details:
- The value sent will have an exponential format if it is a float.
- The value sent will have a trailing space.
- Examples:

12.3456	becomes	1.23456e+01	Note the exponential format and trailing space.
12345	becomes	12345	Note that six digits are sent. There is a trailing space after the 5.

- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **PRINT NUMBER (PORT)**

From	TANK LEVEL	*variable float (the value)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Use PRINT FORMATTED NUMBER (PORT) instead.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: REQUEST PORT, CONFIGURE PORT, PRINT FORMATTED NUMBER (PORT)

PRINT NUMBER AS FIELD (PORT) — Communication

Function: To send a number using a specified minimum length to an open communication port.

Typical Uses:
- To print an integer on a serial printer.
- To send an integer with a fixed length to another device.

Details:
- A value whose length is less than that specified will have leading spaces added as necessary.
- A value whose length is equal to or greater than the specified length will be sent as is.
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT INTEGER
CONSTANT INTEGER	VARIABLE INTEGER
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **PRINT NUMBER AS FIELD (PORT)**

From	TOTAL GALLONS	*variable integer (the value)*
Length	6	*constant integer (total # of characters)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Although floats can be sent using this command, it is not recommended since the results vary greatly.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: REQUEST PORT, CONFIGURE PORT, PRINT FORMATTED NUMBER (PORT)

PRINT STR (OPTOMUX) TO PORT — Communication

Function: To send an OPTOMUX command to OPTOMUX I/O or any device that uses OPTOMUX protocol.

Typical Uses:
- To communicate as a master to existing OPTOMUX I/O.
- To communicate as a master to other computers that understand OPTOMUX protocol.

Details:
- Adds a leading “>“ (character 62) to the message.
- Calculates an eight-bit checksum and appends it to the end of the message as two hex bytes.
- Appends a carriage return (character 13) to the end of the message.
- *RS-232 mode only:* Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will eventually timeout and return a -41 error. No message will be sent if CTS is off. A partial message may be sent if the timeout is too short.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER	VARIABLE INTEGER

Example: **PRINT STR (OPTOMUX) TO PORT**

From	OPTOMUX COMMAND	*variable string (the command)*
To Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER before using this command.
- *RS-232 mode only:* Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- Consider using SEND/RECEIVE PORT (OPTOMUX) instead, since it includes a built-in send and receive.

Dependencies:
- Baud rate, parity, # data bits, # stop bits: Parity must be N; # data bits must be 8; # stop bits must be 1.
- Must use OPTOMUX protocol.

Error Codes:

0 = No error
-40 = Timeout — specified port already in use
-41 = Send timeout — CTS is off or timeout is too short (see SET PORT TIMEOUT DELAY). For ports 4 and 7, this error indicates the transmit buffer is full.
-51 = Invalid port # — use ports 0–3

See Also: RECEIVE FROM PORT (OPTOMUX), SEND/RECEIVE PORT (OPTOMUX), CONFIGURE PORT

PRINT STR WITH CRC TO PORT — Communication

Function: To send a Mistic I/O unit command to a Mistic I/O unit.

Typical Use: To send special commands to Mistic I/O units as detailed in the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270).

Details:
- Supports Opto 22 binary mode only.
- A two-byte CRC (CRC-16 Reverse with a seed of 0) is calculated and appended to the end of the message.
- Not for use with modems, since most modems do not support 11-bit frames.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER	VARIABLE INTEGER

Example: **PRINT STR WITH CRC TO PORT**

From	I/O UNIT COMMAND	*variable string (the command)*
To Port	2	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER before using this command each time.
- Use APPEND CHARACTER to build the message to send.
- No need to use SET PORT TIMEOUT DELAY since the factory default is adequate.
- No need to use CONFIGURE PORT.
- No need to use on ARCNET since all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.

Dependencies:
- I/O units must be in binary mode.

Error Codes:

0 = No error
-40 = Timeout — specified port already in use
-51 = Invalid port # — use port 0–7

See Also: SEND/RECEIVE PORT W/CRC

PRINT STRING (PORT) — Communication

Function: To send a message to an open communication port.

Typical Uses:
- To send data to another device.
- To send peer messages to another Mistic controller via ARCNET.
- To send an alarm message to a serial printer.

Details:
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.
- Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.

Arguments:

ARGUMENT 1
CONSTANT STRING
VARIABLE STRING

Example: **PRINT STRING (PORT)**

From	MESSAGE	*variable string (the data or message)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: PRINT TO PORT, PRINT CHARACTER (PORT), PRINT NEW LINE TO PORT, CONFIGURE PORT

PRINT TIME (PORT) — Communication

Function: To send the time to an open communication port.

Typical Use: To print the time on a serial printer.

Details:
- Sends eight characters in the format hh:mm:ss, where hh = hour (00–23), mm = (00–59), and ss = second (00–59).
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will wait indefinitely.

Arguments: None.

Example: **PRINT TIME (PORT)**

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.
- Use RELEASE ACTIVE PORT when finished to make the port available for other uses.

Dependencies:
- Must use REQUEST PORT first to open the port.
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE (PORT) to actually send the message.

See Also: REQUEST PORT, CONFIGURE PORT

PRINT TO PORT — Communication

Function: To send a message to a closed communication port.

Typical Uses:
- To send data to another device.
- To send peer messages to another Mistic controller via ARCNET.
- To send an alarm message to a serial printer.

Details:
- Ports 0–3 *(RS-232 mode only)*: Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will eventually timeout and return a -41 error. No message will be sent if CTS is off. A partial message may be sent if the timeout is too short.
- Note that all ARCNET communications (ports 4 and 7) are 16-bit CRC error checked.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER	VARIABLE INTEGER

Example: **PRINT TO PORT**

	MESSAGE1	*variable string (the message)*
Port #	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Ports 0–3 *(RS-232 mode only)*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.
- A carriage return (character 13) appended to this message acts as a message delimiter and allows the use of the command RECEIVE FROM PORT in another Mistic controller.

Dependencies:
- Ports 0–3: baud rate, parity, # data bits, # stop bits.
- Ports 4, 6, and 7: Must use PRINT NEW LINE TO PORT to actually send the message.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-41	=	Send timeout — CTS is off or timeout is too short (see SET PORT TIMEOUT DELAY). For ports 4 and 7, this error indicates the transmit buffer is full.
-51	=	Invalid port # — use port 0–7

See Also: PRINT STRING (PORT), PRINT CHR TO PORT , CONFIGURE PORT

RECEIVE FROM PORT — Communication

Function: To get a message from the receive buffer of a closed communication port and move it to a variable string.

Typical Use: To get ASCII messages from weigh scales, barcode readers, data entry terminals, and other Mistic controllers.

Details:
- The message is expected to end with a carriage return (character 13).
- The variable string length must be at least two greater than the length of the longest message expected.
- The carriage return in the receive buffer is deleted as the message is moved to the variable string.
- For ports 0–3, multiple messages can be in the receive buffer as long as each is delimited by a carriage return.
- For ports 4 and 7, only one message can be in the receive buffer. Until that message is removed from the receive buffer, all subsequent messages are discarded without error.
- The status is an error code that indicates how successful this command was. A zero indicates OK; any negative value indicates an error.
- If the first set of characters in the receive buffer that is equal in length to the variable string does not contain a carriage return, these characters will be moved to the variable string without error. In addition, all remaining characters up to and including the first carriage return encountered (if any) *will be deleted* from the receive buffer.
- If the number of characters in the receive buffer is less than the length of the variable string *and* none of the characters is a carriage return, a timeout error (-42) will eventually occur. When this happens, all characters in the receive buffer will be moved to the variable string. If this happens frequently, use SET PORT TIMEOUT DELAY to increase the timeout value. See Notes below.
- If the communication port is already in use, this command will wait for it to become available until a port-in-use timeout error (-40) occurs.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
VARIABLE STRING	CONSTANT INTEGER VARIABLE INTEGER	VARIABLE FLOAT VARIABLE INTEGER

Example: **RECEIVE FROM PORT**

Move To	RECEIVED MESSAGE	*variable string (the message)*
From Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER once before using this command for the first time.
- Always use SET PORT TIMEOUT DELAY once before using this command . As a minimum, use the result of this formula: (longest message length / baud rate) * 40. For example, a 24-character message at 9600 baud results in a delay of 0.1 seconds.
- Always use the condition CHARACTERS WAITING? before this command to avoid an unnecessary timeout error (-42).

RECEIVE FROM PORT *(continued)* Communication

- When there is a single response terminated by a carriage return and a line feed (character 10), use CLEAR RECEIVE BUFFER after this command to drop the line feed character.
- When there are multiple responses terminated by a carriage return and a line feed (character 10), all responses received starting with the second response will have a line feed as the first character in the variable string. To remove it, get the first character of the variable string using GET NTH CHARACTER where n=1. If the nth character is equal to 10, use GET SUBSTRING with *Start At* set to 2 and *Number Of* set greater than or equal to the number of characters expected.
- If a timeout error (-42) occurs *and* a partial string is received *and* this was unexpected, delay for 1 second or so, then use CLEAR RECEIVE BUFFER. This puts the receive buffer back to a known state.
- Do not use this command for binary messages, since they may contain numerous carriage returns at unpredictable locations.

Dependencies:

- Ports 0–3: baud rate, parity, # data bits, # stop bits.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-42	=	Timeout — no carriage return found in the receive buffer within allotted time (see SET PORT TIMEOUT DELAY)
-51	=	Invalid port # — use port 0–7

See Also: GET STRING (PORT), GET CHR FROM PORT, CONFIGURE PORT

RECEIVE FROM PORT (OPTOMUX) — Communication

Function: To get an OPTOMUX response from the receive buffer of a closed communication port and move it to a variable string.

Typical Use: To get OPTOMUX responses from OPTOMUX I/O.

Details:

- The response is expected to start with either an A or an N and expected to end with a carriage return. The two characters preceding the carriage return are expected to be the checksum when data is returned. The checksum is calculated and compared with what was sent. If there is a checksum error, or if "??" was substituted for the checksum characters, a -45 error will be returned. The checksum is not stripped from the response. Some valid responses are: N03, AB2EB9.
- The variable string length must be greater than the longest response expected.
- The carriage return in the receive buffer is deleted as the response is moved to the variable string.
- The status is an error code that indicates how successful this command was. A zero indicates OK; any negative value indicates an error.
- If the number of characters in the receive buffer is less than the length of the variable string and none of characters is a carriage return, a timeout error (-42) will eventually occur. When this happens, all characters in the receive buffer will be moved to the variable string. If this happens frequently, use SET PORT TIMEOUT DELAY to increase the timeout value. See Notes below.
- If the communications port is already in use, this command will wait for it to become available until a port-in-use timeout error (-40) occurs.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
VARIABLE STRING	CONSTANT INTEGER	VARIABLE FLOAT
	VARIABLE INTEGER	VARIABLE INTEGER

Example: **RECEIVE FROM PORT (OPTOMUX)**

Move To	OPTOMUX RESPONSE	*variable string (the response)*
From Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER once before using this command for the first time.
- Always use SET PORT TIMEOUT DELAY once before using this command . As a minimum, use the result of this formula: (longest message length / baud rate) * 40. For example, a 24-character message at 9600 baud results in a delay of 0.1 seconds.
- Always use the condition CHARACTERS WAITING? before this command to avoid an unnecessary timeout error (-42).
- If a timeout error (-42) occurs *and* a partial string is received *and* this was unexpected, delay for 1 second or so, then use CLEAR RECEIVE BUFFER. This puts the receive buffer back to a known state.

RECEIVE FROM PORT (OPTOMUX) *(continued)* Communication

- Consider using SEND/RECEIVE PORT (OPTOMUX) instead, since it includes a built-in send and receive.
- Error -42 indicates that checksum has already been verified and implies that the response is not in standard OPTOMUX format.

Dependencies:

- Ports 0–3: Baud rate, parity, # data bits, # stop bits: Parity must be N; # data bits must be 8; # stop bits must be 1.
- Must use OPTOMUX protocol.

Error Codes:

0 = No error
-40 = Timeout — specified port already in use
-42 = Timeout — no carriage return found in the receive buffer within allotted time (see SET PORT TIMEOUT DELAY)
-43 = Too few characters received
-44 = Response not formatted correctly (illegal first character)
-45 = CRC or checksum failed
-47 = Received a NAK (this is OK — not an error)
-51 = Invalid port # — use port 0–3

See Also: PRINT STR (OPTOMUX) TO PORT, SEND/RECEIVE PORT (OPTOMUX), CONFIGURE PORT

RECEIVE FROM PORT W/CRC — Communication

Function: To get a Mistic I/O unit binary response from the receive buffer of a closed communication port and move it to a variable string.

Typical Use: To get Mistic I/O unit binary responses from Mistic I/O.

Details:

- The response is expected to be from a Mistic I/O unit in binary mode.
- The variable string length must be greater than or equal to the longest response expected.
- The status is an error code that indicates how successful this command was. A zero indicates OK; any other value indicates an error.
- All characters with the exception of the two CRC characters are a part of the CRC calculation.
- The version of CRC used is CRC-16 Reverse with a seed of 0.
- Not for use with modems, since most modems do not support 11-bit frames.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
VARIABLE STRING	CONSTANT INTEGER	VARIABLE FLOAT
	VARIABLE INTEGER	VARIABLE INTEGER

Example: **RECEIVE FROM PORT W/CRC**

Move To	I/O UNIT RESPONSE	*variable string (the response)*
From Port	1	*constant integer (port # to use)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER once before using this command for the first time.
- No need to use SET PORT TIMEOUT DELAY since the factory default is adequate.
- Always use the condition CHARACTERS WAITING? before this command to avoid an unnecessary timeout error (-42).
- If an error occurs, delay for 0.1 second or so, then use CLEAR RECEIVE BUFFER. This puts the receive buffer back to a known state.

Dependencies:

- I/O units must be in binary mode.

Error Codes:

0	=	No error
Queue error 2	=	Bad CRC/checksum
Queue error 3	=	Bad message length received
-40	=	Timeout — specified port already in use
-42	=	Timeout — probably didn't use CHARACTERS WAITING? before this command (see SET PORT TIMEOUT DELAY also)
-48	=	String too short to hold response
-51	=	Invalid port # — use port 0, 1, 2, 3, or 6

See Also: PRINT STR WITH CRC TO PORT, SEND/RECEIVE PORT W/CRC

RELEASE ACTIVE PORT — Communication

Function: To give up exclusive rights to a port.

Typical Use: To allow other charts access to the port after communication is finished.

Details:
- Only works on an open port (one that REQUEST PORT was used to open).

Arguments: None.

Example: **RELEASE ACTIVE PORT**

Notes:
- See the Communication Overview in Chapter 1 for important information.

See Also: REQUEST PORT

REQUEST PORT — Communication

Function: To secure exclusive rights to a port.

Typical Use: To deny other charts access to a particular port before communication. Use prior to commands that rely on the port being open.

Details:
- Only works on a closed port (one that is not in use).
- Must use once to secure access to a port before using commands that rely on the port being open.
- The STATUS variable indicates exclusive access was granted (-1) or the specified port was already in use (0).

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **REQUEST PORT**

Port #	0	*constant integer (port # to use)*
Put Status In	PORT STATUS	*variable integer (status code)*

See Also: RELEASE ACTIVE PORT

SEND/RECEIVE PORT (OPTOMUX) Communication

Function: To communicate as a master with an OPTOMUX device using a closed communication port.

Typical Use: To communicate with OPTOMUX I/O.

Details:

- For use with ports 0–3 only.
- Adds a leading ">" (character 62) to the OPTOMUX message.
- Calculates an eight-bit checksum and appends it to the end of the OPTOMUX message as two hex bytes.
- Appends a carriage return (character 13) to the end of the OPTOMUX message.
- The OPTOMUX response is expected to start with either an A or an N and expected to end with a carriage return.
- The two characters preceding the carriage return are expected to be the checksum when data is returned.
- The checksum is calculated and compared with what was sent. If there is a checksum error, or if "??" was substituted for the checksum characters, a -45 error will be returned. The checksum is not stripped from the message.
- Some valid responses are: N03, AB2EB9.
- The variable string length for the OPTOMUX response must be greater than the length of the longest response expected.
- The carriage return in the receive buffer is deleted as the response is moved to the variable string.
- The status is an error code that indicates how successful this command was. A zero indicates OK; any negative value indicates an error.
- If the number of characters in the receive buffer is less than the length of the variable string and none of the characters is a carriage return, a timeout error (-42) will eventually occur. When this happens, all characters in the receive buffer will be moved to the variable string. If this happens frequently, use SET PORT TIMEOUT DELAY to increase the timeout value. See Notes below.
- If the communications port is already in use, this command will wait for it to become available until a port-in-use timeout error (-40) occurs.
- *RS-232 mode only:* Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will eventually timeout and return a -41 error. No message will be sent if CTS is off. A partial message may be sent if the timeout is too short.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT STRING VARIABLE STRING	CONSTANT INTEGER VARIABLE INTEGER	VARIABLE STRING	VARIABLE FLOAT VARIABLE INTEGER

SEND/RECEIVE PORT (OPTOMUX) *(continued)* Communication

Example: **SEND/RECEIVE PORT (OPTOMUX)**

From	OPTOMUX COMMAND	*variable string (the command)*
To Port	1	*constant integer (port # to use)*
Move To	OPTOMUX RESPONSE	*variable string (the response)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER before using this command each time.
- Always use SET PORT TIMEOUT DELAY once before using this command . As a minimum, use the result of this formula: (longest message length / baud rate) * 40. For example, a 24-character message at 9600 baud results in a delay of 0.1 seconds.
- *RS-232 mode only:* Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.

Dependencies:

- Baud rate, parity, # data bits, # stop bits: Parity must be N; # data bits must be 8; # stop bits must be 1.
- Must use OPTOMUX protocol.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-41	=	Send timeout — CTS is off or timeout is too short (see SET PORT TIMEOUT DELAY). For ports 4 and 7, this error indicates the transmit buffer is full.
-42	=	Timeout — no carriage return found in the receive buffer within allotted time (see SET PORT TIMEOUT DELAY)
-43	=	Too few characters received
-44	=	Response not formatted correctly (illegal first character)
-45	=	CRC or checksum failed
-47	=	Received a NAK (this is OK — not an error)
-51	=	Invalid port # — use port 0–3

See Also: PRINT STR (OPTOMUX) TO PORT, RECEIVE FROM PORT (OPTOMUX), CONFIGURE PORT

SEND/RECEIVE PORT W/CRC — Communication

Function: To send a Mistic I/O unit binary command to a Mistic I/O unit and get the response using a closed communication port.

Typical Use: To send special binary commands to Mistic I/O units as detailed in the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270).

Details:

- For use with ports 0, 1, 2, 3, and 6 only.
- Supports Opto 22 binary mode only.
- Calculates a two-byte CRC (CRC-16 Reverse with a seed of 0) and appends it to the end of the I/O unit command.
- Not for use with modems, since most modems do not support 11-bit frames.
- The response is expected to be from a Mistic I/O unit in binary mode.
- The variable string length for the I/O unit response must be greater than or equal to the length of the longest response expected.
- The status is an error code that indicates how successful this command was. A zero indicates OK; any other value indicates an error.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT STRING VARIABLE STRING	CONSTANT INTEGER VARIABLE INTEGER	VARIABLE STRING	VARIABLE FLOAT VARIABLE INTEGER

Example: **SEND/RECEIVE PORT W/CRC**

From	I/O UNIT COMMAND	*variable string (the command)*
To Port	1	*constant integer (port # to use)*
Move To	I/O UNIT RESPONSE	*variable string (the response)*
Put Status In	ERROR CODE	*variable integer (the error code)*

Notes:

- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER before using this command each time.
- Use APPEND CHARACTER to build the message to send.
- No need to use SET PORT TIMEOUT DELAY since the factory default is adequate.
- No need to use CONFIGURE PORT.

Dependencies:

- I/O units must be in binary mode.

Error Codes:

0	=	No error
Queue error 2	=	Bad CRC/checksum
Queue error 3	=	Bad message length received
-40	=	Timeout — specified port already in use
-42	=	Timeout — no response or timeout too short (see SET PORT TIMEOUT DELAY)
-48	=	String too short to hold response
-51	=	Invalid port # — use port 0, 1, 2, 3, or 6

See Also: PRINT STR WITH CRC TO PORT, RECEIVE FROM PORT W/CRC

SEND/RECEIVE USING PORT N — Communication

Function: To send a ASCII message and get an ASCII response using a closed communication port.

Typical Uses:
- To poll for ASCII messages from weigh scales, barcode readers, data entry terminals, and other Mistic controllers.
- To send data to other devices where an immediate response is expected.

Details:
- For use with ports 0–3 only.
- Appends a carriage return (character 13) to the end of the message sent.
- The response is expected to end with a carriage return (character 13).
- The variable string length for the response must be at least two greater than the length of the longest message expected.
- The carriage return in the receive buffer is deleted as the response is moved to the variable string.
- The status is an error code that indicates how successful this command was. A zero indicates OK; any negative value indicates an error.
- If the first set of characters in the receive buffer that is equal to the length of the variable string does not contain a carriage return, these characters will be moved to the variable string without error and all remaining characters in the receive buffer will be discarded.
- If the number of characters in the receive buffer is less than the length of the variable string and none of the characters is a carriage return, a timeout error (-42) will eventually occur. When this happens, all characters in the receive buffer will be moved to the variable string. If this happens frequently, use SET PORT TIMEOUT DELAY to increase the timeout value. See Notes below.
- If the communication port is already in use, this command will wait for it to become available until a port-in-use timeout error (-40) occurs.
- If the receive buffer is empty, no message will be sent and an error -42 will be returned.
- *RS-232 mode only:* Turns RTS on. Turns RTS off when finished. If CTS is not connected, it is on by default except on COM0 of the M4RTU. If CTS is off or the timeout is too short (see SET PORT TIMEOUT DELAY), this command will eventually timeout and return a -41 error. No message will be sent if CTS is off. A partial message may be sent if the timeout is too short.
- No error checking is performed on any data passed.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT STRING	CONSTANT INTEGER	VARIABLE STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER		VARIABLE INTEGER

Example: **SEND/RECEIVE USING PORT N**

From	COMMAND	*variable string (the message)*
To Port	1	*constant integer (port # to use)*
Move To	RESPONSE	*variable string (the response)*
Put Status In	ERROR CODE	*variable integer (the error code)*

SEND/RECEIVE USING PORT N *(continued)* Communication

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use CLEAR RECEIVE BUFFER before using this command each time.
- Always use SET PORT TIMEOUT DELAY once before using this command . As a minimum, use the result of this formula: (longest message length / baud rate) * 40. For example, a 24-character message at 9600 baud results in a delay of 0.1 seconds.
- When there are multiple responses terminated by a carriage return and a line feed (character 10), all responses received starting with the second response will have a line feed as the first character in the variable string. To remove it, get the first character of the variable string using GET NTH CHARACTER where n=1. If the nth character is equal to 10, use GET SUBSTRING with *Start At* set to 2 and *Number Of* set greater than or equal to the number of characters expected.
- Do not use this command for binary messages, since they may contain numerous carriage returns at unpredictable locations.
- When using this command to communicate with another Mistic controller, use RECEIVE FROM PORT in the other controller.
- *RS-232 mode only*: Always connect RTS to CTS on COM0 of the M4RTU unless RTS and CTS must be connected to a modem, printer, or other device. Never connect anything to CTS unless it must be used to handshake with another device.

Dependencies:
- Baud rate, parity, # data bits, # stop bits.

Error Codes:

0	=	No error
-40	=	Timeout — specified port already in use
-41	=	Send timeout — CTS is off or timeout is too short (see SET PORT TIMEOUT DELAY). For ports 4 and 7, this error indicates the transmit buffer is full.
-42	=	Timeout — no carriage return found in the receive buffer within allotted time (see SET PORT TIMEOUT DELAY)
-51	=	Invalid port # — use port 0–3

See Also: GET STRING (PORT), PRINT TO PORT, GET CHR FROM PORT, CONFIGURE PORT

SET ARCNET DEST. ADDR. — Communication

Function: To set the destination address of the next ARCNET message to be sent.

Typical Use: To direct an ARCNET message to an address other than the address of the last ARCNET message received.

Details:
- No need to use this command when the destination is the same as the last ARCNET message received.
- All references to ARCNET use port 4.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **SET ARCNET DEST. ADDR.**

To	ARCNET DEST	*variable integer (the address)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use this command after receiving an ARCNET message unless responding to the source of the message.

See Also: GET ARCNET DEST. ADDR.

SET LAST CHARACTER — Communication

Function: To inform the communication hardware that the next character sent will be the last in this message.

Typical Use: To turn off RTS after a complete message is sent.

Details:
- For use with ports 0–3 only.
- Must use when the last character of a message is sent as a single character *and* RTS must be turned off to receive a response (as when using half-duplex radio with RS-232 or 2-wire RS-485/422 communication).
- When messages are sent as a string, RTS turns off automatically after the last character in the string is sent.

Arguments: None.

Example: **SET LAST CHARACTER**

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use this command immediately prior to sending the final character of a message if you want RTS to turn off.

Dependencies:
- Must be used prior to a command that sends a single character such as PRINT CHARACTER (PORT) or PRINT CHR TO PORT.

SET NUMBER OF RETRIES — Communication

Function: To change the factory default retry setting.

Typical Use: To change the number of retries performed when there is a communication error.

Details:
- The factory default is two retries, which results in a total of three attempts in succession before reporting an error.
- This setting affects all communication ports simultaneously.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **SET NUMBER OF RETRIES**

3	*constant integer*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- The default number of retries (two) is more than adequate for most situations.
- Before using this command, make sure the timeout value is long enough. See Notes under SET PORT TIMEOUT DELAY for details.

See Also: SET PORT TIMEOUT DELAY

SET PEER DESTINATION ADDRESS — Communication

Function: To set the destination address of the next peer message to be sent.

Typical Use: To direct a peer message to an address other than the address of the last peer message received.

Details:
- No need to use this command when the destination is the same as the last peer message received.
- All references to peer use port 7, which is a special gateway to the ARCNET cable.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **SET PEER DESTINATION ADDRESS**

To	PEER DEST	*variable integer (the address)*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Always use this command after receiving a peer message unless responding to the source of the message.

See Also: GET PEER DESTINATION ADDRESS

SET PORT TIMEOUT DELAY — Communication

Function: To change the default timeout delay setting.

Typical Use: To change the timeout delay (the time before retries are attempted) when there is a communication error.

Details:
- The default value is based on the baud rate for the port and is usually sufficient.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT INTEGER
CONSTANT INTEGER	VARIABLE INTEGER
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET PORT TIMEOUT DELAY**

Delay Sec.	1.5	*constant float*
Port #	2	*constant integer*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- If you choose to change the timeout delay, do so after using the CONFIGURE PORT command.
- Use this command to increase the delay if errors -41 or -42 are a constant problem.
- When sending or receiving long messages (50 or more characters), increase the timeout delay. As a minimum, use the result of this formula: (longest message length / baud rate) * 40. For example, a 24-character message at 9600 baud results in a delay of 0.1 seconds.

Dependencies:
- The CONFIGURE PORT command will overwrite any value set by this command.

See Also: SET NUMBER OF RETRIES, CONFIGURE PORT

VERIFY CHECKSUM ON STRING — Communication

Function: To test the integrity of a message received that uses the OPTOMUX protocol.

Typical Use: To verify checksum on any string similar to the OPTOMUX format.

Details:
- Checksum uses an eight-bit value ranging from 0–255.
- The first character in the received message string is not counted as part of the checksum.
- Expects the first character in the received message to be a ">," an "A," or an "N."
- Expects the last two characters in the received message string to be the ASCII hex checksum.
- Since an OPTOMUX NAK does not return a checksum, a -47 code is returned indicating that a NAK was received. This is not an error!

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: **VERIFY CHECKSUM ON STRING**

	RECEIVED MESSAGE	*variable string*
Put Result In	CHECKSUM STATUS	*variable integer*

Notes:
- See the Communication Overview in Chapter 1 for important information.
- To use on a message where the first character is a part of the checksum, append the received message string to another string with a single ">" (character 62) in it. This will provide the expected legal character to ignore.

Error Codes:

0 = No error
-43 = Too few characters received
-44 = Response not formatted correctly (illegal first character)
-45 = CRC or checksum failed
-47 = Received a NAK (this is OK — not an error)
-49 = Receive string was empty

DIGITAL POINT OPERATIONS

CLEAR ALL LATCHES — Digital Point

Function: To reset all digital input latches on a digital multifunction I/O unit.

Typical Use: To ensure all input on- or off-latches are reset. Usually performed after a power-up sequence.

Details:
- Clears all previously set on- or off-latches associated with input channels on the specified digital multifunction I/O unit regardless of the on/off status of the inputs.
- All input channels automatically have the latch feature.
- An on-latch is set when the input channel changes from off to on.
- An off-latch is set when the input channel changes from on to off.

Arguments:

ARGUMENT 1
DIGITAL MF I/O UNIT
REM SMPL I/O UNIT

Example: **CLEAR ALL LATCHES**

INPUT BOARD #1	*digital multifunction I/O unit*

Notes:
- If using the latching feature on one or more digital inputs, it is a good practice to clear all the latches after power-up or reset.

Dependencies:
- Applies only to remote and local digital multifunction I/O units.

See Also: CLEAR ON-LATCH, CLEAR OFF-LATCH

CLEAR OFF-LATCH — Digital Point

Function: To reset a previously set digital input off-latch.

Typical Use: To reset the off-latch associated with a digital input to catch the next transition.

Details:
- Resets the off-latch of a single digital input regardless of the on/off status of the input.
- The next time the input channel changes from on to off, the off-latch will be set.
- Off-latches are very useful for catching high-speed on-off-on input transitions, since they are processed by the digital multifunction I/O unit locally.

Arguments:

ARGUMENT 1
OFF LATCH

Example:

CLEAR OFF-LATCH

BUTTON #1	*digital input configured with the off-latch feature*

Notes:
- Clear an off-latch after a GET OFF-LATCH VALUE command to re-arm the latch.

Dependencies:
- Applies only to inputs configured with the off-latch feature on digital multifunction I/O units.

See Also: GET OFF-LATCH VALUE, CLEAR ALL LATCHES

CLEAR ON-LATCH — Digital Point

Function: To reset a previously set digital input on-latch.

Typical Use: To reset the on-latch associated with a digital input to catch the next transition.

Details:
- Resets the on-latch of a single digital input regardless of the on/off status of the input.
- The next time the input channel changes from off to on, the on-latch will be set.
- On-latches are very useful for catching high-speed off-on-off input transitions, since they are processed by the digital multifunction I/O unit locally.

Arguments:

ARGUMENT 1
ON LATCH

Example: **CLEAR ON-LATCH**

BUTTON #1	*digital input configured with the on-latch feature*

Notes:
- Clear an on-latch after a GET ON-LATCH VALUE command to re-arm the latch.

Dependencies:
- Applies only to inputs configured with the on-latch feature on digital multifunction I/O units.

See Also: GET ON-LATCH VALUE, CLEAR ALL LATCHES

DISABLE DIGITAL POINT — Digital Point

Function: To disable communication between the program in the Mistic controller and an individual digital channel.

Typical Use: To disconnect the program from a specified digital channel for simulation and program testing.

Details:
- All digital point communication is enabled by default.
- This command does not affect the digital channel in any way. It only disconnects the program in the Mistic controller from the digital channel.
- When communication to a digital channel is disabled, program actions have no effect.
- When a program reads the state of a disabled channel, the last value before the channel was disabled (IVAL) will be returned.
- Likewise, any attempts by the program to change the state of an output channel will affect only the IVAL, not the actual output channel (XVAL). Disabling a digital channel when a program is running has no effect on the program.

Arguments:

ARGUMENT 1
DIGITAL IN DIGITAL OUT

Example: **DISABLE DIGITAL POINT**

START BUTTON	*digital input or output channel*

Notes:
- Use TURN OFF instead if the objective is to shut off a digital output.
- Disabling a digital channel is ideal for a start-up situation, since the program thinks it is reading an input or updating an output as it normally would.
- Use the IVAL field in the Debugger to change the state of an input to on or off.
- Use the XVAL field in the Debugger to change the state of an output to on or off.

See Also: ENABLE DIGITAL POINT

ENABLE DIGITAL POINT — Digital Point

Function: To enable communication between the program in the Mistic controller and an individual digital channel.

Typical Use: To reconnect the program to a specified digital channel after simulation or program testing.

Details:
- All digital channel communication is enabled by default.
- This command does not affect the digital channel in any way. It only connects the program in the Mistic controller with the digital channel.
- When communication to a digital channel is enabled, program actions can affect it.
- When a program reads the state of an enabled input channel, the current status of the channel (XVAL) will be returned to the program (IVAL).
- Likewise, an enabled output channel will update when the program writes a value. The XVAL and IVAL will match at this time.

Arguments:

ARGUMENT 1
DIGITAL IN DIGITAL OUT

Example: **ENABLE DIGITAL POINT**

MOTOR START	*digital input or output channel*

Notes:
- Use TURN ON instead to turn on digital output.
- Use this command to enable a digital channel previously disabled by the DISABLE DIGITAL POINT command.

See Also: DISABLE DIGITAL POINT

GENERATE N PULSES — Digital Point

Function: To output a specified number of pulses of configurable on and off times.

Typical Use: To drive stepper motor controllers, flash indicator lamps, or increment counters.

Details:

- Generates a digital waveform on the specified digital output channel. *On Time* specifies the amount of time in seconds that the channel will remain on during each pulse; *Off Time* specifies the amount of time the channel will remain off.
- The minimum *On Time* and *Off Time* is 0.001 second with a resolution of 0.0001 second, making the maximum frequency 500 Hertz.
- The maximum *On Time* and *Off Time* is 429,496.7000 seconds (4.97 days on, 4.97 days off).
- Valid range for *# of Pulses* is 0 to 2,147,483,647 if an integer is used, 0 to 4,294,967,000 if a float is used.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT FLOAT	CONSTANT FLOAT	CONSTANT FLOAT	SMART DIGITAL OUT
CONSTANT INTEGER	CONSTANT INTEGER	CONSTANT INTEGER	
VARIABLE FLOAT	VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	VARIABLE INTEGER	

Example: **GENERATE N PULSES**

On Time	0.250	*on duration of one pulse (in seconds)*
Off Time	0.500	*off duration of one pulse (in seconds)*
# of Pulses	# PULSES	*number of pulses to output*
To	STEPPER OUT	*digital output*

Notes:

- To stop a currently executing pulse train, use TURN OFF.
- Executing a GENERATE N PULSES command will discontinue any previous GENERATE N PULSES command.
- The minimum on or off time is 0.001 seconds; however, the digital output module's minimum turn-on and turn-off times may be greater. Check the specifications for the module to be used.

Dependencies:

- Applies only to outputs on digital multifunction I/O units.

See Also: TURN OFF, START CONTINUOUS SQUARE WAVE

GET & CLEAR OFF-LATCH VALUE — Digital Point

Function: To read and re-arm a high-speed off-latch associated with a digital input.

Typical Use: To ensure detection of an extremely brief on-to-off transition of a digital input.

Details:
- Reads and re-arms the off-latch of a single digital input.
- The next time the input channel changes from on to off, the off-latch will be set.
- Off-latches detect on-off-on input transitions that would otherwise occur too fast for the Mistic controller to detect, since they are processed by the digital multifunction I/O unit.
- If the latch is not set, the output will turn off. If the latch is set, the output will turn on.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF LATCH	DIGITAL OUT
	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **GET & CLEAR OFF-LATCH VALUE**

Get	BUTTON #3 LATCH	*digital input configured with off-latch feature*
Move To	ALARM HORN	*digital output*

Notes:
- The ability of the digital multifunction I/O unit to detect fast input transitions is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the off-latch feature on digital multifunction I/O units.

See Also: GET OFF-LATCH VALUE, CLEAR OFF-LATCH VALUE, CLEAR ALL LATCHES

GET & CLEAR ON-LATCH VALUE — Digital Point

Function: To read and re-arm a high-speed on-latch associated with a digital input.

Typical Use: To ensure detection of an extremely brief off-to-on transition of a digital input.

Details:
- Reads and re-arms the on-latch of a single digital input.
- The next time the input channel changes from off to on, the on-latch will be set.
- On-latches detect off-on-off input transitions that would otherwise occur too fast for the Mistic controller to detect, since they are processed by the digital multifunction I/O unit.
- The value read is placed in the argument specified by the *Move To* parameter. If the latch is not set, the argument will contain the value 0 (False). If the latch is set, the argument will be set to -1 (True).

Arguments:

ARGUMENT 1	ARGUMENT 2
ON LATCH	DIGITAL OUT VARIABLE FLOAT VARIABLE INTEGER

Example: **GET & CLEAR ON-LATCH VALUE**

Get	E STOP BUTTON	*digital input configured with on-latch feature*
Move To	LATCH VAR	*variable integer (the on-latch value)*

Notes:
- The ability of the digital multifunction I/O unit to detect fast input transitions is limited by the input module’s turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the off-latch feature on digital multifunction I/O units.

See Also: GET ON-LATCH VALUE, CLEAR ON-LATCH VALUE, CLEAR ALL LATCHES

GET AND CLEAR COUNTER VALUE — Digital Point

Function: To read and clear a digital input counter value.

Typical Use: To count pulses from turbine flow meters, magnetic pickups, encoders, proximity switches, etc.

Details:
- Reads the current value of a digital input counter and places it in the *Move To* parameter.
- Sets the counter at the I/O unit to zero.
- Does not stop the counter from continuing to count.
- Valid range is 0 to 4,294,967,296 counts.

Arguments:

ARGUMENT 1	ARGUMENT 2
COUNTER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET AND CLEAR COUNTER VALUE**

From	BOTTLE COUNTER	*digital input configured with counter feature*
Move To	# OF BOTTLES	*variable integer (the counter value)*

Notes:
- The maximum speed at which the counter can operate is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.
- Since 32-bit signed integers can only count up to 2,147,483,647, use a float to hold the counts if exceeding this amount.

Dependencies:
- Always use START COUNTER once before using this command for the first time.
- Applies only to inputs configured with the counter feature on digital multifunction I/O units.

See Also: GET AND CLEAR COUNTER VALUE, START COUNTER, STOP COUNTER, RESET COUNTER

GET AND CLEAR QUADRATURE VALUE — Digital Point

Function: To read and clear a quadrature counter value.

Typical Use: To read incremental encoders for positional or velocity measurement.

Details:
- Reads the current value of a quadrature counter and places it in the *Move To* parameter.
- Resets the counter at the I/O unit to zero.
- Does not stop the quadrature counter from continuing to count.
- Valid range is -2,147,483,648 to 2,147,483,647 counts.
- A positive value indicates forward movement (phase B leads phase A), and a negative value indicates reverse movement (phase A leads phase B).
- A quadrature counter occupies two adjacent channels. *Input module pairs specifically made for quadrature counting must be used.* The first channel must be an even channel number on the digital multifunction I/O unit. For example, positions 0 and 1, 4 and 5 are valid, but 1 and 2, 3 and 4 are not.

Arguments:

ARGUMENT 1	ARGUMENT 2
QUADRATURE COUNTER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET AND CLEAR QUADRATURE VALUE**

From	ENCODER #1	*digital input configured with quadrature feature*
Move To	TABLE POSITION	*variable integer (the quadrature count)*

Notes:
- The maximum encoder RPM will be related to the number of pulses per revolution that the encoder provides.
- Max Encoder RPM = (750,000 Pulses per Minute) / (Encoder Pulses [or lines] per Revolution)

Dependencies:
- Always use START QUADRATURE COUNTER once before using this command for the first time.
- Applies only to input channels configured with the quadrature feature on digital multifunction I/O units.

See Also: GET QUADRATURE VALUE, START QUADRATURE COUNTER, STOP QUADRATURE COUNTER, RESET QUADRATURE COUNTER

GET COUNTER VALUE — Digital Point

Function: To read digital input counter value.

Typical Use: To count pulses from turbine flow meters, magnetic pickups, encoders, proximity switches, etc.

Details:
- Reads the current value of a digital input counter and places it in the *Move To* parameter.
- Does *not* reset the counter at the I/O unit to zero.
- Does not stop the counter from continuing to count.
- Valid range is 0 to 4,294,967,296 counts.

Arguments:

ARGUMENT 1	ARGUMENT 2
COUNTER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET COUNTER VALUE**

From	BOTTLE COUNTER	*digital input configured with counter feature*
Move To	# OF BOTTLES	*variable float (the counter value)*

Notes:
- The maximum speed at which the counter can operate is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.
- Since 32-bit signed integers can only count up to 2,147,483,647, use a float to hold the counts if exceeding this amount.

Dependencies:
- Always use START COUNTER once before using this command for the first time.
- Applies only to inputs configured with the counter feature on digital multifunction I/O units.

See Also: GET AND CLEAR COUNTER VALUE, START COUNTER, STOP COUNTER, RESET COUNTER

GET FREQUENCY — Digital Point

Function: To read digital input frequency value.

Typical Use: To read the speed of rotating machinery, velocity encoders, etc.

Details:
- Reads the current frequency of a digital input and places it in the *Move To* parameter.
- Returns an integer value from 0 to 65,535 (see Notes below).
- Resolution is 1 Hertz.

Arguments:

ARGUMENT 1	ARGUMENT 2
FREQUENCY	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **GET FREQUENCY**

From	SHAFT PICKUP	*digital input configured with frequency feature*
Move To	MOTOR SPEED	*variable integer (the frequency)*

Notes:
- Since the resolution is 1 Hertz, significant errors may be encountered at frequencies less than 100 Hertz. Use GET PERIOD then divide 1 by the period to get the frequency with resolution to 0.2 Hertz at 60 Hertz.
- The maximum frequency that can be read is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the frequency feature on digital multifunction I/O units.

GET OFF-LATCH VALUE — Digital Point

Function: To read the state of an off-latch.

Typical Use: To ensure detection of an extremely brief on-to-off transition of a digital input.

Details:
- Reads an off-latch of a single digital input. Off-latches detect on-to-off input transitions that would otherwise occur too fast for the Mistic controller to detect, since they are processed locally by the digital multifunction I/O unit.
- Places the value read into the argument specified by the *Move To* parameter. The argument will contain the value -1 (True) if the latch is set and a 0 (False) if the latch is not set.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF LATCH	DIGITAL OUT
	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **GET OFF-LATCH VALUE**

Get	START BUTTON	*digital input configured with off-latch feature*
Move To	RELEASED	*variable float (the off-latch value)*

Notes:
- The ability to detect fast input transitions is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the off-latch feature on digital multifunction I/O units.

See Also: GET & CLEAR OFF-LATCH VALUE, CLEAR OFF-LATCH VALUE, CLEAR ALL LATCHES

GET OFF-PULSE MEAS — Digital Point

Function: To read the off time duration of a digital input that has had an on-off-on transition.

Typical Use: To shut down or process interlocking where a momentary pulse of a certain length is required.

Details:
- Gets the duration of the first complete off-pulse applied to the digital input.
- Measurement starts on the first on-to-off transition and stops on the first off-to-on transition.
- Returns a float value representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF PULSE MEAS.	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET OFF-PULSE MEAS**

From	OVERHEAT SWITCH	*digital input configured with off-pulse feature*
Move To	OFF TIME	*variable float (the duration of the pulse)*

Notes:
- Use GET OFF-PULSE MEAS COMP. STAT first to see if a complete off-pulse measurement has occurred.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the off-pulse measurement feature on digital multifunction I/O units.

See Also: GET OFF PULSE MEAS & RESTART, GET OFF-PULSE MEAS COMP. STAT

GET OFF-PULSE MEAS & RESTART — Digital Point

Function: To read and clear the off time duration of a digital input that has had an on-off-on transition.

Typical Use: To shut down or process interlocking where a momentary pulse of a certain length is required.

Details:
- Gets the duration of the first complete off-pulse applied to the digital input.
- Restarts the off-pulse measurement after reading the current value.
- Measurement starts on the first on-to-off transition and stops on the first off-to-on transition.
- Returns a float value representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.
- If used while a measurement is in progress, the measurement is terminated, the data is returned, and a new off-pulse measurement is started.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF PULSE MEAS.	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET OFF-PULSE MEAS & RESTART**

From	STANDBY SWITCH	*digital input configured with off-pulse feature*
Move To	OFF TIME	*variable float (the duration of the pulse)*

Notes:
- Use GET OFF-PULSE MEAS COMP. STAT first to see if a complete off-pulse measurement has occurred.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the off-pulse measurement feature on digital multifunction I/O units.

See Also: GET OFF PULSE MEAS, GET OFF-PULSE MEAS COMP STAT

GET OFF-PULSE MEAS COMP. STAT — Digital Point

Function: To read the completion status of an off-pulse measurement.

Typical Use: To determine that a complete measurement has occurred before reading the measurement.

Details:
- Gets the completion status of an off-pulse measurement and stores it in the *Move To* parameter. The argument will contain a -1 (True) if the measurement is complete or a 0 (False) if it is incomplete.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF PULSE MEAS.	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET OFF-PULSE MEAS COMP. STAT**

From	OVERHEAT SWITCH	*digital input configured with off-pulse feature*
Move To	PULSE COMPLETE	*variable integer (the completion status)*

Notes:
- Use this command to see if a complete off-pulse measurement has occurred. The command will not interfere with a current off-pulse measurement.
- Once the completion status is True, use GET OFF-PULSE MEAS or GET OFF-PULSE MEAS & RESTART to read the value.

Dependencies:
- Applies only to inputs configured with the off-pulse measurement feature on digital multifunction I/O units.

See Also: GET OFF PULSE MEAS, GET OFF-PULSE MEAS & RESTART

GET ON-LATCH VALUE — Digital Point

Function: To read the state of an on-latch.

Typical Use: To ensure detection of an extremely brief off-to-on transition of a digital input.

Details:
- Reads an on-latch of a single digital input. On-latches detect off-to-on input transitions that would otherwise occur too fast for the Mistic controller to detect, since they are processed locally by the digital multifunction I/O unit.
- Places the value read into the argument specified by the *Move To* parameter. The argument will contain the value -1 (True) if the latch is set and a 0 (False) if the latch is not set.

Arguments:

ARGUMENT 1	ARGUMENT 2
ON LATCH	DIGITAL OUT VARIABLE FLOAT VARIABLE INTEGER

Example: **GET ON-LATCH VALUE**

Get	ESTOP BUTTON	*digital input configured with on-latch feature*
Move To	EMERGENCY STOP	*variable float (the on-latch value)*

Notes:
- The ability to detect fast input transitions is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the on-latch feature on digital multifunction I/O units.

See Also: GET & CLEAR ON-LATCH VALUE, CLEAR ON-LATCH VALUE, CLEAR ALL LATCHES

GET ON-PULSE MEAS — Digital Point

Function: To read the on time duration of a digital input that has had an off-on-off transition

Typical Use: To shut down or process interlocking where a momentary pulse of a certain length is required.

Details:
- Gets the duration of the first complete on-pulse applied to the digital input.
- Measurement starts on the first off-to-on transition and stops on the first on-to-off transition.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.

Arguments:

ARGUMENT 1	ARGUMENT 2
ON PULSE MEAS.	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET ON-PULSE MEAS**

From	OVERSPEED SWITCH	*digital input configured with on-pulse feature*
Move To	ON TIME	*variable float (the duration of the pulse)*

Notes:
- Use GET ON-PULSE MEAS COMP. STAT first to see if a complete on-pulse measurement has occurred.
- The accuracy of the value returned is limited by the input module’s turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the on-pulse measurement feature on digital multifunction I/O units.

See Also: GET ON-PULSE MEAS & RESTART, GET ON-PULSE MEAS COMP. STAT

GET ON-PULSE MEAS & RESTART — Digital Point

Function: To read and clear the on time duration of a digital input that has had an off-on-off transition.

Typical Use: To shut down or process interlocking where a momentary pulse of a certain length is required.

Details:
- Gets the duration of the first complete on-pulse applied to the digital input.
- Restarts the on-pulse measurement after reading the current value.
- Measurement starts on the first off-to-on transition and stops on the first on-to-off transition.
- Returns a float value representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.
- If used while a measurement is in progress, the measurement is terminated, the data is returned, and a new on-pulse measurement is started.

Arguments:

ARGUMENT 1	ARGUMENT 2
ON PULSE MEAS.	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET ON-PULSE MEAS & RESTART**

From	STANDBY SWITCH	*digital input configured with on-pulse feature*
Move To	ON TIME	*variable float (the duration of the pulse)*

Notes:
- Use GET ON-PULSE MEAS COMP. STAT first to see if a complete on-pulse measurement has occurred.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the on-pulse measurement feature on digital multifunction I/O units.

See Also: GET ON-PULSE MEAS, GET ON-PULSE MEAS COMP. STAT

GET ON-PULSE MEAS COMP. STAT — Digital Point

Function: To read the completion status of an on-pulse measurement.

Typical Use: To determine that a complete measurement has occurred before reading the measurement.

Details:
- Gets the completion status of an on-pulse measurement and stores it in the *Move To* parameter. The argument will contain a -1 (True) if the measurement is complete or a 0 (False) if it is incomplete.

Arguments:

ARGUMENT 1	ARGUMENT 2
ON PULSE MEAS.	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **GET ON-PULSE MEAS COMP. STAT**

From	PRESSURE SWITCH	*digital input configured with on-pulse feature*
Move To	PULSE COMPLETE	*variable integer (the completion status)*

Notes:
- Use this command to see if a complete on-pulse measurement has occurred. The command will not interfere with a current on-pulse measurement.
- Once the completion status is True, use GET ON-PULSE MEAS or GET ON-PULSE MEAS & RESTART to read the value.

Dependencies:
- Applies only to inputs configured with the on-pulse measurement feature on digital multifunction I/O units.

See Also: GET ON-PULSE MEAS, GET ON-PULSE MEAS & RESTART

GET PERIOD — Digital Point

Function: To read the elapsed time during an on-off-on or an off-on-off transition of a digital input.

Typical Use: To measure the period of a slow shaft rotation.

Details:
- Measurement starts on the first transition (either off-to-on or on-to-off) and stops on the next transition of the same type (one complete cycle).
- Does not restart the period measurement.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.

Arguments:

ARGUMENT 1	ARGUMENT 2
PERIOD	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET PERIOD**

From	SHAFT INPUT	*digital input configured with period feature*
Move To	SHAFT CYCLE	*variable float (the period value)*

Notes:
- This command measures the first complete period only. No period measurement is performed after the first measurement until the GET PERIOD & RESTART command is used.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- The GET PERIOD & RESTART command must be used to start the measurement.
- Applies only to inputs configured with the period feature on digital multifunction I/O units.

See Also: GET PERIOD & RESTART

GET PERIOD & RESTART — Digital Point

Function: To read and clear the elapsed time during an on-off-on or an off-on-off transition of a digital input

Typical Use: To measure the period of a slow shaft rotation.

Details:
- Reads the period value of a digital input and places it in the argument specified by the *Move To* parameter.
- Measurement starts on the first transition (either off-to-on or on-to-off) and stops on the next transition of the same type (one complete cycle).
- Restarts the period measurement after reading.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.

Arguments:

ARGUMENT 1	ARGUMENT 2
PERIOD	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET PERIOD & RESTART**

From	SHAFT INPUT	*digital input configured with period feature*
Move To	SHAFT CYCLE	*variable integer (the period value)*

Notes:
- This command should be used to start the period measurement.
- This command measures the first complete period only. No period measurement is performed after the first measurement until another GET PERIOD & RESTART command is used.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the period feature on digital multifunction I/O units.

See Also: GET PERIOD

GET PERIOD MEAS COMP. STAT — Digital Point

Function: To read the completion status of a period measurement.

Typical Use: To determine that a complete measurement has occurred before reading the measurement.

Details:
- Gets the completion status of a period measurement and stores it in the *Move To* parameter. The argument will contain a -1 (True) if the measurement is complete or a 0 (False) if it is incomplete.

Arguments:

ARGUMENT 1	ARGUMENT 2
PERIOD	VARIABLE FLOAT
	VARIABLE INTEGER

Example: **GET PERIOD MEAS COMP. STAT**

From	OVERHEAT SWITCH	*digital input configured with period feature*
Move To	PULSE COMPLETE	*variable integer (the completion status)*

Notes:
- Use this command to see if a complete period measurement has occurred. The command will not interfere with a current period measurement.
- Once the completion status is True, use GET PERIOD or GET PERIOD & RESTART to read the value.

Dependencies:
- Applies only to inputs configured with the period measurement feature on digital multifunction I/O units.

See Also: GET PERIOD, GET PERIOD & RESTART

GET QUADRATURE VALUE — Digital Point

Function: To read a quadrature counter value.

Typical Use: To read incremental encoders for positional or velocity measurement.

Details:
- Reads the current value of a quadrature counter and places it in an argument specified by the *Move To* parameter.
- Does not reset the counter at the I/O unit to zero.
- Does not stop the quadrature counter from continuing to count.
- Valid range is -2,147,483,648 to 2,147,483,647 counts.
- A positive value indicates forward movement (phase B leads phase A) and a negative value indicates reverse movement (phase A leads phase B).
- A quadrature counter occupies two adjacent channels. *Input module pairs specifically made for quadrature counting must be used.* The first channel must be an even channel number on the digital multifunction I/O unit. For example, positions 0 and 1, 4 and 5 are valid, but 1 and 2, 3 and 4 are not.

Arguments:

ARGUMENT 1	ARGUMENT 2
QUADRATURE COUNTER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET QUADRATURE VALUE**

From	ENCODER #1	*digital input configured with quadrature feature*
Move To	TABLE POSITION	*variable integer (the quadrature count)*

Notes:
- The maximum encoder RPM will be related to the number of pulses per revolution that the encoder provides.
- Max Encoder RPM = (750,000 Pulses per Minute) /)Encoder Pulses [or lines] per Revolution)

Dependencies:
- Always use START QUADRATURE COUNTER once before using this command for the first time.
- Applies only to input channels configured with the quadrature feature on digital multifunction I/O units.

See Also: GET AND CLEAR QUADRATURE VALUE, START QUADRATURE COUNTER, STOP QUADRATURE COUNTER, RESET QUADRATURE COUNTER

GET TOTALIZE OFF VALUE — Digital Point

Function: To read digital input total off time.

Typical Use: To accumulate total off time of a device to possibly indicate down time.

Details:
- Reads the accumulated off time of a digital input since it was last reset.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.
- Does not reset the total.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF TIME TOTALIZER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET TOTALIZE OFF VALUE**

From	HEATER OUTPUT	*digital input configured with totalize-off feature*
Move To	HEATER DOWN TIME	*variable float (the total off time)*

Notes:
- To ensure the totalizer is cleared at start-up, use GET/RESTART TOTALIZE OFF VAL. once before using this command for the first time.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the totalize-off feature on digital multifunction I/O units.

See Also: GET/RESTART TOTALIZE OFF VAL.

GET TOTALIZE ON VALUE — Digital Point

Function: To read digital input total on time.

Typical Use: To accumulate total on time of a device.

Details:
- Reads the accumulated on time of a digital input since it was last read.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.
- Does not reset the total.

Arguments:

ARGUMENT 1	ARGUMENT 2
ON TIME TOTALIZER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET TOTALIZE ON VALUE**

From	PUMP POWER	*digital input configured with totalize-on feature*
Move To	PUMP RUNTIME	*variable float (the total on time)*

Notes:
- To ensure the totalizer is cleared at start-up, use GET/RESTART TOTALIZE ON VAL. once before using this command for the first time.
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies:
- Applies only to inputs configured with the totalize-on feature on digital multifunction I/O units.

See Also: GET/RESTART TOTALIZE ON VAL.

GET/RESTART TOTALIZE OFF VAL. — Digital Point

Function: To read digital input total off time and restart.

Typical Use: To accumulate total off time of a device to possibly indicate down-time.

Details:
- Reads the accumulated off time of a digital input since it was last reset.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Resets the total to zero after execution.
- Maximum duration is 4.97 days.

Arguments:

ARGUMENT 1	ARGUMENT 2
OFF TIME TOTALIZER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET/RESTART TOTALIZE OFF VAL.**

From	POWER STATUS	*digital input configured with totalize-off feature*
Move To	SYSTEM DOWN TIME	*variable integer (the total off time)*

Notes:
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.
- Use GET TOTALIZE OFF VALUE to read the totalized value without resetting it.

Dependencies:
- Applies only to inputs configured with the totalize-off feature on digital multifunction I/O units.

See Also: GET TOTALIZE OFF VALUE

GET/RESTART TOTALIZE ON VAL. — Digital Point

Function: To read digital input total on time and restart.

Typical Use: To accumulate total on time of a device.

Details:
- Reads the accumulated on time of a digital input since it was last reset.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Resets the total to zero after execution.
- Maximum duration is 4.97 days.

Arguments:

ARGUMENT 1	ARGUMENT 2
ON TIME TOTALIZER	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET/RESTART TOTALIZE ON VAL.**

From	CIRC MOTOR PWR	*digital input configured with totalize-on feature*
Move To	MOTOR RUNTIME	*variable integer (the total off time)*

Notes:
- The accuracy of the value returned is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.
- Use GET TOTALIZE ON VALUE to read the totalized value without resetting it.

Dependencies:
- Applies only to inputs configured with the totalize-on feature on digital multifunction I/O units.

See Also: GET TOTALIZE ON VALUE

PULSE OFF — Digital Point

Function: To turn off a digital output for a specified time or to delay turning it on.

Typical Uses:
- To serve as an alternative to the TURN ON command.
- To “reset” another device.

Details:
- Same as using TURN OFF followed by a delay followed by TURN ON, or if the output was off already, same as a delay followed by TURN ON.
- After the off time expires, this command leaves the channel on.
- The time may be specified from 0.0005 to 429,496.7000 seconds (4.97 days), with a resolution of 100 microseconds.
- During the execution of this command, if another PULSE OFF is performed, the current off-pulse is canceled and the new off-pulse is generated.
- The output does not have to be configured with a feature to use this command.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	SMART DIGITAL OUT
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **PULSE OFF**

Seconds	RESET TIME	*time the channel is to remain off*
To	PUMP #2 STOP	*any digital output channel (no feature required)*

Notes:
- A TURN ON command may be used to abort an off-pulse before the end of the off time.
- The minimum off time is 0.0005 seconds; however, the digital output module’s minimum turn-on and turn-off times may be greater. Check the specifications for the module to be used.
- *Caution:* If this command is used more frequently than the specified delay, the output will remain off.

Dependencies:
- Applies only to outputs on digital multifunction I/O units.

See Also: PULSE ON, TURN OFF, TURN ON

PULSE ON — Digital Point

Function: To turn on a digital output for a specified period or to delay turning it off.

Typical Uses:
- As an alternative to the TURN OFF command.
- To “reset” another device.
- To increment a counter.
- To latch devices connected to digital outputs that require a minimum pulse duration to latch, such as motor starters and latching relays.

Details:
- Same as using TURN ON followed by a delay followed by TURN OFF, or if the output was on already, same as a delay followed by TURN OFF.
- After the on time expires, this command leaves the channel off.
- The time may be specified from 0.0005 to 429,496.7000 seconds (4.97 days), with a resolution of 100 microseconds.
- During the execution of this command, if another PULSE ON is performed, the current on-pulse is canceled and the new On-pulse is generated.
- The output does not have to be configured with a feature to use this command.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	SMART DIGITAL OUT
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **PULSE ON**

Seconds	MIN LATCH TIME	*time the channel is to remain on*
To	PUMP #2 RUN	*any digital output channel (no feature required)*

Notes:
- A TURN OFF command may be used to abort an on pulse before the end of the on time.
- The minimum on time is 0.0005 seconds; however, the digital output module’s minimum turn-on and turn-off times may be greater. Check the specifications for the module to be used.
- *Caution:* If this command is used more frequently than the specified delay, the output will remain on.

Dependencies:
- Applies only to outputs on digital multifunction I/O units.

See Also: PULSE OFF, TURN OFF, TURN ON

RESET COUNTER — Digital Point

Function: To reset a digital input counter to zero.

Typical Use: To reset a digital input configured with a counter feature.

Details:
- Resets the specified counter input to zero as soon as it is used.
- Does not stop the counter from continuing to run (as STOP COUNTER does).

Arguments:

ARGUMENT 1
COUNTER

Example: **RESET COUNTER**

BOTTLE COUNTER	*digital input configured with counter feature*

Dependencies:
- Applies only to inputs configured with the counter feature on digital multifunction I/O units.

See Also: GET COUNTER VALUE, GET AND CLEAR COUNTER VALUE, START COUNTER, STOP COUNTER

RESET QUADRATURE COUNTER — Digital Point

Function: To reset a quadrature counter to zero.

Typical Use: To reset a quadrature counter used with incremental encoders.

Details:
- Resets the specified quadrature counter to zero as soon as it is used.
- Does not stop the quadrature counter from continuing to count.
- A quadrature counter occupies two adjacent channels. *Input module pairs specifically made for quadrature counting must be used.* The first channel must be an even channel number on the digital multifunction I/O unit. For example, positions 0 and 1, 4 and 5 are valid, but 1 and 2, 3 and 4 are not.

Arguments:

ARGUMENT 1
QUADRATURE COUNTER

Example: **RESET QUADRATURE COUNTER**

ENCODER #1	*digital input configured with quadrature feature*

Dependencies:
- Applies only to input channels configured with the quadrature feature on digital multifunction I/O units.

See Also: GET QUADRATURE VALUE, GET AND CLEAR QUADRATURE VALUE, START QUADRATURE COUNTER, STOP QUADRATURE COUNTER

SET TIME PROP OUTPUT — Digital Point

Function: To set the time proportional output (TPO) period of an output channel.

Typical Use: To vary the percentage of on time (duty cycle). Commonly used to control heater outputs in a pseudo-analog fashion.

Details:

- Sets the period of a TPO to the specified value.
- The period is specified from 0.1 to 429,496.7000 seconds (4.97 days), with a resolution of 100 microseconds.
- This command must be used before the SET TIME PROP PERCENT command.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	TIME PROP. OUTPUT
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET TIME PROP OUTPUT**

Period	60.0	*time proportion period*
To	HEATER OUTPUT	*digital output configured with TPO feature*

Notes:

- The time proportion period only specifies the total time the output is varied over. SET TIME PROP PERCENT sets the on and off time within this period. For example, a TPO period of 30 seconds and an output of 25 percent will cause the output channel to go on for 7.5 seconds (30 seconds x .25) and off for 22.5 seconds at 30-second intervals.
- Although the minimum TPO period is 0.1 seconds (and the resolution is 100 microseconds), at low percentages the minimum turn-on and turn-off times of the digital output module may be greater. Check the specifications for the module to be used.
- To ensure that the TPO period will always be correct, store this and other changeable I/O unit values in permanent memory at the I/O unit. (You can do so through the Debugger.)
- If the TPO period is not stored in permanent memory at the I/O unit, use this command immediately before SET TIME PROP PERCENT every time. This ensures that the TPO period will be configured properly if the I/O unit has experienced loss of power. However, do not issue these commands more frequently than necessary, since this can be counterproductive.

Dependencies:

- Applies only to output channels configured with the TPO feature on digital multifunction I/O units.

See Also: SET TIME PROP PERCENT

SET TIME PROP PERCENT — Digital Point

Function: To set the on time of an output channel as a percentage.

Typical Use: To vary the net output percentage over time. Commonly used to control heater outputs in a pseudo-analog fashion.

Details:
- Sets the percentage of on time for an output configured as a TPO.
- Valid range is 0 (always off) to 100 (always on).
- A TPO period of 10 seconds and an output of 20 percent will cause the output channel to go on for 2.0 seconds (10 seconds x .20) and off for 8.0 seconds at 10-second intervals.
- Changes to the output percentage take effect at the beginning of the next period.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	TIME PROP. OUTPUT
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET TIME PROP PERCENT**

Percent	NEW OUTPUT	*percentage output*
To	HEATER OUTPUT	*digital output configured with TPO feature*

Notes:
- When using the output of a PID to drive a digital TPO, scale the analog output channel (for the PID) to 0–100. (This analog channel does not have to exist physically, but must be one of the 16 channels on the I/O unit). Use MOVE to copy the PID analog output value to the digital TPO channel periodically.
- At low percentages, the output module's minimum turn-on and turn-off times may affect the accuracy of control. Check the specifications for the module to be used.

Dependencies:
- A SET TIME PROP OUTPUT command must be used at least once before this command to define the time period.
- Applies only to output channels configured with the TPO feature on digital multifunction I/O units.

See Also: SET TIME PROP OUTPUT

START CONTINUOUS SQUARE WAVE — Digital Point

Function: To generate a square wave on an output channel.

Typical Use: To drive stepper motor controllers, pulse indicator lamps, or horns or counters connected to digital outputs.

Details:

- Generates a digital waveform on the specified digital output channel. *On Time* specifies the amount of time in seconds that the channel will remain on during each pulse; *Off Time* specifies the amount of time the channel will remain off.
- The minimum *On Time* and *Off Time* is 0.001 second with a resolution of 0.0001 second, making the maximum frequency 500 Hertz.
- The maximum *On Time* and *Off Time* is 429,496.7000 seconds (4.97 days on, 4.97 days off).
- Timing begins with the off state. If a square wave is already running when this command is used, the new timing will become effective on the next transition (on-to-off or off-to-on).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	SMART DIGITAL OUT
CONSTANT INTEGER	CONSTANT INTEGER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	

Example: **START CONTINUOUS SQUARE WAVE**

On Time	0.100	*on duration of one pulse (in seconds)*
Off Time	0.500	*off duration of one pulse (in seconds)*
To	BLINKING LAMP	*digital output*

Notes:

- Once the pulse train has started, the digital I/O unit maintains the waveform indefinitely.
- Use only to start or change the square wave.
- To stop a currently executing pulse train, use TURN OFF.
- The minimum on or off time is 0.001 second; however, the digital output module's minimum turn-on and turn-off times may be greater. Check the specifications for the module to be used.

Dependencies:

- Applies only to outputs on digital multifunction I/O units.

See Also: TURN OFF, GENERATE N PULSES

START COUNTER — Digital Point

Function: To activate a digital input counter.

Typical Use: Once at the beginning of a program to activate a digital input counter.

Details:
- Must be used to activate the specified counter input.
- Does not reset the counter to zero.
- Retains any previously accumulated counts.

Arguments:

ARGUMENT 1
COUNTER

Example: **START COUNTER**

BAGGAGE COUNTER	*digital input configured with counter feature*

Notes:
- To keep a counter active after a power failure at the I/O unit, use the Debugger to write or "burn" the current I/O unit configuration to EEPROM after the counter is started.
- Use RESET COUNTER to clear a counter to zero.

Dependencies:
- Applies only to inputs configured with the counter feature on digital multifunction I/O units.

See Also: GET COUNTER VALUE, GET AND CLEAR COUNTER VALUE, RESET COUNTER, STOP COUNTER

START QUADRATURE COUNTER — Digital Point

Function: To activate a digital input quadrature counter.

Typical Use: Once at the beginning of a program to activate a quadrature counter.

Details:
- Must use to activate the specified quadrature counter as soon as it is used.
- Does not reset the quadrature counter to zero.
- Retains any previously accumulated counts.
- A quadrature counter occupies two adjacent channels. *Input module pairs specifically made for quadrature counting must be used.* The first channel must be an even channel number on the digital multifunction I/O unit. For example, positions 0 and 1, 4 and 5 are valid, but 1 and 2, 3 and 4 are not.

Arguments:

ARGUMENT 1
QUADRATURE COUNTER

Example: **START QUADRATURE COUNTER**

ENCODER #1	*digital input configured with quadrature feature*

Notes:
- Before using a quadrature counter, you must activate it with the START QUADRATURE COUNTER command or no additional counts will accumulate.
- Use RESET QUADRATURE COUNTER to set the counts to zero.

Dependencies:
- Applies only to input channels configured with the quadrature feature on digital multifunction I/O units.

See Also: GET QUADRATURE VALUE, GET AND CLEAR QUADRATURE VALUE, RESET QUADRATURE COUNTER, STOP QUADRATURE COUNTER

STOP COUNTER — Digital Point

Function: To deactivate a digital input counter.

Typical Use: To inhibit a counter until further notice.

Details:
- Deactivates the specified counter.
- Stops counting incoming pulses to the digital input channel until START COUNTER is used.
- Does not reset the counter to zero.
- Retains any previously accumulated counts.

Arguments:

ARGUMENT 1
COUNTER

Example: **STOP COUNTER**

BEAN COUNTER	*digital input configured with counter feature*

Notes:
- Use RESET COUNTER to set counts to zero.

Dependencies:
- Applies only to inputs configured with the counter feature on digital multifunction I/O units.

See Also: GET COUNTER VALUE, GET AND CLEAR COUNTER VALUE, RESET COUNTER, START COUNTER

STOP QUADRATURE COUNTER — Digital Point

Function: To deactivate a quadrature counter.

Typical Use: To inhibit a quadrature counter until further notice.

Details:
- Stops the specified quadrature counter.
- Stops counting incoming quadrature pulses until START QUADRATURE COUNTER is used.
- Does not reset the quadrature counter to zero.
- Retains any previously accumulated counts.
- A quadrature counter occupies two adjacent channels. *Input module pairs specifically made for quadrature counting must be used.* The first channel must be an even channel number on the digital multifunction I/O unit. For example, positions 0 and 1, 4 and 5 are valid, but 1 and 2, 3 and 4 are not.

Arguments:

ARGUMENT 1
QUADRATURE COUNTER

Example: **STOP QUADRATURE COUNTER**

TABLE POSITION	*digital input configured with quadrature feature*

Notes:
- Use RESET QUADRATURE COUNTER to set quadrature counts to zero.

Dependencies:
- Applies only to input channels configured with the quadrature feature on digital multifunction I/O units.

See Also: GET QUADRATURE VALUE, GET AND CLEAR QUADRATURE VALUE, RESET QUADRATURE COUNTER, START QUADRATURE COUNTER

TURN OFF — Digital Point

Function: To turn off a digital output channel.

Typical Use: To deactivate devices connected to digital outputs, such as motors, pumps, lights, etc.

Details:
- Turns off the specified output.
- Discontinues any previously executing pulse, square wave, or TPO command immediately.
- The output will remain off until directed otherwise.

Arguments:

ARGUMENT 1
DIGITAL OUT

Example:

TURN OFF

THE LIGHTS	*any digital output channel (no feature required)*

Notes:
- Use MOVE to cause an output on one I/O unit to assume the state of an input on another I/O unit.
- Use NOT to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Use event/reactions to cause an output to track an input on the same digital multifunction I/O unit.
- *Speed Tip:* Use DO BINARY WRITE (with a value of 0) or DO BINARY DEACTIVATE (with a value of -1) to turn off all 16 outputs at once.

Dependencies:
- If the output channel or the I/O unit is disabled, no action will occur at the output channel (XVAL). The IVAL, however, will be updated.
- Applies to all digital outputs on digital multifunction I/O units and local simple I/O units.

See Also: DO BINARY WRITE, DO BINARY DEACTIVATE, PULSE ON, PULSE OFF, TURN ON

TURN ON — Digital Point

Function: To turn on a digital output channel.

Typical Use: To activate devices connected to digital outputs, such as motors, pumps, lights, etc.

Details:
- Turns on the specified output.
- Discontinues any previously executing pulse, square wave, or TPO command immediately.
- The output will remain on until directed otherwise.

Arguments:

ARGUMENT 1
DIGITAL OUT

Example: **TURN ON**

INLET VALVE	*any digital output channel (no feature required)*

Notes:
- Use MOVE to cause an output on one I/O unit to assume the state of an input on another I/O unit.
- Use NOT to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Use event/reactions to cause an output to track an input on the same digital multifunction I/O unit.
- *Speed Tip:* Use DO BINARY WRITE (with a value of -1) or DO BINARY ACTIVATE (with a value of -1) to turn on all 16 outputs at once.

Dependencies:
- If the output channel or the I/O unit is disabled, no action will occur at the output channel (XVAL). The IVAL, however, will be updated.
- Applies to all outputs on digital multifunction I/O units and local simple I/O units.

See Also: DO BINARY WRITE, DO BINARY ACTIVATE, PULSE ON, PULSE OFF, TURN OFF

EVENT/REACTION OPERATIONS

CLEAR ALL EVENT LATCHES — Event/Reaction

Function: To reset all 256 event latches on the I/O unit.

Typical Use: In the POWERUP chart, to reset all event latches on the I/O unit to a known or default state.

Details:
- Each event sets a latch at the moment its criteria is True. This command resets all latches.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT
DIGITAL MF I/O UNIT

Example: **CLEAR ALL EVENT LATCHES**

On I/O Unit	ESTOP BUTTONS	*name of digital multifunction I/O unit*

Notes:
- Use with care since this command will erase the history of all event latches.
- Normally CLEAR EVENT LATCH is used to reset a single event latch after it has been evaluated.

Dependencies:
- Event/reactions are not supported on local simple I/O units.

See Also: CLEAR EVENT LATCH

CLEAR EVENT LATCH — Event/Reaction

Function: To reset a specified event latch on the I/O unit.

Typical Use: After an event has been evaluated.

Details:
- To determine that a specified event has occurred, the event latch must be checked. One way to check the event latch is to use the condition HAS EVENT OCCURRED?. To detect the next incident of the event, the event latch must be reset using this command.

Arguments:

ARGUMENT 1
ANALOG E/R DIGITAL E/R

Example: **CLEAR EVENT LATCH**

Event/Reaction	ESTOP BUTTON 1	*name of the event/reaction*

Notes:
- Always use after CLEAR I/O UNIT INTERRUPT (if using interrupts).

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: CLEAR I/O UNIT INTERRUPT, CLEAR ALL EVENT LATCHES, HAS EVENT OCCURRED?

CLEAR I/O UNIT INTERRUPT — Event/Reaction

Function: To reset the interrupt latch, which turns off the interrupt line on the I/O unit.

Typical Use: In the INTERRUPT chart, to reset the interrupt latch immediately after determining that an I/O unit has generated an interrupt.

Details:
- Resets the interrupt latch to off.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT DIGITAL MF I/O UNIT

Example: **CLEAR I/O UNIT INTERRUPT**

I/O UNIT	ESTOP BUTTONS	*name of digital multifunction I/O unit*

Notes:
- Use GENERATING INTERRUPT? to determine if a specified I/O unit has generated an interrupt.
- Clear the interrupt first, then check all event latches, to ensure that a new event latch will generate a new interrupt.

Dependencies:
- Event/reactions are not supported on local simple I/O units.

See Also: GENERATING INTERRUPT?, HAS EVENT OCCURRED?, CLEAR EVENT LATCH

DISABLE EVENT SCANNING — Event/Reaction

Function: To deactivate all event/reactions on the specified I/O unit.

Typical Use: To shut off all event/reactions during a planned shutdown or an emergency stop.

Details:
- Disables the scanning of all event/reactions, directing the I/O unit to stop looking for any events. No logic is executed; no reaction occurs.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT
DIGITAL MF I/O UNIT

Example: **DISABLE EVENT SCANNING**

On I/O Unit	OVERTEMP SENSORS	*name of digital multifunction I/O unit*

Notes:
- To stop a specific event/reaction, use DISABLE SCAN FOR EVENT.

Dependencies:
- Event/reactions are not supported on local simple I/O units.

See Also: DISABLE SCAN FOR EVENT, ENABLE SCAN FOR EVENT, ENABLE EVENT SCANNING

DISABLE EVENT/REACTION — Event/Reaction

Function: To disable communication between the program in the Mistic controller and the specified event/reaction.

Typical Use: To disconnect the program from a specified event/reaction for simulation and program testing.

Details:
- All event/reaction communication is enabled by default.
- Does not affect the event/reaction at the I/O unit in any way. While communication to the event/reaction is disabled, any Cyrano command that refers to it by name will not affect it because the command only has access to the IVAL.
- If the event/reaction is disabled and it's active, reactions *will* occur. If an interrupt is enabled, it will try to interrupt the Mistic controller. However, the program in the Mistic controller will not be able to read or clear any status bits associated with the event/reaction until it is enabled (see ENABLE EVENT/REACTION).

Arguments:

ARGUMENT 1
ANALOG E/R DIGITAL E/R

Example: **DISABLE EVENT/REACTION**

ESTOP BUTTON 1	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- To actually stop an event/reaction, use DISABLE SCAN FOR EVENT.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: ENABLE EVENT/REACTION

DISABLE INTERRUPT ON EVENT — Event/Reaction

Function: To disable interrupt notification for a specified event/reaction.

Typical Use: To accommodate situations where the specified event/reaction is still needed but the interrupt notification is not.

Details:
- See the Event/Reaction Overview in Chapter 1 for important information.

Arguments:

ARGUMENT 1
ANALOG E/R DIGITAL E/R

Example: **DISABLE INTERRUPT ON EVENT**

Event/Reaction	ESTOP BUTTON 1	*name of the event/reaction*

Notes:
- To disable both the interrupt notification and the event/reaction, use DISABLE SCAN FOR EVENT.

Dependencies:
- Event/reactions must be configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: ENABLE INTERRUPT ON EVENT, DISABLE SCAN FOR EVENT

DISABLE SCAN FOR EVENT — Event/Reaction

Function: To deactivate a specific event/reaction.

Typical Use: To shut off a specific event/reaction during a planned shutdown or an emergency stop.

Details:
- Disables the scanning of an event/reaction, directing the I/O unit to stop looking for the event. No logic is executed; no reaction occurs.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **DISABLE SCAN FOR EVENT**

Event/Reaction	ESTOP BUTTON 1	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- To disable all event/reactions, use DISABLE EVENT SCANNING.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: DISABLE EVENT SCANNING, ENABLE SCAN FOR EVENT, ENABLE EVENT SCANNING

ENABLE EVENT/REACTION — Event/Reaction

Function: To enable communication between the program in the Mistic controller and the specified event/reaction.

Typical Use: To reconnect the program to a specified event/reaction after simulation and program testing.

Details:
- All event/reaction communication is enabled by default.
- Does not affect the event/reaction at the I/O unit in any way.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **ENABLE EVENT/REACTION**

ESTOP BUTTON 1	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- To enable all event/reactions, use ENABLE EVENT SCANNING.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: ENABLE EVENT/REACTION, ENABLE EVENT SCANNING

ENABLE EVENT SCANNING — Event/Reaction

Function: To activate all event/reactions on the specified I/O unit.

Typical Use: To reactivate all event/reactions after a planned shutdown or an emergency stop.

Details:
- Whenever scanning for event/reactions is started, all events found to be True on the first scan will be considered to have just occurred. Therefore, the reactions will follow.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT
DIGITAL MF I/O UNIT

Example: **ENABLE EVENT SCANNING**

On I/O Unit	OVERTEMP SENSORS	*name of digital multifunction I/O unit*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- To activate a specific event/reaction, use ENABLE SCAN FOR EVENT.
- Normally used after DISABLE EVENT SCANNING.

Dependencies:
- Event/reactions are not supported on local simple I/O units.

See Also: DISABLE SCAN FOR EVENT, ENABLE SCAN FOR EVENT, DISABLE EVENT SCANNING

ENABLE INTERRUPT ON EVENT — Event/Reaction

Function: To activate interrupt notification for a specified event/reaction.

Typical Use: To provide interrupt notification to the Mistic program so it can resume.

Details:
- The event/reaction must be active (scanning enabled) for the interrupt to work.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **ENABLE INTERRUPT ON EVENT**

Event/Reaction	ACID TANK 1 HIGH LEVEL	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- Use ENABLE EVENT/REACTION to enable a disabled event/reaction.

Dependencies:
- Event/reactions must be configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: DISABLE INTERRUPT ON EVENT, DISABLE SCAN FOR EVENT

ENABLE SCAN FOR EVENT — Event/Reaction

Function: To activate a specific event/reaction.

Typical Use: To reactivate a specific event/reaction after a planned shutdown.

Details:
- If the event is found to be True when scanning for an event/reaction is started, the reaction will occur.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **ENABLE SCAN FOR EVENT**

Event/Reaction	ACID TANK 1 HIGH LEVEL	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- To activate all event/reactions, use ENABLE EVENT SCANNING.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: ENABLE EVENT SCANNING, ENABLE SCAN FOR EVENT, ENABLE EVENT SCANNING

READ E/R HOLD BUFFER — Event/Reaction

Function: To get a value that was stored at the I/O unit as a reaction to a specific event.

Typical Use: To capture a counter value at the moment a digital input turned on (or off).

Details:
- There are 256 32-bit holding buffers, one for each event/reaction. If a channel is configured as a counter and the reaction is to send its value to the hold buffer, the counts will be in the hold buffer for the specified event/reaction.
- Other values, such as period measurements and analog inputs, may also be captured.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG E/R	VARIABLE FLOAT
DIGITAL E/R	VARIABLE INTEGER

Example: **READ E/R HOLD BUFFER**

Event/Reaction	SEQUENCE FINISHED	*name of the event/reaction*
Put Result In	COUNTER VALUE	*variable integer*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- Use HAS EVENT OCCURRED? to determine if there is a value to be read.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

GENERAL PURPOSE OPERATIONS

CALCULATE STRATEGY CRC — General Purpose

Function: Calculates and returns a 16-bit CRC on the strategy that is currently resident in the controller.

Typical Use: Periodically used in an error handler to check the integerity of the running program.

Details:
- Use the result to compare with the original CRC that was automatically calculated during the last download. The original CRC is obtained by using RETRIEVE STRATEGY CRC. These two values should match exactly.

Arguments:

ARGUMENT 1
VARIABLE INTEGER

Example: **CALCULATE STRATEGY CRC**

Put Result In	NEW_CRC_CALC	*variable integer*

Notes:
- This command could take several minutes to execute when 30 tasks are running and the program is very large. Therefore, do not use it in a chart where timing is critical.

See Also: RETRIEVE SAVED CRC

CLEAR ALL ERRORS — General Purpose

Function: To clear the error queue in the controller.

Typical Use: To clear all errors from a full error queue.

Details:
- This function clears all errors in the queue. Normally this is never done. If the user program performs error checking, it will eventually clear the error queue. If no error checking is done, simply let the queue fill up.

Arguments: None.

Example: **CLEAR ALL ERRORS**

Notes:
- Performing a download and run does an automatic CLEAR ALL ERRORS.

See Also: GET ERROR CODE, GET ERROR COUNT, POINT TO NEXT ERROR

DELAY (MSEC) — General Purpose

Function: To slow the execution of program logic and to release the remaining time of a chart's time slice.

Typical Use: To cause a chart to give up the remaining time of its time slice.

Details:
- Units are in milliseconds.
- When this command is used, the chart is suspended immediately, since it would be inefficient to utilize CPU time just to wait.
- The chart is continued automatically at the DELAY (MSEC) command at its next scheduled time in the 32-task queue. If the delay has not expired, the suspend/continue cycle continues.
- The actual minimum delay is usually greater than 1 millisecond and is a function of how many tasks are running concurrently. For example, if there are 10 tasks running, each with a priority of 1, the minimum delay would be 10 x 1 x 0.5 milliseconds = 5 milliseconds.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **DELAY (MSEC)**

1	*constant integer (number of milliseconds)*

Notes:
- For readability, use DELAY (SEC) for delays longer than 10 seconds.
- When high accuracy is needed, reduce the number of tasks running concurrently.
- *Speed Tip:* Use this command in an operation block connected to the False exit of CHARACTERS WAITING? to give up the time slice while waiting. Connect the DELAY (MSEC) operation block back to the CHARACTERS WAITING? condition block.

Dependencies:
- Minimum time is increased as the number of concurrent tasks increases.

Error Codes: Queue error 33 = Overflow error — delay value larger than 2,147,483,647

See Also: DELAY (SEC)

DELAY (SEC) General Purpose

Function: To slow the execution of program logic and to release the remaining time of a chart's time slice.

Typical Use: To pause logic execution in a chart.

Details:
- Units are in seconds with millisecond resolution.
- When this command is used, the chart is suspended immediately, since it would be inefficient to utilize CPU time just to wait.
- The chart is continued automatically at the DELAY (SEC) command at its next scheduled time in the 32-task queue. If the delay has not expired, the suspend/continue cycle continues.
- The actual minimum delay is usually greater than 1 millisecond and is a function of how many tasks are running concurrently. For example, if there are 10 tasks running, each with a priority of 1, the minimum delay would be 10 x 1 x 0.5 milliseconds = 5 milliseconds.

Arguments:

ARGUMENT 1
CONSTANT FLOAT VARIABLE FLOAT

Example: **DELAY (SEC)**

10.525	*constant float (number seconds to delay)*

Notes:
- Use DELAY (MSEC) for delays shorter than 10 seconds.
- When high accuracy is needed, reduce the number of tasks running concurrently.

Dependencies:
- Minimum time is increased as the number of concurrent tasks increases.

See Also: DELAY (MSEC)

GET ERROR CODE — General Purpose

Function: To return the oldest error code in the error queue.

Typical Use: To allow a chart to perform error handling.

Details:
- Returns a zero if the queue is empty.
- The same error code is read each time unless POINT TO NEXT ERROR is used first.
- The error queue can hold up to 64 errors.
- The following is a list of errors that can appear in the error queue:

CODE	ERRORS FROM I/O UNITS (BRICKS)	CODE	ERRORS FROM MISTIC CONTROLLER
1	Undefined command	31	Send timeout; Mistic couldn't send message
2	Bad CRC or checksum	32	Bad table index value
3	Buffer overrun	33	Arithmetic overflow
4	I/O unit has powered up since last access	35	Not a real number
5	Incorrect command length	36	Division by zero
6	Communication watchdog timeout	38	Processor failure or factory software fault
7	Specified data invalid	39	Port already in use
8	Busy error	40	E/R does not have a "read &hold" reaction
9	Command & channel configuration mismatch	41	Invalid E/R hold buffer at I/O unit (brick)
10	Invalid event type	42	ARCNET port busy
11	Invalid time for TPO, sq. wave or pulse	43	Host relock
29	I/O unit response timeout	44	Invalid board type
30	Invalid serial port number	45	String too short to hold data

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET ERROR CODE**

Move to	ERROR CODE	*variable integer*

Notes:
- Use POINT TO NEXT ERROR to drop the oldest error from the queue so the next error can be evaluated.
- Use the Debugger to view the error queue for detailed information.

See Also: CLEAR ALL ERRORS, GET ERROR COUNT, POINT TO NEXT ERROR

GET ERROR COUNT — General Purpose

Function: To determine the number of errors in the queue.

Typical Use: To allow an error handling chart to determine that there are no more errors to process.

Details:
- Returns a zero if the queue is empty.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET ERROR COUNT**

Move to	ERROR COUNT	*variable integer*

Notes:
- To eliminate all errors from the queue, use CLEAR ALL ERRORS.
- Use the Debugger to view the error queue for detailed information.

See Also: CLEAR ALL ERRORS, GET ERROR CODE, POINT TO NEXT ERROR

GET RTU TEMPERATURE — General Purpose

Function: To obtain the temperature inside the M4RTU controller case.

Typical Use: To determine if heating or cooling is required or has failed.

Details:

- The temperature is reported in either Celsius or Fahrenheit depending on how I/O unit 1 on the local bus is configured.
- The temperature range is -40°C to 125°C (-40°F to 257°F).

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example:

GET RTU TEMPERATURE

Move to	RTU TEMP	*variable float*

Notes:

- If I/O unit 1 is not configured, this command returns the temperature in degrees Celsius.
- To read temperature in degrees Fahrenheit, make sure TEMP CNV is set to Degrees F when configuring I/O unit 1. (To verify, select I/O Unit from the Configurator's Configure menu, select the I/O unit, and click CHANGE.)
- Accuracy is:
  ±0.5°C from 0°C to 70°C
  ±1°C from -40°C to 0°C and from 70°C to 85°C
  ±2°C from -55°C to -40°C and from 85°C to 125°C

Dependencies:

- An M4RTU must be in use.

Error Codes:

- If this command is used for a controller other than an M4RTU, an error value of -32,768 is returned.

See Also: GET RTU VOLTAGE

GET RTU VOLTAGE — General Purpose

Function: To read the input voltage furnished to the M4RTU power supply.

Typical Use: To monitor battery voltage supplied to the M4RTU power supply to determine if it's getting low.

Details:
- Reads voltage supplied to the input terminals by others.
- Accuracy is plus or minus five percent.
- Works with both AC and DC.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **GET RTU VOLTAGE**

Move to	RTU VOLTAGE	*variable float*

Dependencies:
- An M4RTU must be in use.

Error Codes:
- If this command is used for a controller other than an M4RTU, an error value of -32,768 is returned.

See Also: GET RTU TEMPERATURE

GET SIZE OF NUMERIC TABLE — General Purpose

Function: To obtain the declared length (size) of a float or integer table.

Typical Use: To determine the last index when reading or writing to a numeric table.

Details:
- A size of 10, for example, means there are 11 elements numbered 0–10.

Arguments:

ARGUMENT 1	ARGUMENT 2
FLOAT TABLE	VARIABLE FLOAT
INTEGER TABLE	VARIABLE INTEGER

Example: **GET SIZE OF NUMERIC TABLE**

Table	CONFIG DATA	*numeric table*
Move to	CONFIG DATA SIZE	*variable integer (the table size)*

Notes:
- Always use to determine table size when program logic must act on all elements of a table. Then if the size of the table is later changed, the program will automatically adjust to the new size.

See Also: GET SIZE OF STRING TABLE

GET SIZE OF STRING TABLE — General Purpose

Function: To obtain the declared length (size) of a string table.

Typical Use: To determine the last index when reading or writing to a string table.

Details:
- A size of 19, for example, means there are 20 elements numbered 0–19.

Arguments:

ARGUMENT 1	ARGUMENT 2
STRING TABLE	VARIABLE FLOAT VARIABLE INTEGER

Example: **GET SIZE OF STRING TABLE**

Table	CONFIG NAMES	*string table*
Move to	CONFIG NAMES SIZE	*variable integer (the table size)*

Notes:
- Always use to determine table size when program logic must act on all elements of a table. Then if the size of the table is later changed, the program will automatically adjust to the new size.

See Also: GET SIZE OF NUMERIC TABLE

GET THIS CONTROLLER'S ADDRESS — General Purpose

Function: To obtain the controller's assigned HOST port address.

Typical Use: To execute program logic branching based on the controller's address or serial port message ID.

Details:
- The range of values returned is from 1 to 255.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET THIS CONTROLLER'S ADDRESS**

Move to	LC ADDR	*variable integer (the address)*

Notes:
- Use to determine if messages received from a non-HOST serial port are for this controller.

MOVE — General Purpose

Function: To copy a digital, analog, or numeric value to another location.

Typical Use: To copy values between objects, even if they are dissimilar types.

Details:

- Cyrano automatically converts the type of Argument 1 to match that of Argument 2. The following rules are employed when copying values between objects of different types:
  - *From Float to Integer:* Floats are rounded up for fractions of 0.5 or greater, otherwise they are rounded down.
  - *From Integer to Float:* Integer values are converted directly to floats.
  - *From Digital Input or Output:* A value of -1 is returned for on, 0 for off.
  - *From Latch:* A value of -1 is returned for set latches, 0 for latches that are not set.
  - *To Digital Output:* A value of 0 turns the output off. Any non-zero value turns the output on.
  - *To Analog Output:* Values are sent as is. Expect some rounding consistent with the analog resolution of the I/O unit. If the value sent is outside the allowable range for the channel, the output will go to the nearest range limit, either zero or full scale.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	DIGITAL OUT
CONSTANT FLOAT	TIME PROP. OUTPUT
CONSTANT INTEGER	VARIABLE FLOAT
COUNTER	VARIABLE INTEGER
DIGITAL IN	VARIABLE TIMER
DIGITAL OUT	
FREQUENCY	
OFF LATCH	
OFF PULSE MEAS.	
OFF TIME TOTALIZER	
ON LATCH	
ON PULSE MEAS.	
ON TIME TOTALIZER	
PERIOD	
QUADRATURE COUNTER	
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **MOVE**

From	DIG1	*digital point*
To	DIG1 STATUS	*variable integer*

Notes:

- After moving a new value to an analog output, anywhere from 0–50 milliseconds will elapse before the analog output is actually updated. Reading the output value during this period will show the previous value. This limitation may be improved in future versions of analog I/O units.

Error Codes: Queue error 33 = Overflow error — integer or float value was too large

See Also: MOVE STRING and all MOVE TO or MOVE FROM table commands.

MOVE FLOAT TABLE TO FLOAT TABLE — General Purpose

Function: To copy a single value from one float table to another.

Typical Use: To reorder the way data are arranged or to copy temporary values to a final location.

Details:
- The two tables can be the same.
- Any value sent to an invalid index is discarded, and an error 32 is added to the queue.
- The valid range for each index is zero to the table length (size).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT INTEGER VARIABLE INTEGER	FLOAT TABLE	CONSTANT INTEGER VARIABLE INTEGER	FLOAT TABLE

Example: **MOVE FLOAT TABLE TO FLOAT TABLE**

Source Index	SOURCE INDEX	*variable integer*
Source Table	FTABLE1	*float table*
Dest. Index	DEST INDEX	*variable integer*
Dest. Table	FTABLE2	*float table*

Notes:
- The contents of one table can be duplicated into another by using this command with the same value for the source and destination index. A loop structure can be used to increment the index for subsequent calls of this command.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE FROM TABLE, MOVE INT TABLE TO INT TABLE, MOVE TO FLOAT TABLE, MOVE TO INTEGER TABLE, SHIFT TABLE

MOVE FROM TABLE — General Purpose

Function: To copy one value from either an integer or float table.

Typical Use: To copy a numeric table value to an I/O point or another numeric variable.

Details:
- All numeric type conversions are automatically handled according to the rules detailed for the MOVE command.
- The valid range for the index is zero to the table length (size).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT INTEGER	FLOAT TABLE	ANALOG OUT
VARIABLE INTEGER	INTEGER TABLE	DIGITAL OUT
		TIME PROP. OUTPUT
		VARIABLE FLOAT
		VARIABLE INTEGER

Example: **MOVE FROM TABLE**

Index	0	*constant integer*
From	LOOK UP TABLE	*float table*
To	PRESS OUT	*analog output (destination for the data)*

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE FLOAT TABLE TO FLOAT TABLE, MOVE INT TABLE TO INT TABLE, MOVE TO FLOAT TABLE, MOVE TO INTEGER TABLE, SHIFT TABLE

MOVE INT TABLE TO INT TABLE — General Purpose

Function: To copy a single value from one integer table to another.

Typical Use: To reorder the way data is arranged or copy temporary values to a final location.

Details:
- The two tables can be the same.
- Any value sent to an invalid index is discarded, and an error 32 is added to the queue.
- The valid range for the index is zero to the table length (size).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT INTEGER VARIABLE INTEGER	INTEGER TABLE	CONSTANT INTEGER VARIABLE INTEGER	INTEGER TABLE

Example: **MOVE INT TABLE TO INT TABLE**

Source Index	SOURCE INDEX	*variable integer*
Source Table	ITABLE1	*integer table*
Dest. Index	DEST INDEX	*variable integer*
Dest. Table	ITABLE2	*integer table*

Notes:
- The contents of one table can be duplicated into another by using this command with the same variable for the source and destination index. A loop structure can be used to increment the index for subsequent calls of this command.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE FROM TABLE, MOVE FLOAT TABLE TO FLOAT TABLE, MOVE TO FLOAT TABLE, MOVE TO INTEGER TABLE, SHIFT TABLE

MOVE TO FLOAT TABLE — General Purpose

Function: To copy a value to a specified float table.

Typical Use: To store numeric data in a float table.

Details:
- All numeric type conversions are automatically handled according to the rules detailed for the MOVE command.
- The valid range for the index is zero to the table length (size).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		

Example: **MOVE TO FLOAT TABLE**

From	DATA VALUE	*variable float*
Index	4	*constant integer*
To	MY TABLE	*float table name*

Error Codes:

Queue error 32 = Bad table index value — index was negative or greater than the table size

Queue error 33 = Overflow — integer or float value was too large

See Also: MOVE FLOAT TABLE TO FLOAT TABLE, MOVE FROM TABLE, MOVE INT TABLE TO INT TABLE, MOVE TO INTEGER TABLE, SHIFT TABLE

MOVE TO INTEGER TABLE — General Purpose

Function: To copy a value to a specified integer table.

Typical Use: To store numeric data in a integer table.

Details:
- All numeric type conversions are automatically handled according to the rules detailed for the MOVE command.
- The valid range for the index is zero to the table length (size).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		

Example: **MOVE TO INTEGER TABLE**

From	MY DIG INPUT	*digital input*
Index	4	*constant integer*
To	MY INT TABLE	*integer table name*

Error Codes:

Queue error 32 = Bad table index value — index was negative or greater than the table size

Queue error 33 = Overflow — integer or float value was too large

See Also: MOVE FLOAT TABLE TO FLOAT TABLE, MOVE FROM TABLE, MOVE INT TABLE TO INT TABLE, MOVE TO FLOAT TABLE, SHIFT TABLE

POINT TO NEXT ERROR General Purpose

Function: To drop the oldest error from the queue and bring the next error to the top of the queue.

Typical Use: To access items in the error queue during error handling within the Cyrano strategy.

Details:
- Must use before the next error in the queue can be evaluated.
- Once this command is executed, the previous error can no longer be accessed.
- Commands that have the word ERROR in their name always evaluate the top (oldest) error in the queue.

Arguments: None.

Example: **POINT TO NEXT ERROR**

Notes:
- Always use the condition ERROR? to determine if there are errors in the queue before using this command.
- Use the Debugger to view the error queue for detailed information.

Dependencies:
- At least one error must exist in the error queue.

See Also: GET ERROR COUNT GET ERROR CODE, GET ERROR BOARD NAME, GET ERROR TASK NAME, ERROR?

RETRIEVE SAVED CRC — General Purpose

Function: Retrieves the 16-bit CRC calculated when the strategy was initially downloaded to the controller.

Typical Use: Periodically used in an error handler to check the integrity of the running program.

Details:
- Use the returned value to compare with a newly calculated CRC that was obtained by using CALCULATE STRATEGY CRC. These two values should match exactly.

Arguments:

ARGUMENT 1
VARIABLE INTEGER

Example: **RETRIEVE SAVED CRC**

Put Result In	ORIGINAL_CRC	*variable integer*

See Also: CALCULATE STRATEGY CRC

SHIFT TABLE — General Purpose

Function: To shift numeric table elements up or down.

Typical Use: To follow items on a conveyor.

Details:
- For positive shift counts, entries shift toward the end of the table. For negative shift counts, entries shift toward the beginning (index zero) of the table.
- Entries at the beginning or end of the table are lost when shifted beyond those limits.
- Zeros are written to entries left empty by shifting.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT INTEGER	FLOAT TABLE
VARIABLE INTEGER	INTEGER TABLE

Example: **SHIFT TABLE**

Shift Count	-5	*constant integer*
Table	MY TABLE	*integer or float table name*

Notes:
- Use MOVE FROM TABLE before this command to capture values that will be shifted out of the table, if they need to be used.
- Use MOVE TO INTEGER TABLE (for example) after this command to fill vacated entries, if desired.

See Also: MOVE FLOAT TABLE TO FLOAT TABLE, MOVE FROM TABLE, MOVE INT TABLE TO INT TABLE, MOVE TO FLOAT TABLE, MOVE TO INTEGER TABLE

\ COMMENT General Purpose

Function: To add a comment to an operation block.

Typical Use: To document commands within a block.

Details:
- Comments are string constants. They use controller memory.

Arguments:

ARGUMENT 1
CONSTANT STRING

Example: **\ COMMENT**
PID Loop Control Start *constant string*

Notes:
- To conserve memory, use the TEXT command button to create comments outside a block.

See Also: \\ COMMENT

\\ COMMENT — General Purpose

Function: To disable one or more commands in an operation block.

Typical Use: To temporarily disable commands within an operation block during debugging.

Details:

- This command is normally used in pairs. Everything between the pair of \\ COMMENT commands is considered a comment and is ignored when the strategy is compiled and downloaded.
- This command is useful for temporarily disabling a group of commands within an operation block while debugging a program.
- If the second \\ COMMENT is omitted, everything from the first \\ COMMENT to the end of the operation block is considered a comment.

Arguments:

ARGUMENT 1
CONSTANT STRING

Example: **\\ COMMENT**

Command
Command
Command

\\ COMMENT

See Also: \ COMMENT

I/O UNIT OPERATIONS

DISABLE I/O UNIT — I/O Unit

Function: To disable communication between the program in the Mistic controller and all channels on the I/O unit.

Typical Uses:
- To prohibit the program in the Mistic controller from reading or writing to the I/O unit for simulation and program testing.
- To gain fast I/O processing. With communication disabled, all logic is executed using values within the Mistic controller.

Details:
- All program references to I/O will be restricted to the use of internal I/O values (IVAL).
- Input IVALs will remain in their current state (unless changed by the user via the Debugger or with special simulation commands).
- Output IVALs will reflect what the program is instructing the outputs to do.
- *Caution:* Event/reactions (if any) will still be operational at the I/O unit. Any outputs that are on may remain on.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT
DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT

Example: **DISABLE I/O UNIT**

VAPOR EXTRACTION	*I/O unit*

Notes:
- Communication to I/O units is normally disabled using the Cyrano Configurator or Debugger.
- If I/O units are disabled to speed logic execution, perform the following in the order shown:
  1. MOVE ANL. I/O UNIT TO TABLE *(with I/O unit still disabled)*: Copies analog output IVALs updated by program.
  2. DO BINARY READ *(with I/O unit still disabled)*: Copies digital output IVALs updated by program.
  3. ENABLE I/O UNIT: Re-establishes communications.
  4. MOVE TABLE TO ANL I/O UNIT: Writes to the table MOVEd to above. Updates analog outputs.
  5. DO BINARY WRITE: Writes to the value read above. Updates digital outputs.
  6. MOVE ANL. I/O UNIT TO TABLE: Updates analog input IVALs.
  7. DO BINARY READ: Updates digital input IVALs.
  8. DISABLE I/O UNIT: Disconnects communications.
  9. Program logic . . . (Not for use with commands that access MIN, MAX, AVERAGE, COUNTS, etc.)
  10. Repeat 1 through 9.

See Also: ENABLE I/O UNIT

DISABLE ON I/O UNIT ERROR — I/O Unit

Function: To disable communication between the program in the Mistic controller and all channels on the I/O unit if the I/O unit generated the top queue error.

Typical Use: Since the I/O unit is automatically disabled after a queue error 29, this command is not currently needed.

Details:
- The Mistic controller generates a queue error 29 (timeout) whenever an I/O unit does not respond. When this happens, all further communication to the I/O unit is disabled to ensure that communication to other I/O units does not slow down.
- I/O unit errors other than 29 will not disable communication.

Arguments: None.

Example: **DISABLE ON I/O UNIT ERROR**

Notes:
- This command is typically used in an error handling chart.

Dependencies:
- For this command to have any effect, the top error in the queue must be generated by an I/O unit, as listed below:
  - — Queue error 2 = Bad CRC/checksum
  - — Queue error 3 = Bad message length received
  - — Queue error 4 = I/O unit has powered up since last access
  - — Queue error 6 = Watchdog timeout has occurred on I/O unit
  - — Queue error 29 = I/O unit did not respond within specified time

See Also: ENABLE ON I/O UNIT ERROR, ERROR ON I/O UNIT?

DO BINARY ACTIVATE — I/O Unit

Function: To turn on multiple digital output channels on the same I/O unit simultaneously with a single command.

Typical Use: To efficiently control a selected group of digital outputs with one command.

Details:
- This command is 16 times faster than using TURN ON 16 times.
- Updates the IVALs and XVALs for all 16 channels.
- Does not affect input channels or channels not specified.
- Truncates floats (if used) to integers.
- Uses only the lowest (least significant) 16 bits of the integer.
- A channel is selected for activation by setting the respective bit in the 16-bit data field to "1."
- If a specific channel is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.
- The least significant bit corresponds to channel zero.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	DIGITAL NMF I/O UNIT
VARIABLE FLOAT	REM SMPL I/O UNIT
VARIABLE INTEGER	

Example: **DO BINARY ACTIVATE**

From	35888	*constant integer*
Move To	LIGHT CONTROL	*digital I/O unit*

The effect of this is illustrated below:

Channel_Number	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bit_Mask 35888 decimal	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
Hex 8C30		**8**				**C**				**3**				**0**		

In this example, channels 15, 11, 10, 5 and 4 will be turned on. The other channels (set to "0" only for this illustration) are unaffected. They will remain in their original state.

Notes:
- Switch the Cyrano Configurator to binary or hex mode before using this command to make it easier to determine the mask value. Use ALT-B to switch to binary mode, ALT-H to switch to hex mode, ALT-D to switch to decimal mode.
- Use BIT SET or BIT CLEAR to change individual bits in a variable integer under program control.

See Also: DO BINARY DEACTIVATE, DO BINARY WRITE

DO BINARY DEACTIVATE — I/O Unit

Function: To turn off multiple digital output channels on the same I/O unit simultaneously with a single command.

Typical Use: To efficiently control a selected group of digital outputs with one command.

Details:
- This command is 16 times faster than using TURN OFF 16 times.
- Updates the IVALs and XVALs for all 16 channels.
- Does not affect input channels or channels not specified.
- Truncates floats (if used) to integers.
- Uses only the lowest (least significant) 16 bits of the integer.
- A channel is selected for deactivation by setting the respective bit in the 16-bit data field to "1."
- If a specific channel is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.
- The least significant bit corresponds to channel zero.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	DIGITAL NMF I/O UNIT
VARIABLE FLOAT	REM SMPL I/O UNIT
VARIABLE INTEGER	

Example: **DO BINARY DEACTIVATE**

From	9217	*constant integer*
Move To	LIGHT CONTROL	*digital I/O unit*

The effect of this is illustrated below:

Channel_Number	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bit_Mask 9217 decimal	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
Hex 2401		**2**				**4**				**0**				**1**		

In this example, channels 13, 10, and 0 will be turned off. The other channels (set to "0" only for this illustration) are unaffected. They will remain in their original state.

Notes:
- Switch the Cyrano Configurator to binary or hex mode before using this command to make it easier to determine the mask value. Use ALT-B to switch to binary mode, ALT-H to switch to hex mode, ALT-D to switch to decimal mode.
- Use BIT SET or BIT CLEAR to change individual bits in a variable integer under program control.

See Also: DO BINARY ACTIVATE, DO BINARY WRITE

DO BINARY READ — I/O Unit

Function: To read the current on/off status of all channels on the specified digital I/O unit.

Typical Use: To efficiently read the status of all digital channels on a single I/O unit with one command.

Details:

- Reads the current on/off status of all 16 channels on the digital I/O unit specified.
- Updates the IVALs and XVALs for all 16 channels.
- Reads outputs as well as inputs.
- Returns status (a 16-bit integer) to the numeric variable specified.
- If a channel is on, there will be a "1" in the respective bit. If the channel is off, there will be a "0" in the respective bit.
- If a specific channel is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.
- The least significant bit corresponds to channel zero.

Arguments:

ARGUMENT 1	ARGUMENT 2
DIGITAL MF I/O UNIT	VARIABLE FLOAT
DIGITAL NMF I/O UNIT	VARIABLE INTEGER
REM SMPL I/O UNIT	

Example: **DO BINARY READ**

From	INPUT BOARD_#1	*digital I/O unit*
Move To	IN BD1 STATUS	*variable integer*

The effect of this is illustrated below:

Channel_Number	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bit_Mask 27714 decimal	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
Hex 6C42		**6**				**C**				**4**				**2**		

In this example, channels 14, 13, 11, 10, 6, and 1 are currently on. The other channels are off.

Notes:

- Use BIT TEST to examine individual bits.

See Also: DO BINARY WRITE

DO BINARY WRITE — I/O Unit

Function: To control multiple digital output channels on the same I/O unit simultaneously with a single command.

Typical Use: To efficiently control a selected group of digital outputs with one command.

Details:
- This command is 16 times faster than using TURN ON or TURN OFF 16 times.
- Updates the IVALs and XVALs for all 16 channels.
- Affects all output channels.
- Does not affect input channels.
- Uses only the lowest (least significant) 16 bits of the integer.
- A channel is selected for activation by setting the respective bit in the 16-bit data field to "1."
- A channel is selected for deactivation by setting the respective bit in the 16-bit data field to "0."
- If a specific channel is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.
- The least significant bit corresponds to channel zero.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	DIGITAL NMF I/O UNIT
VARIABLE FLOAT	REM SMPL I/O UNIT
VARIABLE INTEGER	

Example: **DO BINARY WRITE**

From	146	*constant integer*
Move To	OUT BD1	*digital I/O unit*

The effect of this is illustrated below:

Channel_Number	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Bit_Mask 146 decimal	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**
Hex 0092		**0**				**0**				**9**				**2**		

In this example, channels 7, 4, and 1 are turned on. The other output channels are turned off.

Notes:
- Use BIT SET or BIT CLEAR to change individual bits in a variable integer under program control.

See Also: DO BINARY READ, DO BINARY ACTIVATE, DO BINARY DEACTIVATE

ENABLE I/O UNIT — I/O Unit

Function: To enable communication between the program in the Mistic controller and all channels on the I/O unit.

Typical Use: To re-establish communication between the Mistic controller and the I/O unit after it was automatically disabled due to a timeout error (29).

Details:
- Attempts to communicate with the I/O unit.
- If the communication succeeds and the I/O unit reports that it has lost power since the last communication, all channels will be configured and all event/reactions (if any) will be sent. Counters will have to be restarted under program control.
- If this command fails because the I/O unit specified is still not responding, a new error 29 will be added to the bottom of the error queue.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT
DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT

Example: **ENABLE I/O UNIT**

VAPOR EXTRACTION	*I/O unit*

Notes:
- This command is sometimes useful for debugging and/or system start-up.

Error Codes: Queue error 29 = I/O unit did not respond within specified time

See Also: DISABLE I/O UNIT

ENABLE ON I/O UNIT ERROR — I/O Unit

Function: To enable communication between the program in the Mistic controller and all channels on the I/O unit if the top queue error is a 29.

Typical Use: To re-establish communication between the Mistic controller and the I/O unit after it was automatically disabled due to a timeout error (29).

Details:
- The Mistic controller generates a queue error 29 (timeout) whenever an I/O unit does not respond. When this happens, all further communication to the I/O unit is disabled to ensure that communication to other I/O units does not slow down. This may be undesirable in some cases. This command can be used to re-establish communication.
- If this command fails because the I/O unit specified is still not responding, a new error 29 will be added to the bottom of the error queue.

Arguments: None.

Example: **ENABLE ON I/O UNIT ERROR**

Notes:
- This command is typically used in an error handling chart.
- Always use ERROR ON I/O UNIT? to determine if the top error in the error queue is an I/O unit error before using this command.
- Always use POINT TO NEXT ERROR after using this command.

Dependencies:
- For this command to have any effect, the top error in the queue must be a 29.

Error Codes: Queue error 29 = I/O unit did not respond within specified time

See Also: DISABLE ON I/O UNIT ERROR, ERROR ON I/O UNIT?

GET BAD I/O UNIT ADDRESS — I/O Unit

Function: To return the address of the I/O unit that failed to respond if the top queue error is a 29.

Typical Uses:
- Within an error handler, to log the date and time of a timeout error and the name of the I/O unit that failed to respond.
- Within an error handler, to alert an operator as to which I/O units are off-line.

Details:
- The Mistic controller generates a queue error 29 (timeout) whenever an I/O unit does not respond. This command can be used to determine the address of the I/O unit that failed to respond.

Arguments:

ARGUMENT 1
VARIABLE INTEGER

Example: **GET BAD I/O UNIT ADDRESS**

Put Result In	I/O UNIT ADDR	*variable integer*

Notes:
- This command is typically used in an error handling chart.
- In a system with many I/O units, this command can pinpoint exactly which I/O units are not responding. The result can be put in an integer table or appended to an error message string for display on an MMI screen.
- Always use ERROR ON I/O UNIT? to determine if the top error in the error queue is an I/O unit error before using this command.
- Always use POINT TO NEXT ERROR after using this command.

Dependencies: For this command to have any effect, the top error in the queue must be a 29.

See Also: GET BAD I/O UNIT PORT, ERROR ON I/O UNIT?, POINT TO NEXT ERROR

GET BAD I/O UNIT PORT — I/O Unit

Function: To return the port number of the I/O unit that failed to respond if the top queue error is a 29.

Typical Use: Within an error handler in conjunction with GET BAD I/O UNIT ADDRESS, to log the date and time of a timeout error as well as the name and port number of the I/O unit that failed to respond. Use only when there are several I/O units with the same address on different ports.

Details: The Mistic controller generates a queue error 29 (timeout) whenever an I/O unit does not respond. This command can be used to determine the port number of the I/O unit that failed to respond.

Arguments:

ARGUMENT 1
VARIABLE INTEGER

Example: **GET BAD I/O UNIT PORT**

Put Result In	I/O UNIT PORT	*variable integer*

Notes:

- This command is typically used in an error handling chart.
- In a system with many I/O units, this command can pinpoint exactly which I/O units are not responding. The result can be put in an integer table or appended to an error message string for display on an MMI screen.
- Always use ERROR ON I/O UNIT? to determine if the top error in the error queue is an I/O unit error before using this command.
- Always use POINT TO NEXT ERROR after using this command.

Dependencies: For this command to have any effect, the top error in the queue must be a 29.

See Also: GET BAD I/O UNIT ADDRESS, ERROR ON I/O UNIT?, POINT TO NEXT ERROR

MOVE ANL. I/O UNIT TO TABLE — I/O Unit

Function: To read all 16 channels of an analog I/O unit and move the returned values to a float table.

Typical Use: To efficiently read all 16 channels of analog data on a single I/O unit with one command.

Details:

- This command is four times faster than using MOVE 16 times.
- Reads both inputs and outputs.
- Updates the IVALs and XVALs for all 16 channels.
- Transfers 16 channels of float data (in engineering units) from the analog I/O unit to a float table beginning at the index specified. If there are fewer than 16 elements of data from the specified index to the end of the table, no data will be written to the table and a 32 will be placed in the error queue.
- Channels that are not configured will return a value of 0.0.
- If a specific channel is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be read.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG MF I/O UNIT	CONSTANT INTEGER VARIABLE INTEGER	FLOAT TABLE

Example: **MOVE ANL. I/O UNIT TO TABLE**

From	ANALOG UNIT 255	*analog I/O unit*
Index	16	*constant integer (starting index of table)*
Move To	DATA TABLE	*float table (holds 16 channels of analog data)*

Notes:

- To speed up analog logic execution, use DISABLE I/O UNIT after this command. This forces all references to channels on the I/O unit to use IVAL data rather than getting data from the I/O unit one channel at a time. If this procedure is followed, use ENABLE I/O UNIT before using this command again. See Notes under MOVE TABLE TO ANL I/O UNIT for more information.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE TABLE TO ANL I/O UNIT

MOVE TABLE TO ANL I/O UNIT — I/O Unit

Function: To write values in a float table to all 16 channels of an analog I/O unit.

Typical Use: To efficiently write all 16 channels of analog data on a single I/O unit with one command.

Details:
- This command is four times faster than using MOVE 16 times.
- Updates the IVALs and XVALs for all 16 channels except XVALs for input channels.
- Transfers 16 channels of data from the float table beginning at the index specified to the analog I/O unit. If there are fewer than 16 elements of data from the specified index to the end of the table, no data will be written to the I/O unit and a 32 will be placed in the error queue.
- If a specific channel is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.
- *Caution:* Writes to IVALs of input channels.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT INTEGER VARIABLE INTEGER	FLOAT TABLE	ANALOG MF I/O UNIT

Example: **MOVE TABLE TO ANL I/O UNIT**

Index	16	*constant integer (starting index of table)*
From	DATA TABLE	*float table (holds 16 values)*
To	ANALOG UNIT 255	*analog I/O unit*

Notes:
- If analog I/O units are disabled using DISABLE I/O UNIT to speed up analog logic execution, perform the following in the order shown:
  1. MOVE ANL. I/O UNIT TO TABLE *(with the I/O unit still disabled):* Copies output IVALs updated by program.
  2. ENABLE I/O UNIT: Re-establishes communications.
  3. MOVE TABLE TO ANL I/O UNIT: Writes to the table MOVEd to above. Updates analog outputs.
  4. MOVE ANL. I/O UNIT TO TABLE: Updates analog input IVALs.
  5. DISABLE I/O UNIT: Disconnects communications.
  6. Program logic . . . (not for use with commands that access MIN, MAX, AVERAGE, etc.)
  7. Repeat 1 through 6.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE ANL. I/O UNIT TO TABLE

LOGICAL OPERATIONS

AND — Logical

Function: To perform a logical AND on any two allowable values.

Typical Use: To determine if each of a pair of values is non-zero (True).

Details:
- Performs a logical AND on Arguments 1 and 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
-1	0	0
0	-1	0
-1	-1	-1

- The result is -1 (True) if both values are non-zero, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	DIGITAL OUT
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE FLOAT
DIGITAL IN	DIGITAL IN	VARIABLE INTEGER
DIGITAL OUT	DIGITAL OUT	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	

Example: **AND**

	LIMIT SWITCH1	*digital input*
With	LIMIT SWITCH2	*digital input*
Move To	BOTH SWITCHES CLOSED	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- To AND multiple variables (such as A, B, C, and D) into one variable (such as RESULT), do the following:
  1. AND A with B *Move To* RESULT
  2. AND C with RESULT *Move To* RESULT
  3. AND D with RESULT *Move To* RESULT
- To test for individual bits, use BIT TEST or BIT AND.

See Also: BIT TEST, BIT AND, AND (Condition)

BIT AND — Logical

Function: To perform a 32-bit bitwise AND on any two allowable values.

Typical Use: To clear one or more bits as specified by a "mask" (zero bits will clear).

Details:

- Performs a bitwise AND on Arguments 1 and 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
1	0	0
0	1	0
1	1	1

- Acts on all 32 bits.
- One value is the mask for selecting specific bits in the other value.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	CONSTANT INTEGER	DIGITAL NMF I/O UNIT
DIGITAL MF I/O UNIT	DIGITAL MF I/O UNIT	DIGITAL OUT
DIGITAL NMF I/O UNIT	DIGITAL NMF I/O UNIT	REM SMPL I/O UNIT
REM SMPL I/O UNIT	REM SMPL I/O UNIT	VARIABLE FLOAT
VARIABLE FLOAT	VARIABLE FLOAT	VARIABLE INTEGER
VARIABLE INTEGER	VARIABLE INTEGER	

Example: This example copies the four least significant bits from VALUE to RESULT and sets all remaining bits in RESULT to zero.

BIT AND

	VALUE	*variable integer*
With	15	*constant integer (the mask, binary 1111)*
Move To	RESULT	*variable integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- To clear bits in Argument 1, set a zero for each bit to clear in the mask (all remaining bits must be 1), and make Arguments 1 and 3 the same.
- It may be preferable to set a 1 for each bit to clear in the mask, then use BIT NOT to invert all the bits.
- Use 255 as the mask to keep the lower eight bits.
- To clear only one bit, use BIT CLEAR.
- To test for non-zero values, use AND.

See Also: BIT CLEAR, AND, AND (Condition)

BIT CLEAR — Logical

Function: To clear a specified bit (set it to zero) in an allowable value.

Typical Use: To clear one bit of a particular variable integer.

Details:
- Performs this operation on a *copy* of Argument 1, then moves the copy to Argument 3.
- Valid range for the bit to clear is 0–31.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
DIGITAL MF I/O UNIT	CONSTANT INTEGER	DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT	VARIABLE INTEGER	DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT		REM SMPL I/O UNIT
VARIABLE INTEGER		VARIABLE INTEGER

Example: This example does a binary read of the I/O unit I/O_UNIT_1, clears bit 0, and does a binary write of the data back out to I/O_UNIT_1. This will cause channel 0 of the I/O unit to be turned off. If channel 0 happens to be an input, nothing will happen.

BIT CLEAR

Data Source	I/O_UNIT_1	*digital I/O unit*
Bit to Clear	0	*constant integer*
Put Result In	I/O_UNIT_1	*digital I/O unit*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- Although this command can be used to turn off digital points, it is primarily used to manipulate bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time), fault (latch), and needs acknowledgment.
- To clear bits in Argument 1, make Arguments 1 and 3 the same.
- To clear several bits at once, use BIT AND.

See Also: BIT AND, BIT TEST, BIT SET

BIT NOT — Logical

Function: To invert all 32 bits of an allowable value.

Typical Use: To invert "mask" bits.

Details:
- Inverts Argument 1 and puts result in Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2
0	1
1	0

- Performs this operation on a *copy* of Argument 1, then moves the copy to Argument 2.
- Acts on all 32 bits.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	DIGITAL NMF I/O UNIT
DIGITAL MF I/O UNIT	DIGITAL OUT
DIGITAL NMF I/O UNIT	REM SMPL I/O UNIT
REM SMPL I/O UNIT	VARIABLE FLOAT
VARIABLE FLOAT	VARIABLE INTEGER
VARIABLE INTEGER	

Example: **BIT NOT**

	DATA	*variable integer*
Move To	DATA	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- To invert all bits in Argument 1, make both arguments the same.
- To clear one or more specific bits, use this command to invert the mask bits. Then, BIT AND the mask with the value containing the bits to be cleared.
- To toggle True/False, use NOT.

See Also: NOT, BIT XOR, XOR, BIT NOT? (Condition)

BIT OR — Logical

Function: To perform a 32-bit bitwise OR on any two allowable values.

Typical Use: To set one or more bits as specified by a "mask."

Details:
- Performs a bitwise OR on Arguments 1 and 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
1	0	1
0	1	1
1	1	1

- Combines all bits set to 1 in Arguments 1 and 2. The result (Argument 3) can be put into either of the first two items or into a different item.
- Acts on all 32 bits. One value is the mask for selecting specific bits to set in the other value.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	CONSTANT INTEGER	DIGITAL NMF I/O UNIT
DIGITAL MF I/O UNIT	DIGITAL MF I/O UNIT	DIGITAL OUT
DIGITAL NMF I/O UNIT	DIGITAL NMF I/O UNIT	REM SMPL I/O UNIT
REM SMPL I/O UNIT	REM SMPL I/O UNIT	VARIABLE FLOAT
VARIABLE FLOAT	VARIABLE FLOAT	VARIABLE INTEGER
VARIABLE INTEGER	VARIABLE INTEGER	

Example: This example sets bit 2 in a copy of Argument 1 and puts the result in Argument 3.

BIT OR

	VALUE	*variable integer*
With	4	*constant integer (the "mask," binary 10)*
Move To	RESULT	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- Although this command can be used to turn on digital points, it is used primarily to manipulate bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time), fault (latch), needs acknowledgment, etc.
- To set bits in Argument 1, make Arguments 1 and 3 the same.
- To set only one bit, use BIT SET.
- To test if either of two values is True, use OR.

See Also: BIT SET, OR, BIT XOR, XOR

BIT ROTATE — Logical

Function: To rotate all 32 bits of an allowable value to the left or right.

Typical Use: To shift bits left or right with wraparound.

Details:
- Acts on all 32 bits. All bits rotated past one end reappear at the other end.
- Valid range for the *Count* parameter (Argument 2) is 0–32. If Argument 2 is positive, bits will rotate left. If it is negative, bits will rotate right. If it is zero, no rotation will occur.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT INTEGER	CONSTANT INTEGER	DIGITAL MF I/O UNIT
DIGITAL MF I/O UNIT	VARIABLE INTEGER	DIGITAL NMF I/O UNIT
DIGITAL NMF I/O UNIT		DIGITAL OUT
REM SMPL I/O UNIT		REM SMPL I/O UNIT
VARIABLE INTEGER		VARIABLE FLOAT
		VARIABLE INTEGER

Example: **BIT ROTATE**

	MASK VARIABLE	*variable integer*
Count	4	*constant integer*
Move To	RESULT VARIABLE	*variable integer*

This example shows the bits of a copy of MASK VARIABLE rotated to the left by 4, with the result placed in RESULT VARIABLE. If MASK VARIABLE is -2,147,483,904 (10000000 00000000 00000000 00000000 binary), then after the rotation RESULT VARIABLE would be 8 (00000000 00000000 00000000 00001000 binary).

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- To rotate bits in Argument 1, make Arguments 1 and 3 the same.
- To get rid of all bits that move past either end, use BIT SHIFT.

See Also: BIT SHIFT

BIT SET — Logical

Function: To set a specified bit (set it to 1) in an allowable value.

Typical Use: To set a bit in an integer variable that is used as a flag.

Details:
- Performs this operation on a *copy* of Argument 1, then moves the copy to Argument 3.
- Valid range for Argument 2 is 0–31.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
DIGITAL MF I/O UNIT	CONSTANT INTEGER	DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT	VARIABLE INTEGER	DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT		REM SMPL I/O UNIT
VARIABLE INTEGER		VARIABLE INTEGER

Example: **BIT SET**

Data Source	PUMP3 CTRL BITS	*variable integer*
Bit to Set	15	*constant integer*
Put Result In	I/ PUMP3 CTRL BITS	*variable integer*

If PUMP3 CTRL BITS is 8 (00000000 00000000 00000000 00001000 binary), then after the BIT SET, I/ PUMP3 CTRL BITS would be 32776 (00000000 00000000 10000000 00001000 binary).

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- Although this command can be used to turn on digital points, it is primarily used to manipulate bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time), fault (latch), and needs acknowledgment.
- To set bits in Argument 1, make Arguments 1 and 3 the same.
- To set several bits at once, use BIT OR.

See Also: BIT OR, BIT TEST, BIT CLEAR

BIT SHIFT — Logical

Function: To shift the bits of an allowable value to the right or left.

Typical Use: To evaluate the four bytes of a 32-bit integer one at a time. A faster integer multiply or divide.

Details:
- Functionally equivalent to integer multiply or divide, except faster.
- Acts on all 32 bits. Valid range for the *Count* parameter (Argument 2) is 0–32.
- BIT SHIFT with a *Count* of 2 is the same as multiplying by 4. BIT SHIFT with a *Count* of -3 is the same as dividing by 8.
- All bit positions vacated by the shift are filled with zeros.
- If Argument 2 is positive, bits will shift left. If it is negative, bits will shift right. If it is zero, no shifting will occur.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT INTEGER	CONSTANT INTEGER	DIGITAL MF I/O UNIT
DIGITAL MF I/O UNIT	VARIABLE INTEGER	DIGITAL NMF I/O UNIT
DIGITAL NMF I/O UNIT		DIGITAL OUT
REM SMPL I/O UNIT		REM SMPL I/O UNIT
VARIABLE INTEGER		VARIABLE FLOAT
		VARIABLE INTEGER

Example: **BIT SHIFT**

	MASK VARIABLE	*variable integer*
Count	-8	*constant integer*
Move To	RESULT VARIABLE	*variable integer*

This example shows the bits of a copy of MASK VARIABLE shifted to the right by 8, with the result placed in RESULT VARIABLE. If MASK VARIABLE is -2,147,483,904 (10000000 00000000 00000000 00000000 binary), then after the shift RESULT VARIABLE would be 8,388,608 (00000000 10000000 00000000 00000000 binary).

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- To shift bits in Argument 1, make Arguments 1 and 3 the same.
- To retain all bits that move past either end, use BIT ROTATE.

See Also: BIT ROTATE

BIT TEST — Logical

Function: To determine the status of a specific bit in an allowable value.

Typical Use: To test a bit in an integer variable that is used as a flag.

Details:
- Valid range for the *Bit to Test* parameter (Argument 2) is 0–31.
- If the bit is clear (0), 0 is moved to Argument 3.
- If the bit is set (1), -1 is moved to Argument 3.
- Note that the result can be sent directly to a digital output.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
DIGITAL MF I/O UNIT	CONSTANT INTEGER	DIGITAL OUT
DIGITAL NMF I/O UNIT	VARIABLE INTEGER	VARIABLE INTEGER
REM SMPL I/O UNIT		
VARIABLE INTEGER		

Example: **BIT TEST**

Data Source	PUMP3 CTRL BITS	*variable integer*
Bit to Test	15	*constant integer*
Put Result In	I/ PUMP3 CTRL BITS	*variable integer*

If PUMP3 CTRL BITS is 00000000 00000000 10000000 00001000, the result would be set to True.

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers with this command.
- Although this command can be used to determine the status of digital points, it is primarily used to test bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time), fault (latch), and needs acknowledgment.
- To test several bits at once, use BIT AND.

See Also: BIT CLEAR, BIT SET

BIT XOR — Logical

Function: To perform a 32-bit bitwise EXCLUSIVE OR on any two allowable values.

Typical Uses:
- To toggle one or more bits as specified by a "mask."
- To toggle an integer between zero and any other value.

Details:
- Performs a bitwise EXCLUSIVE OR on Arguments 1 and 2 and puts result in Argument 3. Examples:

BIT MANIPULATION			VALUE MANIPULATION		
ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0	0	22	22
0	1	1	22	22	0
1	0	1	255	65280	65535
1	1	0	0	-1	-1
			-1	0	-1

- Acts on all 32 bits.
- One value is the mask for selecting specific bits in the other value.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	DIGITAL MF I/O UNIT
CONSTANT INTEGER	CONSTANT INTEGER	DIGITAL NMF I/O UNIT
DIGITAL MF I/O UNIT	DIGITAL MF I/O UNIT	DIGITAL OUT
DIGITAL NMF I/O UNIT	DIGITAL NMF I/O UNIT	REM SMPL I/O UNIT
REM SMPL I/O UNIT	REM SMPL I/O UNIT	VARIABLE FLOAT
VARIABLE FLOAT	VARIABLE FLOAT	VARIABLE INTEGER
VARIABLE INTEGER	VARIABLE INTEGER	

Example: **BIT XOR**

	DATA	*variable integer*
With	22	*constant integer (the "mask," binary 10110)*
Move To	DATA NEW	*variable integer*

This example performs a BIT XOR on a copy of DATA with the constant 22 (binary 10110). The result (DATA NEW) has bits 1, 2, and 4 inverted. If DATA = 0, DATA NEW = 22. If DATA = 22, DATA NEW = 0.

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use this command only with integers.
- This command can be used to toggle digital outputs as well as bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time), fault (latch), and needs acknowledgment.
- To toggle bits in Argument 1, make Arguments 1 and 3 the same.
- To toggle a bit, BIT XOR with 1. Zero leaves the bit unchanged.
- To toggle an integer value between 0 and -1, use XOR.

See Also: XOR, BIT NOT, NOT

NOT — Logical

Function: To perform a logical NOT (True/False toggle) on any allowable value.

Typical Uses:
- To invert the logical state of an integer variable.
- To toggle the state of a digital output.
- To have a digital output assume the inverse state of a digital input.

Details:
- Performs a logical NOT on Argument 1 and puts result in Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2
0	-1
-1	0
22	0

- Performs this operation on a *copy* of Argument 1, then moves the copy to Argument 2.
- If Argument 1 is True (non-zero), the result will be False (0). If Argument 1 is False (0), the result will be True (-1).

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	DIGITAL OUT
CONSTANT INTEGER	VARIABLE FLOAT
DIGITAL IN	VARIABLE INTEGER
DIGITAL OUT	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **NOT**

	CURRENT STATE	*variable integer*
Move To	DOUT1	*digital output*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- To invert the TRUE/FALSE state of Argument 1, make both arguments the same.
- To toggle all 32 bits, use BIT NOT.

See Also: BIT NOT

OR — Logical

Function: To perform a logical OR on any two allowable values.

Typical Use: To use the True state of either value to control an output or set an alarm.

Details:

- Performs a logical OR on Arguments 1 and 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
-1	0	-1
0	-1	-1
-1	-1	-1

- The result is -1 (True) if either value is non-zero, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	DIGITAL OUT
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE FLOAT
DIGITAL IN	DIGITAL IN	VARIABLE INTEGER
DIGITAL OUT	DIGITAL OUT	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	

Example: **OR**

	LIMIT SWITCH1	*digital input*
With	LIMIT SWITCH2	*digital input*
Move To	MOTOR1 OUTPUT	*digital output*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- To OR multiple variables (such as A, B, C, and D) into one variable (such as RESULT), do the following:
  1. OR A with B *Move To* RESULT
  2. OR C with RESULT *Move To* RESULT
  3. OR D with RESULT *Move To* RESULT
- To test or manipulate individual bits, use BIT OR.

See Also: BIT OR

SET VARIABLE FALSE — Logical

Function: To move a False (0) value into an allowable value.

Typical Use: To clear a variable after it has been used for program logic.

Details:
- All numeric variables are False by default unless initialized by the user to a non-zero value.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET VARIABLE FALSE**

FLAG-HOPPER FULL *variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- *Speed Tip:* This command is faster than MOVE for moving a 0 to a variable.

See Also: SET VARIABLE TRUE

SET VARIABLE TRUE — Logical

Function: To move a True (-1) value into an allowable value.

Typical Use: To set a variable to -1.

Details:
- All numeric variables are False by default unless initialized by the user to a non-zero value.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET VARIABLE TRUE**

FLAG-JOB DONE	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- *Speed Tip:* This command is faster than MOVE for moving a -1 to a variable.

See Also: SET VARIABLE FALSE

TEST EQUAL — Logical

Function: To determine if two values are equal.

Typical Use: To perform logic branching based on whether an argument equals a set value.

Details:

- Determines if Argument 1 is equal to Argument 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	-1
-1	0	0
255	65280	0
22.22	22.22	-1

- The result is -1 (True) if both values are the same, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	DIGITAL OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	
COUNTER	COUNTER	
DIGITAL IN	DIGITAL IN	
DIGITAL OUT	DIGITAL OUT	
FREQUENCY	FREQUENCY	
OFF LATCH	OFF LATCH	
OFF PULSE MEAS.	OFF PULSE MEAS.	
OFF TIME TOTALIZER	OFF TIME TOTALIZER	
ON LATCH	ON LATCH	
ON PULSE MEAS.	ON PULSE MEAS.	
ON TIME TOTALIZER	ON TIME TOTALIZER	
PERIOD	PERIOD	
QUADRATURE COUNTER	QUADRATURE COUNTER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **TEST EQUAL**

	TOP LEVEL	*variable integer*
With	1000	*constant integer*
Put Result In	FLAG-AT THE TOP	*variable integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- When working with floats, this command is useful for determining if two numeric values are *exactly* the same.
- It may be safer to use TEST GREATER OR EQUAL or TEST LESS OR EQUAL instead, since exact matches of non-integer types are rare.

See Also: Any "TEST . . ." logical operations

TEST GREATER — Logical

Function: To determine if one value is greater than another.

Typical Use: To determine if a counter has reached an upper limit or if an analog value is too high.

Details:

- Determines if Argument 1 is greater than Argument 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
-1	0	0
-1	-3	-1
22.221	22.220	-1

- The result is -1 (True) if Argument 1 is greater than Argument 2, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	DIGITAL OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	
COUNTER	COUNTER	
DIGITAL IN	DIGITAL IN	
DIGITAL OUT	DIGITAL OUT	
FREQUENCY	FREQUENCY	
OFF LATCH	OFF LATCH	
OFF PULSE MEAS.	OFF PULSE MEAS.	
OFF TIME TOTALIZER	OFF TIME TOTALIZER	
ON LATCH	ON LATCH	
ON PULSE MEAS.	ON PULSE MEAS.	
ON TIME TOTALIZER	ON TIME TOTALIZER	
PERIOD	PERIOD	
QUADRATURE COUNTER	QUADRATURE COUNTER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **TEST GREATER**

	MY DATA COUNT	*digital counter*
Greater than	1000	*constant integer*
Put Result In	FLAG-MY DATA IS DONE	*variable integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Consider using TEST GREATER OR EQUAL instead.

See Also: Any “TEST . . .” logical operations

TEST GREATER OR EQUAL — Logical

Function: To determine if one value is greater than or equal to another.

Typical Use: To determine if an analog value has reached a maximum allowable value.

Details:
- Determines if Argument 1 is greater than or equal to Argument 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	-1
1	0	-1
-32768	-32767	0
22221	2222	-1

- The result is -1 (True) if Argument 1 is greater than or equal to Argument 2, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	DIGITAL OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	
COUNTER	COUNTER	
DIGITAL IN	DIGITAL IN	
DIGITAL OUT	DIGITAL OUT	
FREQUENCY	FREQUENCY	
OFF LATCH	OFF LATCH	
OFF PULSE MEAS.	OFF PULSE MEAS.	
OFF TIME TOTALIZER	OFF TIME TOTALIZER	
ON LATCH	ON LATCH	
ON PULSE MEAS.	ON PULSE MEAS.	
ON TIME TOTALIZER	ON TIME TOTALIZER	
PERIOD	PERIOD	
QUADRATURE COUNTER	QUADRATURE COUNTER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **TEST GREATER OR EQUAL**

	ROOM TEMP	*analog input*
> or =	78.5000	*constant float*
Put Result In	FLAG-ROOM TEMP OK	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- When using analog values or digital features in this command, be sure to take into consideration the units that the value is read in and adjust the test values accordingly.

See Also: Any “TEST . . .” logical operations

TEST LESS — Logical

Function: To determine if one value is less than another.

Typical Use: To determine if a tank needs to be filled.

Details:

- Determines if Argument 1 is less than Argument 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
-1	0	-1
-1	-3	0
22.221	22.220	0

- The result is -1 (True) if Argument 1 is less than Argument 2, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	DIGITAL OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	
COUNTER	COUNTER	
DIGITAL IN	DIGITAL IN	
DIGITAL OUT	DIGITAL OUT	
FREQUENCY	FREQUENCY	
OFF LATCH	OFF LATCH	
OFF PULSE MEAS.	OFF PULSE MEAS.	
OFF TIME TOTALIZER	OFF TIME TOTALIZER	
ON LATCH	ON LATCH	
ON PULSE MEAS.	ON PULSE MEAS.	
ON TIME TOTALIZER	ON TIME TOTALIZER	
PERIOD	PERIOD	
QUADRATURE COUNTER	QUADRATURE COUNTER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **TEST LESS**

	TANK LEVEL	*analog input*
Less than	FULL TANK LEVEL	*variable integer*
Put Result In	FLAG-TANK FILL VALVE	*digital output*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Consider using TEST LESS OR EQUAL instead.

See Also: Any “TEST . . .” logical operations

TEST LESS OR EQUAL — Logical

Function: To determine if one value is less than or equal to another.

Typical Use: To determine if a temperature is below or the same as a certain value.

Details:

- Determines if Argument 1 is less than or equal to Argument 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	-1
-1	0	-1
-1	-3	0
22.221	22.220	0

- The result is -1 (True) if Argument 1 is less than or equal to Argument 2, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	DIGITAL OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	
COUNTER	COUNTER	
DIGITAL IN	DIGITAL IN	
DIGITAL OUT	DIGITAL OUT	
FREQUENCY	FREQUENCY	
OFF LATCH	OFF LATCH	
OFF PULSE MEAS.	OFF PULSE MEAS.	
OFF TIME TOTALIZER	OFF TIME TOTALIZER	
ON LATCH	ON LATCH	
ON PULSE MEAS.	ON PULSE MEAS.	
ON TIME TOTALIZER	ON TIME TOTALIZER	
PERIOD	PERIOD	
QUADRATURE COUNTER	QUADRATURE COUNTER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **TEST LESS OR EQUAL**

	TEMPERATURE	*variable float*
< or =	98.6	*constant float*
Put Result In	FLAG-TEMP OK	*variable integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- When using analog values or digital features in this command, be sure to take into consideration the units that the value is read in and adjust the test values accordingly.

See Also: Any “TEST . . .” logical operations

TEST NOT EQUAL — Logical

Function: To determine if two values are different.

Typical Use: To check a counter.

Details:

- Determines if Argument 1 is different from Argument 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
-1	0	-1
255	65280	-1
22.22	22.22	0

- The result is -1 (True) if both values are not the same, 0 (False) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	DIGITAL OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	
COUNTER	COUNTER	
DIGITAL IN	DIGITAL IN	
DIGITAL OUT	DIGITAL OUT	
FREQUENCY	FREQUENCY	
OFF LATCH	OFF LATCH	
OFF PULSE MEAS.	OFF PULSE MEAS.	
OFF TIME TOTALIZER	OFF TIME TOTALIZER	
ON LATCH	ON LATCH	
ON PULSE MEAS.	ON PULSE MEAS.	
ON TIME TOTALIZER	ON TIME TOTALIZER	
PERIOD	PERIOD	
QUADRATURE COUNTER	QUADRATURE COUNTER	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **TEST NOT EQUAL**

	COUNTER VAL	*variable integer*
With	100	*constant integer*
Put Result In	FLAG-NOT DONE	*variable integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.

See Also: Any “TEST . . .” logical operations

XOR **Logical**

Function: To perform a logical EXCLUSIVE OR on any two allowable values.

Typical Use: To toggle a logic state such as a digital output from True to False or False to True.

Details:
- Performs a logical EXCLUSIVE OR on Arguments 1 and 2 and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
0	0	0
0	1	-1
1	0	-1
1	1	0
0	-1	-1
-1	0	-1
-1	-1	0
22	0	-1
22	22	0

- The result is -1 (True) if either Argument 1 or Argument 2 value is non-zero but not both, otherwise the result is 0 (False).
- The result can be sent directly to a digital output if desired.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT FLOAT	CONSTANT FLOAT	DIGITAL OUT
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE FLOAT
DIGITAL IN	DIGITAL IN	VARIABLE INTEGER
DIGITAL OUT	DIGITAL OUT	
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	

Example: **XOR**

	SUPPLY FAN	*digital output*
With	-1	*constant integer*
Move To	SUPPLY FAN	*digital output*

In this example, if SUPPLY FAN is on it will turn off, and vice versa.

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- To manipulate individual bits or toggle a value between zero and any other value, use BIT XOR.

See Also: BIT XOR, NOT EQUAL?

MATHEMATICAL OPERATIONS

COMPLEMENT — Mathematical

Function: To change the sign of a number from positive to negative or from negative to positive.

Typical Use: To make a result positive after subtracting a large number from a small number.

Details:
- Same as multiplying by -1. Thus, -1 becomes 1, 1 becomes -1, etc.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **COMPLEMENT**

TEMPERATURE DIFFERENCE *variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- The complement of zero is zero.
- Executes faster than multiplying by -1.

See Also: BIT NOT, NOT, TAKE ABSOLUTE VALUE OF

DECREMENT VARIABLE — Mathematical

Function: To decrease the value specified by 1.

Typical Use: To control count-down loops and other counting applications.

Details:
- Same as subtracting 1: 9 becomes 8, 0 becomes -1, 22.22 becomes 21.22, etc.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **DECREMENT VARIABLE**
NUM HOLES LEFT TO PUNCH *variable integer*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Executes faster than subtracting 1.

See Also: INCREMENT VARIABLE

DO ADDITION — Mathematical

Function: To add two numeric values.

Typical Use: To add two numbers to get a third number, or to add one number to a running total.

Details:

- Adds Arguments 1 and 2 and places the result in Argument 3.
- Argument 3 can be the same as either of the first two arguments (unless they are read-only, such as analog inputs), or it can be a completely different argument .
- Accommodates different item types such as float, integer, analog, and digital without restriction.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	ANALOG OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **DO ADDITION**

	INGREDIENT 1 WEIGHT	*analog input*
Plus	INGREDIENT 2 WEIGHT	*analog input*
Put Result In	TOTAL WEIGHT	*analog output*

Notes:

- See the Mathematical Overview in Chapter 1 for rounding and other important information.

Error Codes: Queue error 33 = Overflow error — result too large

See Also: INCREMENT VARIABLE, DO SUBTRACTION

DO DIVIDE — Mathematical

Function: To divide two numerical values.

Typical Use: To perform a standard division operation.

Details:
- Divides Argument 1 by Argument 2 and places the result in Argument 3.
- Argument 3 can be the same as either of the first two arguments (unless they are read-only, such as analog inputs), or it can be a completely different argument .
- If Argument 2 is 0, an error 36 (divide by zero) is added to the error queue.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	ANALOG OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **DO DIVIDE**

	TOTAL DISTANCE	*variable float*
By	2.0	*constant float*
Put Result In	HALF DISTANCE	*variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for rounding and other important information.
- Avoid divide-by-zero errors by checking Argument 2 *before* doing the division to be sure it does not equal zero. Use VARIABLE TRUE? (if it's True, it's not zero) or TEST NOT EQUAL (to zero).
- *Speed Tip:* Use BIT SHIFT instead of DO DIVIDE for integer math when the multiplier is 2, 4, 8, 16, 32, 64, etc.

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 36 = Divide by zero

See Also: DO MODULO, DO MULTIPLY, BIT SHIFT

DO MODULO — Mathematical

Function: To generate the remainder resulting from integer division.

Typical Use: To capture the remainder whenever integer modulo calculations are needed.

Details:
- Always results in an integer value. Examples: 40 modulo 16 = 8, 8 modulo 8 = 0.
- If any arguments are floats, they are rounded to integers before the division occurs.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	ANALOG OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **DO MODULO**

	NUM PARTS PRODUCED	*variable integer*
By	MINUTES ELAPSED	*variable integer*
Put Result In	PRODUCTIVITY REMAINDER	*variable integer*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.

See Also: DO DIVIDE, DO MULTIPLY

DO MULTIPLY — Mathematical

Function: To multiply two numeric values.

Typical Use: To multiply two numbers to get a third number or to modify one of the original numbers.

Details:
- Multiplies Arguments 1 and 2 and places the result in Argument 3.
- Argument 3 can be the same as either of the first two arguments (unless they are read-only, such as analog inputs), or it can be a completely different argument .

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	ANALOG OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **DO MULTIPLY**

	INGREDIENT 1 WEIGHT	*analog input*
Times	TEMPERATURE ADJUST	*variable float*
Put Result In	CORRECTED WEIGHT	*analog output*

Notes:
- See the Mathematical Overview in Chapter 1 for rounding and other important information.
- *Speed Tip:* Use BIT SHIFT instead for integer math where the multiplier is 2, 4, 8, 16, 32, 64, etc.

Error Codes: Queue error 33 = Overflow error — result too large

See Also: DO DIVIDE, BIT SHIFT

DO SUBTRACTION — Mathematical

Function: To find the difference between two numeric values

Typical Use: To subtract two numbers to get a third number, or to reduce the first number by the amount of the second.

Details:
- Subtracts Argument 2 from Argument 1 and places the result in Argument 3.
- Argument 3 can be the same as either of the first two arguments (unless they are read-only, such as analog inputs), or it can be a completely different argument .

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	ANALOG OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **DO SUBTRACTION**

	NUM WIDGETS TO PRODUCE	*variable integer*
Minus	NUM WIDGETS PRODUCED	*variable integer*
Put Result In	NUM WIDGETS LEFT TO MAKE	*variable integer*

Notes:
- See the Mathematical Overview in Chapter 1 for rounding and other important information.

Error Codes: Queue error 33 = Overflow error — result too large

See Also: DECREMENT VARIABLE, DO ADDITION

INCREMENT VARIABLE — Mathematical

Function: To increase the value specified by 1.

Typical Use: To control loop counters and other counting applications.

Details:
- Same as adding 1: 8 becomes 9, -1 becomes 0, 12.33 becomes 13.33, etc.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **INCREMENT VARIABLE**

LOOP COUNTER *variable integer*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Executes faster than adding 1.

See Also: DECREMENT VARIABLE

RAISE E TO — Mathematical

Function: To raise the constant e to a specified power.

Typical Use: To solve mathematical equations where the constant e is required.

Details:
- Raises e to the power specified in Argument 1.
- The constant e, the base of the natural system of logarithms, has a value of 2.7182818.
- The power (Argument 1) must be between -88.33654 and 88.72283.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **RAISE E TO**

	GAS PRESSURE	*analog input*
Put Result In	PRESSURE CALCULATION	*variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.

Error Codes: Queue error 33 = Overflow error — result too large

See Also: TAKE NATURAL LOG OF, RAISE TO POWER

RAISE TO POWER — Mathematical

Function: To raise a value to a specified power.

Typical Use: To solve exponentiation calculations.

Details:
- Raises Argument 1 to the power specified by Argument 2 and places the result in Argument 3.
- For use with positive numbers only.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	ANALOG IN	ANALOG OUT
ANALOG OUT	ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	VARIABLE FLOAT	
VARIABLE INTEGER	VARIABLE INTEGER	
VARIABLE TIMER	VARIABLE TIMER	

Example: **RAISE TO POWER**

	10	*constant integer*
To The	2	*constant integer*
Put Result In	TEN SQUARED	*variable integer*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Multiplying a number by itself is faster than raising a number to the power of 2.

Error Codes:
Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: RAISE E TO, TAKE SQUARE ROOT OF

TAKE ABSOLUTE VALUE OF — Mathematical

Function: To ensure that a value is positive.

Typical Use: To ensure a positive value when the result of a subtraction operation may be negative.

Details:
- Copies Argument 1 to Argument 2, dropping the minus sign if it exists.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
VARIABLE FLOAT	VARIABLE INTEGER
VARIABLE INTEGER	

Example: **TAKE ABSOLUTE VALUE OF**

	NEGATIVE VALUE	*variable float*
Put Result In	POSITIVE VALUE	*variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- To change a negative value to a positive value, make Arguments 1 and 2 the same.
- Use to convert a -1 Boolean result to a 1 for programs communicating with the Mistic controller that represent logical True with 1 rather than -1. This is required only when such programs read Boolean values from the Mistic controller.

See Also: COMPLEMENT

TAKE ARC COS OF — Mathematical

Function: To derive the angular value from a cosine value.

Typical Use: To solve trigonometric calculations.

Details:
- Calculates the arc cosine of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be a cosine value with a range of -1.0 to 1.0.
- The angular value returned is in radians with a range of 0 to 6.283185. (To convert radians to degrees, multiply by 57.29578.)

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE ARC COS OF**

	X	*variable float*
Put Result In	RADIANS	*variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Use TAKE COS OF if the angle is known and the cosine is desired.

Error Codes:
Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE COS OF, TAKE ARC SIN OF, TAKE ARC TAN OF

TAKE ARC SIN OF — Mathematical

Function: To derive the angular value from a sine value.

Typical Use: To solve trigonometric calculations.

Details:
- Calculates the arc sine of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be a sine value with a range of -1.0 to 1.0.
- The angular value returned is in radians with a range of 0 to 6.283185. (To convert radians to degrees, multiply by 57.29578.)

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE ARC SIN OF**

	X	*variable float*
Put Result In	RADIANS	*variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Use TAKE SIN OF if the angle is known and the sine is desired.

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE SIN OF, TAKE ARC COS OF, TAKE ARC TAN OF

TAKE ARC TAN OF — Mathematical

Function: To derive the angular value from a tangent value.

Typical Use: To solve trigonometric calculations.

Details:
- Calculates the arc tangent of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be a tangent value.
- The angular value returned is in radians with a range of 0 to 6.283185. (To convert radians to degrees, multiply by 57.29578.)

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE ARC TAN OF**

	X	*variable float*
Put Result In	RADIANS	*variable float*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Use TAKE TAN OF if the angle is known and the tangent is desired.

Error Codes:
Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE ARC COS OF, TAKE ARC SIN OF

TAKE COS OF — Mathematical

Function: To derive the cosine of an angle.

Typical Use: To solve trigonometric and electrical power calculations.

Details:

- Calculates the cosine of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be expressed in radians. (To convert degrees to radians, divide by 57.29578.)
- Negative radians are converted to positive radians before the cosine is calculated.
- Radian values in excess of 6.283185 (360°) will be treated as increments of 6.283185 before the cosine is calculated.
- The result is a sinusoidal value ranging from -1.0 to 1.0 that repeats every 6.283185 radians (360°).
- The following are examples of cosine calculations:

RADIANS	DEGREES	RESULT
0.0	0.0	1.0
0.785398	45	0.707106
1.570796	90	0.0
2.356194	135	-0.707106
3.141592	180	-1.0
3.926991	225	-0.707106
4.712388	270	0.0
5.497787	315	0.707106
6.283185	360	1.0

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE COS OF**

	RADIANS	*variable float*
Put Result In	COSINE	*variable float*

Notes:

- See the Mathematical Overview in Chapter 1 for important information.
- Electrical power factor is equal to the cosine of the angle by which the current lags the voltage.
- Use TAKE ARC COS OF if the cosine is known and the angle is desired.

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE ARC COS OF, TAKE SIN OF, TAKE TAN OF

TAKE COSH OF — Mathematical

Function: To derive the hyperbolic cosine of a value.

Typical Use: To solve hyperbolic calculations.

Details:
- Calculates the hyperbolic cosine of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be a value from -88.33654 to 88.72283.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE COSH OF**

	2.0	*constant float*
Put Result In	ANSWER	*variable float*

Error Codes: Queue error 33 = Overflow error — result too large

See Also: TAKE SINH OF, TAKE TANH OF

TAKE NATURAL LOG OF — Mathematical

Function: To calculate the natural log (base e) of a value.

Typical Use: To solve natural log calculations.

Details:
- Takes the natural log of Argument 1 and places the result in Argument 2.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE NATURAL LOG OF**

	FERMENTATION RATE	*variable float*
Put Result In	RATE CALCULATION	*variable float*

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE NATURAL LOG OF, RAISE TO POWER

TAKE SIN OF — Mathematical

Function: To derive the sine of an angle.

Typical Use: To solve trigonometric calculations.

Details:

- Calculates the sine of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be expressed in radians. (To convert degrees to radians, divide by 57.29578.)
- Negative radians are converted to positive radians before the sine is calculated.
- Radian values in excess of 6.283185 (360°) will be treated as increments of 6.283185 before the sine is calculated.
- The result is a sinusoidal value ranging from -1.0 to 1.0 that repeats every 6.283185 radians (360°).
- The following are examples of sine calculations:

RADIANS	DEGREES	RESULT
0.0	0.0	0.0
0.785398	45	0.707106
1.570796	90	1.0
2.356194	135	0.707106
3.141592	180	0.0
3.926991	225	-0.707106
4.712388	270	-1.0
5.497787	315	-0.707106
6.283185	360	0.0

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE SIN OF**

	RADIANS	*variable float*
Put Result In	SINE	*variable float*

Notes:

- See the Mathematical Overview in Chapter 1 for important information.
- Use TAKE ARC SIN OF if the sine is known and the angle is desired.

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE ARC SIN OF, TAKE COS OF, TAKE TAN OF

TAKE SINH OF — Mathematical

Function: To derive the hyperbolic sine of a value.

Typical Use: To solve hyperbolic calculations.

Details:
- Calculates the hyperbolic sine of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be a value from -88.33654 to 88.72283.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE SINH OF**

	2.0	*constant float*
Put Result In	ANSWER	*variable float*

Error Codes: Queue error 33 = Overflow error — result too large

See Also: TAKE COSH OF, TAKE TANH OF

TAKE SQUARE ROOT OF — Mathematical

Function: To calculate the square root of a value.

Typical Use: To solve square root calculations.

Details:
- Takes the square root of Argument 1 and places the result in Argument 2.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE SQUARE ROOT OF**

	4	*constant integer*
Put Result In	TWO	*variable integer*

Notes:
- See the Mathematical Overview in Chapter 1 for important information.
- Executes faster than raising a number to the 0.5 power.
- Taking the square root of a negative value will result in zero.
- To convert a differential pressure value representing flow to the proper engineering units, convert its current value to a number between 0 and 1, take the square root of this number, then convert it to the desired engineering units. For example: A 0–100" flow signal that represents 0–50,000 CFH has a value of 50. 50/100 = 0.5. The square root of 0.5 is 0.7071. 0.7071 times 50,000 = 35355 CFH.

Error Codes:
Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: RAISE TO POWER

TAKE TAN OF — Mathematical

Function: To derive the tangent of an angle.

Typical Use: To solve electrical Q factor and trigonometric calculations.

Details:

- Calculates the tangent of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be expressed in radians. (To convert degrees to radians, divide by 57.29578.)
- Negative radians are converted to positive radians before the tangent is calculated.
- Radian values in excess of 6.283185 (360°) will be treated as increments of 6.283185 before the tangent is calculated.
- The result is an extremely nonlinear value ranging from -¥ to ¥ that repeats every 6.283185 radians (360°).

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE TAN OF**

	RADIANS	*variable float*
Put Result In	TANGENT	*variable float*

Notes:

- See the Mathematical Overview in Chapter 1 for important information.
- Electrical Q factor is equal to the tangent of the angle by which the current lags the voltage in a coil.
- Use TAKE ARC TAN OF if the tangent is known and the angle is desired.

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE ARC TAN OF, TAKE COS OF, TAKE SIN OF

TAKE TANH OF — Mathematical

Function: To derive the hyperbolic tangent of a value.

Typical Use: To solve hyperbolic calculations.

Details:
- Calculates the hyperbolic tangent of Argument 1 and places the result in Argument 2.
- Argument 1 (the operand) must be a value between -8.21 and 8.665.
- The result is a value ranging from -1.0 to 1.0.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG OUT
ANALOG OUT	VARIABLE FLOAT
CONSTANT FLOAT	VARIABLE INTEGER
CONSTANT INTEGER	VARIABLE TIMER
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: **TAKE TANH OF**

	2.0	*constant float*
Put Result In	ANSWER	*variable float*

Error Codes:

Queue error 33 = Overflow error — result too large
Queue error 35 = Not a number — result invalid

See Also: TAKE COSH OF, TAKE SINH OF

PID OPERATIONS

DISABLE PID LOOP — PID

Function: To disable communication between the program in the Mistic controller and the PID.

Typical Use: To disconnect the program from a specified PID for simulation and program testing.

Details:
- All PID communication is enabled by default.
- Does not affect the PID at the I/O unit in any way. While communication to the PID is disabled, any Cyrano command that refers to it by name will not affect it because the command will only have access to the IVAL.
- No changes can be made to the PID by the program in the Mistic controller while the PID is disabled.

Arguments:

ARGUMENT 1
PID LOOP

Example: **DISABLE PID LOOP**

HEATER 3	*PID loop name*

Notes:
- To stop updating the PID output, use SET PID MANUAL MODE instead of DISABLE PID LOOP.
- Many additional PID loop control features are available, including DEACTIVATE PID OUTPUT. See the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270) or consult the Opto 22 BBS.

Dependencies:
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET PID MANUAL MODE

ENABLE PID LOOP — PID

Function: To enable communication between the program in the Mistic controller and the PID.

Typical Use: To reconnect the program to a specified PID after simulation or program testing.

Details:
- All PID communication is enabled by default.
- Does not affect the PID at the I/O unit in any way. While communication to the PID is enabled, any Cyrano command that refers to it by name will have full access.

Arguments:

ARGUMENT 1
PID LOOP

Example:

ENABLE PID LOOP

HEATER 3	*PID loop name*

Notes:
- Many additional PID loop control features are available, including ACTIVATE PID OUTPUT. See the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270) or consult the Opto 22 BBS.

Dependencies:
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: DISABLE PID LOOP

READ OUTPUT RATE OF CHANGE — PID

Function: To read the output rate-of-change limit of the PID.

Typical Use: To verify that the output rate-of-change limit is as expected.

Details:
- The output rate-of-change value defines how much the PID output can change per scan period. The units are the same as those defined for the PID output channel.
- The default value is the span of the output channel. This allows the PID output to move as much as 100% per scan period. For example, if the PID output channel is 4–20 mA, 16.00 would be returned by default, representing 100% of the span.

Arguments:

ARGUMENT 1	ARGUMENT 2
PID LOOP	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ OUTPUT RATE OF CHANGE**

From	HEATER 3	*PID loop name*
Move To	PID RATE LIMIT	*variable float (the value read)*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Many additional PID loop control features are available. See the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270) or consult the Opto 22 BBS.

Dependencies:
- Communication to the PID must be enabled for this command to read the actual value from the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET OUTPUT RATE OF CHANGE, SET PID SCAN RATE

READ PID INPUT — PID

Function: To read the input value (also known as the process variable) of the PID.

Typical Use: To verify that the input to the PID is within the working range.

Details:
- The value read has the same engineering units as the specified PID input channel.
- A value of -32,768 means the input is out of range and the PID output is no longer being updated.

Arguments:

ARGUMENT 1	ARGUMENT 2
PID LOOP	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ PID INPUT**

From	HEATER 3	*PID loop name*
Move To	PID INPUT VALUE	*variable float (the value read)*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Use to detect bad or out-of-range PID input values. When such a value is found, use the MOVE command to change the PID output as required.

Dependencies:
- Communication to the PID must be enabled for this command to read the actual value from the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP

READ PID OUTPUT — PID

Function: To read the output value of the PID.

Typical Use: To read the PID output and send it to a digital time proportional output (TPO) on a digital I/O unit.

Details:
- The value read has the same engineering units as the specified PID output channel.

Arguments:

ARGUMENT 1	ARGUMENT 2
PID LOOP	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ PID OUTPUT**

From	HEATER 3	*PID loop name*
Move To	PID OUTPUT VALUE	*variable float (the value read)*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Define the output channel as one of the upper eight channels (these channels do not have to physically exist).
- Scale this output channel 0–100, since the digital TPO wants to see a range of 0–100.
- Use SET TIME PROP PERCENT to send the value read from the PID output to the digital TPO. Do this based on elapsed time. For example, if the TPO period is five seconds, send the value read at least every five seconds.
- This command can also be used to detect when the PID output updates (which is always at the end of the scan period).

Dependencies:
- Communication to the PID must be enabled for this command to read the actual value from the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET TIME PROP PERCENT, SET TIME PROP OUTPUT

READ PID SETPOINT — PID

Function: To read the setpoint value of the PID.

Typical Use: To verify that the setpoint of the PID is as expected and to store the setpoint for later use.

Details:
- The value read has the same engineering units as the specified PID setpoint.
- The setpoint can be an analog channel or it can come from the program in the Mistic controller using SET PID SETPOINT.

Arguments:

ARGUMENT 1	ARGUMENT 2
PID LOOP	VARIABLE FLOAT VARIABLE INTEGER

Example: **READ PID SETPOINT**

From	HEATER 3	*PID loop name*
Move To	PID SETPOINT VALUE	*variable float (the value read)*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Can be used to detect and log changes made to the PID setpoint.

Dependencies:
- Communication to the PID must be enabled for this command to read the actual value from the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET PID SETPOINT.

SET D TERM — PID

Function: To change the derivative value of the PID.

Typical Use: To improve PID performance in systems with long delays.

Details:
- The derivative is used to determine how much effect the change-in-slope of the PID input should have on the PID output.
- Derivative is useful in predicting the future value of the PID input based on the change in trend of the PID input as recorded during the last three scan periods.
- Derivative is used in systems with long delays between the time that the PID output changes and the time that the PID input responds to the change.
- Too much derivative results in excessive amounts of PID output change.
- Too little derivative results in a PID output that is always out of phase with the PID input in systems with long delays.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	PID LOOP
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET D TERM**

From	D TERM VALUE	*variable float (the value to send)*
To	HEATER 3	*PID loop name*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Leave the derivative at zero unless you are sure you need it and until the gain and integral have been determined.
- The derivative is multiplied by the gain. Hence, for example, if the gain is doubled, you may wish to cut the derivative in half to keep its effect the same.
- Typical derivative values range from 0.001 to 20.
- Use sparingly. A little derivative goes a long way!

Dependencies:
- The P term (gain) must not be zero.
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP

SET I TERM PID

Function: To change the integral value of the PID.

Typical Use: To improve PID performance in systems with steady-state errors.

Details:

- The integral is used to reduce the error between the PID setpoint and the PID input to zero under steady-state conditions. Its value determines how much the error affects the PID output.
- Always use a positive integral value. Do not use zero.
- Too much integral results in excessive amounts of PID output change.
- Too little integral results in long lasting errors between the PID input and the PID setpoint.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	PID LOOP
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET I TERM**

From	I TERM VALUE	*variable float (the value to send)*
To	HEATER 3	*PID loop name*

Notes:

- See the PID Overview in Chapter 1 for important information.
- Use an initial value of 1.0 until a better value is determined.
- The integral is multiplied by the gain. Hence, for example, if the gain is doubled, you may wish to cut the integral in half to keep its effect the same.
- Typical integral values range from 0.1 to 20.

Dependencies:

- P term (gain) must not be zero.
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP

SET OUTPUT RATE OF CHANGE — PID

Function: To change the output rate-of-change limit of the PID.

Typical Use: To slow down the PID output rate-of-change as it responds to large input or setpoint changes.

Details:
- Slows the PID output rate-of-change when a large change occurs to the setpoint or the input.
- The output rate-of-change value defines how much the PID output can change per scan period. The units are the same as those defined for the PID output channel.
- The default value is the span of the output channel. This allows the PID output to move as much as 100% per scan period. For example, if the PID output channel is 4–20 mA, 16.00 would be returned by default, representing 100% of the span.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	PID LOOP
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET OUTPUT RATE OF CHANGE**

From	PID RATE LIMIT	*variable float (the value to send)*
To	HEATER 3	*PID loop name*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Tune the loop before reducing the output rate-of-change.
- Set the output rate-of-change back to 100% before retuning the PID.
- Many additional PID loop control features are available. See the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270) or consult the Opto 22 BBS.

Dependencies:
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, READ OUTPUT RATE OF CHANGE, SET PID SCAN RATE

SET P TERM PID

Function: To change the gain value of the PID.

Typical Use: To tune the PID for more or less aggressive performance.

Details:
- Gain is the inverse of "proportional band," a term used in many PID applications.
- Gain is used to determine the amount of PID output response to a change in PID input or PID setpoint.
- Always use a non-zero gain value.
- Gain has a direct multiplying effect on the integral and derivative values.
- Use a negative gain to reverse the direction of the PID output (typical for cooling applications).
- Too much gain results in excessive amounts of PID output change.
- Too little gain results in long lasting errors between the PID input and the PID setpoint.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	PID LOOP
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET P TERM**

From	GAIN	*variable float (the value to send)*
To	HEATER 3	*PID loop name*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Use an initial value of 1.0 or -1.0 until a better value is determined.
- Typical gain values range from 1 to 40 and -1 to -40.
- Use more gain to improve response to step changes.
- Use less gain to improve stability.

Dependencies:
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP

SET PID AUTO MODE — PID

Function: To change the mode of the PID to auto.

Typical Use: To put the PID in auto mode from manual mode.

Details:
- While in auto mode, the PID output functions normally.

Arguments:

ARGUMENT 1
PID LOOP

Example: **SET PID AUTO MODE**

HEATER 3	*PID loop name*

Notes:
- Use SET PID SETPOINT after using this command to restore the PID setpoint to its original value. This assumes that "setpoint tracking" is enabled (as it is by factory default) and that the original setpoint was saved prior to switching to manual mode.
- Even when the PID is in auto mode, the PID output can be changed manually. Use the MOVE command, the Debugger, or an MMI to write directly to the PID output analog channel. The new PID output value will be the starting value used at the end of the next PID scan period. This procedure can be helpful in pre-setting the PID output where it needs to be.

Dependencies:
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET PID MANUAL MODE

SET PID INPUT — PID

Function: To send an input value (also known as the process variable) to the PID when its input does not come from an analog input channel on the same I/O unit.

Typical Use: To get an input from another I/O unit and forward it to the PID.

Details:

- Use this command based on a timed interval. For example, if the PID scan rate is 1 second, send the input value to the PID approximately every second (anywhere from 0.9 seconds to 1.0 seconds would be adequate).

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	PID LOOP
ANALOG OUT	
CONSTANT FLOAT	
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET PID INPUT**

To	INPUT VALUE	*variable float (the value to send)*
On	HEATER 3	*PID loop name*

Notes:

- See the PID Overview in Chapter 1 for important information.
- Do not send the input value to the PID any slower than the PID scan rate, since this will adversely affect the PID performance.
- Sending the input value to the PID more than 10 times per second can slow the performance of event/reactions on the I/O unit.

Dependencies:

- Must configure the PID input to be FROM HOST.
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET PID SCAN RATE

SET PID MANUAL MODE — PID

Function: To change the mode of the PID to manual.

Typical Use: To put the PID in manual mode for maintenance, for testing, or simply to turn it off.

Details:

- While in manual mode, the PID output is not updated by the PID calculation. Instead, it retains its last value.
- To change the PID output value, use the MOVE command, the Debugger, or an MMI to write directly to the PID output analog channel. The new PID output value will be the starting value when the PID is changed to auto mode.
- While in manual mode, the PID setpoint is changed to match the PID input value. Although this provides for a "bumpless transfer" when switching back to auto mode, the original PID setpoint is lost. This feature can be disabled by changing the PID control word. See the Mistic *Analog and Digital Commands Manual* (Opto 22 form 270) or consult the Opto 22 BBS.

Arguments:

ARGUMENT 1
PID LOOP

Example: **SET PID MANUAL MODE**

HEATER 3	*PID loop name*

Notes:

- Use READ PID SETPOINT first to save the PID setpoint to a variable float.

Dependencies:

- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, SET PID AUTO MODE

SET PID SCAN RATE — PID

Function: To change the scan rate (update period) for a PID calculation.

Typical Use: To adapt a PID to the characteristics of the closed-loop control system under program control.

Details:
- This is the most important parameter of all the configurable PID parameters. Note that the loop may be impossible to tune if the scan rate is significantly different from the loop dead time.
- The value to send is in seconds. Values range from 0.1 to 6553.5 seconds in 0.1-second increments. The default is 0.1 seconds.
- This command is useful for adapting a PID to work for either heating or cooling when the heat mode has a different loop dead time than the cool mode.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	PID LOOP
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET PID SCAN RATE**

To	SCAN RATE	*variable float (the value to send)*
On	HEATER 3	*PID loop name*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Do not use frequently since this will adversely affect the PID performance.

Dependencies:
- Communication to the PID must be enabled for this command to send the value to the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP

SET PID SETPOINT — PID

Function: To change the setpoint value of the PID.

Typical Use: To raise or lower the setpoint or to restore it to its original value.

Details:
- The value to send has the same engineering units as the specified PID input.
- Values are the same as those for the PID input.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	PID LOOP
ANALOG OUT	
CONSTANT FLOAT	
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **SET PID SETPOINT**

From	PID SETPOINT VALUE	*variable float (the value to send)*
To	HEATER 3	*PID loop name*

Notes:
- See the PID Overview in Chapter 1 for important information.
- Sending the setpoint value to the PID more than 10 times per second can slow the performance of event/reactions on the I/O unit.
- Send a new setpoint value only when necessary.

Dependencies:
- Communication to the PID must be enabled for this command to read the actual value from the PID.
- Requires an analog multifunction I/O unit (HRD I/O units are not supported).

See Also: ENABLE PID LOOP, READ PID SETPOINT

STRING OPERATIONS

APPEND CHARACTER — String

Function: To add a character to the end of a variable string.

Typical Use: To build strings consisting of non-printable or binary characters.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- The character is represented by an ASCII value. A space is a character 32 and a “1” is a character 49.
- Appending a value of zero is legal and will append a null byte.
- If the appended value is greater than 255 (hex FF) or less than 0, the value will be truncated to eight bits, i.e., -2 becomes hex FE and 257 (hex 101) becomes 1.
- Floats (if used) are automatically rounded to integers before conversion.
- If the string cannot hold any more characters, the character will not be appended.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	VARIABLE STRING
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: The following example appends a “!” to a string (for example, “Hello” would become “Hello!”):

APPEND CHARACTER

From	33	*constant integer for “!”*
To	HELLO STRING	*variable string*

The following example appends an ETX (character 3) to a string. An ETX or some other terminating character may be required when sending commands to serial devices, such as barcode printers, scales, or single-loop controllers.

APPEND CHARACTER

From	3	*constant integer for “ETX”*
To	COMMAND STRING	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- Always use MOVE STRING before using this command if the string needs to be cleared. Moving an empty string (“”) to a variable string will clear it.

Dependencies:
- The variable string must be wide enough to hold one more character.

See Also: APPEND STRING

APPEND STRING — String

Function: To add a string to the end of another variable string.

Typical Use: To build strings.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- If the variable string cannot hold all of the appended string, the remaining portion of the string to be appended will be discarded.
- Single characters can be appended (yielding the same result as an APPEND CHARACTER). For example, to append a “space,” use the space bar rather than the number 32.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING VARIABLE STRING	VARIABLE STRING

Example: The following example appends the string “ world” to a string. For example, “Hello” would become “Hello world” (note the space before the “w” in “ world”).

APPEND STRING

From	“ world”	*constant string*
To	HELLO STRING	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- Always use MOVE STRING before using this command if the string needs to be cleared. Moving an empty string (“”) to a variable string will clear it.

Dependencies:
- The variable string must be wide enough to hold the appended string.

See Also: APPEND CHARACTER

CONV. FORMATTED # TO HEX STR — String

Function: To convert an integer to a formatted hex string having a specified length, or to convert a float to an eight-byte IEEE hex format.

Typical Uses:
- To allow efficient transfer of numeric data via a serial port. (The largest number can be sent using only eight hex characters.)
- To print a hex number or to send it to another device with a fixed length.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- The *Length* parameter (Argument 2) specifies the final length of the resulting string. Leading zeros are added if required.
- To send a float value in native IEEE format, set Argument 2 to 8 and use a variable or constant float. Use CONV. IEEE HEX STRING TO NUMBER to convert the eight hex characters back to a float.
- If the resulting hex string is wider than the specified length, the most significant hex characters will be discarded.
- If the declared width of the variable string is less than the specified length, the remaining portion (least significant characters) of the formatted string will be discarded.
- Upper case is used for all hex characters, i.e., 1000 decimal is represented as 3E8 rather than 3e8.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	VARIABLE STRING
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
VARIABLE FLOAT		
VARIABLE INTEGER		

Example: The following example converts a decimal integer to a hex string. If MY ADDRESS has the value 255, the resulting hex string would be “00FF” because *Length* is 4. If *Length* had been 2, the hex string would have become “FF.”

CONV. FORMATTED # TO HEX STR

From	MY ADDRESS	*variable integer*
Length	4	*constant integer*
Move to	ADDRESS AS HEX	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- *Caution:* Do not use a float where an integer would suffice. Floats are not automatically converted to integers with this command.
- Must use a *Length* of 8 when converting a float.

Dependencies:
- The variable string must be wide enough to hold the hex string.

See Also: CONVERT FORMATTED # TO STR., CONVERT NUMBER TO HEX STRING, CONVERT NUMBER TO STRING, CONVERT NUMBER TO STR. FIELD

CONV. IEEE HEX STRING TO NUMBER — String

Function: To convert a hex string representing an IEEE float in native IEEE format to a number.

Typical Use: To retrieve the float value previously stored as hex after using CONV. FORMATTED # TO HEX STR.

Details:
- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Use between Mistic controllers or other computers that use the IEEE format when efficiency of communications is desired.
- The four bytes in memory (in IEEE float format) that hold the float value are converted to eight hex bytes.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: The following example converts a hex string into a float value. For example, if STRING FROM PORT contains "418E6666" then MY FLOAT VAL becomes 17.8.

CONV. IEEE HEX STRING TO NUMBER

From	STRING FROM PORT	*variable string*
Move To	MY FLOAT VAL	*variable float*

Notes:
- See the String Overview in Chapter 1 for important information.

See Also: CONV. FORMATTED # TO HEX STR, CONVERT HEX STRING TO NUMBER

CONV. FLOATING POINT # TO STR. (formerly CONV. FORMATTED # TO STR.) — String

Function: To convert a float to a formatted string having a specified length and number of digits to the right of the decimal.

Typical Use: To print a float or send it to another device using a specific format or length.

Details:

- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.
- The *Length* parameter (Argument 2) specifies the final length of the resulting string, including the decimal point. Leading spaces (character 32) are added if required.
- The *Decimals* parameter (Argument 3) specifies the number of digits to the right of the decimal point.
- Rounding occurs whenever digits on the right must be dropped.
- Digits to the left of the decimal point are never dropped.
- If the whole number portion (digits to the left of the decimal plus the decimal itself) of the resulting string would be larger than its allocated space, the resulting string will be filled with asterisks to alert you to the problem. For example, if the value to convert is 123.4567 with a *Length* value of 5 and a *Decimals* value of 2, the space allocated to the whole number portion is only three (5 - 2). Since four characters ("123.") are required, the formatted number "123.46" will not fit, so "*****" will be moved to the destination string.
- If the declared width of the variable string is less than the specified length, the remaining portion (least significant characters) of the formatted string will be discarded.
- Although integers can also be converted, significant rounding errors will occur for values of 1,000,000 or more.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
ANALOG IN	CONSTANT INTEGER	CONSTANT INTEGER	VARIABLE STRING
ANALOG OUT	VARIABLE INTEGER	VARIABLE INTEGER	
CONSTANT FLOAT			
CONSTANT INTEGER			
VARIABLE FLOAT			
VARIABLE INTEGER			

Example: The following example converts a decimal number in variable MY VALUE to a string (for example, if MY VALUE is 12.3435, the string becomes "12.34"):

CONV. FLOATING POINT # TO STR.

From	MY VALUE	*variable float*
Length	5	*constant integer*
Decimals	2	*constant integer*
Move to	VALUE AS STRING	*variable string*

Notes:

- See the String Overview in Chapter 1 for important information.
- Set decimals to zero to get an integer. Normal rounding will occur.

Dependencies:

- The variable string must be wide enough to hold the resulting formatted string.

See Also: CONV. STR. TO FLOATING POINT #, CONVERT NUMBER TO STRING

CONVERT HEX STRING TO NUMBER — String

Function: To convert a hex string value to an integer value.

Typical Use: To accommodate communications where values may be represented by hex strings.

Details:

- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- An empty string results in a value of zero.
- Conversion is not case-sensitive. For example, the strings “FF,” “ff,” “fF,” and “Ff” all convert to a value of 255.
- Legal hex characters are “0” through “1,” “A” through “F,” and “a” through “f.”
- A string containing an illegal character will be converted up to the point just before the illegal character. For example, the strings “AG” and “A 123” will both convert to 10 (the value of “A”).
- Leading spaces in strings will convert to zeros.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: **CONVERT HEX STRING TO NUMBER**

From	STRING FROM PORT	*variable string*
Move To	INT VALUE	*variable integer*

Notes:

- See the String Overview in Chapter 1 for important information.
- Must use CONV. IEEE HEX STRING TO NUMBER if the hex string contains an IEEE float.

See Also: CONVERT NUMBER TO HEX STRING, CONVERT STRING TO NUMBER, CONV. IEEE HEX STRING TO NUMBER

CONVERT NUMBER TO HEX STRING — String

Function: To convert a decimal integer to a hex string.

Typical Uses:
- To send an integer value with a predetermined length to another Mistic controller.
- To print a hex representation of a number or to send it to another device.

Details:
- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Does not add leading zeros or spaces.
- If the resulting string is too big, the string will be truncated. No error will be reported and memory will not be corrupted.
- If the declared width of the variable string is less than the resulting hex string length, the remaining portion of the hex string (least significant characters) will be discarded.
- Upper case is used for all hex characters, i.e., 1000 decimal is represented as 3E8 rather than 3e8.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE STRING
ANALOG OUT	
CONSTANT FLOAT	
CONSTANT INTEGER	
DIGITAL MF I/O UNIT	
DIGITAL NMF I/O UNIT	
REM SMPL I/O UNIT	
VARIABLE FLOAT	
VARIABLE INTEGER	
VARIABLE TIMER	

Example: The following example converts a number in MY ADDRESS to a hex string (for example, if MY ADDRESS has the value 256, the hex string becomes "100"):

CONVERT NUMBER TO HEX STRING

From	MY ADDRESS	*variable integer*
Move to	ADDRESS AS HEX	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- Must use CONV. FORMATTED # TO HEX STR when converting floats.

Dependencies:
- The variable string must be wide enough to hold the resulting hex string.

See Also: CONV. FORMATTED # TO HEX STR, CONVERT FORMATTED # TO STR., CONVERT NUMBER TO STRING, CONVERT NUMBER TO STR. FIELD

CONVERT NUMBER TO STR. FIELD — String

Function: To convert a number to a string using a specified minimum length.

Typical Use: To fix the length of an integer before sending it to a serial printer or to another device.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- The resulting string length will be greater than or equal to the length specified in the *Length* parameter (Argument 2).
- If the declared width of the variable string is less than the resulting string length, the remaining portion of the string (characters on the right) will be discarded.
- A value whose length is less than that specified will have leading spaces added as necessary.
- A value whose length is equal to or greater than the specified length will be sent as is.
- Examples:

23456	becomes	23456	There are six digits (one leading space in front of the 2).
0	becomes	0	There are six digits (five leading spaces in front of the 0).
2345678	becomes	2345678	The six-digit specified length is ignored.
12.3	becomes	1.23e01	The six-digit specified length is ignored.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	VARIABLE STRING
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
VARIABLE FLOAT		
VARIABLE INTEGER		

Example: **CONVERT NUMBER TO STR. FIELD**

From	VALUE	*variable integer*
Length	6	*constant integer*
Move to	VALUE AS STRING	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- Use CONV. FORMATTED # TO STR. to better control the resulting format, if desired.

Dependencies:
- The variable string must be wide enough to hold the resulting string.

See Also: CONV. FORMATTED # TO HEX STR, CONVERT FORMATTED # TO STR., CONVERT NUMBER TO STRING, CONVERT NUMBER TO HEX STRING

CONVERT NUMBER TO STRING — String

Function: To convert a decimal number to a string.

Typical Use: To print a number or send it to another device.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Represents floating point values in scientific notation (e.g., 1.234e+01 rather than 12.34).
- If the declared width of the variable string is less than the resulting string length, the remaining portion of the string (characters on the right) will be discarded.
- Examples:

12.3456	becomes	1.23456e+01	Note the exponential format for floats.
12345	becomes	12345	Note no change for integers.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE STRING
ANALOG OUT	
CONSTANT FLOAT	
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: The following example converts a decimal number in MY VALUE to a string (for example, if MY VALUE is 12.34, the string becomes “1.234e+01”; if MY VALUE is the integer value 1234, the string becomes “1234”):

CONVERT NUMBER TO STRING

From	MY VALUE	*variable float*
Move to	VALUE AS STRING	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- To avoid scientific notation or to have greater control over format, use CONV. FORMATTED # TO STR. instead.

Dependencies:
- The variable string must be wide enough to hold the resulting string.

See Also: CONV. STR. TO INTEGER #, CONV. FLOATING POINT # TO STR.,

CONV. STR. TO FLOATING POINT # (formerly CONVERT STRING TO NUMBER) — String

Function: To convert a string to a float value.

Typical Use: To accommodate communications or operator entry, since all characters from these sources are strings.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Although this command can be used to convert a string to an integer, significant rounding errors will occur for values of 1,000,000 or more.
- Valid, convertible characters are 0 to 9, the decimal point, and “e” (natural log base). Spaces are also considered valid, although they are not converted. Note in particular that commas are invalid.
- Strings are analyzed from left to right.
- Spaces divide text blocks within a string.
- If a space appears to the right of a valid text block, the space and all characters to its right will be ignored. For example, “123 4” and “123.0 X” both convert to 123.0.
- If an invalid character is found, the string will be converted to 0.0. For example, “X 22.2 4” and “1,234 45” both convert to 0.0, since the X in the first string and the comma in the second are invalid. Note, however, that “45 1,234” would convert to 45.0, since the invalid character (“,”) would be ignored once the valid text block (“45”) was found.
- The following are string-to-float conversion examples:

STRING	FLOAT
“”	0.0
“A12”	0.0
“123P”	0.0
“123 P”	123.0
“123.456”	123.456
“22 33 44”	22.0
“ 22.11”	22.11
“1,234.00”	0.0
“1234.00”	1234.0
“1.23e01”	12.3

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	VARIABLE STRING
ANALOG OUT	
CONSTANT FLOAT	
CONSTANT INTEGER	
VARIABLE FLOAT	
VARIABLE INTEGER	

Example: **CONV. STR. TO FLOATING POINT #**

From	STRING FROM PORT	*variable string*
Move To	FLOAT VALUE	*variable float*

Notes:
- See the String Overview in Chapter 1 for important information.

See Also: CONV. FLOATING POINT # TO STR.**,** CONV. STR. TO INTEGER #

CONV. STR. TO INTEGER # String

Function: To convert a string to an integer value.

Typical Use: To accommodate communications or operator entry, since all characters from these sources are strings.

Details:

- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Valid, convertible characters are 0 to 9. Decimals are valid if they are a part of text that can be considered a float value. Spaces are also considered valid, although they are not converted. Note in particular that commas are invalid.
- Strings are analyzed from left to right.
- Text that could be read as a float value is rounded to an integer value. For example, "123.6" converts to 124.
- Spaces divide text blocks within a string.
- If a space appears to the right of a valid text block, the space and all characters to its right will be ignored. For example, "123 4" and "123.0 X" both convert to 123.
- If an invalid character is found, the string will be converted to 0. For example, "X 22 4" and "1,234 45" both convert to 0, since the X in the first string and the comma in the second are invalid. Note, however, that "45 1,234" would convert to 45, since the invalid character (",") would be ignored once the valid text block ("45") was found.
- The following are string-to-integer conversion examples:

STRING	INTEGER
""	0
"A12"	0
"123P"	0
"123 P"	123
"123.456"	123
"22 33 44"	22
" 22.51"	23
"1,234"	0
"1234.00"	1234

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: **CONV. STR. TO INTEGER #**

From	STRING FROM PORT	*variable string*
Move To	INT VAL	*variable integer*

Notes:

- See the String Overview in Chapter 1 for important information.
- Avoid alpha characters. Stick with 0 to 9.

See Also: CONV. STR. TO FLOATING POINT #, CONVERT NUMBER TO STRING

GET NTH CHARACTER — String

Function: To get the decimal ASCII value for a character in a string.

Typical Use: To examine characters in a string one by one especially when the characters may not be printable ASCII.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Valid range for the *Index* parameter (Argument 2) is 1 to the string length.
- A negative result (-46) indicates an error in the value of the *Index* parameter used.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT INTEGER	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER	VARIABLE INTEGER

Example: The following example gets the decimal ASCII value for a character in the string “ABC.” If the *Index* is 1, the returned value will be 65 (the decimal ASCII value for “A”).

GET NTH CHARACTER

	“ABC”	*constant string*
Index	INDEX	*variable integer*
Put Result In	ASCII VAL	*variable integer*

Notes:
- See the String Overview in Chapter 1 for important information.
- Use to search a string for a particular character, such as a carriage return (character 13).
- To avoid searching past the end of the string, use GET STRING LENGTH to determine the end of the string.

Error Codes: -46 = Bad limit — index was negative or greater than the string length

See Also: GET SUBSTRING, APPEND CHARACTER, GET STRING LENGTH

GET STRING LENGTH — String

Function: To get the length of a string.

Typical Use: To determine if a string is empty prior to searching it for a character.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- An empty string has a length of zero.
- The string length is not the same as the width. Width is the maximum string length and is set in the Cyrano Configurator; it does not change at run time. String length, on the other hand, may change dynamically as the string is modified at run time.
- Spaces and nulls count as part of the length.
- A string with width 10 containing “Hello ” has a length of six (five for “Hello” plus one for the trailing space).

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: The following example gets the length of the string MY STRING (for example, if MY STRING is “ABC” then STRING LEN is 3):

GET STRING LENGTH

	MY STRING	*constant string*
Put Result In	STRING LEN	*variable integer*

Notes:
- See the String Overview in Chapter 1 for important information.
- Use before GET NTH CHARACTER to stay within the string length.

See Also: GET NTH CHARACTER

GET SUBSTRING — String

Function: To copy a portion of a string.

Typical Uses:
- To parse or extract data from a string.
- To skip leading or trailing characters.
- To extract data from strings that may contain starting and ending character sequences generated by barcode readers or scales.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Valid range for *Start At* (Argument 2) is 1 to the string length. If it is less than 1, 1 will be assumed.
- If the combination of *Start At* (Argument 2) and *Number Of* (Argument 3) extend beyond the length of the source string, only the available portion of the source string will be returned.
- The following are examples of this command applied to the string “MONTUEWEDTHRFRI”:

START AT	NUMBER OF	SUBSTRING RETURNED
1	3	“MON”
4	3	“TUE”
1	4	“MONT”
14	3	“RI”
16	5	“”

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	ARGUMENT 4
CONSTANT STRING VARIABLE STRING	CONSTANT INTEGER VARIABLE INTEGER	CONSTANT INTEGER VARIABLE INTEGER	VARIABLE STRING

Example: The following example gets a single day from the string “MONTUEWEDTHRFRI”:

GET SUBSTRING

	“MONTUEWEDTHRFRI”	*constant string*
Start At	INDEX	*variable integer*
Number Of	3	*constant integer*
Move To	STRING	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.
- You can get text that follows a delimiter (such as a space) within a string. Create a loop that first uses GET NTH CHARACTER to extract a character, then compares it to the delimiter (character 32 in the case of a space). If the character is equal to the delimiter, add 1 to the N argument and use the new N as the *Start At* parameter above.
- See MOVE FROM STRING TABLE for a similar example.

See Also: GET NTH CHARACTER

MOVE FROM STRING TABLE — String

Function: To copy a string from a string table.

Typical Uses:
- To create a numeric-to-string lookup table.
- To retrieve strings from a table for further processing.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Valid range for *Index* (Argument 1) is zero to the table length (size).

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT INTEGER VARIABLE INTEGER	STRING TABLE	VARIABLE STRING

Example: The following example performs a numeric-to-string-table lookup. Given the numeric value for the day of week, the command below gets the name of the day of week from a string table. Use GET DAY OF WEEK to get the value to use for the *Index.*

MOVE FROM STRING TABLE

Index	INDEX	*variable integer*
From	STRING TABLE	*string table*
To	STRING	*variable string*

The results of this command are as follows:

INDEX	STRING
0	“SUN”
1	“MON”
2	“TUE”
3	“WED”
4	“THU”
5	“FRI”
6	“SAT”

Notes:
- See the String Overview in Chapter 1 for important information.
- A string table is a good way to correlate a number to a string.
- Use MOVE TO STRING TABLE or the Init utility to load the table with data.
- Multiple string tables can be used to create small databases of information. For example, one string table could contain a product name and another could contain the product ID code or barcode. It is essential to keep all related information at the same *Index* in each table.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE TO STRING TABLE, EQUAL TO STRING TABLE DATA, GET SUBSTRING, GET SIZE OF STRING TABLE

MOVE STRING — String

Function: To copy the contents of one string to another.

Typical Use: To save, initialize, or clear strings.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- If the width of the destination variable string is less than the width of the source, the remaining portion of the source string (characters on the right) will be discarded.
- The contents of the destination string are replaced with the source string.
- The length of the destination string will become that of the source string unless the declared width of the destination is less than the length of the source, in which case the length of the destination will match its declared width.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE STRING
VARIABLE STRING	

Example: The following example initializes a string variable to “Hello”:

MOVE STRING

From	“Hello”	*constant string*
To	HELLO STRING	*variable string*

The following example clears a variable string:

MOVE STRING

From	“”	*constant string (empty)*
Move to	MY STRING	*variable string*

Notes:
- See the String Overview in Chapter 1 for important information.

Dependencies:
- The destination variable string must be wide enough to hold the source string.

See Also: APPEND STRING, COPY TIME TO STRING, GET STRING (PORT), PRINT STRING (PORT)

MOVE TO STRING TABLE — String

Function: To put a string into a string table.

Typical Use: To load strings into a table for later retrieval.

Details:
- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.
- Valid range for *Index* (Argument 2) is zero to the table length (size).
- Strings with a length greater than the width of the table will be truncated to fit.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT INTEGER	STRING TABLE
VARIABLE STRING	VARIABLE INTEGER	

Example: **MOVE TO STRING TABLE**

From	"MON"	*constant string*
Index	INDEX	*variable integer*
To	STRING TABLE	*string table*

Notes:
- See the String Overview in Chapter 1 for important information.
- Use to log key events or application errors as if the string table were a "virtual line printer." For example, a string table called EVENT LOG could be used as a circular buffer to store strings containing the time, the date, and a description such as "12-25-96, 1:00:00, Clogged chimney alarm." A variable integer would also be required to "remember" the next available *Index* (where the next entry goes).
- Many additional string commands are available. These are "external" commands that require library support. Consult the Opto 22 BBS.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: MOVE FROM STRING TABLE, GET SIZE OF STRING TABLE

TEST EQUAL STRINGS — String

Function: To compare two strings for equality.

Typical Use: To check passwords or barcodes for an exact match.

Details:

- Determines if Arguments 1 and 2 are equal and puts result in Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
"OPTO"	"OPTO"	-1
"OPTO"	"Opto"	0
"22"	"22"	-1
"2 2"	"22"	0

- The result is -1 (True) if both strings are exactly the same, 0 (False) otherwise.
- Only an exact match on all characters (including leading or trailing spaces) will return a True.
- This test is case-sensitive. For example, a "T" does not equal a "t."
- The result can be sent directly to a digital output if desired.
- This operation is functionally equivalent to the STRING EQUAL condition.
- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT STRING	DIGITAL OUT
VARIABLE STRING	VARIABLE STRING	VARIABLE FLOAT
		VARIABLE INTEGER

Example: The following example compares a password variable to a string constant. The resulting value in IS AUTHORIZED could be used at several points in the program to determine if the user has sufficient authorization.

TEST EQUAL STRINGS

	PASSWORD	*variable string*
With	"LISA"	*constant string*
Put Result In	IS AUTHORIZED	*variable integer*

The following example compares a barcode to a string retrieved from a string table. This instruction would be located in a loop that retrieves each entry from a string table and performs this comparison.

TEST EQUAL STRINGS

	BARCODE	*variable string*
With	BARCODE FROM LIST	*variable string*
Put Result In	IS IN LIST	*variable integer*

Notes:

- See the String Overview in Chapter 1 for important information.
- Use EQUAL TO STRING TABLE DATA to compare with strings in a table.

See Also: STRING EQUAL, EQUAL TO STRING TABLE DATA

VERIFY CRC ON STRING — String

Function: To check the integrity of the contents of a string with imbedded CRC-16 Reverse characters.

Typical Use: In communications, to validate a received string before use.

Details:
- Quotes (“”) are used for readability only. They are not part of the string. Do not type them or expect to see them.
- The last two characters of the string must be the CRC characters.
- All characters with the exception of the two CRC characters are a part of the CRC calculation.
- The version of CRC used is CRC-16 Reverse with a seed of 0.
- Returns a zero for OK.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	VARIABLE FLOAT
VARIABLE STRING	VARIABLE INTEGER

Example: **VERIFY CRC ON STRING**

	RECV$	*variable string*
Put Result In	CRC STATUS	*variable integer*

Notes:
- See the String Overview in Chapter 1 for important information.

Error Codes:
0 = No error
-45 = CRC or checksum failed

See Also: SEND/RECEIVE PORT W/CRC, VERIFY CHECKSUM ON STRING

TIME/DATE OPERATIONS

COPY DATE TO STRING (EUR) Time/Date

Function: To read the date from the Mistic controller's real-time clock/calendar and put it into a string variable in the standard European format dd/mm/yy, where dd = day (01–31), mm = month (01–12), and yy = year (00–99).

Typical Use: To date stamp an event in a Cyrano program.

Details:
- If the current date is March 1, 1995, this operation would place the string "01/03/95" into the *String* parameter (Argument 1).
- The destination string should have a minimum width of eight.

Arguments:

ARGUMENT 1
VARIABLE STRING

Example: **COPY DATE TO STRING (EUR)**

String	EUROPEAN DATE STRING	*variable string*

Notes:
- This is a one-time read of the date. If the date changes, you will need to execute the command again to get the current date.

Error Codes: -48 = String too short

See Also: COPY DATE TO STRING (US), COPY TIME TO STRING, SET DATE, SET TIME

COPY DATE TO STRING (US) — Time/Date

Function: To read the date from the Mistic controller's real-time clock/calendar and put it into a string variable in the standard United States format mm/dd/yy, where mm = month (01–12), dd = day (01–31), and yy = year (00–99).

Typical Use: To date stamp an event in a Cyrano program.

Details:
- If the current date is March 1, 1995, this operation would place the string "03/01/95" into the *String* parameter (Argument 1).
- The destination string should have a minimum width of eight.

Arguments:

ARGUMENT 1
VARIABLE STRING

Example: **COPY DATE TO STRING (US)**

String	US DATE STRING	*variable string*

Notes:
- This is a one-time read of the date. If the date changes, you will need to execute the command again to get the current date.

Error Codes: -48 = String too short

See Also: COPY DATE TO STRING (EUR), COPY TIME TO STRING, SET DATE, SET TIME

COPY TIME TO STRING — Time/Date

Function: To read the time from the Mistic controller's real-time clock/calendar and put it into a string variable in the format hh:mm:ss, where hh = hours (00–23), mm = minutes (00–59), and ss = seconds (00–59).

Typical Use: To time stamp an event in a Cyrano program.

Details:

- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the current time is 2:35 p.m., this operation would place the string "14:35:00" into the *String* parameter (Argument 1).
- The destination string should have a minimum width of eight.

Arguments:

ARGUMENT 1
VARIABLE STRING

Example: **COPY TIME TO STRING**

String	TIME STRING	*variable string*

Notes:

- This is a one-time read of the time. If the time changes, you will need to execute the command again to get the current time.
- Put this command in a small program loop that executes frequently to ensure that the string always contains the current time.

Error Codes: -48 = String too short

See Also: COPY DATE TO STRING (EUR), COPY DATE TO STRING (US), SET DATE, SET TIME

GET DAY — Time/Date

Function: To read the day of the month (1 through 31) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in a Cyrano program based on the day of the month.

Details:
- The destination variable can be an integer or a float, although an integer is preferred.
- If the current date is March 2, 1995, this operation would place the value 2 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET DAY**

Move To	DAY OF MONTH	*variable integer*

Notes:
- This is a one-time read of the day of the month. If the date changes, you will need to execute this command again to get the current day of the month.
- To detect the start of a new day, use GET DAY and put the result into a variable called DAY OF MONTH. Do this once in the POWERUP chart and then continually in another chart. In this other chart, move DAY OF MONTH to DAY OF MONTH(LAST) just before executing GET DAY, then compare DAY OF MONTH with DAY OF MONTH(LAST) using NOT EQUAL? When they are not equal, midnight has just occurred.

See Also: GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

GET DAY OF WEEK — Time/Date

Function: To read the number of the day of the week (0 through 6) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in a Cyrano program based on the day of the week.

Details:
- The destination variable can be an integer or a float, although an integer is preferred.
- Days are numbered as follows:

Sunday = 0	Tuesday = 2	Thursday = 4	Saturday = 6
Monday = 1	Wednesday = 3	Friday = 5	

- If the current day is a Wednesday, this operation would place the value 3 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **GET DAY OF WEEK**

Move To	DAY OF WEEK	*variable integer*

Notes:
- This is a one-time read of the day of the week. If the date changes, you will need to execute this command again to get the current day of the week.
- It is advisable to use this operation once in the POWERUP chart and once after midnight rollover thereafter. See Notes for GET DAY.

See Also: GET DAY, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

GET HOURS — Time/Date

Function: To read the hour (0 through 23) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in a Cyrano program based on the hour of the day or to log an event.

Details:

- The destination variable can be an integer or a float, although an integer is preferred.
- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the current time is 2:35 p.m. (14:35:00), this operation would place the value 14 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET HOURS**

Move To	HOURS	*variable integer*

Notes:

- This is a one-time read of the hour. If the hour changes, you will need to execute this command again to get the current hour.
- Put this command in a small program loop that executes frequently to ensure that the variable always contains the current hour.

See Also: GET DAY, GET DAY OF WEEK, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

GET MINUTES — Time/Date

Function: To read the minute (0 through 59) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in a Cyrano program based on minutes past the hour or to log an event.

Details:
- The destination variable can be an integer or a float, although an integer is preferred.
- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the current time is 2:35 p.m. (14:35:00), this operation would place the value 35 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET MINUTES**

Move To	MINUTES	*variable integer*

Notes:
- This is a one-time read of the minutes. If the minute changes, you will need to execute this command again to get the current minute value.
- Put this command in a small program loop that executes frequently to ensure that the variable always contains the current minute value.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

GET MONTH — Time/Date

Function: To read the month value (1 through 12) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To determine when to begin and end Daylight Savings Time.

Details:
- The destination variable can be an integer or a float, although an integer is preferred.
- If the current date is March 2, 1995, this operation would place the value 3 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **GET MONTH**

Move To	MONTH	*variable integer*

Notes:
- This is a one-time read of the month. If the month changes, you will need to execute this command again to get the value of the current month.
- Put this command in a small program loop that executes frequently to ensure that the variable always contains the current month value.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

GET SECONDS — Time/Date

Function: To read the second (0 through 59) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To use seconds information in a Cyrano program.

Details:
- The destination variable can be an integer or a float, although an integer is preferred.
- If the current time is 08:51:26, this operation would place the value 26 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET SECONDS**

Move To	SECONDS	*variable integer*

Notes:
- This is a one-time read of the seconds. If the second changes, you will need to execute this command again to get the value of the current second.
- Put this command in a small program loop that executes frequently to ensure that the variable always contains the current seconds value.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

GET YEAR — Time/Date

Function: To read the year value (00 through 99) from the Mistic controller's real-time clock/calendar and put it into a numeric variable.

Typical Use: To use year information in a Cyrano program.

Details:
- The destination variable can be an integer or a float, although an integer is preferred.
- If the current date is March 2, 1995, this operation would place the value 95 into the *Move To* parameter (Argument 1).

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **GET YEAR**

Move To	YEAR	*variable integer*

Notes:
- This is a one-time read of the year. If the year changes, you will need to execute this command again to get the value of the current year.
- Put this command in a small program loop that executes frequently to ensure that the variable always contains the current year value.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MONTH, GET MINUTES, GET SECONDS, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

SET DATE — Time/Date

Function: To set the date in the Mistic controller's real-time clock/calendar to the value contained in a string variable, using the standard United States format mm/dd/yy, where mm = month (01–12), dd = day (01–31), and yy = year (00–99).

Typical Use: To set the date from a Cyrano program.

Details:
- The destination can be a variable string or a constant string.
- If the desired date to set is March 1, 1995, the *From* parameter (Argument 1) should contain the string "03/01/95."
- Executing this command would set the Mistic controller's real-time clock/calendar to March 1, 1995.
- Updates day of week also.
- All erroneous date strings are ignored.

Arguments:

ARGUMENT 1
CONSTANT STRING
VARIABLE STRING

Example: **SET DATE**

From	US DATE STRING	*variable string*

Notes:
- The Cyrano Debugger always sets the date, time, and day of week to the PC clock at the end of a download.
- To change the date, use a variable integer as a change trigger. Set the trigger variable True after the date string has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- Do not issue this command continuously.

See Also: COPY DATE TO STRING (EUR), COPY DATE TO STRING (US), COPY TIME TO STRING

SET DAY — Time/Date

Function: To set the day of the month (1 through 31) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the day of the month from a Cyrano program.

Details:

- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- If the desired day of the month to set is March 2, 1995, the *To* parameter (Argument 1) should contain the value 2.
- Executing this command would then set the day of the month in the Mistic controller's real-time clock/calendar.
- Updates day of week also.
- All erroneous day values are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET DAY**

To	DAY OF MONTH	*variable integer*

Notes:

- Use to change the DAY to test program logic. Use a variable integer as a change trigger. Set the trigger variable True after the DAY OF MONTH variable has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

SET DAY OF WEEK — Time/Date

Function: To set the day of the week value (0 through 6) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the day of the week from a Cyrano program.

Details:
- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- Days are numbered as follows:

Sunday = 0	Tuesday = 2	Thursday = 4	Saturday = 6
Monday = 1	Wednesday = 3	Friday = 5	

- If the desired day of week to set is Wednesday, then the *To* parameter (Argument 1) should contain the value 3.
- Executing this command would set the day of the week in the Mistic controller's real-time clock/calendar.
- All erroneous day of week values are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET DAY OF WEEK**

To	DAY OF WEEK	*variable integer*

Notes:
- Use to change the day of the week to test program logic. Use a variable integer as a change trigger. Set the trigger variable True after the *To* parameter (DAY OF WEEK, in the example above) has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

SET HOURS — Time/Date

Function: To set the hours value (0 through 23) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the hours value from a Cyrano program.

Details:
- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the desired hour to set is 2 p.m. (14:00:00), the *To* parameter (Argument 1) should contain the value 14.
- Executing this command would set the hours value in the Mistic controller's real-time clock/calendar.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous hour values are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET HOURS**

To	HOURS	*variable integer*

Notes:
- Use to change the HOUR to test program logic. Use a variable integer as a change trigger. Set the trigger variable True after the HOURS variable has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

SET MINUTES — Time/Date

Function: To set the minutes (0 through 59) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the minutes value from a Cyrano program.

Details:
- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the desired time to set is 2:35 p.m. (14:35:00), the *To* parameter (Argument 1) should contain the value 35.
- Executing this command would set the minutes value in the Mistic controller's real-time clock/calendar.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous values for minutes are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET MINUTES**

To	MINUTES	*variable integer*

Notes:
- Use to change the MINUTES to test program logic. Use a variable integer as a change trigger. Set the trigger variable True after the MINUTES variable has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET SECONDS, SET YEAR

SET MONTH — Time/Date

Function: To set the month value (1 through 12) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the month from a Cyrano program.

Details:
- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- If the desired month to set is March, the *To* parameter (Argument 1) should contain the value 3.
- Executing this command would set the month in the Mistic controller's real-time clock/calendar.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous month values are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET MONTH**

To	MONTH	*variable integer*

Notes:
- Use to change the MONTH to test program logic. Use a variable integer as a change trigger. Set the trigger variable True after the MONTH variable has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET SECONDS, SET YEAR

SET SECONDS — Time/Date

Function: To set the seconds (0 through 59) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the seconds value from a Cyrano program.

Details:
- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the desired time to set is 2:35:26 p.m., then the *To* parameter (Argument 1) should contain the value 26.
- Executing this command would set the seconds value in the Mistic controller's real-time clock/calendar.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous values for seconds are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET SECONDS**

To	SECONDS	*variable integer*

Notes:
- Use to change the SECONDS to test program logic. Use a variable integer as a change trigger. Set the trigger variable True after the SECONDS variable has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET MONTH, SET YEAR

SET TIME — Time/Date

Function: To set the time in the Mistic controller's real-time clock/calendar from a string variable.

Typical Use: To set the time from a Cyrano program.

Details:
- The *From* parameter (Argument 1) can be a constant or variable string, although a variable string is preferred.
- Time is in 24-hour format. For example, 8 a.m. = 08:00:00, 1 p.m. = 13:00:00, and 11:59:00 p.m. = 23:59:00.
- If the desired time to set is 2:35:00 p.m., the *From* parameter (Argument 1) should contain the string "14:35:00."
- Executing this command would set the time value in the Mistic controller's real-time clock/calendar.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous time strings are ignored.

Arguments:

ARGUMENT 1
CONSTANT STRING
VARIABLE STRING

Example: **SET TIME**

From	TIME STRING	*variable string*

Notes:
- The Cyrano Debugger always sets the date, time, and day of week to the PC clock at the end of a download.
- To change the time, use a variable integer as a change trigger. Set the trigger variable True after the time string has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- Do not issue this command continuously.

See Also: COPY DATE TO STRING (EUR), COPY DATE TO STRING (US), COPY TIME TO STRING, SET DATE.

SET YEAR — Time/Date

Function: To set the year (00 through 99) in the Mistic controller's real-time clock/calendar.

Typical Use: To set the year from a Cyrano program.

Details:
- The *To* parameter (Argument 1) can be an integer or a float, although an integer is preferred.
- If the desired year to set is 1995, the *To* parameter (Argument 1) should contain the value 95.
- Executing this command would set the year (00 through 99) in the Mistic controller's real-time clock/calendar.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous month values are ignored.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
VARIABLE FLOAT
VARIABLE INTEGER

Example: **SET YEAR**

To	YEAR	*variable integer*

Notes:
- The Cyrano Debugger always sets the date, time, and day of week to the PC clock at the end of a download.
- To change the year, use a variable integer as a change trigger. Set the trigger variable True after the year variable has the desired value. When the trigger is True, the program executes this command, then sets the trigger variable False.
- The Mistic controller's real-time clock/calendar will automatically increment the time and date after they are set.
- Do not issue this command continuously.

See Also: GET DAY, GET DAY OF WEEK, GET HOURS, GET MINUTES, GET MONTH, GET SECONDS, GET YEAR, SET DAY, SET DAY OF WEEK, SET HOURS, SET MINUTES, SET SECONDS, SET YEAR

CONDITIONS

CHAPTER 3

OVERVIEW

This appendix provides reference data on all Cyrano condition commands.

To locate a command, look it up in the index below or browse through the appropriate command group (Chart, Digital Point, etc.) in this chapter.

INDEX OF CONDITION COMMAND GROUPS

INDEX OF CONDITION COMMANDS

CHART CONDITIONS

CALLING CHART RUNNING? Chart

Function: To check if the calling chart (the one that started this chart) is in the running state.

Typical Use: To determine the status of the chart that started this chart.

Details:
- Evaluates True if the calling chart is running, False if not.

Arguments: None.

Example: **CALLING CHART RUNNING?**

Notes:
- See the Chart Overview in Chapter 1 for important information.

See Also: CONTINUE CALLING CHART, CALLING CHART SUSPENDED?, CALLING CHART STOPPED?

CALLING CHART STOPPED? Chart

Function: To check if the calling chart (the one that started this chart) is in the stopped state.

Typical Use: To determine the status of the chart that started this chart.

Details:
- Evaluates True if the calling chart is stopped, False if not.

Arguments: None.

Example: **CALLING CHART STOPPED?**

Notes:
- See the Chart Overview in Chapter 1 for important information.

See Also: CONTINUE CALLING CHART, CALLING CHART SUSPENDED?, CALLING CHART RUNNING?

CALLING CHART SUSPENDED? Chart

Function: To check if the calling chart (the one that started this chart) is in the suspended state.

Typical Use: Called before CONTINUE CALLING CHART to ensure its success.

Details:
- Evaluates True if the calling chart is suspended, False if not.

Arguments: None.

Example: **CALLING CHART SUSPENDED?**

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Always use before CONTINUE CALLING CHART to ensure its success. See the CONTINUE CALLING CHART operation for details.

See Also: CONTINUE CALLING CHART, CALLING CHART STOPPED?, CALLING CHART RUNNING?

CHART RUNNING? Chart

Function: To check if the specified chart is in the running state.

Typical Use: To determine the status of the specified chart.

Details:
- Evaluates True if the specified chart is running, False if not.

Arguments:

ARGUMENT 1
CHART

Example:

Is	CHART_B	*chart name*
CHART RUNNING?		

Notes:
- See the Chart Overview in Chapter 1 for important information.

See Also: CHART SUSPENDED?, CHART STOPPED?

CHART STOPPED? Chart

Function: To check if the specified chart is in the stopped state.

Typical Use: Used before START CHART to ensure its success when it is imperative that START CHART succeed.

Details:
- Evaluates True if the specified chart is stopped, False if not.

Arguments:

ARGUMENT 1
CHART

CHART STOPPED?

Example: *Is* CHART_B *chart name*
CHART STOPPED?

Notes:
- See the Chart Overview in Chapter 1 for important information.
- When a chart calls a START CHART followed immediately by a SUSPEND CHART to suspend itself, it depends on the target chart to continue it later. Hence, it is imperative that the target chart be started, otherwise the original (calling) chart will remain suspended. This condition can determine if the target chart has started.

See Also: CHART SUSPENDED?, CHART RUNNING?

CHART SUSPENDED? Chart

Function: To check if the specified chart is in the suspended state.

Typical Use: To determine the status of the specified chart.

Details:
- Evaluates True if the specified chart is suspended, False if not.

Arguments:

ARGUMENT 1
CHART

Example:

Is	CHART_B	*chart name*
CHART SUSPENDED?		

Notes:
- See the Chart Overview in Chapter 1 for important information.
- Use before CONTINUE CHART to ensure success.

See Also: CHART

DIGITAL POINT CONDITIONS

CLOSED? — Digital Point

Function: To determine if a digital input or output is on.

Typical Use: To determine the status of a digital input or output channel.

Details:
- Evaluates True if the specified channel is on, False if the channel is off.

Arguments:

ARGUMENT 1
DIGITAL IN
DIGITAL OUT

Example:

	START SWITCH	*any digital input or output channel*
CLOSED?		

Notes:
- May be used with either input or output channels.
- This condition is identical to the ON? condition.
- *Speed Tip:* Use DO BINARY READ to get the state of all 16 channels at once. Then use BIT TEST to determine the state of individual channels.

Dependencies:
- Applies to all inputs and outputs on digital multifunction I/O units and local simple I/O units.

See Also: OPEN?, ON?, OFF?

LATCH SET? Digital Point

Function: To determine if a digital input on-latch or off-latch is set.

Typical Use: To see if a momentary button was pressed.

Details:
- Evaluates True if the specified on-latch or off-latch is set, False if not.

Arguments:

ARGUMENT 1
OFF LATCH
OFF TIME TOTALIZER
ON LATCH
ON TIME TOTALIZER
QUADRATURE COUNTER

Example: E STOP *digital input configured with an on- or off-latch feature*

LATCH SET?

Notes:
- Don't confuse the latch status with the on or off status of a channel. An on-latch may be set even though the channel is currently off.

Dependencies:
- Applies only to inputs configured with the on- or off-latch feature on digital multifunction I/O units.

See Also: ON?, OFF?

OFF? — Digital Point

Function: To determine if a digital input or output is off.

Typical Use: To determine the status of a digital input or output channel.

Details:
- Evaluates True if the specified channel is off, False if the channel is on.
- *Speed Tip:* Use DO BINARY READ to get the state of all 16 channels at once. Then use BIT TEST to determine the state of individual channels.

Arguments:

ARGUMENT 1
DIGITAL IN
DIGITAL OUT

Example: SAFETY INTERLOCK *digital input or output channel*
OFF?

Notes:
- May be used with either input or output channels.
- This condition is identical to the OPEN? condition.

Dependencies:
- Applies to all inputs and outputs on digital multifunction I/O units and local simple I/O units.

See Also: CLOSED?, OPEN?, ON?

ON? Digital Point

Function: To determine if a digital input or output is on.

Typical Use: To determine the status of a digital input or output channel.

Details:
- Evaluates True if the specified channel is on, False if the channel is off.

Arguments:

ARGUMENT 1
DIGITAL IN
DIGITAL OUT

Example:

	MOTOR POWER	*digital input or output channel*
ON?		

Notes:
- May be used with either input or output channels.
- This condition is identical to the CLOSED? condition.
- *Speed Tip:* Use DO BINARY READ to get the state of all 16 channels at once. Then use BIT TEST to determine the state of individual channels.

Dependencies:
- Applies to all inputs and outputs on digital multifunction I/O units and local simple I/O units.

See Also: OPEN?, CLOSED?, OFF?

OPEN? — Digital Point

Function: To determine if a digital input or output is off.

Typical Use: To determine the status of a digital input or output channel.

Details:
- Evaluates True if the specified channel is off, False if the channel is on.

Arguments:

ARGUMENT 1
DIGITAL IN
DIGITAL OUT

Example:

	BRAKE RELEASE	*digital input or output channel*
OPEN?		

Notes:
- May be used with either input or output channels.
- This condition is identical to the OFF? condition.
- *Speed Tip:* Use DO BINARY READ to get the state of all 16 channels at once. Then use BIT TEST to determine the state of individual channels.

Dependencies:
- Applies to all inputs and outputs on digital multifunction I/O units and local simple I/O units.

See Also: CLOSED?, ON?, OFF?

EVENT/REACTION CONDITIONS

EVENT SCANNING DISABLED? — Event/Reaction

Function: To determine if a specific event/reaction is active or not.

Typical Use: To verify the active/inactive state of a specific event/reaction.

Details:
- Evaluates True if the specified event/reaction is not being scanned, False if it is being scanned.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **EVENT SCANNING DISABLED?**

Event/Reaction SEQUENCE FINISHED *name of the event/reaction*

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.

See Also: EVENT SCANNING ENABLED?

EVENT SCANNING ENABLED? — Event/Reaction

Function: To determine if a specific event/reaction is active or not.

Typical Use: To verify the active/inactive state of a specific event/reaction.

Details:
- Evaluates True if the specified event/reaction is being scanned, False if it's not being scanned.

Arguments:

ARGUMENT 1
ANALOG E/R DIGITAL E/R

Example: **EVENT SCANNING ENABLED?**

Event/Reaction	SEQUENCE FINISHED	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: EVENT SCANNING DISABLED?

GENERATING INTERRUPT? — Event/Reaction

Function: To determine if a specific I/O unit is generating an interrupt.

Typical Use: In the INTERRUPT chart, to determine which I/O unit is generating an interrupt when more than one is configured to do so.

Details:
- Evaluates True if the specified I/O unit is generating an interrupt, False if it's not.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT DIGITAL MF I/O UNIT

Example:

	OVERTEMP SENSORS	*name of I/O unit*
GENERATING INTERRUPT?		

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- Use CLEAR I/O UNIT INTERRUPT immediately after determining the interrupt is on. Then use HAS EVENT OCCURRED? for each event/reaction configured to interrupt.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: HAS EVENT OCCURRED?, CLEAR I/O UNIT INTERRUPT

HAS EVENT OCCURRED? — Event/Reaction

Function: To determine if a specific event has occurred.

Typical Use: To determine which event caused an interrupt.

Details:
- Evaluates True if the specified event/reaction has occurred, False if it has not.
- When the event occurs, its event latch is set. It will remain set until cleared with CLEAR EVENT LATCH.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **HAS EVENT OCCURRED?**

SEQUENCE FINISHED	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- The current state of the event is not relevant to this condition. See IS EVENT OCCURRING?
- Always use CLEAR EVENT LATCH after the event has occurred. This allows detection of subsequent events.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: IS EVENT OCCURRING?, CLEAR EVENT LATCH, CLEAR I/O UNIT INTERRUPT, GENERATING INTERRUPT?

IS EVENT OCCURRING? Event/Reaction

Function: To determine if the criteria for a specific event is currently true.

Typical Use: To determine if a specific situation still exists.

Details:
- Evaluates True if the criteria for the specified event are still true, False if the criteria are no longer true.

Arguments:

ARGUMENT 1
ANALOG E/R DIGITAL E/R

Example: **IS EVENT OCCURRING?**

SEQUENCE FINISHED	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.
- This is an easy way to test for an I/O state pattern.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: HAS EVENT OCCURRED?

INTERRUPT DISABLED FOR EVENT? — Event/Reaction

Function: To determine if the interrupt for a specific event/reaction is inactive.

Typical Use: To verify the active/inactive state of the interrupt for a specific event/reaction.

Details:
- Evaluates True if the interrupt for the specified event/reaction is not active, False if it is active.
- Event/reactions still occur when the interrupt is disabled as long as they are active.

Arguments:

ARGUMENT 1
ANALOG E/R
DIGITAL E/R

Example: **INTERRUPT DISABLED FOR EVENT?**

SEQUENCE FINISHED	*name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: ENABLE INTERRUPT ON EVENT, INTERRUPT ENABLED FOR EVENT?

INTERRUPT ENABLED FOR EVENT? Event/Reaction

Function: To determine if the interrupt for a specific event/reaction is active.

Typical Use: To verify the active/inactive state of the interrupt for a specific event/reaction.

Details:
- Evaluates True if the interrupt for the specified event/reaction is active, False if it is not active.
- Event/reactions still occur when the interrupt is disabled as long as they are active.

Arguments:

ARGUMENT 1
ANALOG E/R DIGITAL E/R

Example: **INTERRUPT ENABLED FOR EVENT?**

SEQUENCE FINISHED *name of the event/reaction*

Notes:
- See the Event/Reaction Overview in Chapter 1 for important information.

Dependencies:
- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on local simple I/O units.

See Also: ENABLE INTERRUPT ON EVENT, INTERRUPT DISABLED FOR EVENT?

GENERAL PURPOSE CONDITIONS

CAUSED A CHART ERROR? General Purpose

Function: To determine if the specified chart caused the current error in the error queue.

Typical Use: To determine which chart caused the current error.

Details:
- Evaluates True if the specified chart caused the error, False otherwise.
- The current error is the oldest one and is always at the top of the error queue.

Arguments:

ARGUMENT 1
CHART

Example: **CAUSED A CHART ERROR?**

Has	POWERUP	*chart name*

Notes:
- Use the Debugger to view the error queue for detailed information.

Dependencies:
- Prior to using this call, you should ensure that the error of interest is pointed to by using the POINT TO NEXT ERROR command.

See Also: GET ERROR CODE, POINT TO NEXT ERROR

CAUSED AN I/O UNIT ERROR? General Purpose

Function: To determine if the specified I/O unit caused the top error in the error queue.

Typical Use: To determine which I/O unit caused an error.

Details:
- Evaluates True if the specified I/O unit caused the error, False otherwise.
- Must use ERROR ON I/O UNIT? before using this command, since this command assumes the top error is an I/O error.

Arguments:

ARGUMENT 1
ANALOG MF I/O UNIT
DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT

Example: **CAUSED AN I/O UNIT ERROR?**

Has	DIG BRICK 1	*I/O unit name*

Notes:
- Be sure the top error in the queue is an I/O error.
- Use the Debugger to view the error queue for detailed information.

Dependencies:
- Must use ERROR ON I/O UNIT? before using this command.

See Also: ERROR ON I/O UNIT?, GET ERROR CODE, POINT TO NEXT ERROR

CHARACTERS WAITING (PORT)? General Purpose

Function: To determine if there are characters in the receive buffer of an open communication port.

Typical Use: To communicate with other Mistic controllers and other serial devices.

Details:
- Evaluates True if there is at least one character in the receive buffer, False otherwise.

Arguments: None.

Example: **CHARACTERS WAITING (PORT)?**

Notes:
- See the Communication Overview in Chapter 1 for important information.
- Must use before commands such as GET CHAR (PORT) and GET STRING (PORT), otherwise these commands will wait indefinitely.

Dependencies:
- Must use REQUEST PORT first to open the port.

See Also: # OF CHARACTERS WAITING (PORT), # OF CHARACTERS WAITING FROM PORT, CHARACTERS WAITING?

CHARACTERS WAITING? General Purpose

Function: To determine if there are characters in the receive buffer of a closed communication port.

Typical Use: To communicate with other Mistic controllers and other serial devices.

Details:
- Evaluates True if there is at least one character in the receive buffer, False otherwise.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **CHARACTERS WAITING?**

Port	1	*constant integer (port # to use)*

Notes:
- See the Communication Overview in Chapter 1 for important information.

Error Codes: -40 = Timeout — specified port already in use

See Also: # OF CHARACTERS WAITING (PORT), # OF CHARACTERS WAITING FROM PORT, CHARACTERS WAITING (PORT)?

ERROR ON I/O UNIT? General Purpose

Function: To determine if the top error in the error queue is an I/O-related error.

Typical Use: To determine if further error handling for I/O units should be performed.

Details:
- Evaluates True if the current error in the error queue is an I/O unit error, False otherwise.
- Queue errors 2 through 29 are considered I/O unit errors, with 29 being the most common.

Arguments: None.

Example: **ERROR ON I/O UNIT?**

Notes:
- Use CAUSED AN I/O UNIT ERROR? to determine which I/O unit caused the error.

Error Codes: Use the Debugger to view the error queue for detailed information.

See Also: CAUSED AN I/O UNIT ERROR?, POINT TO NEXT ERROR

ERROR? General Purpose

Function: To determine if there is an error in the error queue.

Typical Use: To determine if further error handling should be performed.

Details:
- Evaluates True if there is an error in the error queue, False otherwise.

Arguments: None.

Example: **ERROR?**

Notes:
- Use ERROR ON I/O UNIT? to determine if it is an I/O related error.
- Use the Debugger to view the error queue for detailed information.

See Also: ERROR ON I/O UNIT?

IS ARCNET CONNECTED? General Purpose

Function: To determine if the Mistic controller is connected to an active ARCNET link.

Typical Use: To detect a failure of the ARCNET link so that a backup communication path can be enabled.

Details:
- Evaluates True if there is at least one other active ARCNET device on the link, False otherwise.
- This "active" ARCNET device can be another Mistic controller or a PC, etc.

Arguments: None.

Example: **IS ARCNET CONNECTED?**

Notes:
- See the Communication Overview in Chapter 1 for important information.

Dependencies:
- This command does not work with LC32 controllers that do not have Flash memory.

See Also: RECEIVED MESSAGE FROM HOST?, IS ARCNET NODE PRESENT?

IS ARCNET MSG ADDR EQUAL TO? General Purpose

Function: To determine if the message received in the ARCNET port originated from a specified address.

Typical Use: To determine the source of the last ARCNET message received.

Details:
- Evaluates True if the addresses match, False otherwise.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **IS ARCNET MSG ADDR EQUAL TO?**

Address	3	*constant integer (node address)*

Notes:
- See the Communication Overview in Chapter 1 for important information.

See Also: IS ARCNET NODE PRESENT?

IS ARCNET NODE PRESENT? General Purpose

Function: To determine if a specific node on the ARCNET is present.

Typical Use: To determine if a specific node on the ARCNET link has gone off line.

Details:
- Evaluates True if the specified node responds, False otherwise.
- The ARCNET chip set cannot directly detect the presence of the next logical node on the network. The next logical node is defined as the first address found on the link either immediately before or after the Mistic controller's address. Knowledge of the addresses of each device on the network can be used with this function to determine if the next logical node is present.
- If there are Mistic controllers at addresses 1 and 2, and if there is a PC at address 3, then the controller at address 1 can determine if the ARCNET card in the PC at 3 is responding. If it is, this implies that the node at address 2 must exist also.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **IS ARCNET NODE PRESENT?**

Node Number	247	*constant integer (node address)*

Notes:
- See the Communication Overview in Chapter 1 for important information.

Dependencies:
- This command does not work with LC32 controllers that do not have Flash memory.

See Also: IS ARCNET CONNECTED?, IS ARCNET MSG ADDR EQUAL TO?

LOW BATTERY? General Purpose

Function: To determine if the battery backing up the static RAM on the controller is weak.

Typical Use: To determine if the battery needs to be replaced.

Details:
- Evaluates True if the voltage for the battery backing up static RAM is low, False otherwise.

Arguments: None.

Example: **LOW BATTERY?**

Notes:
- On the LC32, if the keypad (port 5) is in use by a chart, this condition will return False.

Error Codes:

Queue error 39	=	Port already in use — LC32 keypad (port 5) is in use by another chart
Queue error 29	=	Timeout — LC32 keypad (port 5) does not respond

See Also: GET RTU VOLTAGE

RECEIVED MESSAGE FROM HOST? General Purpose

Function: To determine if a message has been received on the specified HOST port.

Typical Use: To determine if an MMI has stopped communicating to the Mistic controller.

Details:
- Evaluates True if a message has been received on the specified HOST port since the last use of this command, False otherwise.

Arguments:

ARGUMENT 1
CONSTANT INTEGER
VARIABLE INTEGER

Example: **RECEIVED MESSAGE FROM HOST?**

Host Port #	4	*constant integer (the HOST port number)*

Notes:
- See the Communication Overview in Chapter 1 for important information.

Error Codes: Queue error 30 = Incorrect port number — use 0 to 4

See Also: IS ARCNET NODE PRESENT?, IS ARCNET MSG ADDR EQUAL TO?

TIMER EXPIRED? General Purpose

Function: To determine if the specified timer has counted down to zero.

Typical Use: To determine if it is time to take an appropriate action.

Details: • Evaluates True if the specified timer has reached zero, False otherwise.

Arguments:

ARGUMENT 1
VARIABLE TIMER

Example: **TIMER EXPIRED?**

Is	EGG TIMER	*variable timer (the timer variable)*

Notes: • Although the timer resolution is 1 millisecond, the accuracy of a time period is limited by the number of charts running concurrently as well as by the charts' priority.

See Also: MOVE

VARIABLE FALSE? — General Purpose

Function: To determine if the specified variable is zero.

Typical Use: To determine if further processing should take place.

Details:
- Evaluates True if the specified variable has a value of zero, False otherwise.

Arguments:

ARGUMENT 1
VARIABLE FLOAT VARIABLE INTEGER

Example: **VARIABLE FALSE?**

Is	PRESSURE DIFFERENCE	*variable integer*

See Also: VARIABLE TRUE?

VARIABLE TRUE? General Purpose

Function: To determine if the specified variable is non-zero.

Typical Use: To determine if further processing should take place.

Details:
- Evaluates True if the specified variable has a non-zero value, False otherwise.

Arguments:

ARGUMENT 1
VARIABLE FLOAT
VARIABLE INTEGER

Example: **VARIABLE TRUE?**

Is PRESSURE DIFFERENCE *variable integer*

See Also: VARIABLE FALSE?

\ COMMENT General Purpose

Function: To add a comment to a condition block.

Typical Use: To document commands within a condition block.

Details: • Comments are string constants. They use controller memory.

Arguments:

ARGUMENT 1
CONSTANT STRING

Example: **\ COMMENT**
Check for PID Loop Enabled *constant string*

Notes: • Use text outside a block for comments to conserve memory.

See Also: \\ COMMENT

\\ COMMENT — General Purpose

Function: To disable one or more conditions in a condition block.

Typical Use: To temporarily disable conditions within a condition block during debugging.

Details:

- This command is normally used in pairs. Everything between the pair of \\ COMMENT commands is considered a comment and is ignored when the strategy is compiled and downloaded. This is useful for temporarily disabling a group of conditions within a condition block while debugging a program.
- If the second \\ COMMENT is omitted, everything from the first \\ COMMENT to the end of the condition block is considered a comment.

Arguments:

ARGUMENT 1
CONSTANT STRING

Example:

\\ COMMENT
Condition
Condition
Condition
\\ COMMENT

See Also: \ COMMENT

LOGICAL CONDITIONS

AND — Logical

Function: To perform a logical AND on any two allowable values.

Typical Use: Used in place of calling VARIABLE TRUE? twice.

Details:
- Performs a logical AND on Arguments 1 and 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
-1	0	False
0	-1	False
-1	-1	True

- Evaluates True if both values are non-zero, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **AND**

Is	LIMIT SWITCH1	*digital input*
Is	LIMIT SWITCH2	*digital input*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- Multiple values can be ANDed by repeating this condition or the VARIABLE TRUE? condition several times in the same block.
- Use BIT AND if the objective is to test for individual bits.
- Executes faster than using VARIABLE TRUE? twice.

See Also: BIT AND, VARIABLE TRUE?, VARIABLE FALSE?

BIT AND **Logical**

Function: To perform a 32-bit bitwise AND on any two allowable values.

Typical Use: To determine if the individual bits of one value match the on bits of a mask value.

Details:

- Performs a bitwise AND on Arguments 1 and 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
1	0	False
0	1	False
1	1	True

- Evaluates True if any bit set to 1 in the mask (Argument 2) is also set to 1 in Argument 1. Evaluates False if all of the mask's 1 bits are set to 0 in Argument 1.
- Acts on all 32 bits.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
DIGITAL MF I/O UNIT	DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT	DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT	REM SMPL I/O UNIT
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: This example reads the current state of all channels on a digital I/O unit and BIT ANDs the value with the constant 33,280 (1000 0010 0000 0000 binary). Evaluates True if either channel 15 or 9 is on, False if both channels are off.

BIT AND

Is	BRICK 1	*digital I/O unit*
	33280	*constant integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital I/O units with this command.
- Use 255 as the constant to check the lower eight channels.

See Also: AND, BIT OR

BIT NOT? Logical

Function: To invert all 32 bits of an allowable value and determine if the result is True or False.

Typical Use: To determine if any bit is off.

Details:
- Inverts Argument 1 and evaluates whether the result is True or False. Examples:

ARGUMENT 1	RESULT
0	True
1	False

- Evaluates True if any bit is set to 0, False otherwise.
- Acts on all 32 bits.

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT
VARIABLE FLOAT
VARIABLE INTEGER

Example: This example reads the state of all channels of the specified digital I/O unit and then inverts them. Evaluates True if any channel is off, False otherwise.

BIT NOT?

Is	BRICK1	*digital I/O unit*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital I/O units with this command.
- Use NOT if the objective is to toggle the value between True and False.

See Also: BIT ON?, BIT OFF?

BIT OFF? Logical

Function: To test the False status of a specific bit in an allowable value.

Typical Use: To test a bit used as a flag in an integer variable.

Details:
- Evaluates True if the bit in Argument 1 specified by Argument 2 is set to 0. Evaluates False if the bit is set to 1.
- Valid range for the *Bit to Test* parameter (Argument 2) is 0–31.

Arguments:

ARGUMENT 1	ARGUMENT 2
DIGITAL MF I/O UNIT	CONSTANT INTEGER
DIGITAL NMF I/O UNIT	VARIABLE INTEGER
REM SMPL I/O UNIT	
VARIABLE INTEGER	

Example: This example evaluates to True if channel 15 of I/O UNIT 1 is off, False otherwise.

BIT OFF?

Data Source	I/O UNIT 1	*digital I/O unit*
Bit to Test	15	*constant integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital I/O units with this command. Although this condition can be used to determine the status of digital points, it is primarily used to test bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time or latch), and needs acknowledgment.
- Use BIT AND if the objective is to test several bits at once.

See Also: BIT ON?, BIT AND, BIT TEST (operation)

BIT ON? — Logical

Function: To test the True status of a specific bit in an allowable value.

Typical Use: To test a bit used as a flag in an integer variable.

Details:
- Evaluates True if the bit specified in Argument 2 is set to 1 in Argument 1. Evaluates False if the bit is set to 0.
- Valid range for the *Bit to Test* parameter (Argument 2) is 0–31.

Arguments:

ARGUMENT 1	ARGUMENT 2
DIGITAL MF I/O UNIT	CONSTANT INTEGER
DIGITAL NMF I/O UNIT	VARIABLE INTEGER
REM SMPL I/O UNIT	
VARIABLE INTEGER	

Example: This example evaluates to True if channel 0 of I/O UNIT 1 is on, False otherwise.

BIT ON?

Data Source	I/O UNIT 1	*digital I/O unit*
Bit to Test	0	*constant integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital I/O units with this command. Although this condition can be used to determine the status of digital points, it is primarily used to test bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time or latch), and needs acknowledgment.
- Use BIT AND if the objective is to test several bits at once.

See Also: BIT OFF?, BIT AND, BIT TEST (operation)

BIT OR — Logical

Function: To perform a 32-bit bitwise OR on any two allowable values.

Typical Use: To determine if any bit is set to 1 in either of two values.

Details:
- Performs a bitwise OR on Arguments 1 and 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
1	0	True
0	1	True
1	1	True

- Evaluates to True if any bit is set to 1 in either of the two allowable values, False otherwise.
- Acts on all 32 bits.
- Functionally equivalent to the OR condition.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
DIGITAL MF I/O UNIT	DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT	DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT	REM SMPL I/O UNIT
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **BIT OR**

Is	FAULT BITS 1	*variable integer*
	FAULT BITS 2	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital I/O units with this command. Although this condition can be used to determine the status of digital points, it is primarily used to test bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time or latch), and needs acknowledgment.
- Use BIT ON? or BIT OFF? if the objective is to test only one bit.

See Also: BIT ON?, BIT OFF?, OR

BIT XOR — Logical

Function: To determine the inequality of any two allowable values.

Typical Use: To detect a change of state of any bit in either of two values.

Details:
- Performs a bitwise XOR on Arguments 1 and 2. Examples:

BIT TEST			VALUE TEST		
ARGUMENT 1	ARGUMENT 2	RESULT	ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False	0	0	False
0	1	True	-1	0	True
1	0	True	255	65280	True
1	1	False	22	22	False

- Evaluates True if the two allowable values are not equal, False if they are equal.
- Acts on all 32 bits.
- Functionally equivalent to the NOT EQUAL condition when used with integer types.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
DIGITAL MF I/O UNIT	DIGITAL MF I/O UNIT
DIGITAL NMF I/O UNIT	DIGITAL NMF I/O UNIT
REM SMPL I/O UNIT	REM SMPL I/O UNIT
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **BIT XOR**

Is	BRICK 0	*digital I/O unit*
	PREV BRICK 0	*variable integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital I/O units with this command. Although this condition can be used to determine the status of digital points, it is primarily used to test bits in an integer variable. These bits can be used as flags to carry information such as status, control, fault (real-time or latch), and needs acknowledgment.
- Use the False exit if the objective is to test for an exact match, or use the EQUAL condition if using numeric values.

See Also: EQUAL, BIT AND, BIT NOT, BIT OR

EQUAL — Logical

Function: To determine the equality of two values.

Typical Use: To branch program logic based on the sequence number of the process.

Details:
- Determines if Argument 1 is equal to Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
-1	-1	True
-1	1	False
22.22	22.22	True
22.22	22.221	False

- Evaluates True if both values are the same, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG IN
ANALOG OUT	ANALOG OUT
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
COUNTER	COUNTER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
FREQUENCY	FREQUENCY
OFF LATCH	OFF LATCH
OFF PULSE MEAS.	OFF PULSE MEAS.
OFF TIME TOTALIZER	OFF TIME TOTALIZER
ON LATCH	ON LATCH
ON PULSE MEAS.	ON PULSE MEAS.
ON TIME TOTALIZER	ON TIME TOTALIZER
PERIOD	PERIOD
QUADRATURE COUNTER	QUADRATURE COUNTER
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE TIMER	VARIABLE TIMER

Example: **EQUAL**

Is	BATCH STEP	*variable integer*
To	4	*constant integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER OR EQUAL or LESS OR EQUAL when testing floats or analog values, since exact matches are rare.
- Use WITHIN LIMITS? to test for an approximate match.
- Use either NOT EQUAL or the False exit if the objective is to test for inequality.

See Also: GREATER, LESS, NOT EQUAL, GREATER OR EQUAL, LESS OR EQUAL, WITHIN LIMITS?

EQUAL TO FLOAT TABLE DATA — Logical

Function: To determine if a numeric value is exactly equal to the specified value in a float table.

Typical Use: To perform lookup table matching.

Details:

- Determines if one value (Argument 1) is equal to another (a value at index Argument 2 in float table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0.0	0.0	True
0.0001	0.0	False
-98.765	-98.765	True
22.22	22.22	True

- Evaluates True if both values are exactly the same, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **EQUAL TO FLOAT TABLE DATA**

Is	THIS READING	*variable float*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF READINGS	*float table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER OR EQ TO FLT TABLE DATA or LESS OR EQ TO FLT TABLE DATA when testing floats or analog values unless an exact match is required.
- Use either NOT EQUAL TO FLOAT TABLE DATA or the False exit if the objective is to test for inequality.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....FLOAT TABLE DATA" conditions

EQUAL TO INTEGER TABLE DATA — Logical

Function: To determine if a numeric value is exactly equal to the specified value in an integer table.

Typical Use: To perform lookup table matching.

Details:

- Determines if one value (Argument 1) is equal to another (a value at index Argument 2 in integer table Argument 3). Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	True
1	0	False
-32768	-32768	True
2222	2222	True

- Evaluates True if both values are exactly the same, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **EQUAL TO INTEGER TABLE DATA**

Is	THIS READING	*variable integer*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF COUNTS	*integer table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER OR EQ TO INT TABLE DATA or LESS OR EQ TO INT TABLE DATA when testing integer values unless an exact match is required.
- Use either NOT EQUAL TO INTEGER TABLE DATA or the False exit if the objective is to test for inequality.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....INTEGER TABLE DATA" conditions

GREATER — Logical

Function: To determine if one numeric value is greater than another.

Typical Use: To determine if a counter has reached an upper limit.

Details:
- Determines if Argument 1 is greater than Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
-1	0	False
-1	-3	True
22.221	22.220	True

- Evaluates True if Argument 1 is greater than Argument 2, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG IN
ANALOG OUT	ANALOG OUT
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
COUNTER	COUNTER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
FREQUENCY	FREQUENCY
OFF LATCH	OFF LATCH
OFF PULSE MEAS.	OFF PULSE MEAS.
OFF TIME TOTALIZER	OFF TIME TOTALIZER
ON LATCH	ON LATCH
ON PULSE MEAS.	ON PULSE MEAS.
ON TIME TOTALIZER	ON TIME TOTALIZER
PERIOD	PERIOD
QUADRATURE COUNTER	QUADRATURE COUNTER
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE TIMER	VARIABLE TIMER

Example: **GREATER**

Is	CALCULATED VALUE	*variable integer*
Than	1000	*constant integer*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- Use WITHIN LIMITS? to test for an approximate match.
- Use either LESS OR EQUAL or the False exit if the objective is to test for less than or equal.

See Also: LESS, NOT EQUAL, GREATER OR EQUAL, LESS OR EQUAL, WITHIN LIMITS?

GREATER OR EQ TO FLT TABLE DATA — Logical

Function: To determine if a numeric value is greater than or equal to a specified value in a float table.

Typical Use: To store peak values.

Details:

- Determines if one value (Argument 1) is greater than or equal to another (a value at index Argument 2 in float table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0.0	0.0	True
0.0001	0.0	True
-98.765	-98.765	True
22.22	22.222	False

- Evaluates True if the first value is greater than or equal to the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **GREATER OR EQ TO FLT TABLE DATA**

Is	THIS READING	*variable float*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF READINGS	*float table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either LESS THAN FLOAT TABLE DATA or the False exit if the objective is to test for less than.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....FLOAT TABLE DATA" conditions

GREATER OR EQ TO INT TABLE DATA — Logical

Function: To determine if a numeric value is greater than or equal to a specified value in an integer table.

Typical Use: To store peak values.

Details:

- Determines if one value (Argument 1) is greater than or equal to another (a value at index Argument 2 in integer table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0	0	True
1	0	True
-32768	-32767	False
22221	2222	True

- Evaluates True if the first value is greater than or equal to the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **GREATER OR EQ TO INT TABLE DATA**

Is	THIS READING	*variable integer*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF RANGES	*integer table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either LESS THAN INTEGER TABLE DATA or the False exit if the objective is to test for less than.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....INTEGER TABLE DATA" conditions

GREATER OR EQUAL — Logical

Function: To determine if one numeric value is greater than or equal to another.

Typical Use: To determine if a value has reached an upper limit.

Details:

- Determines if Argument 1 is greater than or equal to Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	True
1	0	True
-32768	-32767	False
22221	2222	True

- Evaluates True if the first value is greater than or equal to the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG IN
ANALOG OUT	ANALOG OUT
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
COUNTER	COUNTER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
FREQUENCY	FREQUENCY
OFF LATCH	OFF LATCH
OFF PULSE MEAS.	OFF PULSE MEAS.
OFF TIME TOTALIZER	OFF TIME TOTALIZER
ON LATCH	ON LATCH
ON PULSE MEAS.	ON PULSE MEAS.
ON TIME TOTALIZER	ON TIME TOTALIZER
PERIOD	PERIOD
QUADRATURE COUNTER	QUADRATURE COUNTER
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE TIMER	VARIABLE TIMER

Example: **GREATER OR EQUAL**

Is	ROOM TEMP	*analog input*
To	78.5000	*constant float*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use WITHIN LIMITS? to test for an approximate match.
- Use either the LESS condition or the False exit if the objective is to test for less than.
- When using analog values or digital features in this command, be sure to take into consideration the units that the value is read in and adjust the test values accordingly.

See Also: LESS, NOT EQUAL, LESS OR EQUAL, WITHIN LIMITS?

GREATER THAN FLOAT TABLE DATA — Logical

Function: To determine if a numeric value is greater than a specified value in a float table.

Typical Use: To store peak values.

Details:

- Determines if one value (Argument 1) is greater than another (a value at index Argument 2 in float table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0.0	0.0	False
0.0001	0.0	True
-98.765	-98.765	False
22.22	22.22	False

- Evaluates True if the first value is greater than the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **GREATER THAN FLOAT TABLE DATA**

Is	THIS READING	*variable float*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF READINGS	*float table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either LESS OR EQ TO FLT TABLE DATA or the False exit if the objective is to test for less than or equal to.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....FLOAT TABLE DATA" conditions

GREATER THAN INTEGER TABLE DATA — Logical

Function: To determine if a numeric value is greater than a specified value in an integer table.

Typical Use: To store peak values.

Details:

- Determines if one value (Argument 1) is greater than another (a value at index Argument 2 in integer table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0	0	False
1	0	True
-32768	-32767	False
22221	2222	True

- Evaluates True if the first value is greater than the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example:

GREATER THAN INTEGER TABLE DATA

Is	THIS READING	*variable integer*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF RANGES	*integer table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either LESS OR EQ TO INT TABLE DATA or the False exit if the objective is to test for less than or equal to.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....INTEGER TABLE DATA" conditions

LESS — Logical

Function: To determine if one numeric value is less than another.

Typical Use: To determine if a value is too low.

Details:
- Determines if Argument 1 is less than Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
-1	0	True
-1	-3	False
22.221	22.220	False

- Evaluates True if the first value is less than the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG IN
ANALOG OUT	ANALOG OUT
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
COUNTER	COUNTER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
FREQUENCY	FREQUENCY
OFF LATCH	OFF LATCH
OFF PULSE MEAS.	OFF PULSE MEAS.
OFF TIME TOTALIZER	OFF TIME TOTALIZER
ON LATCH	ON LATCH
ON PULSE MEAS.	ON PULSE MEAS.
ON TIME TOTALIZER	ON TIME TOTALIZER
PERIOD	PERIOD
QUADRATURE COUNTER	QUADRATURE COUNTER
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE TIMER	VARIABLE TIMER

Example: **LESS**

Is	TANK LEVEL	*analog input*
Than	FILL SETPOINT	*variable float*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- Use WITHIN LIMITS? to test for an approximate match.
- Use either GREATER OR EQUAL or the False exit if the objective is to test for greater than or equal to.

See Also: GREATER, NOT EQUAL, EQUAL, GREATER OR EQUAL

LESS OR EQUAL — Logical

Function: To determine if one numeric value is less than or equal to another.

Typical Use: To determine if a value is too low.

Details:

- Determines if Argument 1 is less than or equal to Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	True
-1	0	True
-1	-3	False
22.221	22.220	False

- Evaluates True if the first value is less than or equal to the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG IN
ANALOG OUT	ANALOG OUT
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
COUNTER	COUNTER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
FREQUENCY	FREQUENCY
OFF LATCH	OFF LATCH
OFF PULSE MEAS.	OFF PULSE MEAS.
OFF TIME TOTALIZER	OFF TIME TOTALIZER
ON LATCH	ON LATCH
ON PULSE MEAS.	ON PULSE MEAS.
ON TIME TOTALIZER	ON TIME TOTALIZER
PERIOD	PERIOD
QUADRATURE COUNTER	QUADRATURE COUNTER
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE TIMER	VARIABLE TIMER

Example: **LESS OR EQUAL**

Is	TEMPERATURE	*variable float*
To	98.60	*constant float*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use WITHIN LIMITS? to test for an approximate match.
- Use either the GREATER condition or the False exit if the objective is to test for greater than.

See Also: GREATER OR EQUAL, NOT EQUAL, GREATER

LESS OR EQ TO FLT TABLE DATA — Logical

Function: To determine if a numeric value is less than or equal to a specified value in a float table.

Typical Use: To store low values.

Details:
- Determines if one value (Argument 1) is less than or equal to another (a value at index Argument 2 in float table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0.0	0.0	True
0.0001	0.0	False
-98.765	-98.765	True
22.22	22.222	True

- Evaluates True if the first value is less than or equal to the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **LESS OR EQ TO FLT TABLE DATA**

Is	THIS READING	*variable float*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF READINGS	*float table*

Notes:
- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER THAN FLOAT TABLE DATA or the False exit if the objective is to test for greater than.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....FLOAT TABLE DATA" conditions

LESS OR EQ TO INT TABLE DATA — Logical

Function: To determine if a numeric value is less than or equal to a specified value in an integer table.

Typical Use: To store low values.

Details:

- Determines if one value (Argument 1) is less than or equal to another (a value at index Argument 2 in integer table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0	0	True
1	0	False
-32768	-32767	True
22221	2222	False

- Evaluates True if the first value is less than or equal to the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **LESS OR EQ TO INT TABLE DATA**

Is	THIS READING	*variable integer*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF RANGES	*integer table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER THAN INTEGER TABLE DATA or the False exit if the objective is to test for greater than.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....INTEGER TABLE DATA" conditions

LESS THAN FLOAT TABLE DATA — Logical

Function: To determine if a numeric value is less than a specified value in a float table.

Typical Use: To store low values.

Details:

- Determines if one value (Argument 1) is less than another (a value at index Argument 2 in float table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0.0	0.0	False
0.0001	0.0	False
-98.766	-98.765	True
22.22	22.22	False

- Evaluates True if the first value is less than the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **LESS THAN FLOAT TABLE DATA**

Is	THIS READING	*variable float*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF READINGS	*float table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER OR EQ TO FLT TABLE DATA or the False exit if the objective is to test for greater than or equal to.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....FLOAT TABLE DATA" conditions

LESS THAN INTEGER TABLE DATA — Logical

Function: To determine if a numeric value is less than a specified value in an integer table.

Typical Use: To store low values.

Details:

- Determines if one value (Argument 1) is less than another (a value at index Argument 2 in integer table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0	0	False
1	0	False
-32768	-32767	True
22221	2222	False

- Evaluates True if the first value is less than the second, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **LESS THAN INTEGER TABLE DATA**

Is	THIS READING	*variable integer*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF RANGES	*integer table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either GREATER OR EQ TO INT TABLE DATA or the False exit if the objective is to test for greater than or equal to.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....INTEGER TABLE DATA" conditions

NOT EQUAL — Logical

Function: To determine if two values are different.

Typical Use: To perform reverse logic.

Details:

- Determines if Argument 1 is different from Argument 2. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
-1	0	True
255	65280	True
22.22	22.22	False

- Evaluates True if the two values are different, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2
ANALOG IN	ANALOG IN
ANALOG OUT	ANALOG OUT
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
COUNTER	COUNTER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
FREQUENCY	FREQUENCY
OFF LATCH	OFF LATCH
OFF PULSE MEAS.	OFF PULSE MEAS.
OFF TIME TOTALIZER	OFF TIME TOTALIZER
ON LATCH	ON LATCH
ON PULSE MEAS.	ON PULSE MEAS.
ON TIME TOTALIZER	ON TIME TOTALIZER
PERIOD	PERIOD
QUADRATURE COUNTER	QUADRATURE COUNTER
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER
VARIABLE TIMER	VARIABLE TIMER

Example: **NOT EQUAL**

Is	BATCH STEP	*variable integer*
To	4	*constant integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use WITHIN LIMITS? to test for an approximate match.
- Use either the EQUAL condition or the False exit if the objective is to test for equality.

See Also: GREATER, LESS, LESS OR EQUAL, GREATER OR EQUAL, EQUAL

NOT EQUAL TO FLOAT TABLE DATA — Logical

Function: To determine if a numeric value is different from a specified value in a float table.

Typical Use: To perform reverse logic.

Details:

- Determines if one value (Argument 1) is different from another (a value at index Argument 2 in float table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0.0	0.0	False
0.0001	0.0	True
-98.765	-98.765	False
22.22	22.22	False

- Evaluates True if the two values are different, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	FLOAT TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **NOT EQUAL TO FLOAT TABLE DATA**

Is	THIS READING	*variable float*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF READINGS	*float table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either EQUAL TO FLOAT TABLE DATA or the False exit if the objective is to test for equality.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....FLOAT TABLE DATA" conditions

NOT EQUAL TO INTEGER TABLE DATA — Logical

Function: To determine if a numeric value is different from a specified value in an integer table.

Typical Use: To perform reverse logic.

Details:

- Determines if one value (Argument 1) is different from another (a value at index Argument 2 in integer table Argument 3). Examples:

VALUE 1	VALUE 2	RESULT
0	0	False
1	0	True
-32768	-32768	False
2222	2222	False

- Evaluates True if the two values are different, False otherwise.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT INTEGER	INTEGER TABLE
ANALOG OUT	VARIABLE INTEGER	
CONSTANT FLOAT		
CONSTANT INTEGER		
COUNTER		
DIGITAL IN		
DIGITAL OUT		
FREQUENCY		
OFF LATCH		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON LATCH		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
QUADRATURE COUNTER		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: **NOT EQUAL TO INTEGER TABLE DATA**

Is	THIS READING	*variable integer*
At Index	TABLE INDEX	*variable integer*
Of Table	TABLE OF COUNTS	*integer table*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use either EQUAL TO INTEGER TABLE DATA or the False exit if the objective is to test for equality.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: Other "....INTEGER TABLE DATA" conditions

NOT? Logical

Function: To determine if a value is False (zero, off).

Typical Use: To perform False testing.

Details:

- Determines if Argument 1 is False. Examples:

ARGUMENT 1	RESULT
0	True
-1	False
22	False

- Evaluates True if Argument 1 is False (zero, off). Evaluates False if Argument 1 is True (non-zero, on).
- Functionally equivalent to VARIABLE FALSE?

Arguments:

ARGUMENT 1
CONSTANT FLOAT
CONSTANT INTEGER
DIGITAL IN
DIGITAL OUT
VARIABLE FLOAT
VARIABLE INTEGER

Example: **NOT?**

Is CURRENT STATE *variable integer*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- Use either VARIABLE TRUE? or the False exit if the objective is to determine whether a value is True (non-zero).

See Also: AND, OR, XOR, VARIABLE TRUE?

OR — Logical

Function: To determine if either or both of two values are True.

Typical Use: To OR two values within an AND type condition block.

Details:

- Determines if Argument 1 or Argument 2 is non-zero. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
-1	0	True
0	-1	True
-1	-1	True

- Evaluates True if either argument is True (non-zero, on). Evaluates False if both arguments are False (zero, off).

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **OR**

Is	LIMIT SWITCH1	*digital input*
Is	LIMIT SWITCH2	*digital input*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- Use either VARIABLE FALSE? or the False exit if the objective is to determine whether both values are False (zero, off).
- Multiple uses of OR within a condition block result in the OR pairs being ANDed.

See Also: NOT, AND, XOR

WITHIN LIMITS? Logical

Function: To determine if a value is greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To check if a temperature is within an acceptable range.

Details:

- Determines if Argument 1 is no less than Argument 2 and no greater than Argument 3. Examples:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3	RESULT
0.0	0.0	100.0	True
-32768	0.0	100.0	False
72.1	68.0	72.0	False
-1.0	-45.0	45.0	True

- Evaluates True if Argument 1 falls between Arguments 2 and 3 or equals either value. Evaluates False if Argument 1 is less than Argument 2 or greater than Argument 3.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
ANALOG IN	CONSTANT FLOAT	CONSTANT FLOAT
ANALOG OUT	CONSTANT INTEGER	CONSTANT INTEGER
CONSTANT FLOAT	VARIABLE FLOAT	VARIABLE FLOAT
CONSTANT INTEGER	VARIABLE INTEGER	VARIABLE INTEGER
COUNTER		
FREQUENCY		
OFF PULSE MEAS.		
OFF TIME TOTALIZER		
ON PULSE MEAS.		
ON TIME TOTALIZER		
PERIOD		
TIME PROP. OUTPUT		
VARIABLE FLOAT		
VARIABLE INTEGER		
VARIABLE TIMER		

Example: This example evaluates True if CURRENT TEMP is greater than or equal to COLDEST TEMP and less than or equal to HOTTEST TEMP. It evaluates False otherwise.

WITHIN LIMITS?

Is	CURRENT TEMP	*variable float*
Low Limit	COLDEST TEMP	*variable float*
High Limit	HOTTEST TEMP	*variable float*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- Use to replace two conditions: LESS OR EQUAL and GREATER OR EQUAL.

See Also: LESS OR EQUAL, GREATER OR EQUAL

XOR — Logical

Function: To determine if two values are at opposite True/False states.

Typical Use: To determine if a logic value has changed state.

Details:

- Determines if Argument 1 and Argument 2 have different True/False states. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
0	0	False
0	1	True
1	0	True
1	1	False
0	-1	True
-1	0	True
-1	-1	False
22	0	True
22	-4	False

- Evaluates True if one item is True (non-zero, on) and the other is False (zero, off). Evaluates False if both items are True or if both items are False.
- Functionally equivalent to the NOT EQUAL condition when using allowable values.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT FLOAT	CONSTANT FLOAT
CONSTANT INTEGER	CONSTANT INTEGER
DIGITAL IN	DIGITAL IN
DIGITAL OUT	DIGITAL OUT
VARIABLE FLOAT	VARIABLE FLOAT
VARIABLE INTEGER	VARIABLE INTEGER

Example: **XOR**

Is	LIMIT SWITCH1 PREV	*variable integer*
Is	LIMIT SWITCH1	*digital input*

Notes:

- See the Logical Overview in Chapter 1 for important information.
- It is advisable to use only integers or digital channels with this command.
- Use the False exit if the objective is to test two values for equivalent True/False states.

See Also: NOT, AND, OR

STRING CONDITIONS

EQUAL TO STRING TABLE DATA — String

Function: To compare two strings for equality.

Typical Use: To check passwords or barcodes for an exact match with an entry in a string table.

Details:
- Determines if one string (Argument 1) is equal to another (a string at index Argument 2 in string table Argument 3). Examples:

STRING 1	STRING 2	RESULT
"OPTO"	"OPTO"	True
"OPTO"	"Opto"	False
"22"	"22"	True
"2 2"	"22"	False

- Evaluates True if both strings are exactly the same, False otherwise.
- Only an exact match on all characters (including leading or trailing spaces) will return a True.
- This test is case-sensitive. For example, a "T" does not equal a "t."
- Valid range for the *At Index* parameter (Argument 2) is zero to the table length (size).
- Functionally equivalent to the TEST EQUAL STRINGS operation.
- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.

Arguments:

ARGUMENT 1	ARGUMENT 2	ARGUMENT 3
CONSTANT STRING	CONSTANT INTEGER	STRING TABLE
VARIABLE STRING	VARIABLE INTEGER	

Example: The following example compares a new barcode to a string in a string table. This could be done in a loop to see if the new barcode exists in a table.

EQUAL TO STRING TABLE DATA

Is	NEW BARCODE	*variable string with barcode*
At Index	LOOP INDEX	*variable integer*
Of Table	CURRENT PRODUCTS	*string table*

Notes:
- See the String Overview in Chapter 1 for important information.
- Many additional string commands are available. These are "external" commands that require library support. Consult the Opto 22 BBS.

Error Codes: Queue error 32 = Bad table index value — index was negative or greater than the table size

See Also: TEST EQUAL STRINGS, STRING EQUAL, GET SIZE OF STRING TABLE

STRING EQUAL — String

Function: To compare two strings for equality.

Typical Use: To check passwords or barcodes for an exact match.

Details:

- Determines if strings Argument 1 and Argument 2 are equal. Examples:

ARGUMENT 1	ARGUMENT 2	RESULT
"OPTO"	"OPTO"	True
"OPTO"	"Opto"	False
"22"	"22"	True
"2 2"	"22"	False

- Evaluates True if both strings are exactly the same, False otherwise.
- Only an exact match on all characters (including leading or trailing spaces) will return a True.
- This test is case-sensitive. For example, a "T" does not equal a "t."
- Functionally equivalent to the TEST EQUAL STRINGS operation.
- Quotes ("") are used for readability only. They are not part of the string. Do not type them or expect to see them.

Arguments:

ARGUMENT 1	ARGUMENT 2
CONSTANT STRING	CONSTANT STRING
VARIABLE STRING	VARIABLE STRING

Example: **STRING EQUAL**

Is	NEW ENTRY	*variable string*
To	PASSWORD	*variable string*

Notes:

- See the String Overview in Chapter 1 for important information.
- Use EQUAL TO STRING TABLE DATA to compare with strings in a table.

See Also: TEST EQUAL STRINGS, EQUAL TO STRING TABLE DATA

CHAPTER 4

ERROR CODES

I/O UNIT ERRORS

These errors are reported to the error queue by the Mistic controller.

1 I/O unit received an inappropriate command

Generated by: Mistic I/O unit

Possible causes: • A digital I/O unit with the address and port of an analog I/O unit or vice versa.

2 Bad CRC on a message to or from an I/O unit

Generated by: Mistic I/O unit or the Mistic controller

Possible causes: • Incorrect or loose communications wiring.

• High noise level on the communications line.

• Missing terminator on the ends of the communication cable.

• Twisted pair cable not used.

• Two or more I/O units with the same address.

3 Length of received message is too long

Generated by: Mistic I/O unit

Possible causes: • Incorrect or loose communications wiring.

• High noise level on the communications line.

• Missing terminator on the ends of the communication cable.

• Twisted pair cable not used.

• Two or more I/O units with the same address.

• Improperly formatted message sent to I/O unit.

4 Powerup has occurred

Generated by: Mistic I/O unit

Possible causes: • Power has been cycled on the I/O unit.

Important notes: • **This is not an error.** It is notification that communication to the I/O unit has been re-established following an error 29.

5 I/O unit received insufficient data in a particular data field

Generated by: Mistic I/O unit

Possible causes:
- A digital I/O unit with the address and port of an analog I/O unit.

6 I/O unit watchdog timeout

Generated by: Mistic I/O unit

Possible causes:
- Lack of communication to the I/O unit.

Important notes:
- **This is not an error.** It is notification that communication to the I/O unit was interrupted.
- The command that returned this code will not be executed.

7 I/O unit received invalid data in a particular data field

Generated by: Mistic I/O unit

Possible causes:
- Sending a value greater than 65535 to an analog channel.
- Sending a RAMP TO POINT command with a value of 0 units/sec.
- Sending a GENERATE N PULSES command with an on or off time value that is too small.

9 I/O unit has an invalid module type

Generated by: Mistic I/O unit

Possible causes:
- No module installed.
- An output module installed in a channel configured as an input.
- Sending an input command to an analog channel with either no module installed or an output module installed.
- Sending an output command to an analog channel with an input module installed.
- Sending an analog single-point I/O unit command to an analog HRD I/O unit or vice-versa.
- An event/reaction referencing a digital output that was later changed to a digital input.

10 I/O unit has an invalid event/reaction entry

Generated by: Mistic I/O unit

Possible causes:
- An attempt to enable an event interrupt on a null entry in the event/reaction table.
- An illegal reaction command specified.

11 I/O unit digital time delay capability exceeded

Generated by: Mistic I/O unit

Possible causes:
- An attempt to start a square wave, generate N pulses, or set up a TPO with a value of less than 10 milliseconds or with more than eight output channels on the same digital I/O unit.

29 No response from I/O unit

Generated by: Mistic controller

Possible causes:
- Improper jumper settings (address, baud, protocol) at the I/O unit.
- Low power supply voltage at the I/O unit.
- No power at the I/O unit.
- Bad communication link to the I/O unit.
- Missing terminator on the ends of the communication cable.
- Two or more I/O units with the same address.

GENERAL ERRORS

These errors are reported to the error queue by the Mistic controller.

30 Invalid port number

Generated by: Mistic controller

Possible causes:
- Sending a peer message to the same Mistic controller (analogous to talking to yourself).
- Firmware error.

31 Send timeout

Generated by: Mistic controller

Possible causes:
- CTS low on an RS-232 HOST port.
- Sending long strings to a HOST port.
- Port timeout delay too short.
- Firmware error.

32 Bad table index

Generated by: Mistic controller

Possible causes:
- Negative table index value.
- Table index value greater than the table length.

33 Numeric overflow

Generated by: Mistic controller

Possible causes: • The result of a calculation is larger than the numeric type used.

35 Not a number

Generated by: Mistic controller

Possible causes: • A math operation resulting in a complex or imaginary number, such as the square root of a negative number.

36 Divide by zero

Generated by: Mistic controller

Possible causes: • A math operation tried to divide a constant or variable by 0.

38 Bus fault

Generated by: Mistic controller

Possible causes: • Attempt to access invalid memory areas (firmware or library bug).

• Failure in the Mistic controller hardware.

39 Port already in use

Generated by: Mistic controller

Possible causes: • Attempt to have more than one port “open” in a chart.

41 Invalid E/R hold buffer

Generated by: Mistic I/O unit

Possible causes: • Attempt to read the event/reaction data hold buffer for an event/reaction that is not configured with a “read and hold” reaction.

45 String too short to hold data

Generated by: Mistic controller

Possible causes: • String variable too short for data specified.

• Attempt to put the date or time in a string with a length less than eight.

ERRORS REPORTED TO HOST PORT DEVICES

These errors are reported to HOST port devices (such as the Cyrano Debugger and MMI) by the Mistic controller.

0 "Is not Unique": Mistic controller received a new word that already existed

Generated by: Mistic controller during a download

Important note: • **This is not an error.** It is notification that the word just defined has actually been redefined, since a word by the original name already existed.

1 "Undefined Command": Mistic controller received an undefined command

Generated by: Mistic controller during a download

Possible causes: • Old Mistic controller firmware.

• Not selecting YES for MISTIC.LIB under Download Options in the Cyrano Configurator when using commands that require library support.

• MISTIC.LIB does not include the referenced command.

• Typo in a file being downloaded (MISTIC.LIB, MISTIC.###, MISTIC.INC, or MISTIC.TRM).

2 Mistic controller received a message with a bad CRC

Generated by: Mistic controller while communicating with a PC

Possible causes: • Incorrect or loose communications wiring.

• High noise level on the communications line.

• Missing terminator on the ends of the communication cable (RS-485/422 only).

• Twisted pair cable not used (RS-485/422 only).

• Incorrect cable used for ARCNET or Ethernet connections.

4 Mistic controller power-up clear

Generated by: Mistic controller

Possible causes: • Mistic controller lost power since last communication.

Important notes: • **This is not an error.** It is a notification only.

• The command that returned this code will not be executed.

5 Mistic controller received insufficient data in a particular data field

Generated by: Mistic controller

Possible causes: • Error in command syntax.

38 Mistic controller bus fault

Generated by: Mistic controller

Possible causes:
- An attempt to access invalid or protected memory areas (firmware or library bug).
- Failure in the Mistic controller hardware.

40 I/O unit not configured

Generated by: Mistic controller

Possible causes:
- Communicating to an I/O unit that has not been configured.
- I/O unit configured as the wrong type.

42 "Mistic Controller Busy": Mistic controller HOST port is busy

Generated by: Mistic controller

Possible causes:
- Another device has locked the HOST port while downloading to the Mistic controller.

Important notes:
- **This is not an error.** It is notification that the Mistic controller is busy.
- The command that returned this code will not be executed.

43 HOST port relock

Generated by: Mistic controller.

Possible causes:
- An error occurring after the UNLOCK command.

50 "Empty Stack": Mistic controller stack is empty

Generated by: Mistic controller during a download or while running

Possible causes:
- A command requesting more items from the stack than are available.
- An ENDIF (Forth "THEN") without a corresponding IF.
- An UNTIL (Forth "UNTIL") without a corresponding LOOP ... UNTIL (Forth "BEGIN").
- An ENDSWITCH (Forth "ENDCASE") without a corresponding SWITCH (Forth "CASE").
- A BREAK (Forth "ENDOF") without a corresponding CASE (Forth "OF").

51 "Dictionary Full": Mistic controller dictionary is full

Generated by: Mistic controller during a download

Possible causes:
- Allocated word dictionary space is full. Too many words were downloaded.

52 "Stack Full": Mistic controller stack is full

Generated by: Mistic controller

Possible causes: • A command leaving one or more items on the stack.

53 "Compilation Only": Mistic controller compilation only

Generated by: Mistic controller during a download

Possible causes: • A word that can only be used within a definition being used outside the definition.

54 "Execution Only": Mistic controller execution only

Generated by: Mistic controller during a download

Possible causes:

- A word being defined within a word that is being defined.
- A missing semicolon from a Forth word definition.

55 "DEF Not Finished": Mistic controller DEF not finished

Generated by: Mistic controller during a download

Possible causes:

- An unfinished loop construct used when terminating the definition (for example, an IF without a THEN).
- A missing semicolon from a Forth word definition.

58 "Out of Memory": Mistic controller is out of memory

Generated by: Mistic controller during a download

Possible causes: • A program too large to fit in memory.

59 "Invalid Data": Mistic controller received invalid data

Generated by: Mistic controller during a download or while running

Possible causes: • Data type or range invalid for the command.

— ARCNET card not responding at configured address

Generated by: Cyrano

Possible causes: • Invalid ARCNET BASE address.

— Send error: Controller offline or address incorrect

Generated by: Cyrano

Possible causes:
- I/O defined on the same port used by the HOST task.
- Controller powered off.
- Controller configured with wrong address.
- Timeout interval too short.

— Timeout error

Generated by: Cyrano

Possible causes: • Communication cannot be completed within the time specified through the CONFIGURE MISTIC COMMUNICATIONS dialog box.

COMMUNICATION AND STRING COMMAND ERRORS

These errors are reported to the *Put Result In* parameter of Cyrano by the Mistic controller.

-40 Port busy

Generated by: Mistic controller

Possible causes: • Specified port already in use.

-41 Send timeout

Generated by: Mistic controller

Possible causes:
- CTS low on a serial port in RS-232 mode.
- Sending long strings to a serial port.
- Serial port timeout delay too short.
- For ports 4 and 7, transmit buffer is full.

-42 No response

Generated by: Mistic controller

Possible causes:
- No carriage return (character 13) in the receive buffer.
- Serial port timeout delay too short.

-43 Not enough data returned

Generated by: Mistic controller

Possible causes:
- Invalid response to an OPTOMUX command.

-44 Invalid data returned

Generated by: Mistic controller

Possible causes:
- Illegal first character in response to an OPTOMUX command.

-45 Bad checksum or CRC on received message

Generated by: Mistic controller

Possible causes:
- Incorrect or loose communications wiring.
- High noise level on the communications line.
- Missing terminator on the ends of the communication cable.
- Twisted pair cable not used.
- Two or more devices with the same address.

-46 Invalid limits

Generated by: Mistic controller when using GET NTH CHARACTER

Possible causes:
- Index into a string variable was negative or greater than the STRING LENGTH.

-47 NAK returned

Generated by: Mistic controller

Possible causes:
- NAK returned in response to an OPTOMUX command (usually followed by an error code).

-48 String too short

Generated by: Mistic controller

Possible causes:
- Target string variable too short to hold response to an OPTOMUX or CRC command.

-49 String was empty

Generated by: Mistic controller

Possible causes: • A command that expected a string variable to contain one or more characters.

-50 Invalid characters in string

Generated by: Mistic controller

Possible causes: • Unexpected characters in the string passed to the CONFIGURE PORT command.

MOTION CONTROL ERRORS

These errors are reported to the error queue by the Mistic controller.

128 Invalid command

Indicates that the motion I/O unit did not recognize the command. Make sure the motion axis is configured as the correct type (servo or stepper) at the correct address. Also check that the motion I/O unit's firmware is consistent with the current version of Cyrano (see page vi). If it isn't, contact Opto 22 for an update.

129 Limit switch active

Occurs when a START MOTION operation is executed while a limit switch is active.

130 Cannot load acceleration

Occurs when a SET ACCELERATION operation is executed while the axis running. The acceleration can be set only when motion is stopped.

131 Cannot set direction

Occurs when a SET DIRECTION FORWARD or SET DIRECTION REVERSE operation is executed and the axis is configured for position-relative or position-absolute mode. The SET DIRECTION operations are valid only when the axis is configured for velocity mode.

132 Cannot load position

Occurs when a SET TARGET POSITION operation is executed and the axis is configured for velocity mode. The SET TARGET POSITION operation is valid only when the axis is configured for position-relative or position-absolute mode.

133 Busy executing FIND HOME sequence

Occurs when an operation is executed that affects motion while the specified axis is busy executing a FIND HOME operation. Your strategy must wait until the FIND HOME sequence is complete before attempting the operation.

134 Limit and home switches disabled

Occurs when an operation is executed that depends on the limit or home switches (e.g., FIND HOME, FIND INDEX) and the inputs are disabled. The inputs must be enabled and have their polarities properly configured before they are used.

135 Trigger buffer enabled

Occurs when an operation is executed that involves the trigger or watchdog buffer while the buffer is in use. Disable the buffer first before trying to program it.

136 Both limit switches True

Occurs when an operation is executed that may be affected by the limit switches and both limit switches are indicating a True condition. This may indicate a mechanical failure, or the polarity of the inputs may not be set up correctly.

137 Trigger buffer empty

Occurs when execution of operations in the trigger buffer is attempted (via use of the FORCE TRIGGER operation or activation of a configured trigger input) and the trigger buffer is empty.

138 Servo command sent to stepper

Occurs when a command issued to a stepper axis on a multi-axis motion brick is not a valid stepper axis command (the command was a servo ONLY command).

139 No velocity mode support

Occurs because the stepper axis does not support velocity mode.

140 Motion step count too high

Occurs when the commanded motion (in relative or absolute mode) exceeds 0x00200000 (2,097,152) steps in a single move.

141 Clear position counter

Occurs when a CLEAR POSITION COUNTER is issued while a stepper axis is in motion. You must wait for a MOTION COMPLETE before issuing this command.

142 Feedback encoder missing

A stepper motor must be equipped with a feedback device (encoder) to work in closed-loop mode.

143 Start motion

Occurs when a START MOTION is issued while a stepper axis is in motion. You must wait for a MOTION COMPLETE before issuing this command.

144 Cannot execute FIND INDEX while axis is running

Occurs when the FIND INDEX operation is executed while the specified axis is running. Your strategy should make sure that the move is complete before attempting to execute the FIND INDEX operation.

145 Invalid data

Occurs if a FIND INDEX operation is executed and the *Encoder Lines per Revolution* parameter is configured as zero.

146 Busy executing FIND INDEX

Occurs when an operation is executed that affects motion, and the axis is busy executing a FIND INDEX operation. Your strategy must wait until the FIND INDEX sequence is complete before attempting the operation.

147 Limit switch True

Occurs when a FIND INDEX operation is executed and one of the limit switches is active. Make sure the axis is in a position where it will not trip a limit switch when the FIND INDEX operation is executed.

148 Anticipation time break point warning

This warning occurs when an anticipation time break point is set with a time that calculates to less time than is required to complete the move. In this case, the break point output will trip immediately at the beginning of the move. Remember, anticipation time is calculated with respect to the end of the motion profile of the axis.

149 Invalid scale factor

Occurs if the scale factor is configured with a value that is zero or less.

150 Scale factor divide by zero

Occurs if a divide-by-zero error occurs while the axis is doing scale factor calculations.

151 Anticipation time break point

Occurs when a SET ANTICIPATION TIME BREAKPT. operation is executed and the specified axis is in velocity mode.

152 FIND HOME sequence active

Occurs when an operation is executed that interferes with a FIND HOME sequence that is in progress on the axis.

153 Following error

Occurs if the axis did not start motion because of an existing following error condition.

154 Cannot change loop mode

Occurs when a SET OPEN LOOP MODE or SET CLOSED LOOP MODE command is issued while a stepper axis is in motion. You must wait for a MOTION COMPLETE before issuing this command.

155 FIND INDEX command error

Occurs because there is no feedback (encoder) on an open loop stepper, so there is no index to find. The FIND INDEX command cannot be issued in open loop mode.

156 ENABLE BREAKPOINT command error

Occurs when an ENABLE BREAKPOINT command is issued while a stepper axis is in motion. You must wait for a MOTION COMPLETE before issuing this command.

157 Trigger buffer storage already enabled

Occurs because a BEGIN GLOBAL TRIGGER BUFFER PRESTORE command was followed by another BEGIN TRIGGER BUFFER PRESTORE command (multi or single) without an END GLOBAL TRIGGER BUFFER PRESTORE command in between. Once global trigger buffer storage has begun, it must be completed before any new trigger buffer storage can begin (on any axis on the brick).

160 BEGIN TRIGGER BUFFER command error

Occurs when a BEGIN TRIGGER BUFFER PRESTORE or BEGIN WATCHDOG BUFFER PRESTORE operation is executed while the specified axis is in trigger buffer storage mode. An END TRIGGER BUFFER PRESTORE or END WATCHDOG BUFFER PRESTORE operation must first be executed.

161 Trigger buffer full

Occurs when the trigger buffer or watchdog buffer is in storage mode and an operation is executed (sent to the buffer for storage) that causes the buffer to overflow. Reduce the number of operations being stored in the buffer.

162 TRIGGER BUFFER command error

Occurs when an operation that reads or returns status information is executed while the axis is in trigger or watchdog buffer storage mode. Only operations that control motion and do not return values may be stored in the buffers.

163 END TRIGGER BUFFER command error

Occurs when an END TRIGGER BUFFER PRESTORE or END WATCHDOG BUFFER PRESTORE operation is executed and the specified axis is not in trigger or watchdog buffer storage mode.

164 Trigger buffer delimiter error

Occurs when an INSERT BUFFER DELIMITER operation is executed and the specified axis is not in trigger or watchdog buffer storage mode.

PRODUCT SUPPORT

APPENDIX A

If you have any questions about this product, contact Opto 22 Product Support Monday through Friday, 8 a.m. to 5 p.m. Pacific Time.

Phone:	800-TEK-OPTO (835-6786) 951-695-3080
Fax:	951-695-3017
E-mail:	support@opto22.com
Opto 22 Web site:	www.opto22.com

When calling for technical support, be prepared to provide the following information about your system to the Product Support engineer:

- Software and version being used
- Controller firmware version
- PC configuration
- A complete description of your hardware and operating systems, including:
  — jumper configuration
  — accessories installed (such as expansion daughter cards)
  — type of power supply
  — types of I/O units installed
  — third-party devices installed (e.g., barcode readers)
- Specific error messages seen

INDEX