ioCONTROL COMMAND REFERENCE

FORM 1301-060308—MARCH, 2006

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
Email: support@opto22.com
Web: support.opto22.com

ioControl Command Reference
Form 1301-060308—MARCH, 2006

Opto 22 warrants all of its products to be free from defects in material or workmanship for 30 months from the manufacturing date code. This warranty is limited to the original cost of the unit only and does not cover installation, labor, or any other contingent costs. Opto 22 I/O modules and solid-state relays with date codes of 1/96 or later are guaranteed for life. This lifetime warranty excludes reed relay, SNAP serial communication modules, SNAP PID modules, and modules that contain mechanical contacts or switches. Opto 22 does not warrant any product, components, or parts not manufactured by Opto 22; for these items, the warranty from the original manufacturer applies. These products include, but are not limited to, OptoTerminal-G70, OptoTerminal-G75, and Sony Ericsson GT-48; see the product data sheet for specific warranty information. Refer to Opto 22 form number 1042 for complete warranty information.

Table of Contents

Welcome

Welcome to ioControl™, Opto 22's visual control language for SNAP Ultimate I/O™ and other Opto 22 control systems. ioControl provides a complete and powerful set of commands for all your industrial control needs.

About this Reference

This command reference describes in detail all ioControl programming commands, or instructions.The commands are listed alphabetically. To find a command by its command group, such as Analog Point commands or Control Engine commands, see the chart starting on page xx.

The *ioControl User's Guide*, in a separate binder, explains how to install and use ioControl. For helpful information on using commands, see Chapter 10, "Programming with Commands," in the user's guide.

This reference assumes that you are already familiar with Microsoft® Windows® on your personal computer. If you are not familiar with Windows or your PC, refer to the documentation from Microsoft and your computer manufacturer.

Information Key

Pro

Commands with the Pro icon are available only in ioControl Professional, not in the basic version of ioControl.

Other Resources

Documents and Online Help

To help you understand and use ioControl systems, the following resources are provided:

- **Online Help** is available in ioControl and in most of the utility applications. To open online Help, choose Help→Contents and Index in any screen.
- ***ioControl User's Guide*** shows how to install and use ioControl.
- ***ioControl Command Reference*** contains detailed information about each command (instruction) available in ioControl.
- A **quick reference card**, located in the front pocket of the *ioControl Command Reference*, lists all ioControl commands plus their OptoScript™ code equivalents and arguments.
- ***ioManager User's Guide*** and other guides provided with specific hardware help you install, configure, and use controllers and I/O units.

Online versions (Adobe® Acrobat® format) of ioControl documents are provided on the CD that came with your controller or purchase of Professional software and are also available from the Help menu in ioControl. To view a document, select Help→Manuals, and then choose a document from the submenu.

When you purchase ioControl Professional or ioProject Professional, you also receive a complete set of printed documents.

Resources are also available on the Opto 22 Web site at www.opto22.com. You can conveniently access the Web site using the Help menu in ioControl. Select Help→Opto 22 on the Web, and then select an online resource from the submenu.

Product Support

If you have any questions about ioControl, you can call, fax, or email Opto 22 Product Support.

Phone:	800-TEK-OPTO (835-6786) 951-695-3080 (Hours are Monday through Friday, 7 a.m. to 5 p.m. Pacific Time)
Fax:	951-695-3017
Email:	support@opto22.com
Opto 22 Web site:	support.opto22.com

NOTE: Email messages and phone calls to Opto 22 Product Support are grouped together and answered in the order received.

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
- Firmware versions
- PC configuration (type of processor, speed, memory, operating system)
- A complete description of your hardware and operating systems, including:
  - type of power supply
  - types of I/O units installed
  - third-party devices installed (for example, barcode readers)
- Specific error messages seen.

Commands by Command Group

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
Digital Point	Clear All Latches	C-20	ClearAllLatches(*On I/O Unit*)
	Clear Counter	C-22	ClearCounter(*On Point*)
	Clear Off-Latch	C-26	ClearOffLatch(*On Point*)
	Clear On-Latch	C-27	ClearOnLatch(*On Point*)
	Generate N Pulses	G-5	GenerateNPulses(*On Time (Seconds), Off Time (Seconds), Number of Pulses, On Point*)
	Get & Clear Counter	G-18	GetClearCounter(*From Point*)
	Get & Clear Off-Latch	G-25	GetClearOffLatch(*From Point*)
	Get & Restart Off-Pulse Measurement	G-27	GetRestartOffPulseMeasurement(*From Point*)
	Get & Restart Off-Time Totalizer	G-28	GetRestartOffTimeTotalizer(*From Point*)
	Get & Restart On-Pulse Measurement	G-29	GetRestartOnPulseMeasurement(*From Point*)
	Get & Restart On-Time Totalizer	G-30	GetRestartOnTimeTotalizer(*From Point*)
	Get & Restart Period	G-31	GetRestartPeriod(*From Point*)
	Get & Clear On-Latch	G-26	GetClearOnLatch(*From Point*)
	Get Counter	G-48	GetCounter(*From Point*)
	Get Frequency	G-56	GetFrequency(*From Point*)
	Get Off-Latch	G-102	GetOffLatch(*From Point*)
	Get Off-Pulse Measurement	G-103	GetOffPulseMeasurement(*From Point*)
	Get Off-Pulse Measurement Complete Status	G-104	GetOffPulseMeasurementCompleteStatus(*From Point*)
	Get Off-Time Totalizer	G-105	GetOffTimeTotalizer(*From Point*)
	Get On-Latch	G-106	GetOnLatch(*From Point*)
	Get On-Pulse Measurement	G-107	GetOnPulseMeasurement(*From Point*)
	Get On-Pulse Measurement Complete Status	G-108	GetOnPulseMeasurementCompleteStatus(*From Point*)
	Get On-Time Totalizer	G-109	GetOnTimeTotalizer(*From Point*)
	Get Period	G-110	GetPeriod(*From Point*)
	Get Period Measurement Complete Status	G-111	GetPeriodMeasurementCompleteStatus(*From Point*)
	Off?	O-1	IsOff(*Point*)
	Off-Latch Set?	O-2	IsOffLatchSet(*On Point*)
	On?	O-3	IsOffLatchSet(*On Point*)
	On-Latch Set?	O-4	IsOn(*Point*)
	Set TPO Percent	S-84	SetTpoPercent(*To Percent, On Point*)
	Set TPO Period	S-85	SetTpoPeriod(*To Seconds, On Point*)
	Start Continuous Square Wave	S-94	StartContinuousSquareWave(*On Time (Seconds)*, Off Time (*Seconds*), *On Point*)
	Start Counter	S-95	StartCounter(*On Point*)
	Start Off-Pulse	S-96	StartOffPulse(*Off Time (Seconds), On Point*)
	Start On-Pulse	S-97	StartOnPulse(*On Time (Seconds), On Point*)
	Stop Counter	S-101	StopCounter(*On Point*)
	Turn Off	T-25	TurnOff(*Output*)
	Turn On	T-27	TurnOn(*Output*)

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
Analog Point	Calculate & Set Analog Gain	C-1	CalcSetAnalogGain(On Point)
	Calculate & Set Analog Offset	C-2	CalcSetAnalogOffset(On Point)
	Get & Clear Analog Filtered Value	G-14	GetClearAnalogFilteredValue(From)
	Get & Clear Analog Maximum Value	G-15	GetClearAnalogMaxValue(From)
	Get & Clear Analog Minimum Value	G-16	GetClearAnalogMinValue(From)
	Get & Clear Analog Totalizer Value	G-17	GetClearAnalogTotalizerValue(From)
	Get Analog Filtered Value	G-38	GetAnalogFilteredValue(From)
	Get Analog Maximum Value	G-39	GetAnalogMaxValue(From)
	Get Analog Minimum Value	G-40	GetAnalogMinValue(From)
	Get Analog Square Root Filtered Value	G-41	GetAnalogSquareRootFilteredValue(From)
	Get Analog Sqaure Root Value	G-42	GetAnalogSquareRootValue(From)
	Get Analog Totalizer Value	G-43	GetAnalogTotalizerValue(From)
	Ramp Analog Output	R-3	RampAnalogOutput(Ramp Endpoint, Units/Sec, Point to Ramp)
	Set Analog Filter Weight	S-7	SetAnalogFilterWeight(To, On Point)
	Set Analog Gain	S-8	SetAnalogGain(To, On Point)
	Set Analog Load Cell Fast Settle Level	S-9	SetAnalogLoadCellFastSettleLevel(To, On Point)
	Set Analog Load Cell Filter Weight	S-10	SetAnalogLoadCellFilterWeight(To, On Point)
	Set Analog Offset	S-11	SetAnalogOffset(To, On Point)
	Set Analog Totalizer Rate	S-12	SetAnalogTotalizerRate(To Seconds, On Point)
	Set Analog TPO Period	S-14	SetAnalogTpoPeriod(To, On Point)
Chart	Call Chart	C-4	CallChart(Chart)
	Calling Chart Running?	C-5	IsCallingChartRunning()
	Calling Chart Stopped?	C-5	IsCallingChartStopped()
	Calling Chart Suspended?	C-6	IsCallingChartSuspended()
	Chart Running?	C-9	IsChartRunning(Chart)
	Chart Stopped?	C-10	IsChartStopped(Chart)
	Chart Suspended?	C-11	IsChartSuspended(Chart)
	Continue Calling Chart	C-37	ContinueCallingChart()
	Continue Chart	C-38	ContinueChart(Chart)
	Get Chart Status	G-45	GetChartStatus(Chart)
	Start Chart	S-93	StartChart(Chart)
	Stop Chart	S-99	StopChart(Chart)
	Suspend Chart	S-106	SuspendChart(Chart)
Miscellaneous	Comment (Block)	C-30	/* block comment */
	Comment (Single Line)	C-31	// single line comment
	Float Valid?	F-3	IsFloatValid(Float)
	Generate Reverse CRC-16 on Table (32 bit)	G-9	GenerateReverseCrc16OnTable32(Start Value, Table, Starting Element, Number of Elements)
	Get Length of Table	G-83	GetLengthOfTable(Table)
	Get Type From Name	G-142	GetTypeFromName(Name)
	Get Value From Name	G-143	GetValueFromName(Name, Put Result In)
	Move	M-6	x = y;
	Move from Numeric Table Element	M-8	x = nt[0];
	Move Numeric Table Element to Numeric Table	M-13	nt1[0] = nt2[5];
	Move Numeric Table to Numeric Table	M-15	MoveNumTableToNumTable(From Table, From Index, To Table, To Index, Length)
	Move to Numeric Table Element	M-17	nt[0] = x;
	Move to Numeric Table Elements	M-18	MoveToNumTableElements(From, Start Index, End Index, Of Table)
	Shift Numeric Table Elements	S-90	ShiftNumTableElements(Shift Count, Table)

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
Error Handling	Add Message to Queue	A-4	AddMessageToQueue(*Severity, Message*)
	Add User Error to Queue	A-5	AddUserErrorToQueue(*Error Number*)
	Add User I/O Unit Error to Queue	A-6	AddUserIoUnitErrorToQueue(*Error Number, I/O Unit*)
	Caused a Chart Error?	C-7	HasChartCausedError(*Chart*)
	Caused an I/O Unit Error?	C-8	HasIoUnitCausedError(*I/O Unit*)
	Clear All Errors	C-18	ClearAllErrors()
	Copy Current Error to String	C-58	CurrentErrorToString(*Delimiter, String*)
	Disable I/O Unit Causing Current Error	D-13	DisableIoUnitCausingCurrentError()
	Enable I/O Unit Causing Current Error	E-9	EnableIoUnitCausingCurrentError()
	Error?	E-19	IsErrorPresent()
	Error on I/O Unit?	E-20	IsErrorOnIoUnit()
	Get Error Code of Current Error	G-52	GetErrorCodeOfCurrentError()
	Get Error Count	G-53	GetErrorCount()
	Get ID of Block Causing Current Error	G-64	GetIdOfBlockCausingCurrentError()
	Get Line Causing Current Error	G-84	GetLineCausingCurrentError()
	Get Name of Chart Causing Current Error	G-98	GetNameOfChartCausingCurrentError(*Put in*)
	Get Name of I/O Unit Causing Current Error	G-99	GetNameOfIoUnitCausingCurrentError(*Put in*)
	Get Severity of Current Error	G-137	GetSeverityOfCurrentError()
	Remove Current Error and Point to Next Error	R-22	RemoveCurrentError()
	Stop Chart on Error	S-100	StopChartOnError()
	Suspend Chart on Error	S-107	SuspendChartOnError()
I/O Unit	Get I/O Unit as Binary Value	G-65	GetIoUnitAsBinaryValue(*I/O Unit*)
	Get Target Address State	G-141	GetTargetAddressState(*Enable Mask, Active Mask, I/O Unit*)
	I/O Unit Ready?	I-4	IsIoUnitReady(*I/O Unit*)
	IVAL Move Numeric Table to I/O Unit	I-5	IvalMoveNumTableToIoUnit(*Start at Index, Of Table, Move to*)
	Move I/O Unit to Numeric Table	M-12	MoveIoUnitToNumTable(*I/O Unit, Starting Index, Of Table*)
	Move Numeric Table to I/O Unit	M-14	MoveNumTableToIoUnit(*Start at Index, Of Table, Move to*)
	Set All Target Address States	S-5	SetAllTargetAddressStates(*Must-On Mask, Must-Off Mask, Active Mask*)
	Set I/O Unit from MOMO Masks	S-29	SetIoUnitFromMomo(*Must-On Mask, Must-Off Mask, Digital I/O Unit*)
	Set Target Address State	S-81	SetTargetAddressState(*Must-On Mask, Must-Off Mask, Active Mask, I/O Unit*)
	Write I/O Unit Configuration to EEPROM	W-2	WriteIoUnitConfigToEeprom(*On I/O Unit*)

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
I/O Unit—Scratch Pad	Get I/O Unit Scratch Pad Bits	G-69	`GetIoUnitScratchPadBits(`*I/O Unit, Put Result in*`)`
	Get I/O Unit Scratch Pad Float Element	G-70	`GetIoUnitScratchPadFloatElement(`*I/O Unit, Index, Put Result in*`)`
	Get I/O Unit Scratch Pad Float Table	G-72	`GetIoUnitScratchPadFloatTable(`*I/O Unit, Length, From Index, To Index, To Table*`)`
	Get I/O Unit Scratch Pad Integer 32 Element	G-74	`GetIoUnitScratchPadInt32Element(`*I/O Unit, Index, Put Result in*`)`
	Get I/O Unit Scratch Pad Integer 32 Table	G-76	`GetIoUnitScratchPadInt32Table(`*I/O Unit, Length, From Index, To Index, To Table*`)`
	Get I/O Unit Scratch Pad String Element	G-78	`GetIoUnitScratchPadStringElement(`*I/O Unit, Index, Put Result in*`)`
	Get I/O Unit Scratch Pad String Table	G-80	`GetIoUnitScratchPadString(`*I/O Unit, Length, From Index, To Index, To Table*`)`
	Set I/O Unit Scratch Pad Bits from MOMO Mask	S-31	`SetIoUnitScratchPadBitsFromMomo(`*I/O Unit, Must-On Mask, Must-Off Mask*`)`
	Set I/O Unit Scratch Pad Float Element	S-32	`SetIoUnitScratchPadFloatElement(`*I/O Unit, Index, From*`)`
	Set I/O Unit Scratch Pad Float Table	S-34	`SetIoUnitScratchPadFloatTable(`*I/O Unit, Length, To Index, From Index, From Table*`)`
	Set I/O Unit Scratch Pad Integer 32 Element	S-36	`SetIoUnitScratchPadInt32Element(`*I/O Unit, Index, From*`)`
	Set I/O Unit Scratch Pad Integer 32 Table	S-38	`SetIoUnitScratchPadInt32Table(`*I/O Unit, Length, To Index, From Index, From Table*`)`
	Set I/O Unit Scratch Pad String Element	S-40	`SetIoUnitScratchPadStringElement(`*I/O Unit, Index, From*`)`
	Set I/O Unit Scratch Pad String Table	S-41	`SetIoUnitScratchPadStringTable(`*I/O Unit, Length, To Index, From Index, From Table*`)`
I/O Unit—Memory Map	Read Number from I/O Unit Memory Map	R-5	`ReadNumFromIoUnitMemMap(`*I/O Unit, Mem address, To*`)`
	Read Numeric Table from I/O Unit Memory Map	R-7	`ReadNumTableFromIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, To*`)`
	Read String from I/O Unit Memory Map	R-9	`ReadStrFromIoUnitMemMap(`*Length, I/O Unit, Mem address, To*`)`
	Read String Table from I/O Unit Memory Map	R-11	`ReadStrTableFromIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, To*`)`
	Write Number to I/O Unit Memory Map	W-3	`WriteNumToIoUnitMemMap(`*I/O Unit, Mem address, Variable*`)`
	Write Numeric Table to I/O Unit Memory Map	W-4	`WriteNumTableToIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, Table*`)`
	Write String to I/O Unit Memory Map	W-9	`WriteStrToIoUnitMemMap(`*I/O Unit, Mem address, Variable*`)`
	Write String Table to I/O Unit Memory Map	W-7	`WriteStrTableToIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, Table*`)`
I/O Unit—Event Msg	Get I/O Unit Event Message State	G-67	`GetIoUnitEventMsgState(`*I/O Unit, Event Message #, Put Result in*`)`
	Get I/O Unit Event Message Text	G-68	`GetIoUnitEventMsgText(`*I/O Unit, Event Message #, Put Result in*`)`
	Set I/O Unit Event Message State	S-26	`SetIoUnitEventMsgState(`*I/O Unit, Event Message #, State*`)`
	Set I/O Unit Event Message Text	S-27	`SetIoUnitEventMsgText(`*I/O Unit, Event Message #, Message Text*`)`

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
Time/Date	Copy Date to String (DD/MM/YYYY)	C-59	DateToStringDDMMYYYY(*String*)
	Copy Date to String (MM/DD/YYYY)	C-60	DateToStringMMDDYYYY(*String*)
	Copy Time to String	C-61	TimeToString(*String*)
	Get Day	G-49	GetDay()
	Get Day of Week	G-50	GetDayOfWeek()
	Get Hours	G-63	GetHours()
	Get Julian Day	G-82	GetJulianDay()
	Get Minutes	G-86	GetMinutes()
	Get Month	G-97	GetMonth()
	Get Seconds	G-135	GetSeconds()
	Get Seconds Since Midnight	G-136	GetSecondsSinceMidnight()
	Get System Time	G-140	GetSystemTime()
	Get Year	G-145	GetYear()
	Set Date	S-16	SetDate(*To*)
	Set Day	S-17	SetDay(*To*)
	Set Hours	S-25	SetHours(*To*)
	Set Minutes	S-43	SetMinutes(*To*)
	Set Month	S-57	SetMonth(*To*)
	Set Seconds	S-78	SetSeconds(*To*)
	Set Time	S-83	SetTime(*To*)
	Set Year	S-89	SetYear(*To*)
Timing	Continue Timer	C-39	ContinueTimer(*Timer*)
	Delay (mSec)	D-2	DelayMsec(*Milliseconds*)
	Delay (Sec)	D-3	DelaySec(*Seconds*)
	Down Timer Expired?	D-21	HasDownTimerExpired(*Down Timer*)
	Pause Timer	P-1	PauseTimer(*Timer*)
	Set Down Timer Preset Value	S-21	SetDownTimerPreset(*Target Value, Down Timer*)
	Set Up Timer Target Value	S-86	SetUpTimerTarget(*Target Value, Up Timer*)
	Start Timer	S-96	StartTimer(*Timer*)
	Stop Timer	S-102	StopTimer(*Timer*)
	Timer Expired?	T-10	HasTimerExpired(*Timer*)
	Up Timer Target Time Reached?	U-1	HasUpTimerReachedTargetTime(*Up Timer*)
Control Engine	Calculate Strategy CRC	C-3	CalcStrategyCrc()
	Erase Files in Permanent Storage	E-18	EraseFilesInPermanentStorage()
	Get Available File Space	G-44	GetAvailableFileSpace(*File System Type*)
	Get Control Engine Address	G-46	GetEngineAddress()
	Get Control Engine Type	G-47	GetEngineType()
	Get Firmware Version	G-55	GetFirmwareVersion(*Put in*)
	Load Files From Permanent Storage	L-7	LoadFilesFromPermanentStorage()
	Retrieve Strategy CRC	R-23	RetrieveStrategyCrc()
	Save Files To Permanent Storage	S-1	SaveFilesToPermanentStorage()

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
PID—Mistic	Clamp Mistic PID Output	C-16	`ClampMisticPidOutput(`*High Clamp, Low Clamp, On PID Loop*`)`
	Clamp Mistic PID Setpoint	C-17	`ClampMisticPidSetpoint(`*High Clamp, Low Clamp, On PID Loop*`)`
	Disable Mistic PID Output	D-14	`DisableMisticPidOutput(`*Of PID Loop*`)`
	Disable Mistic PID Output Tracking in Manual Mode	D-15	`DisableMisticPidOutputTrackingInManualMode(`*On PID Loop*`)`
	Disable Mistic PID Setpoint Tracking in Manual Mode	D-16	`DisableMisticPidSetpointTrackingInManualMode(`*On PID Loop*`)`
	Enable Mistic PID Output	E-10	`EnableMisticPidOutput(`*On PID Loop*`)`
	Enable Mistic PID Output Tracking in Manual Mode	E-11	`EnableMisticPidOutputTrackingInManualMode(`*On PID Loop*`)`
	Enable Mistic PID Setpoint Tracking in Manual Mode	E-12	`EnableMisticPidSetpointTrackingInManualMode(`*On PID Loop*`)`
	Get Mistic PID Control Word	G-87	`GetMisticPidControlWord(`*From PID Loop*`)`
	Get Mistic PID D Term	G-88	`GetMisticPidDTerm(`*From PID Loop*`)`
	Get Mistic PID I Term	G-89	`GetMisticPidITerm(`*From PID Loop*`)`
	Get Mistic PID Input	G-90	`GetMisticPidInput(`*PID Loop*`)`
	Get Mistic PID Mode	G-91	`GetMisticPidMode(`*PID Loop*`)`
	Get Mistic PID Output	G-92	`GetMisticPidOutput(`*PID Loop*`)`
	Get Mistic PID Output Rate of Change	G-93	`GetMisticPidOutputRateOfChange(`*From PID Loop*`)`
	Get Mistic PID P Term	G-94	`GetMisticPidPTerm(`*From PID Loop*`)`
	Get Mistic PID Scan Rate	G-95	`GetMisticPidScanRate(`*From PID Loop*`)`
	Get Mistic PID Setpoint	G-96	`GetMisticPidSetpoint(`*PID Loop*`)`
	Set Mistic PID Control Word	S-44	`SetMisticPidControlWord(`*On-Mask, Off-Mask, For PID Loop*`)`
	Set Mistic PID D Term	S-45	`SetMisticPidDTerm(`*To, On PID Loop*`)`
	Set Mistic PID I Term	S-46	`SetMisticPidITerm(`*To, On PID Loop*`)`
	Set Mistic PID Input	S-47	`SetMisticPidInput(`*PID Loop, Input*`)`
	Set Mistic PID Mode to Auto	S-48	`SetMisticPidModeToAuto(`*On PID Loop*`)`
	Set Mistic PID Mode to Manual	S-49	`SetMisticPidModeToManual(`*On PID Loop*`)`
	Set Mistic PID Output Rate of Change	S-50	`SetMisticPidOutputRateOfChange(`*To, On PID Loop*`)`
	Set Mistic PID P Term	S-51	`SetMisticPidPTerm(`*To, On PID Loop*`)`
	Set Mistic PID Scan Rate	S-52	`SetMisticPidScanRate(`*To, On PID Loop*`)`
	Set Mistic PID Setpoint	S-53	`SetMisticPidSetpoint(`*PID Loop, Setpoint*`)`
Deprecated	Set Digital I/O Unit from MOMO Masks	S-18	`SetDigitalIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Digital I/O Unit*`)`
	Set Digital-64 I/O Unit from MOMO Masks	S-19	`SetDigital64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Digital-64 I/O Unit*`)`
	Set Mixed I/O Unit from MOMO Masks	S-55	`SetMixedIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Mixed I/O Unit*`)`
	Set Mixed 64 I/O Unit from MOMO Masks	S-54	`SetMixed64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Mixed 64 I/O Unit*`)`
	Set Simple 64 I/O Unit from MOMO Masks	S-79	`SetSimple64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Simple 64 I/O Unit*`)`
	IVAL Set Digital Binary	I-8	`IvalSetDigitalBinary(`*On Mask, Off Mask, On I/O Unit*`)`
	IVAL Set Digital-64 I/O Unit from MOMO Masks	I-9	`IvalSetDigital64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Digital 64 I/O Unit*`)`
	IVAL Set Mixed I/O Unit from MOMO Masks	I-16	`IvalSetMixedIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Mixed I/O Unit*`)`
	IVAL Set Mixed 64 I/O Unit from MOMO Masks	I-15	`IvalSetMixed64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Mixed 64 I/O Unit*`)`
	IVAL Set Simple 64 I/O Unit from MOMO Masks	I-25	`IvalSetSimple64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Simple 64 I/O Unit*`)`

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
Communication	Accept Incoming Communication	A-2	AcceptIncomingCommunication(*Communication Handle*)
	Clear Communication Receive Buffer	C-21	ClearCommunicationReceiveBuffer(*Communication Handle*)
	Clear Receive Buffer	C-29	ClearReceiveBuffer
	Close Communication	C-29	CloseCommunication(*Communication Handle*)
	Communication Open?	C-32	IsCommunicationOpen(*Communication Handle*)
	Get Communication Handle Value	G-46	GetCommunicationHandleValue(*From, To*)
	Get End-Of-Message Terminator	G-51	GetEndOfMessageTerminator (*Communication Handle*)
	Get Number of Characters Waiting	G-101	GetNumCharsWaiting(*On Communication Handle*)
	Listen for Incoming Communication	L-5	ListenForIncomingCommunication(*Communication Handle*)
	Open Outgoing Communication	O-4	OpenOutgoingCommunication(*Communication Handle*)
	Receive Character	R-13	ReceiveChar(*Communication Handle*)
	Receive N Characters	R-14	ReceiveNChars(*Put In, Number of Characters, Communication Handle*)
	Receive Numeric Table	R-16	ReceiveNumTable(*Length, Start at Index, Of Table, Communication Handle*)
	Receive Pointer Table	R-17	ReceivePtrTable(*Length, Start at Index, Of Table, Communication Handle*)
	Receive String	R-19	ReceiveString(*Put In, Communication Handle*)
	Receive String Table	R-21	ReceiveStrTable(*Length, Start at Index, Of Table, Communication Handle*)
	Send Communication Handle Command	S-2	SendCommunicationHandleCommand(*Communication Handle, Command*)
	Set Communication Handle Value	S-15	SetCommunicationHandleValue(*Value, Communication Handle*)
	Set End-Of-Message Terminator	S-22	SetEndOfMessageTerminator (*Communication Handle, To Character*)
	Transfer N Characters	T-11	TransferNChars(*Destination Handle, Source Handle, Num Chars*)
	Transmit Character	T-13	TransmitChar(*Character, Communication Handle*)
	Transmit NewLine	T-14	TransmitNewLine(*Communication Handle*)
	Transmit Numeric Table	T-15	TransmitNumTable(*Length, Start at Index, Of Table, Communication Handle*)
	Transmit Pointer Table	T-16	TransmitPtrTable(*Length, Start at Index, Of Table, Communication Handle*)
	Transmit/Receive Mistic I/O Hex String	T-18	TransReceMisticIoHexStringWithCrc(*Hex String, On Port, Put Result in*)
	Transmit/Receive String	T-20	TransmitReceiveString(*String, Communication Handle, Put Result in*)
	Transmit String Table	T-23	TransmitStrTable(*Length, Start at Index, Of Table, Communication Handle*)
	Transmit String	T-22	TransmitString(*String, Communication Handle*)
Event/Reaction	Clear All Event Latches	C-19	ClearAllEventLatches(*On I/O Unit*)
	Clear Event Latch	C-23	ClearEventLatch(*On Event/Reaction*)
	Disable Scanning for All Events	D-17	DisableScanningForAllEvents(*On I/O Unit*)
	Disable Scanning for Event	D-18	DisableScanningForEvent(*Event/Reaction*)
	Disable Scanning of Event/Reaction Group	D-19	DisableScanningOfEventReactionGroup(*E/R Group*)
	Enable Scanning for All Events	E-13	EnableScanningForAllEvents(*On I/O Unit*)
	Enable Scanning for Event	E-14	EnableScanningForEvent(*Event/Reaction*)
	Enable Scanning of Event/Reaction Group	E-15	EnableScanningOfEventReactionGroup()
	Event Occurred?	E-21	HasEventOccurred(*Event/Reaction*)
	Event Occurring?	E-22	IsEventOccurring(*Event/Reaction*)
	Event/Reaction Communication Enabled?	E-23	IsEventReactionCommEnabled(*Event/Reaction*)
	Event Scanning Disabled?	E-25	IsEventScanningDisabled(*Event/Reaction*)
	Event Scanning Enabled?	E-26	IsEventScanningEnabled(*Event/Reaction*)
	Get & Clear Event Latches	G-19	GetClearEventLatches(*E/R Group*)
	Get Event Latches	G-54	GetEventLatches(*E/R Group*)
	Read Event/Reaction Hold Buffer	R-4	ReadEventReactionHoldBuffer(*Event/Reaction*)

	ioControl Command	See pg	OptoScript Equivalent (Arguments)
Simulation	Communication to All I/O Points Enabled?	C-33	`IsCommToAllIoPointsEnabled()`
	Communication To All I/O Units Enabled?	C-34	`IsCommToAllIoUnitsEnabled()`
	Disable Communication to All I/O Points	D-4	`DisableCommuncationToAllIoPoints()`
	Disable Communication to All I/O Units	D-5	`DisableCommunicationToAllIoUnits()`
	Disable Communication to Event/Reaction	D-6	`DisableCommunicationToEventReaction` (*Event/Reaction*)
	Disable Communication to I/O Unit	D-7	`DisableCommunicationToIoUnit(`*I/O Unit*`)`
	Disable Communication to Mistic PID Loop	D-9	`DisableCommunicationtoMisticPidLoop(`*PID Loop*`)`
	Disable Communication to PID Loop	D-10	`DisableCommunicationtoPidLoop(`*PID Loop*`)`
	Disable Communication to Point	D-11	`DisableCommunicationToPoint(`*Point*`)`
	Disable Event/Reaction Group	D-12	`DisableEventReactionGroup(`*E/R Group*`)`
	Enable Communication to All I/O Points	E-1	`EnableCommunicationToAllIoPoints()`
	Enable Communication to All I/O Units	E-2	`EnableCommunicationToAllIoUnits()`
	Enable Communication to Event/Reaction	E-3	`EnableCommunicationToEventReaction(`*Event/Reaction*`)`
	Enable Communication to I/O Unit	E-4	`EnableCommunicationToIoUnit(`*I/O Unit*`)`
	Enable Communication to Mistic PID Loop	E-5	`EnableCommunicationToMisticPidLoop(`*PID Loop*`)`
	Enable Communication to PID Loop	E-6	`EnableCommunicationtoPidLoop(`*PID Loop*`)`
	Enable Communication to Point	E-7	`EnableCommunicationToPoint(`*Point*`)`
	Enable Event/Reaction Group	E-8	`EnableEventReactionGroup(`*E/R Group*`)`
	Event/Reaction Communication Enabled?	E-23	`IsEventReactionCommEnabled (`*Event/Reaction*`)`
	Event/Reaction Group Communication Enabled?	E-24	`IsEventReactionGroupEnabled(`*E/R Group*`)`
	I/O Point Communication Enabled?	I-2	`IsIoPointCommEnabled(`*I/O Point*`)`
	I/O Unit Communication Enabled?	I-3	`IsIoUnitCommEnabled(`*I/O Unit*`)`
	IVAL Set Analog Point	I-6	`IvalSetAnalogPoint(`*To, On Point*`)`
	IVAL Set Counter	I-7	`IvalSetCounter(`*To, On Point*`)`
	IVAL Set I/O Unit from MOMO Masks	I-8	`SetIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Digital I/O Unit*`)`
	IVAL Set Frequency	I-10	`IvalSetFrequency(`*To, On Point*`)`
	IVAL Set Mistic PID Control Word	I-13	`IvalSetPidControlWord(`*On Mask, Off Mask, For PID Loop*`)`
	IVAL Set Mistic PID Process Term	I-14	`IvalSetMisticPidProcessTerm(`*To, On PID Loop*`)`
	IVAL Set Off-Latch	I-18	`IvalSetOffLatch(`*To, On Point*`)`
	IVAL Set Off-Pulse	I-19	`IvalSetOffPulse(`*To, On Point*`)`
	IVAL Set Off-Totalizer	I-20	`IvalSetOffTotalizer(`*To, On Point*`)`
	IVAL Set On-Latch	I-21	`IvalSetOnLatch(`*To, On Point*`)`
	IVAL Set On-Pulse	I-22	`IvalSetOnPulse(`*To, On Point*`)`
	IVAL Set On-Totalizer	I-23	`IvalSetOnTotalizer(`*To, On Point*`)`
	IVAL Set Period	I-24	`IvalSetPeriod(`*To, On Point*`)`
	IVAL Set TPO Percent	I-26	`IvalSetTpoPercent(`*To, On Point*`)`
	IVAL Set TPO Period	I-27	`IvalSetTpoPeriod(`*Value, On Point*`)`
	IVAL Turn Off	I-28	`IvalTurnOff(`*Point*`)`
	IVAL Turn On	I-29	`IvalTurnOn(`*Point*`)`
	Mistic PID Loop Communication Enabled?	M-4	`IsMisticPidLoopCommEnabled(`*PID Loop*`)`
	PID Loop Communication Enabled?	P-2	`IsPidLoopCommEnabled(`*PID Loop*`)`
Pointer	Clear Pointer	C-28	`pn1 = null;`
	Clear Pointer Table Element	C-28	`pt[0] = null;`
	Get Pointer From Name	G-134	`GetPointerFromName(`*Name, Pointer*`)`
	Move from Pointer Table Element	M-9	`pn = pt[0];`
	Move to Pointer	M-19	`pn = &n;`
	Move to Pointer Table Element	M-21	`pt[0] = &n;`
	Pointer Equal to Null?	P-3	`pn == null`
	Pointer Table Element Equal to Null?	P-4	`pt[0] == null`

ioControl Command	See pg	OptoScript Equivalent (Arguments)
Mathematical		
Absolute Value	A-1	`AbsoluteValue(`*Of*`)`
Add	A-3	`x + y`
Arccosine	A-11	`Arccosine(`*Of*`)`
Arcsine	A-12	`Arcsine(`*Of*`)`
Arctangent	A-13	`Arctangent(`*Of*`)`
Clamp Float Table Element	C-12	`ClampFloatTableElement(`*High Limit, Low Limit, Element Index, Of Float Table*`)`
Clamp Float Variable	C-13	`ClampFloatVariable(`*High Limit, Low Limit, Float Variable*`)`
Clamp Integer 32 Table Element	C-14	`ClampInt32TableElement(`*High Limit, Low Limit, Element Index, Of Integer 32 Table*`)`
Clamp Integer 32 Variable	C-15	`ClampInt32Variable(`*High Limit, Low Limit, Integer 32 Variable*`)`
Complement	C-36	`-x`
Cosine	C-62	`Cosine(`*Of*`)`
Decrement Variable	D-1	`DecrementVariable(`*Variable*`)`
Divide	D-20	`x / y`
Generate Random Number	G-6	`GenerateRandomNumber()`
Hyperbolic Cosine	H-1	`HyperbolicCosine(`*Of*`)`
Hyperbolic Sine	H-2	`HyperbolicSine(`*Of*`)`
Hyperbolic Tangent	H-3	`HyperbolicTangent(`*Of*`)`
Increment Variable	I-1	`IncrementVariable(`*Variable*`)`
Maximum	M-2	`Max(`*Compare, With*`)`
Minimum	M-3	`Min(`*Compare, With*`)`
Modulo	M-5	`x % y`
Multiply	M-25	`x * y`
Natural Log	N-1	`NaturalLog(`*Of*`)`
Raise e to Power	R-1	`RaiseEToPower(`*Exponent*`)`
Raise to Power	R-2	`Power(`*Raise, To the*`)`
Round	R-24	`Round(`*Value*`)`
Seed Random Number	S-4	`SeedRandomNumber()`
Sine	S-91	`Sine(`*Of*`)`
Square Root	S-92	`SquareRoot(`*Of*`)`
Subtract	S-105	`x - y`
Tangent	T-1	`Tangent(`*Of*`)`
Truncate	T-24	`Truncate(`*Value*`)`
String		
Append Character to String	A-9	`s1 += 'a';`
Append String to String	A-10	`s1 += s2;`
Compare Strings	C-35	`CompareStrings(`*String 1, String 2*`)`
Convert Float to String	C-40	`FloatToString(`*Convert, Length, Decimals, Put Result in*`)`
Convert Hex String to Number	C-41	`HexStringToNumber(`*Convert*`)`
Convert IEEE Hex String to Number	C-42	`IEEEHexStringToNumber(`*Convert*`)`
Convert Integer 32 to IP Address String	C-43	`Int32ToIpAddressString(`*Convert, Put Result In*`)`
Convert IP Address String to Integer 32	C-44	`IpAddressStringToInt32(`*Convert*`)`
Convert Mistic I/O Hex String to Float	C-45	`MisticIoHexToFloat(`*Convert*`)`
Convert Number to Formatted Hex String	C-46	`NumberToFormattedHexString(`*Convert, Length, Put Result in*`)`
Convert Number to Hex String	C-48	`NumberToHexString(`*Convert, Put Result in*`)`
Convert Number to Mistic I/O Hex String	C-49	`NumberToMisticIoHex(`*Convert, Put Result in*`)`
Convert Number to String	C-50	`NumberToString(`*Convert, Put Result in*`)`
Convert Number to String Field	C-51	`NumberToStringField(`*Convert, Length, Put Result in*`)`
Convert String to Float	C-52	`StringToFloat(`*Convert*`)`
Convert String to Integer 32	C-54	`StringToInt32(`*Convert*`)`
Convert String to Integer 64	C-55	`StringToInt64(`*Convert*`)`
Convert String to Lower Case	C-56	`StringToLowerCase(`*Convert*`)`
Convert String to Upper Case	C-57	`StringToUpperCase(`*Convert*`)`
Find Character in String	F-1	`FindCharacterInString(`*Find, Start at Index, Of String*`)`
Find Substring in String	F-2	`FindSubstringInString(`*Find, Start at Index, Of String*`)`

ioControl Command	See pg	OptoScript Equivalent (Arguments)
Generate Checksum on String	G-1	`GenerateChecksumOnString(`*Start Value, On String*`)`
Generate Forward CCITT on String	G-3	`GenerateForwardCcittOnString(`*Start Value, On String*`)`
Generate Forward CRC-16 on String	G-4	`GenerateForwardCrc16OnString(`*Start Value, On String*`)`
Generate Reverse CCITT on String	G-7	`GenerateReverseCcittOnString(`*Start Value, On String*`)`
Generate Reverse CRC-16 on String	G-8	`GenerateReverseCrc16OnString(`*Start Value, On String*`)`
Get Nth Character	G-100	`GetNthCharacter(`*From String, Index*`)`
Get String Length	G-138	`GetStringLength(`*Of String*`)`
Get Substring	G-139	`GetSubstring(`*From String, Start at Index, Num. Characters, Put Result in*`)`
Move from String Table Element	M-10	`s = st[0];`
Move String	M-16	`s1 = s2;`
Move to String Table Element	M-23	`st[0] = s;`
Move to String Table Elements	M-24	`MoveToStrTableElements(`*From, Start Index, End Index, Of Table*`)`
Set Nth Character	S-58	`SetNthCharacter(`*To, In String, At Index*`)`
String Equal?	S-103	`s1 == s2`
String Equal to String Table Element?	S-104	`s == st[0]`
Test Equal Strings	T-3	*See* String Equal?
Verify Checksum on String	V-3	`VerifyChecksumOnString(`*Start Value, On String*`)`
Verify Forward CCITT on String	V-4	`VerifyForwardCcittOnString(`*Start Value, On String*`)`
Verify Forward CRC-16 on String	V-5	`VerifyForwardCrc16OnString(`*Start Value, On String*`)`
Verify Reverse CCITT on String	V-6	`VerifyReverseCcittOnString(`*Start Value, On String*`)`
Verify Reverse CRC-16 on String	V-7	`VerifyReverseCrc16OnString(`*Start Value, On String*`)`

PID—Ethernet

ioControl Command	See pg	OptoScript Equivalent (Arguments)
Get PID Configuration Flags	G-112	`GetPidConfigFlags(`*PID Loop*`)`
Get PID Current Input	G-113	`GetPidCurrentInput(`*PID Loop*`)`
Get PID Current Setpoint	G-114	`GetPidCurrentSetpoint(`*PID Loop*`)`
Get PID Feed Forward	G-115	`GetPidFeedForward(`*PID Loop*`)`
Get PID Feed Forward Gain	G-116	`GetPidFeedForwardGain(`*PID Loop*`)`
Get PID Forced Output When Input Over Range	G-117	`GetPidForcedOutputWhenInputOverRange(`*PID Loop*`)`
Get PID Forced Output When Input Under Range	G-118	`GetPidForcedOutputWhenInputUnderRange(`*PID Loop*`)`
Get PID Gain	G-119	`GetPidGain(`*PID Loop*`)`
Get PID Input	G-120	`GetPidInput(`*PID Loop*`)`
Get PID Input High Range	G-121	`GetPidInputHighRange(`*PID Loop*`)`
Get PID Input Low Range	G-122	`GetPidInputLowRange(`*PID Loop*`)`
Get PID Max Output Change	G-123	`GetPidMaxOutputChange(`*PID Loop*`)`
Get PID Min Output Change	G-124	`GetPidMinOutputChange(`*PID Loop*`)`
Get PID Mode	G-125	`GetPidMode(`*PID Loop*`)`
Get PID Output	G-126	`GetPidOutput(`*PID Loop*`)`
Get PID Output High Clamp	G-127	`GetPidOutputHighClamp(`*PID Loop*`)`
Get PID Output Low Clamp	G-128	`GetPidOutputLowClamp(`*PID Loop*`)`
Get PID Scan Time	G-129	`GetPidScanTime(`*PID Loop*`)`
Get PID Setpoint	G-130	`GetPidSetpoint(`*PID Loop*`)`
Get PID Status Flags	G-131	`GetPidStatusFlags(`*PID Loop*`)`
Get PID Tune Derivative	G-132	`GetPidTuneDerivative(`*PID Loop*`)`
Get PID Tune Integral	G-133	`GetPidTuneIntegral(`*PID Loop*`)`
Set PID Configuration Flags	S-59	`SetPidConfigFlags(`*PID Loop, Configuration Flags*`)`
Set PID Feed Forward	S-60	`SetPidFeedForward(`*PID Loop, Feed Forward*`)`
Set PID Feed Forward Gain	S-61	`SetPidFeedForwardGain(`*PID Loop, Feed Fwd Gain*`)`
Set PID Forced Output When Input Over Range	S-62	`SetPidForcedOutputWhenInputOverRange(`*PID Loop, Forced Output*`)`
Set PID Forced Output When Input Under Range	S-63	`SetPidForcedOutputWhenInputUnderRange(`*PID Loop, Forced Output*`)`
Set PID Gain	S-64	`SetPidGain(`*PID Loop, Gain*`)`
Set PID Input	S-65	`SetPidInput(`*PID Loop, Input*`)`
Set PID Input High Range	S-66	`SetPidInputHighRange(`*PID Loop, High Range*`)`

ioControl Command	See pg	OptoScript Equivalent (Arguments)
Set PID Input Low Range	S-67	`SetPidInputLowRange(`*PID Loop, Low Range*`)`
Set PID Max Output Change	S-68	`SetPidMaxOutputChange(`*PID Loop, Max Change*`)`
Set PID Min Output Change	S-69	`SetPidMinOutputChange(`*PID Loop, Min Change*`)`
Set PID Mode	S-70	`SetPidMode(`*PID Loop, Mode*`)`
Set PID Output	S-71	`SetPidOutput(`*PID Loop, Output*`)`
Set PID Output High Clamp	S-72	`SetPidOutputHighClamp(`*PID Loop, High Clamp*`)`
Set PID Output Low Clamp	S-73	`SetPidOutputLowClamp(`*PID Loop, Low Clamp*`)`
Set PID Scan Time	S-74	`SetPidScanTime(`*PID Loop, Scan Time*`)`
Set PID Setpoint	S-75	`SetPidSetpoint(`*PID Loop, Setpoint*`)`
Set PID Tune Derivative	S-76	`SetPidTuneDerivative(`*PID Loop, Derivative*`)`
Set PID Tune Integral	S-77	`SetPidTuneIntegral(`*PID Loop, Integral*`)`

Logical

AND	A-7	`x and y`
AND?	A-8	*See* AND
Bit AND	B-1	`x bitand y`
Bit AND?	B-3	*See* Bit AND
Bit Clear	B-4	`BitClear(`*Item, Bit to Clear*`)`
Bit NOT	B-5	`bitnot x`
Bit NOT?	B-7	*See* Bit NOT
Bit Off?	B-8	`IsBitOff(`*In, Bit*`)`
Bit On?	B-9	`IsBitOn(`*In, Bit*`)`
Bit OR	B-10	`x bitor y`
Bit OR?	B-12	*See* Bit OR
Bit Rotate	B-13	`BitRotate(`*Item, Count*`)`
Bit Set	B-14	`BitSet(`*Item, Bit to Set*`)`
Bit Shift	B-15	`x << nBitsToShift`
Bit Test	B-17	`BitTest(`*Item, Bit to Test*`)`
Bit XOR	B-18	`x bitxor y`
Bit XOR?	B-20	*See* Bit XOR
Equal?	E-16	`x == y`
Equal to Numeric Table Element?	E-17	`n == nt[0]`
Get High Bits of Integer 64	G-62	`GetHighBitsOfInt64(`*High Bits From*`)`
Get Low Bits of Integer 64	G-85	`GetLowBitsOfInt64(`*Integer 64*`)`
Greater?	G-146	`x > y`
Greater Than Numeric Table Element?	G-147	`x > nt[0]`
Greater Than or Equal?	G-148	`x >= y`
Greater Than or Equal to Numeric Table Element?	G-149	`x >= nt[0]`
Less?	L-1	`x < y`
Less Than Numeric Table Element?	L-2	`x < nt[0]`
Less Than or Equal?	L-3	`x <= y`
Less Than or Equal to Numeric Table Element?	L-4	`x <= nt[0]`
Make Integer 64	M-1	`MakeInt64(`*High Integer, Low Integer*`)`
Move 32 Bits	M-7	`Move32Bits(`*From, To*`)`
NOT	N-2	`not x`
NOT?	N-3	`not x`
Not Equal?	N-4	`x <> y`
Not Equal to Numeric Table Element?	N-5	`n <> nt[0]`
Numeric Table Element Bit Clear	N-6	`NumTableElementBitClear(`*Element Index, Of Integer Table, Bit to Clear*`)`
Numeric Table Element Bit Set	N-7	`NumTableElementBitSet(`*Element Index, Of Integer Table, Bit to Set*`)`
Numeric Table Element Bit Test	N-8	`NumTableElementBitTest(`*Element Index, Of Integer Table, Bit to Test*`)`
OR	O-6	`x or y`
OR?	O-7	*See* OR
Set Variable False	S-87	`SetVariableFalse(`*Variable*`)`
Set Variable True	S-88	`SetVariableTrue(`*Variable*`)`
Test Equal	T-2	*See* Equal?
Test Greater	T-4	*See* Greater?
Test Greater or Equal	T-5	*See* Greater Than or Equal?

ioControl Command	See pg	OptoScript Equivalent (Arguments)
Test Less	T-6	*See* Less?
Test Less or Equal	T-7	*See* Less Than or Equal?
Test Not Equal	T-8	*See* Not Equal?
Test Within Limits	T-9	*See* Within Limits?
Variable False?	V-1	`IsVariableFalse(`*`Variable`*`)`
Variable True?	V-2	`IsVariableTrue(`*`Variable`*`)`
Within Limits?	W-1	`IsWithinLimits(`*`Value, Low Limit, High Limit`*`)`
XOR	X-1	`x xor y`
XOR?	X-2	*See* XOR

High Density Digital

Clear HDD Module Off-Latches	C-24	`ClearHddModuleOffLatches(`*`I/O Unit, Module Number, Clear Mask`*`)`
Clear HDD Module On-Latches	C-25	`ClearHddModuleOnLatches(`*`I/O Unit, Module Number, Clear Mask`*`)`
Get & Clear All HDD Module Off-Latches	G-10	`GetClearAllHddModuleOffLatches(`*`I/O Unit, Start Index, Put Result In`*`)`
Get & Clear All HDD Module On-Latches	G-12	`GetClearAllHddModuleOnLatches(`*`I/O Unit, Start Index, Put Result In`*`)`
Get & Clear HDD Module Counter	G-20	`GetClearHddModuleCounter(`*`I/O Unit, Module Number, Point Number, Put Result In`*`)`
Get & Clear HDD Module Counters	G-21	`GetClearHddModuleCounters(`*`I/O Unit, Module Number, Start Table Index, Put Result In`*`)`
Get & Clear HDD Module Off-Latches	G-22	`GetClearHddModuleOffLatches(`*`I/O Unit, Module Number, Put Result In`*`)`
Get & Clear HDD Module On-Latches	G-24	`GetClearHddModuleOnLatches(`*`I/O Unit, Module Number, Put Result In`*`)`
Get All HDD Module Off-Latches	G-32	`GetAllHddModuleOffLatches(`*`I/O Unit, Start Index, Put Result In`*`)`
Get All HDD Module On-Latches	G-26	`GetAllHddModuleOnLatches(`*`I/O Unit, Start Index, Put Result In`*`)`
Get All HDD Module States	G-36	`GetAllHddModuleStates(`*`I/O Unit, Start Index, Put Result In`*`)`
Get HDD Module Counters	G-57	`GetHddModuleCounters(`*`I/O Unit, Module Number, Start Table Index, Put Result In`*`)`
Get HDD Module Off-Latches	G-58	`GetHddModuleOffLatches(`*`I/O Unit, Module Number, Put Result In`*`)`
Get HDD Module On-Latches	G-60	`GetHddModuleOnLatches(`*`I/O Unit, Module Number, Put Result In`*`)`
Get HDD Module States	G-61	`GetHddModuleStates(`*`I/O Unit, Module Number, Put Result In`*`)`
Set HDD Module from MOMO Masks	S-23	`SetHddModulefromMOMOMasks(`*`I/O Unit, Module Number, Must-On Mask, Must-Off Mask`*`)`
Turn Off HDD Module Point	T-26	`TurnOffHDDModulePoint(`*`I/O Unit, Module Number, Point Number`*`)`
Turn On HDD Module Point	T-28	`TurnOnHddModulePoint(`*`I/O Unit, Module Number, Point Number`*`)`

Absolute Value

Mathematical Action

Function: To ensure that a value is positive.

Typical Use: To ensure a positive value when the result of a computation may be negative.

Details: Copies *Argument 1* to *Argument 2*, dropping the minus sign if it exists.

Arguments:

Argument 1 **Of**	Argument 2 **Put Result in**
Analog Input	Analog Output
Analog Output	Float Variable
Float Variable	Integer 32 Variable
Integer 32 Variable	Integer 64 Variable
Integer 64 Variable	

Standard Example:

Absolute Value

Of	Negative_Value	*Float Variable*
Put Result in	Positive_Value	*Float Variable*

OptoScript Example:

`AbsoluteValue(`*Of***`)`**

```
Positive_Value = AbsoluteValue(Negative_Value);
```

This is a function command; it returns the positive value. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- To change a negative value to a positive value, make *Argument 1* and *Argument 2* the same.

See Also: Complement (page C-36)

Accept Incoming Communication

Communication Action

Function: In TCP/IP communication, to establish a connection. (In this case the control engine acts as the slave, and the communication is opened by the master.)

Typical Use: To accept an incoming communication.

Details:

- Applies to communication via TCP communication handles only.
- Always use Listen for Incoming Communication once on each port to start the process before using this command to complete it. If you don't use the listen command first, you'll receive a -441 (Could not listen on socket) error.

Arguments:

Argument 1	**Argument 2**
Communication Handle	**Put Result In**
Communication Handle	Float Variable
	Integer 32 Variable

Standard Example:

Accept Incoming Communication

Communication Handle	Ultimate_A	*Communication Handle*
Put Result In	STATUS	*Integer 32 Variable*

OptoScript Example:

`AcceptIncomingCommunication(`*Communication Handle***`)`**

```
STATUS = AcceptIncomingCommunication(Ultimate_A);
```

This is a function command; it returns one of the status codes listed below. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide.*
- It is only necessary to use the Listen for Incoming Communication command once per port, even if you use the Accept command several times.
- For those familiar with sockets programming, ioControl uses a limited sockets implementation. To use this command again, create another communication handle and Accept Incoming Communication on the new handle.
- The session may be closed by the master. To determine whether the session is still open, use the commands Get Number of Characters Waiting, Communication Open? or Receive String. (Get Number of Characters Waiting is the best method.)

Status Codes:

0 = Success

-10 = Invalid port number. Check format of serial port in the communication handle string.

-36 = Invalid command. Use this command only with a TCP communication handle; for other communication handles, use Open Outgoing Communication instead.

-47 = Open failed. Handle has already been opened.

-203 = Unknown driver on communication handle.

-441 = Could not listen on socket.

-442 = Could not accept on socket. No devices are currently attempting to connect on this port.

See Also: Listen for Incoming Communication (page L-5), Get Number of Characters Waiting (page G-101), Receive String (page R-19), Open Outgoing Communication (page O-4), Communication Open? (page C-32)

Add

Mathematical Action

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:

- The standard ioControl command adds *Argument 1* and *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.
- Accommodates different item types such as float, integer, and analog without restriction.

Arguments:

Argument 1 **[Value]**	**Argument 2** **Plus**	**Argument 3** **Put Result In**
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `+` operator.

```
Total_Weight = Ingredient_1_Weight + Ingredient_2_Weight;
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `+` operator has many uses. For more information on mathematical expressions in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.

Queue Errors: -13 = Overflow error—result too large.

See Also: Increment Variable (page I-1), Subtract (page S-105)

Add Message to Queue

Error Handling Action

Function: To place your own message into the message queue.

Typical Use: To add diagnostic or debugging messages to the queue.

Details:

- Valid severity values are:

  4 = Info
  8 = Warning
  16 = Error

- The queue holds a total of 1000 errors and messages.
- Quotes (“”) are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1	Argument 2
Severity	**Message**
Integer 32 Literal	String Literal
Integer 32 Variable	String Variable

Standard Example: This example shows the string in quotes for clarity only; do not use quotes in standard commands.

Add Message to Queue

Severity	16	*Integer 32 Literal*
Message	"Pressure Tank Exploded"	*String Literal*

OptoScript Example: **`AddMessageToQueue(`***Severity, Message***`)`**

```
AddMessageToQueue(16, "Pressure Tank Exploded");
```

This is a procedure command; it does not return a value.

Queue Error: -83 = Invalid severity value

See Also: Add User Error to Queue (page A-5)

Add User Error to Queue

Error Handling Action

Function: Enables the user to force a program error into the message queue.

Typical Use: Simulating errors offline to test a user-written error handler.

Details:
- Adds a user-defined error number to the message queue. Any number from -22001 to -23000 may be used for this purpose.
- The queue holds a total of 1000 errors and messages.

Arguments:

Argument 1
Error Number
Integer 32 Literal
Integer 32 Variable

Standard Example:

Add User Error to Queue

Error Number	-22001	*Integer 32 Literal*

OptoScript Example:

`AddUserErrorToQueue(`*Error Number***`)`**

```
AddUserErrorToQueue(-22001);
```

This is a procedure command; it does not return a value.

Notes: Also see Add Message to Queue, which is more flexible.

See Also: Add Message to Queue (page A-4), Add User I/O Unit Error to Queue (page A-6), Get Error Code of Current Error (page G-52)

Add User I/O Unit Error to Queue

Error Handling Action

Function: Enables the user to force an I/O unit error into the message queue.

Typical Use: Simulating I/O unit errors offline to test a user-written error handler.

Details:
- Adds a standard predefined I/O unit error number to the message queue.
- The queue holds a total of 1000 errors and messages.

Arguments:

Argument 1	**Argument 2**
Error Number	**I/O Unit**
Integer 32 Literal	B100*
Integer 32 Variable	B200*
	B3000 (Analog)*
	B3000 (Digital)*
	G4A8R, G4RAX*
	G4D16R*
	G4D32RS*
	SNAP-ENET-D64*
	SNAP-UP1-D64
	SNAP-UP1-M64
	SNAP-ENET-S64
	SNAP-B3000-ENET, SNAP-ENET-RTC
	SNAP-UP1-ADS
	SNAP-PAC-R1
	SNAP-PAC-R2
	SNAP-BRS*

* ioControl Professional only

Standard Example:

Add User I/O Unit Error to Queue

Error Number	-52	*Integer 32 Literal*
I/O Unit	My_UIO	*SNAP-UP1-ADS*

OptoScript Example:

`AddUserIoUnitErrorToQueue(`*Error Number, I/O Unit***`)`**

```
AddUserIoUnitErrorToQueue(-52, My_UIO);
```

This is a procedure command; it does not return a value.

Notes: See the Error Codes appendix in the *ioControl User's Guide* for a complete list.

See Also: Add User Error to Queue (page A-5), Get Error Code of Current Error (page G-52)

AND

Logical Action

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:

- The standard ioControl command performs a logical AND on *Argument 1* and *Argument 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 True (non-zero) if both values are non-zero, False (0) otherwise.
- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 **[Value]**	**Argument 2** **With**	**Argument 3** **Put Result in**
Digital Input	Digital Input	Digital Output
Digital Output	Digital Output	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

AND

	Limit_Switch1	*Digital Input*
With	Limit_Switch2	*Digital Input*
Put Result in	Both_Switches_Closed	*Integer Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `and` operator.

```
Both_Switches_Closed = Limit_Switch1 and Limit_Switch2;
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `and` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- It is advisable to use only integers or digital points with this command.
- In OptoScript code, you can combine logical operators and AND multiple variables, for example: `x = a and b and c and d;`
- In standard ioControl code, to AND multiple variables (such as A, B, C, and D) into one variable (such as ANSWER), do the following:
  1. AND A with B, Put Result in ANSWER.
  2. AND C with ANSWER, Put Result in ANSWER.
  3. AND D with ANSWER, Put Result in ANSWER.
- To test for individual bits, use Bit Test or Bit AND.

See Also: Bit Test (page B-17), Bit AND (page B-1), AND? (page A-8)

AND?

Logical Condition

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 *Argument 1* and *Argument 2*. Examples:

Argument 1	Argument 2	Result
0	0	0
1	0	0
0	1	0
1	1	1

- Evaluates True (non-zero) if both values are non-zero, False (0) otherwise.

Arguments:

Argument 1 **Is**	**Argument 2** **[Value]**
Digital Input	Digital Input
Digital Output	Digital Output
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable

Standard Example:

AND?	*Is*	Limit_Switch1	*Digital Input*
		Limit_Switch2	*Digital Input*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `and` operator.

```
if (Limit_Switch1 and Limit_Switch2) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `and` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- It is advisable to use only integers or digital points with this command.
- In OptoScript code, you can combine logical operators and AND multiple variables, for example: `if (a and b and c and d) then`
- In standard ioControl code, multiple values can be AND?ed 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? (page B-3) Variable True? (page V-2) Variable False? (page V-1)

Append Character to String

String Action

Function: To add a character to the end of a string variable.

Typical Use: To build strings consisting of non-printable or binary characters.

Details:

- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- The character is represented by an ASCII value. (See the ASCII table in Chapter 10 of the *ioControl User's Guide.)* 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; for example, -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**
Append	**To**
Float Literal	String Variable
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example: The following example appends a "!" to a string (for example, "Hello" would become "Hello!"):

Append Character to String

Append	33	*Integer 32 Literal*
To	Hello_String	*String Variable*

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 to String

Append	3	*Integer 32 Literal*
To	Command_String	*String Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `+=` operator and the `Chr` keyword. The OptoScript code for the first example above could be either of the following lines:

```
Hello_String += Chr(33);
Hello_String += Chr('!');
```

The OptoScript code for the second example would be:

```
Command_String += Chr(3);
```

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on using strings in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To clear a string, use Move String before using this command. Moving an empty string ("") to a string variable will clear it.

Dependencies: The string variable must be wide enough to hold one more character.

See Also: Append String to String (page A-10)

Append String to String

String Action

Function: To add a string to the end of another string variable.

Typical Use: To build strings.

Details:

- Quotes (“”) are used in OptoScript code, but not in standard ioControl code.
- If the string variable 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 to String). For example, to append a “space,” use the space bar rather than the number 32.

Arguments:

Argument 1	**Argument 2**
Append	**To**
String Literal	String Variable
String Variable	

Standard 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”). Quotes are shown here for clarity only; do not use them in the standard command.

Append String to String

Append	“ world”	*String Literal*
To	Hello_String	*String Variable*

OptoScript Example: OptoScript doesn’t use a command; the function is built in. Use the `+=` operator. Quotes are required in OptoScript code.

```
Hello_String += " world";
```

Notes:

- See “String Commands” in Chapter 10 of the *ioControl User’s Guide*.
- For more information on using strings in OptoScript code, see Chapter 11 of the *ioControl User’s Guide*. For example, in OptoScript, you can append several strings at once, as shown:

```
string1 = string2 + string3 + string4;
```

- To clear a string, use Move String before using this command. Moving an empty string (“”) to a string variable will clear it.

Dependencies: The string variable must be wide enough to hold the appended string.

See Also: Append Character to String (page A-9)

Arccosine

Mathematical Action

Function: To derive the angular value from a cosine value.

Typical Use: To solve trigonometric calculations.

Details:

- Calculates the arccosine 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 pi.
  (To convert radians to degrees, multiply by 180/pi.)

Arguments:

Argument 1 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Arccosine

Of	X	*Float Variable*
Put Result in	RADIANS	*Float Variable*

OptoScript Example:

`Arccosine(`*`Of`*`)`

```
RADIANS = Arccosine(X);
```

This is a function command, it returns the angular value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See “Mathematical Commands” in Chapter 10 of the *ioControl User's Guide*.
- Use Cosine if the angle is known and the cosine is desired.

Queue Errors:

-13 = Overflow error—result too large.

-14 = Not a number—result invalid.

See Also: Cosine (page C-62), Arcsine (page A-12), Arctangent (page A-13)

Arcsine

Mathematical Action

Function: To derive the angular value from a sine value.

Typical Use: To solve trigonometric calculations.

Details:

- Calculates the arcsine 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 –pi/2 to pi/2. (To convert radians to degrees, multiply by 180/pi.)

Arguments:

Argument 1 **Of**	Argument 2 **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Arcsine

Of	X	*Float Variable*
Put Result in	RADIANS	*Float Variable*

OptoScript Example:

`Arcsine(`*`Of`*`)`

```
RADIANS = Arcsine(X);
```

This is a function command, it returns the angular value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See “Mathematical Commands” in Chapter 10 of the *ioControl User's Guide*.
- Use Sine if the angle is known and the sine is desired.

Queue Errors:

-13 = Overflow error—result too large.

-14 = Not a number—result invalid.

See Also: Sine (page S-91), Arccosine (page A-11), Arctangent (page A-13)

Arctangent

Mathematical Action

Function: To derive the angular value from a tangent value.

Typical Use: To solve trigonometric calculations.

Details:
- Calculates the arctangent 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 –pi/2 to pi/2. (To convert radians to degrees, multiply by 180/pi.)

Arguments:

Argument 1 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Arctangent

Of	X	*Float Variable*
Put Result in	RADIANS	*Float Variable*

OptoScript Example:

`Arctangent(`*Of*`)`

```
RADIANS = Arctangent(X);
```

This is a function command, it returns the angular value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use Tangent if the angle is known and the tangent is desired.

Queue Errors: -13 = Overflow error—result too large.

-14 = Not a number—result invalid.

See Also: Arccosine (page A-11), Arcsine (page A-12), Tangent (page T-1)

Bit AND

Logical Action

Function: To perform a 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 *Argument 1* and *Argument 2* and puts result in *Argument 3*. One value is the mask for selecting specific bits in the other value. Examples:

Argument 1	Argument 2	Argument 3
0	0	0
8	0	0
0	8	0
8	8	8

- Acts on all bits.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
[Value]	**With**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal	Integer 64 Variable
Integer 64 Variable	Integer 64 Variable	SNAP-ENET-D64*
SNAP-ENET-D64*	SNAP-ENET-D64*	SNAP-UP1-D64*
SNAP-UP1-D64*	SNAP-UP1-D64*	SNAP-UP1-M64*
SNAP-UP1-M64*	SNAP-UP1-M64*	SNAP-ENET-S64*
SNAP-ENET-S64*	SNAP-ENET-S64*	
* Standard commands only	* Standard commands only	* Standard commands only

Standard 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	*Integer 32 Variable*
With	15	*Integer 32 Literal*
Put Result in	RESULT	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `bitand` operator.

```
RESULT = VALUE bitand 15;
```

Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. The following example ands the bits from two variables and writes the inverted result to an I/O unit:

```
SetDigital64IoUnitFromMomo(nnTemp1 bitand nnTemp2,
bitnot (nnTemp1 bitand nnTemp2),
Dig_IO_Unit);
```

This example moves a value from an I/O unit, ands the bits with a variable, and writes the inverted result to the same I/O unit:

```
nnTemp1 = GetIoUnitAsBinaryValue(Dig_IO_Unit);
nnTemp1 = nnTemp1 bitand nnVariable;
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, Dig_IO_Unit);
```

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User’s Guide*.
- 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 *Argument 1* and *Argument 3* the same.
- You may prefer to set a 1 for each bit to clear in the mask, then use Bit NOT to invert all 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 (page B-4), Bit NOT (page B-5), AND (page A-7), AND? (page A-8) Bit AND? (page B-3)

Bit AND?

Logical Condition

Function: To perform a 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 *Argument 1* and *Argument 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,* False otherwise.
- Acts on all bits.

Arguments:

Argument 1 Is	Argument 2 [Value]
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
SNAP-ENET-D64*	SNAP-ENET-D64*
SNAP-UP1-D64*	SNAP-UP1-D64*
SNAP-UP1-M64*	SNAP-UP1-M64*
SNAP-ENET-S64*	SNAP-ENET-S64*

* Standard commands only   * Standard commands only

Standard Example: This example reads the current state of all points on a digital I/O unit and Bit AND?s the value with the constant 33,280 (1000 0010 0000 0000 binary). Evaluates True if either point 15 or 9 is on, False if both points are off.

	Is	DIG_1	*SNAP-UP1-D64*
Bit AND?			
		33280	*Integer 32 Literal*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `bitand` operator. Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. In this example, the value of DIG_1 has been moved to a variable so it can be anded:

```
if (GetIoUnitAsBinaryValue(DIG_1) bitand 33280i64) then
```

The following is a simpler example; it ands the bits from two variables:

```
if (nVariable1 bitand nVariable2) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use 255 as the constant to check the lower eight points.

See Also: AND? (page A-8) Bit OR? (page B-12)

Bit Clear

Logical Action

Function: To clear a specified bit (set it to zero) in an allowable value.

Typical Use: To clear one bit of a particular integer variable.

Details:

- Performs this action on a *copy* of *Argument 1*, then moves the copy to *Argument 3*.
- For integer 32 variables, the valid range for the bit to clear is 0–31. For SNAP digital 64 I/O units and integer 64 variables, the valid range is 0–63.
- Note that the types for *Argument 2* are 32-bit integers, because an integer 32 provides enough range to handle either a 32- or a 64-bit shift.

Arguments:

Argument 1 **[Value]**	**Argument 2** **Bit to Clear**	**Argument 3** **Put Result in**
Integer 32 Variable	Integer 32 Literal	Integer 32 Variable
Integer 64 Variable	Integer 32 Variable	Integer 64 Variable
SNAP-ENET-D64		SNAP-ENET-D64*
SNAP-UP1-D64		SNAP-UP1-D64*
SNAP-UP1-M64		SNAP-UP1-M64*
SNAP-ENET-S64		SNAP-ENET-S64*

* Standard commands only

Standard Example: This example does a binary read of the I/O unit IO_UNIT_1, clears bit 0, and does a binary write of the data back out to IO_UNIT_1. This will cause point 0 of the I/O unit to be turned off. If point 0 happens to be an input, nothing will happen.

Bit Clear

	IO_UNIT_1	*SNAP-UP1-D64*
Bit to Clear	0	*Integer 64 Literal*
Put Result in	IO_UNIT_1	*SNAP-UP1-D64*

OptoScript Example: **BitClear(***Item, Bit to Clear***)**

```
nBitCleared = BitClear(IO_UNIT_1, 0);
```

This is a function command; it returns the value with the specified bit cleared. This example is different from the standard example, because in OptoScript the returned value cannot be an I/O unit.

To turn off a point as in the standard example, you could use the following OptoScript code:

```
SetDigital64IoUnitFromMomo(0, 1i64 << nPointToClear, IO_Unit_1);
```

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.
- 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, or fault (real-time or latch).
- To clear bits in *Argument 1*, make *Argument 1* and *Argument 3* the same.
- To clear several bits at once, use Bit AND.

See Also: Bit AND (page B-1), Bit Test (page B-17), Bit Set (page B-14)

Bit NOT

Logical Action

Function: To invert all 32 or 64 bits of a value.

Typical Use: To invert bits.

Details:

- Inverts *Argument 1* and puts result in *Argument 2*. Examples:

Argument 1	Argument 2
0	-1
-1 or 1	0

- Performs this action on a *copy* of *Argument 1*, then moves the copy to *Argument 2*.
- Acts on all bits.

Arguments:

Argument 1 [Value]	Argument 2 Put Result in
Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Variable
Integer 64 Variable	SNAP-ENET-D64*
SNAP-ENET-D64*	SNAP-UP1-D64*
SNAP-UP1-D64*	SNAP-UP1-M64*
SNAP-UP1-M64*	SNAP-ENET-S64*
SNAP-ENET-S64*	
* Standard commands only	* Standard commands only

Standard Example:

Bit NOT

	DATA	*Integer 32 Variable*
Put Result in	DATA	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `bitnot` operator.

```
DATA = bitnot DATA;
```

Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. This example moves a value from an I/O unit, bitnots the value, and writes the result to the same I/O unit:

```
nnTemp1 = GetIoUnitAsBinaryValue(Dig_IO_Unit);
SetDigital64IoUnitFromMomo(bitnot nnTemp1, nnTemp1, Dig_IO_Unit);
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To invert all bits in *Argument 1*, make both *Arguments* the same.

- To clear one or more specific bits, use this command to invert a mask set with the bits to be cleared. Then, Bit AND the mask with the value to clear those bits. For example, suppose you want to clear bits 0, 1, and 2.

Create a mask with those bits set	0000 0111
Do a bitnot on the mask, giving:	1111 1000
Bit AND this value with the value to be cleared:	0110 1001
Those bits are cleared:	0110 1000

- To toggle True/False, use NOT.

See Also: NOT (page N-2), Bit XOR (page B-18), XOR (page X-1), Bit Set (page B-14), Bit NOT? (page B-7)

Bit NOT?

Logical Condition

Function: To invert all 32 or 64 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 bits.

Arguments:

Argument 1
Is
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable
SNAP-ENET-D64*
SNAP-UP1-D64*
SNAP-UP1-M64*
SNAP-ENET-S64*

* Standard commands only

Standard Example: This example reads the state of all points of the specified digital I/O unit and then inverts them. Evaluates True if any point is off, False otherwise.

	Is	DIG_1	*SNAP-UP1-D64*
Bit NOT?			

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the bitnot operator. Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. In this example, the value of DIG_1 is moved to a variable so the bitnot operator can be used:

```
nnTemp1 = GetIoUnitAsBinaryValue(DIG_1);
if (bitnot nnTemp1) then
```

The following is a simpler example; it bitnots a variable:

```
if (bitnot nVariable2) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use NOT if the objective is to toggle the value between True and False.

See Also: Bit On? (page B-9) Bit Off? (page B-8)

Bit Off?

Logical Condition

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.
- Note that the types for *Argument 2* are 32-bit integers, because the top of the valid range, a value of 63, requires only 6 bits.

Arguments:

Argument 1 **In**	**Argument 2** **Bit**
Integer 32 Variable	Integer 32 Literal
Integer 64 Variable	Integer 32 Variable
SNAP-ENET-D64	
SNAP-UP1-D64	
SNAP-UP1-M64	
SNAP-ENET-S64	

Standard Example: This example evaluates to True if point 15 of I/O UNIT 1 is off, False otherwise.

	In	IO_UNIT_1	*SNAP-ENET-D64*
Bit Off?			
	Bit	15	*Integer 32 Literal*

OptoScript Example: **IsBitOff(***In*, *Bit***)**

```
if (IsBitOff(IO_UNIT_1, 15)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information on OptoScript.

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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, or fault (real-time or latch).
- Use Bit AND? if the objective is to test several bits at once.

See Also: Bit On? (page B-9) Bit AND? (page B-3) Bit Test (page B-17)

Bit On?

Logical Condition

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.
- Note that the types for *Argument 2* are 32-bit integers, because the top of the valid range, a value of 63, requires only 6 bits.

Arguments:

Argument 1 **In**	Argument 2 **Bit**
Integer 32 Variable	Integer 32 Literal
Integer 64 Variable	Integer 32 Variable
SNAP-ENET-D64	
SNAP-UP1-D64	
SNAP-UP1-M64	
SNAP-ENET-S64	

Standard Example: This example evaluates to True if point 0 of I/O UNIT 1 is on, False otherwise.

	In	IO_UNIT_1	*SNAP-ENET-D64*
Bit On?			
	Bit	0	*Integer 32 Literal*

OptoScript Example: **`IsBitOn(`***In, Bit***`)`**

```
if (IsBitOn(IO_UNIT_1, 0)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information on OptoScript.

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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, or fault (real-time or latch).
- Use Bit AND? if the objective is to test several bits at once.

See Also: Bit Off? (page B-8) Bit AND? (page B-3) Bit Test (page B-17)

Bit OR

Logical Action

Function: To perform a bitwise OR on two values.

Typical Use: To set one or more bits as specified by a mask.

Details:

- Performs a bitwise OR on *Argument 1* and *Argument 2* and puts result in *Argument 3*. Examples:

Argument 1	Argument 2	Argument 3
0	0	0
0xF	0	0xF
0	0xF	0xF
0xF	0xF	0xF

- Combines all bits set to 1 in *Argument 1* and *Argument 2*. The result (*Argument 3*) can be put into either of the first two items or into a different item.
- Acts on all bits.

Arguments:

Argument 1 [Value]	Argument 2 With	Argument 3 Put Result in
Integer 32 Literal	Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal	Integer 64 Variable
Integer 64 Variable	Integer 64 Variable	SNAP-ENET-D64*
SNAP-ENET-D64*	SNAP-ENET-D64*	SNAP-UP1-D64*
SNAP-UP1-D64*	SNAP-UP1-D64*	SNAP-UP1-M64*
SNAP-UP1-M64*	SNAP-UP1-M64*	SNAP-ENET-S64*
SNAP-ENET-S64*	SNAP-ENET-S64*	
* Standard commands only	* Standard commands only	* Standard commands only

Standard Example: This example sets bit 2 in a copy of *Argument 1* and puts the result in *Argument 3*.

Bit OR

	VALUE	*Integer 32 Variable*
With	4	*Integer 32 Literal*
Put Result in	RESULT	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `bitor` operator.

```
RESULT = VALUE bitor 4;
```

Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. The following example ors the bits from two variables and writes the result to an I/O unit:

```
SetDigital64IoUnitFromMomo(nnTemp1 bitor nnTemp2,
bitnot (nnTemp1 bitor nnTemp2),
Dig_IO_Unit);
```

This example moves a value from an I/O unit, ors the bits with a variable, and writes to the same I/O unit:

```
nnTemp1 = GetIoUnitAsBinaryValue(Dig_IO_Unit);
nnTemp1 = nnTemp1 bitor nVariable;
```

```
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, Dig_IO_Unit);
```

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User’s Guide*.
- 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, or fault (real-time or latch).
- To set bits in *Argument 1*, make *Argument 1* and *Argument 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 (page B-14), OR (page O-6), Bit XOR (page B-18), XOR (page X-1)

Bit OR?

Logical Condition

Function: To perform a 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 *Argument 1* and *Argument 2*. Examples:

Argument 1	Argument 2	Results
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 bits.

Arguments:

Argument 1 Is	Argument 2 [Value]
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
SNAP-ENET-D64*	SNAP-ENET-D64*
SNAP-UP1-D64*	SNAP-UP1-D64*
SNAP-UP1-M64*	SNAP-UP1-M64*
SNAP-ENET-S64*	SNAP-ENET-S64*
* Standard commands only	* Standard commands only

Standard Example:

Bit Or?	*Is*	Fault_Bits_1	*Integer 32 Variable*
		Fault_Bits_2	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `bitor` operator.

```
if (Fault_Bits_1 bitor Fault_Bits_2) then
```

Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code.

```
if (GetIoUnitAsBinaryValue(Dig_IO_Unit) bitor nInteger) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- 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, or fault (real-time or latch).
- Use Bit On? or Bit Off? if the objective is to test only one bit.

See Also: Bit On? (page B-9) Bit Off? (page B-8) OR? (page O-7)

Bit Rotate

Logical Action

Function: To rotate all 32 or 64 bits of an allowable value to the left or right.

Typical Use: To shift bits left or right with wraparound.

Details:

- Acts on all bits. All bits rotated past one end reappear at the other end. If *Argument 2* is positive, bits rotate left. If it is negative, bits rotate right. If it is zero, no rotation occurs.
- Note that the types for *Argument 2* are 32-bit integers, because an integer 32 provides enough range to handle either a 32- or a 64-bit shift.

Arguments:

Argument 1 [Value]	Argument 2 Count	Argument 3 Move To
Integer 32 Literal	Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
Integer 64 Literal		Integer 64 Variable
Integer 64 Variable		SNAP-ENET-D64*
SNAP-ENET-D64		SNAP-UP1-D64*
SNAP-UP1-D64		SNAP-UP1-M64*
SNAP-UP1-M64		SNAP-ENET-S64*
SNAP-ENET-S64		

* Standard commands only

Standard Example:

Bit Rotate

	Mask_Variable	*Integer 32 Variable*
Count	4	*Integer 32 Literal*
Move To	Result_Variable	*Integer 32 Variable*

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).

OptoScript Example:

`BitRotate(`*Item*, *Count***`)`**

```
Result_Variable = BitRotate(Mask_Variable, 4);
```

This is a function command; it returns the result of the bit rotation. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. In OptoScript code it cannot be consumed by an I/O unit, however. See Chapter 11 of the *ioControl User's Guide* for more information on OptoScript.

Although the returned value cannot be consumed by an I/O unit, you can accomplish the same thing by using OptoScript code such as the following:

```
nnTemp1 = BitRotate(Dig_IO_Unit, nCount);
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, Dig_IO_Unit);
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- To rotate bits in *Argument 1*, make *Argument 1* and *Argument 3* the same.
- To get rid of all bits that move past either end, use Bit Shift.

See Also: Bit Shift (page B-15)

Bit Set

Logical Action

Function: To set a specified bit (set it to 1) in an allowable value.

Typical Use: To set a bit in an integer variable.

Details:

- Performs this action on a *copy* of *Argument 1*, then moves the copy to *Argument 3*.
- Note that the types for *Argument 2* are 32-bit integers, because an integer 32 provides enough range to handle either a 32- or a 64-bit shift.

Arguments:

Argument 1 **[Value]**	Argument 2 **Bit to Set**	Argument 3 **Put Result in**
Integer 32 Variable	Integer 32 Literal	Integer 32 Variable
Integer 64 Variable	Integer 32 Variable	Integer 64 Variable
SNAP-ENET-D64		SNAP-ENET-D64*
SNAP-UP1-D64		SNAP-UP1-D64*
SNAP-UP1-M64		SNAP-UP1-M64*
SNAP-ENET-S64		SNAP-ENET-S64*

* Standard commands only

Standard Example:

Bit Set

	Pump3_Ctrl_Bits	*Integer 32 Variable*
Bit to Set	15	*Integer 32 Literal*
Put Result in	Pump3_Ctrl_Bits	*Integer 32 Variable*

If Pump3_Ctrl_Bits is 8 (00000000 00000000 00000000 00001000 binary), then after the Bit Set, Pump3_Ctrl_Bits would be 32776 (00000000 00000000 10000000 00001000 binary).

OptoScript Example:

BitSet(*Item, Bit to Set***)**

```
Pump3_Ctrl_Bits = BitSet(Pump3_Ctrl_Bits, 15);
```

This is a function command; it returns the value with the specified bit set. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. It cannot be consumed by an I/O unit, however. See Chapter 11 of the *ioControl User's Guide* for more information on OptoScript.

Although the returned value cannot be consumed by an I/O unit, you can accomplish the same thing by using OptoScript code such as the following:

```
SetDigital64IoUnitFromMomo(1i64 << nPointToSet, 0, Dig_IO_Unit);
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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, or fault (real-time or latch).
- To set bits in *Argument 1*, make *Argument 1* and *Argument 3* the same.
- To set several bits at once, use Bit OR.

See Also: Bit OR (page B-10), Bit Test (page B-17), Bit Clear (page B-4)

Bit Shift

Logical Action

Function: To shift the bits of a value to the right or left.

Typical Use: To evaluate the four bytes of a 32-bit integer or the eight bytes of a 64-bit integer one at a time. A way to multiply or divide integers by a base 2 number.

Details:

- Functionally equivalent to integer multiplication or division by powers of two. 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.
- In the standard ioControl command, 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.
- Acts on all bits. All bit positions vacated by the shift are filled with zeros.
- Note that the types for *Argument 2* are 32-bit integers, because an integer 32 provides enough range to handle either a 32- or a 64-bit shift.

Arguments:

Argument 1 **[Value]**	**Argument 2** **Count**	**Argument 3** **Put Result in**
Integer 32 Literal	Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
Integer 64 Literal		Integer 64 Variable
Integer 64 Variable		SNAP-ENET-D64*
SNAP-ENET-D64*		SNAP-UP1-D64*
SNAP-UP1-D64*		SNAP-UP1-M64*
SNAP-UP1-M64*		SNAP-ENET-S64*
SNAP-ENET-S64*		

* Standard commands only (Argument 1)

* Standard commands only (Argument 3)

Standard Example:

Bit Shift

	Mask_Variable	*Integer 32 Variable*
Count	-8	*Integer 32 Literal*
Put Result in	Result_Variable	*Integer 32 Variable*

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 -2,147,483,648 (10000000 00000000 00000000 00000000 binary), then after the shift Result_Variable would be 8,388,608 (00000000 10000000 00000000 00000000 binary).

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<<` (left shift) or `>>` (right shift) operators. Note that the result of the bit shift cannot be put into an I/O unit.

```
Result_Variable = Mask_Variable >> 8;
```

Although the result of the bit shift cannot be put into an I/O unit, you can accomplish the same thing by using OptoScript code. The following example shifts bits in a variable and writes the result to an I/O unit:

```
nnTemp1 = nnTemp1 >> 8;
```

```
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, Dig_IO_Unit);
```

This example moves a value from an I/O unit, shifts bits, and writes to the same I/O unit:

```
nnTemp1 = GetIoUnitAsBinaryValue(Dig_IO_Unit);
nnTemp1 = nnTemp1 >> 8;
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, Dig_IO_Unit);
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on logical operators such as `>>` and `<<` in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To shift bits in *Argument 1*, make *Argument 1* and *Argument 3* the same.
- To retain all bits that move past either end, use Bit Rotate.

See Also: Bit Rotate (page B-13)

Bit Test

Logical Action

Function: To determine the status of a specific bit.

Typical Use: To test a flag bit in an integer variable.

Details:

- Note that the types for *Argument 2* are 32-bit integers, because the top of the valid range, a value of 63, requires only 6 bits.
- If the bit is clear (0), False (0) is moved to *Argument 3*.
- If the bit is set (1), True (non-zero) is moved to *Argument 3*.
- The result can also be sent directly to a digital output.

Arguments:

Argument 1 [Value]	Argument 2 Bit to Test	Argument 3 Put Result in
Integer 32 Variable	Integer 32 Literal	Digital Output
Integer 64 Variable	Integer 32 Variable	Integer 32 Variable
SNAP-ENET-D64		
SNAP-UP1-D64		
SNAP-UP1-M64		
SNAP-ENET-S64		

Standard Example:

Bit Test

	Pump3_Ctrl_Bits	*Integer 32 Variable*
Bit to Test	15	*Integer 32 Literal*
Put Result in	Pump3_Ctrl_Bits	*Integer 32 Variable*

If Pump3_Ctrl_Bits is 00000000 00000000 10000000 00001000, the result would be set to True.

OptoScript Example:

`BitTest(`*Item, Bit to Test***`)`**

```
Pump3_Ctrl_Bits = BitTest(Pump3_Ctrl_Bits, 15);
```

This is a function command; it returns a value of False (0, bit is clear) or True (non-zero, bit is set). The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information on OptoScript.

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User's Guide*.
- 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, or fault (real-time or latch).
- To test several bits at once, use Bit AND.

See Also: Bit Clear (page B-4), Bit Set (page B-14), Bit On? (page B-9)

Bit XOR

Logical Action

Function: To perform a 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 *Argument 1* and *Argument 2* and puts the result in *Argument 3*. Examples:

Argument 1	Argument 2	Argument 3
0	0	0
0	1	1
1	0	1
1	1	0

- Acts on all bits. One value is the mask for selecting specific bits in the other value.

Arguments:

Argument 1 [Value]	Argument 2 With	Argument 3 Put Result in
Integer 32 Literal	Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal	Integer 64 Variable
Integer 64 Variable	Integer 64 Variable	SNAP-ENET-D64*
SNAP-ENET-D64*	SNAP-ENET-D64*	SNAP-UP1-D64*
SNAP-UP1-D64*	SNAP-UP1-D64*	SNAP-UP1-M64*
SNAP-UP1-M64*	SNAP-UP1-M64*	SNAP-ENET-S64*
SNAP-ENET-S64*	SNAP-ENET-S64*	
* Standard commands only	* Standard commands only	* Standard commands only

Standard Example:

Bit XOR

	Data	*Integer 32 Variable*
With	22	*Integer 32 Literal*
Put Result in	Data_New	*Integer 32 Variable*

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.

OptoScript Example:

OptoScript doesn't use a command; the function is built in. Use the `bitxor` operator.

```
Data_New = Data bitxor 22;
```

Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. The following example xors the bits from two variables and writes the result to an I/O unit:

```
SetDigital64IoUnitFromMomo(nnTemp1 bitxor nnTemp2,
bitnot(nnTemp1 bitxor nnTemp2),
Dig_IO_Unit);
```

This example moves a value from an I/O unit, xors the bits with a variable, and writes to the same I/O unit:

```
nnTemp1 = GetIoUnitAsBinaryValue(Dig_IO_Unit);
nnTemp1 = nnTemp1 bitxor nnVariable;
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, Dig_IO_Unit);
```

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User’s Guide*.
- 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, or fault (real-time or latch).
- To toggle bits in *Argument 1*, make *Argument 1* and *Argument 3* the same.
- To toggle a bit, Bit XOR with 1. Zero leaves the bit unchanged.

See Also: XOR (page X-1), Bit NOT (page B-5), NOT (page N-2), Bit XOR? (page B-20)

Bit XOR?

Logical Condition

Function: To determine the bitwise difference 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 *Argument 1* and *Argument 2*. Examples:

Argument 1	Argument 2	Result
0	0	False
0	1	True
1	0	True
1	1	False

- Evaluates True if the two allowable values are not equal, False if they are equal.
- Acts on all bits.
- Functionally equivalent to the Not Equal? condition when used with integer types.

Arguments:

Argument 1 **Is**	**Argument 2** **[Value]**
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
SNAP-ENET-D64*	SNAP-ENET-D64*
SNAP-UP1-D64*	SNAP-UP1-D64*
SNAP-UP1-M64*	SNAP-UP1-M64*
SNAP-ENET-S64*	SNAP-ENET-S64*
* Standard commands only	* Standard commands only

Standard Example:

Is	DIG_1	*SNAP-ENET-D64*
Bit XOR?		
	PREV_DIG_1	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `bitxor` operator. Note that for this command, I/O units cannot be used the same way as in the standard command. However, you can accomplish the same thing using OptoScript code. In this example, the value of DIG_1 is moved to a variable so the `bitxor` operator can be used:

```
if (GetIoUnitAsBinaryValue(DIG_1) bitxor PREV_DIG_1) then
```

The following is a simpler example; it bitxors two variables:

```
if (nVariable1 bitxor nVariable2) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- 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, or fault (real-time or latch).

- 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? (page E-16) Bit AND? (page B-3) Bit NOT (page B-5), Bit XOR (page B-18), Bit OR? (page B-12)

Calculate & Set Analog Gain

Analog Point Action

Function: To improve the accuracy of an analog input signal.

Typical Uses: To improve calibration on a temperature input.

Details:

- Reads the current value of a specified analog input and interprets it as the maximum (100 percent, 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 percent (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 flash memory at the I/O unit (recommended).
- The default gain value is 1.0. The valid range for gain is any floating point number.

Arguments:

Argument 1 **On Point**	**Argument 2** **Put Result in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Calculate & Set Analog Gain

On Point	Boiler_Temperature	*Analog Input*
Put Result in	Gain_Coefficient	*Float Variable*

OptoScript Example:

`CalcSetAnalogGain(`*On Point***`)`**

```
Gain_Coefficient = CalcSetAnalogGain(Boiler_Temperature);
```

This is a function command; it returns the calculated gain. The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Instead of using this command, it is recommended that you calibrate inputs when configuring I/O points in ioManager. See Opto 22 form 1440, the *ioManager User's Guide*, for instructions.

Dependencies:

- Always use Calculate & Set Analog Offset before using this command.
- Always set the analog input to the full-scale (100 percent) value before using this command.

See Also: Calculate & Set Analog Offset (page C-2), Set Analog Gain (page S-8), Set Analog Offset (page S-11)

Calculate & Set Analog Offset

Analog Point Action

Function: To improve accuracy of an analog input signal.

Typical Uses: To improve calibration on a temperature input.

Details:

- Reads the current value of a specified analog input and interprets it as the minimum (0 percent, 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 percent (zero scale).
- Stores the calculated offset in *Argument 2* for subsequent use by Set Analog Offset.
- 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 flash memory at the I/O unit (recommended).

Arguments:

Argument 1 **On Point**	Argument 2 **Put Result in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Calculate & Set Analog Offset

On Point	Boiler_Temperature	*Analog Input*
Put Result in	OFFSET	*Integer 32 Variable*

OptoScript Example:

`CalcSetAnalogOffset(`*On Point***`)`**

```
OFFSET = CalcSetAnalogOffset(Boiler_Temperature);
```

This is a function command; it returns the calculated offset. The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This command is intended to be used in conjunction with Calculate & Set Analog Gain.
- Instead of using this command, it is recommended that you calibrate inputs when configuring I/O points in ioManager. See Opto 22 form 1440, the *ioManager User's Guide*, for instructions.

See Also: Calculate & Set Analog Gain (page C-1), Set Analog Gain (page S-8), Set Analog Offset (page S-11)

Calculate Strategy CRC

Control Engine Action

Function: Calculates and returns a 16-bit CRC on the program in RAM.

Typical Use: Periodically used in an error handler to check the integrity 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
Put Result in
Integer 32 Variable

Standard Example:

Calculate Strategy CRC

Put Result in	New_CRC_Calc	*Integer 32 Variable*

OptoScript Example:

`CalcStrategyCrc()`

```
New_CRC-Calc = CalcStrategyCrc();
```

This is a function command; it returns the 16-bit CRC. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: This command could take several minutes to execute when seven tasks are running and the program is very large. Therefore, do not use it in a chart where timing is critical.

See Also: Retrieve Strategy CRC (page R-23)

Call Chart

Chart Action

Function: Starts another chart and immediately suspends the calling chart. Automatically continues the calling chart when the called chart ends.

Typical Use: Allows a main or "executive" chart to easily orchestrate the execution of other charts that typically have a dedicated function, thereby reducing the total number of charts running concurrently.

Details:

- This command is functionally a combination of three other commands, Start Chart, Suspend Chart, and Continue Calling Chart. It attempts to start the specified chart and if successful, suspends the chart that issued the command. There is no need to check the returned status if it's known that the called chart is stopped and that there is room in the task queue for another chart. When the called chart finishes, the calling chart automatically continues.
- The status variable indicates success (0) or failure (error code -5 if the task is already running).

Arguments:

Argument 1 **Chart**	Argument 2 **Put Status in**
Chart	Float Variable Integer 32 Variable

Standard Example:

Call Chart

Chart	Tank_Monitor	*Chart*
Put Status in	Call_Status	*Integer 32 Variable*

OptoScript Example:

`CallChart(`*Chart***`)`**

```
Call_Status = CallChart(Tank_Monitor);
```

This is a function command; it returns a zero (indicating success) or an error (indicating failure). The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This command should be used judiciously. It can take up to 100 ms for the called chart to start. Once the called chart has completed its logic, it can take another 100 ms to resume the calling chart. Use this command only when timing is not critical. Otherwise, instead of Call Chart, use a chart that runs continuously and uses subroutines for any kind of repetitive logic.
- Typically used to chain charts so that they run sequentially rather than concurrently.
- Can be used by concurrently running charts calling a sub-chart that performs a common function. For this use, the status must be checked to ensure success.

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

See Also: Continue Calling Chart (page C-37), Start Chart (page S-93), Suspend Chart (page S-106)

Calling Chart Running?

Chart Condition

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.

Standard Example: **Calling Chart Running?**

OptoScript Example: **`IsCallingChartRunning()`**

```
Chart_Status = IsCallingChartRunning();
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Continue Calling Chart (page C-37), Calling Chart Suspended? (page C-6) Calling Chart Stopped? (page C-5)

Calling Chart Stopped?

Chart Condition

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.

Standard Example: **Calling Chart Stopped?**

OptoScript Example: **`IsCallingChartStopped()`**

```
Chart_Status = IsCallingChartStopped();
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Continue Calling Chart (page C-37), Calling Chart Suspended? (page C-6) Calling Chart Running? (page C-5)

Calling Chart Suspended?

Chart Condition

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.

Standard Example: **Calling Chart Suspended?**

OptoScript Example: **`IsCallingChartSuspended()`**

```
Chart_Status = IsCallingChartSuspended();
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- Always use before Continue Calling Chart to ensure its success. See the Continue Calling Chart action for details.

See Also: Continue Calling Chart (page C-37), Calling Chart Stopped? (page C-5) Calling Chart Running? (page C-5)

Caused a Chart Error?

Error Handling Condition

Function: To determine if the specified chart caused the current error in the message 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 message queue.

Arguments:

Argument 1
Has
Chart

Standard Example:

Has	POWERUP	*Chart*

Caused a Chart Error?

OptoScript Example:

`HasChartCausedError(`*Chart*`)`

```
if (HasChartCausedError(POWERUP)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Use Debug mode to view the message queue for detailed information.

Dependencies: Prior to using this call, you should ensure that the error of interest is pointed to by using the Remove Current Error and Point to Next Error command.

See Also: Get Error Code of Current Error (page G-52), Remove Current Error and Point to Next Error (page R-22)

Caused an I/O Unit Error?

Error Handling Condition

Function: To determine if the specified I/O unit caused the top error in the message 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.
- You 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
Has
B100
B200
B3000 (Analog)
B3000 (Digital)
G4A8R, G4RAX
G4D16R
G4D32RS
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-PAC-R1
SNAP-PAC-R2

Standard Example:

	Has	DIG_UNIT_1	*SNAP-UP1-D64*
Caused an I/O Unit Error?			

OptoScript Example:

HasIoUnitCausedError(*I/O Unit***)**

```
if (HasIoUnitCausedError(DIG_UNIT_1)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Be sure the top error in the queue is an I/O error.
- Use Debug mode to view the message queue for detailed information.

Dependencies: You must use Error on I/O Unit? before using this command.

See Also: Error on I/O Unit? (page E-20), Get Error Code of Current Error (page G-52), Remove Current Error and Point to Next Error (page R-22)

Chart Running?

Chart Condition

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
Is
Chart

Standard Example:

Chart Running?		
Is	CHART_B	*Chart*

OptoScript Example:

IsChartRunning(*Chart***)**

```
Chart_Status = IsChartRunning(Chart_B);
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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 command can determine if the target chart has started.

See Also: Chart Suspended? (page C-11) Chart Stopped? (page C-10) Call Chart (page C-4), Start Chart (page S-93), Stop Chart (page S-99)

Chart Stopped?

Chart Condition

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
Is
Chart

Standard Example:

	Is	CHART_B	*Chart*

Chart Stopped?

OptoScript Example:

IsChartStopped(*Chart***)**

```
Chart_Status = IsChartStopped(Chart_B);
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Chart Suspended? (page C-11) Chart Running? (page C-9) Call Chart (page C-4), Start Chart (page S-93), Stop Chart (page S-99)

Chart Suspended?

Chart Condition

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
Is
Chart

Standard Example:

Chart Suspended?		
Is	CHART_B	*Chart*

OptoScript Example:

IsChartSuspended(*Chart***)**

```
Chart_Status = IsChartSuspended(Chart_B);
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use before Continue Chart to ensure success.

See Also: Chart Running? (page C-9), Chart Stopped? (page C-10), Continue Chart (page C-38), Suspend Chart (page S-106)

Clamp Float Table Element

Mathematical Action

Function: To force a table element value to be greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To keep values within a desired range. Very useful on analog input signals to prevent out-of-range values from being evaluated as real values.

Details:

- A table element value greater than the high limit will be set to the high limit. A table element value less than the low limit will be set to the low limit. Any other value is left unchanged.
- Use this command before evaluating the table value each time.

Arguments:

Argument 1 **High Limit**	Argument 2 **Low Limit**	Argument 3 **Element Index**	Argument 4 **Of Table**
Float Literal	Float Literal	Integer 32 Literal	Float Table
Float Variable	Float Variable	Integer 32 Variable	
Integer 32 Literal	Integer 32 Literal		
Integer 32 Variable	Integer 32 Variable		

Standard Example:

Clamp Float Table Element

High Limit	Max_Flow_Rate	*Float Variable*
Low Limit	Low_Flow_Cutoff	*Float Variable*
Element Index	4	*Integer 32 Literal*
Of Table	Flow_Data	*Float Table*

OptoScript Example:

`ClampFloatTableElement(`*High Limit, Low Limit, Element Index, Of Float Table***`)`**

```
ClampFloatTableElement(Max_Flow_Rate, Low_Flow_Cutoff, 4, Flow_Data);
```

This is a procedure command; it does not return a value.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Clamp Integer 32 Table Element (page C-14), Clamp Float Variable (page C-13), Clamp Integer 32 Variable (page C-15)

Clamp Float Variable

Mathematical Action

Function: To force a variable value to be greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To keep values within a desired range. Very useful on analog input signals to prevent out-of-range values from being evaluated as real values.

Details:

- A variable value greater than the high limit will be set to the high limit. A variable value less than the low limit will be set to the low limit. Any other value is left unchanged.
- Use this command before evaluating the variable value each time.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
High Limit	**Low Limit**	**Float Variable**
Float Literal	Float Literal	Float Variable
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Clamp Float Variable

High Limit	Max_Flow_Rate	*Float Variable*
Low Limit	Low_Flow_Cutoff	*Float Variable*
Float Variable	Flow_Var	*Float Variable*

OptoScript Example:

`ClampFloatVariable(` *High Limit, Low Limit, Float Variable to Clamp* **`)`**

```
ClampFloatVariable(Max_Flow_Rate, Low_Flow_Cutoff, Flow_Var);
```

This is a procedure command; it does not return a value.

See Also: Clamp Integer 32 Variable (page C-15), Clamp Float Table Element (page C-12), Clamp Integer 32 Table Element (page C-14)

Clamp Integer 32 Table Element

Mathematical Action

Function: To force a table element value to be greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To keep values within a desired range. Very useful to prevent out-of-range values from being evaluated as real values.

Details:

- A table element value greater than the high limit will be set to the high limit. A table element value less than the low limit will be set to the low limit. Any other value is left unchanged.
- Use this command before evaluating the table value each time.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
High Limit	**Low Limit**	**Element Index**	**Of Integer 32 Table**
Float Literal	Float Literal	Integer 32 Literal	Integer 32 Table
Float Variable	Float Variable	Integer 32 Variable	
Integer 32 Literal	Integer 32 Literal		
Integer 32 Variable	Integer 32 Variable		

Standard Example:

Clamp Integer 32 Table Element

High Limit	Max_Flow_Rate	*Float Variable*
Low Limit	Low_Flow_Cutoff	*Float Variable*
Element Index	4	*Integer 32 Literal*
Of Integer 32 Table	Flow_Data	*Integer 32 Table*

OptoScript Example:

`ClampInt32TableElement(`*High Limit, Low Limit, Element Index, Of Integer 32 Table***`)`**

```
ClampInt32TableElement(Max_Flow_Rate, Low_Flow_Cutoff, 4, Flow_Data);
```

This is a procedure command; it does not return a value.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Clamp Float Table Element (page C-12), Clamp Integer 32 Variable (page C-15), Clamp Float Variable (page C-13)

Clamp Integer 32 Variable

Mathematical Action

Function: To force a variable value to be greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To keep values within a desired range. Very useful to prevent out-of-range values from being evaluated as real values.

Details:
- A variable value greater than the high limit will be set to the high limit. A variable value less than the low limit will be set to the low limit. Any other value is left unchanged.
- Use this command before evaluating the variable value each time.

Arguments:

Argument 1	Argument 2	Argument 3
High Limit	**Low Limit**	**Integer 32 Variable**
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Clamp Integer 32 Variable

High Limit	Max_Flow_Rate	*Float Variable*
Low Limit	Low_Flow_Cutoff	*Float Variable*
Integer 32 Variable	Flow_Var	*Integer 32 Variable*

OptoScript Example:

`ClampInt32Variable(`*High Limit, Low Limit, Integer 32 Variable to Clamp***`)`**

```
ClampInt32Variable(Max_Flow_Rate, Low_Flow_Cutoff, Flow_Var);
```

This is a procedure command; it does not return a value.

See Also: Clamp Float Variable (page C-13), Clamp Integer 32 Table Element (page C-14), Clamp Float Table Element (page C-12)

Clamp Mistic PID Output

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To force a PID output value to be greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To keep the PID output within a desired range while it is fully operational in auto mode.

Details:

- A calculated PID output value greater than the high limit will be set to the high limit. A calculated PID output value less than the low limit will be set to the low limit. Any other calculated PID output value is left unchanged.
- If this command is sent when the PID is in manual mode, the command will not be executed.
- This command takes effect at the next PID scan interval.

Arguments:

Argument 1	Argument 2	Argument 3
High Clamp	**Low Clamp**	**On PID Loop**
Float Literal	Float Literal	PID Loop
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Clamp Mistic PID Output

High Clamp	Max_PID_output	*Float Variable*
Low Clamp	Min_PID_output	*Float Variable*
On PID Loop	Extruder_zone8	*PID Loop*

OptoScript Example:

ClampMisticPidOutput(*High Clamp, Low Clamp, On PID Loop***)**

```
ClampMisticPidOutput(Max_PID_output, Min_PID_output, Extruder_zone8);
```

This is a procedure command; it does not return a value.

Dependencies: Will not clamp values written directly to the analog output channel by anything else besides the PID on the I/O unit.

See Also: Clamp Mistic PID Setpoint (page C-17)

Clamp Mistic PID Setpoint

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To force a PID setpoint value to be greater than or equal to a low limit *and* less than or equal to a high limit.

Typical Use: To keep an operator from moving the PID setpoint outside a desired range.

Details:
- A setpoint value greater than the high limit will be set to the high limit. A setpoint value less than the low limit will be set to the low limit. Any other setpoint value is left unchanged.
- If this command is sent when the PID is in manual mode, the command will not be executed.
- This command takes effect at the next PID scan interval.

Arguments:

Argument 1	Argument 2	Argument 3
High Clamp	**Low Clamp**	**On PID Loop**
Float Literal	Float Literal	PID Loop
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Clamp Mistic PID Setpoint

High Clamp	Max_PID_output	*Float Variable*
Low Clamp	Min_PID_output	*Float Variable*
On PID Loop	Extruder_zone8	*PID Loop*

OptoScript Example:

`ClampMisticPidSetpoint(`*High Clamp, Low Clamp, On PID Loop***`)`**

```
ClampMisticPidSetpoint(Max_PID_output, Min_PID_output, Extruder_zone8);
```

This is a procedure command; it does not return a value.

See Also: Clamp Mistic PID Output (page C-16)

Clear All Errors

Error Handling Action

Function: To clear the message queue.

Typical Use: To clear all errors from a full message queue.

Details: This function clears all errors and messages in the queue. Normally this is not necessary. If your program performs error checking, it will eventually clear the message queue. If no error checking is done, simply let the queue fill up. The queue holds a total of 1000 errors and messages.

Arguments: None.

Standard Example: **Clear All Errors**

OptoScript Example:

```
ClearAllErrors()
ClearAllErrors();
```

This is a procedure command; it does not return a value.

Notes: Downloading and running a strategy automatically clears all errors.

Errors can also be cleared when inspecting the control engine in Debug mode or from ioTerminal, by clicking View Errors and then clicking Clear Errors.

See Also: Get Error Code of Current Error (page G-52), Get Error Count (page G-53), Remove Current Error and Point to Next Error (page R-22), Get ID of Block Causing Current Error (page G-64), Get Name of I/O Unit Causing Current Error (page G-99)

Clear All Event Latches

Event/Reaction Action

NOTE: This command is for mistic I/O units only.

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
On I/O Unit
B100
B200
B3000 (Analog)
B3000 (Digital)
G4A8R, G4RAX
G4D16R
SNAP-BRS

Standard Example:

Clear All Event Latches

On I/O Unit	ESTOP_BUTTONS	*G4A8R, G4RAX*

OptoScript Example:

ClearAllEventLatches(*On I/O Unit***)**

```
ClearAllEventLatches(ESTOP_BUTTONS);
```

This is a procedure command; it does not return a value.

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 Off-Latch (page C-26)

Clear All Latches

Digital Point Action

Function: To reset all standard digital input latches on a digital or mixed I/O unit.

Typical Use: To ensure all input on- or off-latches are reset. Usually performed after a powerup sequence.

Details:

- Standard digital only. For high-density digital, see Clear All HDD Module On-Latches.
- Clears all previously set on- or off-latches associated with input points on the specified I/O unit regardless of the on/off status of the inputs.
- All input points automatically have the latch feature.
- An on-latch is set when the input point changes from off to on.
- An off-latch is set when the input point changes from on to off.

Arguments:

Argument 1
On I/O Unit
B100*
B3000 (Digital)*
G4D16R*
G4D32RS*
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-ENET-S64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-PAC-R1
SNAP-PAC-R2
SNAP-BRS*

* ioControl Professional only

Standard Example:

Clear All Latches

On I/O Unit	INPUT_BOARD_1	*SNAP-ENET-D64*

OptoScript Example:

`ClearAllLatches(`*On I/O Unit***`)`**

```
ClearAllLatches(INPUT_BOARD_1);
```

This is a procedure command; it does not return a value.

Queue Errors: -52 = Invalid connection—the I/O unit or point is disabled.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

Notes: If using the latching feature on one or more digital inputs, it is a good practice to clear all the latches after powerup or reset.

See Also: Clear On-Latch (page C-27), Clear Off-Latch (page C-26)

Clear Communication Receive Buffer

Communication Action

Function: To clear the receive buffer of a communication handle.

Typical Use: To discard any data waiting to be received on a specific communication handle (for TCP and other communication handles that use a receive buffer).

Details: This command is the equivalent of a Get Number of Characters Waiting command followed by a Receive N Characters command, when the characters received are discarded.

Arguments:

Argument 1
Communication Handle
Communication Handle

Standard Example:

Clear Communication Receive Buffer

Communication Handle	UIO_B	*Communication Handle*

OptoScript Example:

`ClearCommunicationReceiveBuffer(`*Communication Handle***`)`**

```
ClearCommunicationReceiveBuffer(UIO_B);
```

This is a procedure command; it does not return a value.

Notes:

- See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide*.
- If this command is used with a communication handle that cannot receive data (for example, the ftp communication handle), the command will have no effect.
- This command replaces the obsolete Clear Receive Buffer command.

See Also: Close Communication (page C-29), Get Number of Characters Waiting (page G-101), Receive N Characters (page R-14)

Clear Counter

Digital Point Action

Function: To reset a standard digital input counter or quadrature counter to zero.

Typical Use: To reset a digital input configured with a counter or quadrature counter feature.

Details:

- Standard digital only. For high-density digital, see Get & Clear HDD Module Counter.
- Resets the specified counter or quadrature counter input to zero as soon as it is used.
- Does not stop the counter or quadrature counter from continuing to run (as Stop Counter does).
- A quadrature counter occupies two adjacent points, so quadrature modules appear with only points 00 and 02 available.

Arguments:

Argument 1
On Point
Counter
Quadrature Counter

Standard Example:

Clear Counter
On Point Bottle_Counter *Counter*

OptoScript Example:

ClearCounter(*On Point***)**

```
ClearCounter(Bottle_Counter);
```

This is a procedure command; it does not return a value.

Dependencies: Applies only to standard digital inputs configured with the counter or quadrature counter feature.

See Also: Get Counter (page G-48), Get & Clear Counter (page G-18), Start Continuous Square Wave (page S-94), Stop Counter (page S-101)

Clear Event Latch

Event/Reaction Action

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 Event Occurred? To detect the next incident of the event, the event latch must be reset using this command.

Arguments:

Argument 1
On Event/Reaction
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Clear Event Latch

On Event/Reaction	ESTOP_BUTTON_1	*Analog Event/Reaction*

OptoScript Example:

ClearEventLatch(*On Event/Reaction***)**

```
ClearEventLatch(ESTOP_BUTTON_1);
```

This is a procedure command; it does not return a value.

Dependencies:

- Event/reactions must be named and configured on the I/O unit before they can be referenced.
- Event/reactions are not supported on simple I/O units.

See Also: Event Occurred? (page E-21)

Clear HDD Module Off-Latches

High Density Digital Module Action

Function: To reset specific off-latches on a high-density digital input module.

Typical Use: To clear some off-latches and not clear others on the same module.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Uses a bitmask to indicate the off-latches to clear. The least significant bit corresponds to point zero. To clear the off-latch on a point, set its respective bit to a value of 1. To leave a point unaffected, set its bit to a value of 0.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Module Number**	Argument 3 **Clear Mask**	Argument 4 **Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Variable

Standard Example:

Clear HDD Module Off-Latches

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Module Number	6	*Integer 32 Variable*
Clear Mask	0x060000C2	*Integer 32 Literal*
Put Status in	OffLatch_Status	*Integer 32 Variable*

The effect of this command is illustrated below. Off-latches for point numbers 1, 6, 7, 25, and 26 are cleared.

	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0
Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			

OptoScript Example:

`ClearHddModuleOffLatches(`*I/O Unit, Module Number, Clear Mask***`)`**

```
OffLatch_Status = ClearHddModuleOffLatches(UIO_A, 6, 0x060000C2);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- Usually used after Get HDD Module Off-Latches. To read and reset all the off-latches on one module at once, use Get & Clear HDD Module Off-Latches. To read and reset all off-latches on all high-density modules on the I/O unit, use Get & Clear All HDD Module Off-Latches.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes: 0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Clear HDD Module On-Latches (page C-25), Get & Clear HDD Module Off-Latches (page G-22), Get & Clear All HDD Module Off-Latches (page G-10)

Clear HDD Module On-Latches

High Density Digital Module Action

Function: To reset specific on-latches on a high-density digital input module.

Typical Use: To clear some on-latches and not clear others on the same module.

Details:
- Works only on high-density digital modules, not on standard digital modules.
- Uses a bitmask to indicate the on-latches to clear. The least significant bit corresponds to point zero. To clear the on-latch on a point, set its respective bit to a value of 1. To leave a point unaffected, set its bit to a value of 0.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Module Number**	**Clear Mask**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable	Integer 32 Variable	
SNAP-UP1-ADS			
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Clear HDD Module On-Latches

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Module Number	6	*Integer 32 Variable*
Clear Mask	0x060000C2	*Integer 32 Literal*
Put Status in	OnLatch_Status	*Integer 32 Variable*

The effect of this command is illustrated below. On-latches for point numbers 1, 6, 7, 25, and 26 are cleared.

	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0
Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			

OptoScript Example:

`ClearHddModuleOnLatches(`*I/O Unit, Module Number, Clear Mask***`)`**

```
OnLatch_Status = ClearHddModuleOnLatches(UIO_A, 6, 0x060000C2);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- Usually used after Get HDD Module On-Latches. To read and reset all the on-latches on one module at once, use Get & Clear HDD Module On-Latches.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Clear HDD Module Off-Latches (page C-24), Get & Clear HDD Module On-Latches (page G-24), Get & Clear All HDD Module On-Latches (page G-12)

Clear Off-Latch

Digital Point Action

Function: To reset a previously set standard digital input off-latch.

Typical Use: To reset the off-latch associated with a digital input to catch the next transition.

Details:

- Standard digital only. For high-density digital, see Clear HDD Module Off-Latches.
- Resets the off-latch of a single digital input regardless of the on/off status of the input.
- The next time the input point 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.

Arguments:

Argument 1
On Point
Digital Input

Standard Example:

Clear Off-Latch

On Point	BUTTON_1	*Digital Input*

OptoScript Example:

`ClearOffLatch(`*On Point***`)`**

```
ClearOffLatch(BUTTON_1);
```

This is a procedure command; it does not return a value.

Queue Errors:

-52 = Invalid connection—the I/O unit or point is disabled.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

Notes: Clear an off-latch after a Get Off-Latch command to re-arm the latch.

See Also: Get Off-Latch (page G-102), Clear All Latches (page C-20)

Clear On-Latch

Digital Point Action

Function: To reset a previously set standard digital input on-latch.

Typical Use: To reset the on-latch associated with a digital input to catch the next transition.

Details:

- Standard digital only. For high-density digital, see Clear HDD Module On-Latches.
- Resets the on-latch of a single digital input regardless of the on/off status of the input.
- The next time the input point 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.

Arguments:

Argument 1
On Point
Digital Input

Standard Example:

Clear On-Latch

On Point	Button_1	*Digital Input*

OptoScript Example:

ClearOnLatch(*On Point***)**

```
ClearOnLatch(Button_1);
```

This is a procedure command; it does not return a value.

Queue Errors: -52 = Invalid connection—the I/O unit or point is disabled.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

Notes: Clear an on-latch after a Get On-Latch command to re-arm the latch.

See Also: Get On-Latch (page G-106), Clear All Latches (page C-20)

Clear Pointer

Pointers Action

Function: To NULL out a pointer.

Typical Use: To clear a pointer so that it no longer points to an object.

Arguments:

Argument 1
Pointer
Pointer Variable

Standard Example:

Clear Pointer		
Pointer	IO_Pointer	*Pointer Variable*

OptoScript Example: OptoScript doesn't use a command; the functionality is built in. Assign `null` to the pointer:

```
IO_Pointer = null;
```

Notes: Operations cannot be performed on NULL pointers. NULL pointers do not point to any object.

See Also: Move to Pointer (page M-19), Clear Pointer Table Element (page C-28)

Clear Pointer Table Element

Pointers Action

Function: To NULL out the specified element of a pointer table.

Typical Use: To clear an element in a pointer table so that it no longer points to any object.

Arguments:

Argument 1	Argument 2
Index	**Of Table**
Integer 32 Literal	Pointer Table
Integer 32 Variable	

Standard Example:

Clear Pointer Table Element		
Index	17	*Integer 32 Literal*
Of Table	IO_POINTER_TABLE	*Pointer Table*

OptoScript Example: OptoScript doesn't use a command; the functionality is built in. Assign `null` to the pointer:

```
IO_POINTER_TABLE[17] = null;
```

Notes: Operations cannot be performed on a NULL pointer.

Queue Errors: -12 = Invalid table index value—index was negative or greater than the table size.

See Also: Move to Pointer Table Element (page M-21)

Clear Receive Buffer

Communication Action

Function: Obsolete command. Use Clear Communication Receive Buffer instead.

Close Communication

Communication Action

Function: To disconnect the previously established communication link, or to send the data currently buffered in the temporary FTP file.

Typical Use: To end communication with the other entity (for example, a device on the network or a file) that was specified by a communication handle.

Arguments:

Argument 1	Argument 2
Communication Handle	**Put Status in**
Communication Handle	Integer 32 Variable

Standard Example:

Close Communication

Communication Handle	UIO_A	*Communication Handle*
Put Status in	Ethernet_Status	*Integer 32 Variable*

OptoScript Example:

CloseCommunication(*Communication Handle***)**

```
Ethernet_Status = CloseCommunication(UIO_A);
```

This is a function command; it returns a status code as shown below.

Notes:

- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.
- When using an FTP communication handle, the data to be sent via FTP is held in a temporary FTP file until either this command is used or the FTP destination file is changed using Send Communication Handle Command.

Status Codes:

0 = Success

-37 = Lock port timeout.

-52 = Invalid connection—not opened.

-78 = No destination given (FTP destination file).

See Also: Open Outgoing Communication (page O-4), Send Communication Handle Command (page S-2)

Comment (Block)

Miscellaneous Action or Condition

Function: To disable one or more commands in an action or condition block.

Typical Use: To temporarily disable commands within an action or condition block during debugging.

Details:

- This command is normally used in pairs. Everything between the pair of Comment (Block) commands is considered a comment and is ignored when the strategy is compiled and downloaded. In the Instructions dialog box, commands that are commented out appear in gray.
- This command is useful for temporarily disabling a group of commands within an action block while debugging a program.
- If the second Comment (Block) is omitted, everything from the first Comment (Block) to the end of the action block is considered a comment.

Arguments: None.

Standard Example:

Comment (Block)
  Action or Condition
  Action or Condition
  Action or Condition
Comment (Block)

OptoScript Example: OptoScript doesn't use a command; the functionality is built in. Use a slash and an asterisk before the block comment, and an asterisk and a slash after the block comment:

```
/* block comment */
```

See Also: Comment (Single Line) (page C-31)

Comment (OptoControl Conversion Issue)

Miscellaneous Action or Condition

Function: A Comment is inserted automatically when a command does not convert from OptoControl to ioControl. This command is not added to a strategy by a user.

Typical Use: To locate areas in a strategy where a command did not convert.

Details: To find Comments in a strategy:

1. In the Configure Mode, choose Edit→Find. The Find dialog box appears.
2. Under Search Scope, select Global.
3. Under Search For, select Instruction and Action.
4. Under Instruction, select Comment (OptoControl Conversion Issue), then click Find. A list appears that identifies each Comment.

Comment (Single Line)

Miscellaneous Action or Condition

Function: To add a comment to an action or condition block.

Typical Use: To document commands within a block.

Arguments:

Argument 1
[Value]
String Literal

Standard Example:

Comment (Single Line)
PROCESS_CONTROL_START *String Literal*

OptoScript Example: OptoScript doesn't use a command; the functionality is built in. Use two slashes before the comment.

```
// single line comment
```

See Also: Comment (Block) (page C-30)

Communication Open?

Communication Condition

Function: To determine if the specified communication is still online.

Typical Use: To determine if the communication handle was successfully opened or is still open, before attempting to send communication.

Arguments:

Argument 1
Communication Handle
Communication Handle

Standard Example:

Communication Open?
UIO_A *Communication Handle*

OptoScript Example:

IsCommunicationOpen(*Communication Handle***)**

```
if (IsCommunicationOpen(UIO_A)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.
- This command will return false only if the Close Communication command has been called on the communication handle, or if the handle was closed during an unsuccessful operation. For example, an unrecoverable failure during a Transmit command could cause the handle to be closed.
- Using TCP, this command will return a true (non-zero) even if the other side has closed. This situation is called a "half open" connection. Even though the other side has closed, there may still be characters buffered by the control engine. Make sure the characters are received (and the communication handle closed, if appropriate) so that sessions aren't used up by a half-open state.

See Also: Accept Incoming Communication (page A-2), Open Outgoing Communication (page O-4), Close Communication (page C-29)

Communication to All I/O Points Enabled?

Simulation Condition

Function: To determine whether communication between the program in the control engine and all analog and digital points is enabled.

Typical Use: For simulation and testing. An I/O point might be disabled if you do not want to communicate with it during testing.

Details: All analog and digital point communication is enabled by default. It can be turned off for individual points in the configuration dialog box or by using the command Disable Communication to Point. Use this command to find out if communication has been disabled.

Arguments: None

Standard Example: Communication to All I/O Points Enabled?

OptoScript Example: **`IsCommToAllIoPointsEnabled()`**

```
if (IsCommToAllIoPointsEnabled()) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This command is much faster than checking points individually.
- Be aware that I/O points may not be reachable even if communication is enabled. For example, the I/O unit may be turned off or unplugged, but its points may still be enabled. To determine whether an I/O unit is reachable, use I/O Unit Ready?

See Also: Disable Communication to All I/O Points (page D-4), Enable Communication to All I/O Points (page E-1), Disable Communication to Point (page D-11), I/O Point Communication Enabled? (page I-2)

Communication to All I/O Units Enabled?

Simulation Condition

Function: To determine whether communication between the program in the control engine and all I/O units is enabled.

Typical Use: For simulation and testing. An I/O unit might be disabled if you do not want to communicate with it during testing.

Arguments: None.

Standard Example: **Communication to All I/O Units Enabled?**

OptoScript Example: **`IsCommToAllIoUnitsEnabled()`**

```
if (IsCommToAllIoUnitsEnabled()) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This command is much faster than checking I/O units individually.
- Be aware that the I/O unit may not be reachable even if communication is enabled. For example, the I/O unit may be turned off or unplugged, but its points and the unit itself may still be enabled. To determine whether an I/O unit is reachable, use I/O Unit Ready?

See Also: Disable Communication to All I/O Units (page D-5), Enable Communication to All I/O Units (page E-2), Disable Communication to I/O Unit (page D-7), I/O Unit Communication Enabled? (page I-3)

Compare Strings

String Action

Function: To compare two strings to see if they are the same or if one is less than the other.

Typical Use: To sort strings.

Details:

- Strings are compared character by character according to their ASCII value. See the ASCII table in the "String Commands" section of Chapter 10 in the *ioControl User's Guide*. Note that number values are lower than letter values and that all uppercase letter values are lower than all lowercase letter values.
- If the strings are different lengths, they are compared up to the length of the shorter string. If the compared portions are equal, the shorter string is found to be less than the longer one.
- The result returned indicates the relationship between the two strings:

  -1 = String 1 less than String 2
  0 = Strings equal
  1 = String 1 greater than String 2

  Examples:

String 1	String 2	Result	Relationship
"abcDEF"	"abcDEF"	0	Strings equal
"abcDEF"	"abcdef"	-1	String 1 less
"abcDEF"	"ABCDEF"	1	String 1 greater
"abcDEF"	"abcdEF"	-1	String 1 less
"abcDEF"	"AbcDEF"	1	String 1 greater
"abcDEF"	"abcDE"	1	String 1 greater
"abcDEF"	"abcDEFG"	-1	String 1 less
"abcDEF"	"aBcDEF"	1	String 1 greater
"abcDEF"	"9abcDEF"	1	String 1 greater
"abcDEF"	"DEFabc"	1	String 1 greater

- Quotes ("") are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1	Argument 2	Argument 3
Compare	**With**	**Put Result In**
String Literal	String Literal	Integer 32 Variable
String Variable	String Variable	

Example:

Compare Strings

Compare	Search_Name	*String Variable*
With	Current_Name	*String Variable*
Put Result In	String_Test	*Integer 32 Variable*

OptoScript Example:

CompareStrings(*String 1, String 2***)**

```
String_Test = CompareStrings(Search_Name, Current_Name);
```

This is a function command; it returns one of the values shown above (-1, 0, or 1). The returned value can be consumed by a variable (as shown in the example) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide*.

Notes: See "String Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Test Equal Strings (page T-3)

Complement

Mathematical Action

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. The command Absolute Value is another, better way to accomplish the same thing.

Details: Same as multiplying by -1, but executes faster. Thus, -1 becomes 1, 1 becomes -1, etc.

Arguments:

Argument 1
[Value]
Float Variable
Integer 32 Variable
Integer 64 Variable

Standard Example:

Complement		
	Temperature_Difference	*Float Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the minus sign:

```
- Temperature_Difference
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- The complement of zero is zero.

See Also: Bit NOT (page B-5), NOT (page N-2), Absolute Value (page A-1)

Continue Calling Chart

Chart Action

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

Typical Use: To restart a suspended chart.

Details:

- This command is not normally needed, since a called chart, when finished, automatically continues the chart that called it. Use this command only if you need to restart the calling chart before the chart it called is finished.
- 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 task queue.
- The STATUS variable indicates success (0) or failure (non-zero). 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 action. The True exit will lead directly to the Continue Calling Chart action, 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
Put Status in
Float Variable
Integer 32 Variable

Standard Example:

Continue Calling Chart		
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`ContinueCallingChart()`

```
STATUS = ContinueCallingChart();
```

This is a function command; it returns a non-zero (indicating success) or a zero (indicating failure).

Notes: See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Call Chart (page C-4), Calling Chart Suspended? (page C-6)

Continue Chart

Chart Action

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 task queue at the point at which it was suspended.
- Suspended charts give up their time slice.
- The STATUS variable indicates success (0) or failure (non-zero).
- 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	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Standard Example:

Continue Chart

Chart	CHART_A	*Chart*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`ContinueChart(`*`Chart`*`)`

```
= ContinueChart(CHART_A);
```

This is a function command; it returns a zero (indicating success) or a non-zero (indicating failure).

Notes:

- This command should be used judiciously. It can take up to 100 ms for the chart to continue. Use this command only when timing is not critical. Otherwise, instead of Continue Chart, use a chart that runs continuously and uses subroutines for any kind of repetitive logic.
- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- Loop on Chart Suspended? before this command if success is critical.

See Also: Suspend Chart (page S-106), Chart Suspended? (page C-11)

Continue Timer

Timing Action

Function: To continue a paused timer variable.

Typical Use: Used with Pause Timer command to track total on/off (up/down, fwd/reverse) time.

Details: The timer variable must have been paused with the Pause Timer command. It continues from the value at which it was paused.

Arguments:

Argument 1
Timer
Down Timer Variable
Up Timer Variable

Standard Example:

Continue Timer

Timer	OVEN_TIMER	*Down Timer Variable*

OptoScript Example:

ContinueTimer(*Timer***)**

```
ContinueTimer(OVEN_TIMER);
```

This is a procedure command; it does not return a value.

Notes: None

See Also: Start Off-Pulse (page S-96), Stop Timer (page S-102), Pause Timer (page P-1), Set Down Timer Preset Value (page S-21), Set Up Timer Target Value (page S-86)

Convert Float to String

String Action

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:

- 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 string variable is less than the specified length, "*****" will be moved to the destination string.
- 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
Convert	**Length**	**Decimals**	**Put Result in**
Analog Input	Integer 32 Literal	Integer 32 Literal	String Variable
Analog Output	Integer 32 Variable	Integer 32 Variable	
Float Literal			
Float Variable			
Integer 32 Literal			
Integer 32 Variable			

Standard 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"):

Convert Float to String

Convert	My_Value	*Float Variable*
Length	5	*Integer 32 Literal*
Decimals	2	*Integer 32 Literal*
Put Result in	Value_as_String	*String Variable*

OptoScript Example: **`FloatToString(`***Convert, Length, Decimals, Put Result in***`)`**

```
FloatToString(My_Value, 5, 2, Value_as_String);
```

This is a procedure command; it does not return a value.

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on using strings in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Set decimals to zero to get an integer. Normal rounding will occur.

Dependencies: The string variable must be wide enough to hold the resulting formatted string.

See Also: Convert String to Float (page C-52), Convert Number to String (page C-50), Convert Number to String Field (page C-51)

Convert Hex String to Number

String Action

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 in OptoScript code, but not in standard ioControl code.
- 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 "9," "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 a string convert the result to a zero.

Arguments:

Argument 1 **Convert**	Argument 2 **Put Result in**
String Literal	Float Variable
String Variable	Integer 32 Variable

Standard Example:

Convert Hex String to Number

Convert	String_From_Port	*String Variable*
Put Result in	Int_Value	*Integer 32 Variable*

OptoScript Example:

HexStringToNumber(*Convert***)**

```
Int_Value = HexStringToNumber(String_From_Port);
```

This is a function command; it returns the converted number. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- If the hex string contains an IEEE float, you must use Convert IEEE Hex String to Number.

See Also: Convert Number to Hex String (page C-48), Convert String to Float (page C-52), Convert String to Integer 32 (page C-54), Convert IEEE Hex String to Number (page C-42)

Convert IEEE Hex String to Number

String Action

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 Convert Number to Formatted Hex String.

Details:

- Quotes (“”) are used in OptoScript code, but not in standard ioControl code.
- Use between control engines or other computers that use the IEEE format.
- The eight hex characters are converted to four bytes (IEEE float format).
- The hex string must be in Motorola or Big Endian format (most significant byte on the left, in the least significant address).

Arguments:

Argument 1 **Convert**	Argument 2 **Put Result in**
String Literal	Float Variable
String Variable	Integer 32 Variable

Standard Example: The following example converts a hex string into a float value. For example, if STRING FROM PORT contains “418E6666” then MY FLOAT VALUE becomes 17.8.

Convert IEEE Hex String to Number

Convert	STRING_FROM_PORT	*String Variable*
Put Result in	MY_FLOAT_VALUE	*Float Variable*

OptoScript Example: **`IEEEHexStringToNumber(`***Convert***`)`**

```
MY_FLOAT_VALUE = IEEEHexStringToNumber(STRING_FROM_PORT);
```

This is a function command; it returns the converted number. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User’s Guide* for more information.

Notes: See “String Commands” in Chapter 10 of the *ioControl User’s Guide*.

See Also: Convert Number to Formatted Hex String (page C-46), Convert Hex String to Number (page C-41)

Convert Integer 32 to IP Address String

String Action

Function: To convert an integer 32 value to an IP address string.

Typical Use: To convert an IP address stored as an integer into a human-readable string, such as "10.192.54.155"

Arguments:

Argument 1	**Argument 2**
Convert	**Put Result in**
Integer 32 Literal	String Variable
Integer 32 Variable	

Standard Example:

Convert Integer 32 to IP Address String

Convert	IP_Integer	*Integer 32 Variable*
Put Result in	IP_String	*String Variable*

OptoScript Example:

`Int32ToIpAddressString(`*Convert, Put Result In***`)`**

```
IpAddressStringToInt32(IP_Integer, IP_String);
```

This is a function command; it returns the converted string.

Notes: See "String Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Convert IP Address String to Integer 32 (page C-44)

Convert IP Address String to Integer 32

String Action

Function: To convert an IP address string value to an integer 32 value.

Typical Use: To convert an IP address stored as a string (for example, "10.192.54.155") to an integer (in this example, 0x0AC0369B)

Details: Quotes ("") are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1	**Argument 2**
Convert	**Put Result in**
String Literal	Integer 32 Variable
String Variable	

Standard Example:

Convert IP Address String to Integer 32

Convert	IP_String	*String Variable*
Put Result in	IP_Integer	*Integer 32 Variable*

OptoScript Example:

`IpAddressStringToInt32(`*Convert***`)`**

```
IP_Integer = IpAddressStringToInt32(IP_String);
```

This is a function command; it returns the converted number. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "String Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Convert Integer 32 to IP Address String (page C-43)

Convert Mistic I/O Hex String to Float

String Action

Function: Converts a float value represented as an eight-character hex response from an I/O unit to a float number.

Typical Use: Reading analog values in engineering units from an I/O unit.

Details:

- I/O units use integers to represent all numeric values. Float values are handled using a 16-bit signed integer for the whole number part and a 16-bit unsigned integer for the fractional part. Each count in the fractional part represents 0.000015259. These four bytes become eight bytes when represented in hex.
- Legal range is -32768 to 32767.

Arguments:

Argument 1	Argument 2
Hex String	**Put Result in**
String Literal	Float Variable
String Variable	

Standard Example:

Convert Mistic I/O Hex String to Float

Hex String	IO_Response	*String Variable*
Put Result in	Eunit_Value	*Float Variable*

OptoScript Example:

`MisticIoHexToFloat(`*Convert***`)`**

```
Eunit_Value = MisticIoHexToFloat(IO_Response);
```

This is a function command; it returns the converted float. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Use Convert Hex String to Number instead when the hex response represents a count.

Dependencies: Use Transmit/Receive Mistic I/O Hex String first.

See Also: Transmit/Receive Mistic I/O Hex String (page T-18), Convert Number to Mistic I/O Hex String (page C-49), Convert Hex String to Number (page C-41)

Convert Number to Formatted Hex String

String Action

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 print a hex number or to send it to another device with a fixed length.

Details:

- The *Length* parameter (*Argument 2*) specifies the final length of the resulting string. Leading zeros are added if required.
- You must use a Length of 8 when converting a float or a negative number.
- To send a float value in native IEEE format, set the value of *Argument 2* to 8, and use a float variable or literal. If less than eight characters are used, asterisks appear in the Put Result In argument, and error -3 (Buffer overrun or invalid length error) appears in the message queue. Use Convert 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 string is filled with asterisks and an error -3 is reported.
- If the declared width of the string variable is less than the specified length, error -3 (Buffer overrun or invalid length error) appears in the message queue. If the value can be represented by the string width, the value is stored in the variable. Otherwise, the string is filled with asterisks.
- If the declared width is not long enough to represent the value, error -23 (Destination string too short) appears in the message queue, and the string is filled with asterisks.
- Upper case is used for all hex characters; for example, 1,000 decimal is represented as 3E8 rather than 3e8.

Arguments:

Argument 1 **Convert**	Argument 2 **Length**	Argument 3 **Put Result in**
Analog Input	Integer 32 Literal	String Variable
Analog Output	Integer 32 Variable	
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		

Standard 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."

Convert Number to Formatted Hex String

Convert	My_Address	*Integer 32 Variable*
Length	4	*Integer 32 Literal*
Put Result in	Address_as_Hex	*String Variable*

OptoScript Example:

`NumberToFormattedHexString(`*Convert*, *Length*, *Put Result in***`)`**

```
NumberToFormattedHexString(My_Address, 4, Address_as_Hex);
```

This is a procedure command; it does not return a value.

Notes:

- See “String Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Caution: Do not use a float where an integer would suffice. Floats are not automatically converted to integers with this command.

Queue Errors:

-3 = Buffer overrun or invalid length error. If a float value or negative number is used, the string width must be at least 8.

-23 = Destination string too short. The string width is not long enough to represent the number.

Dependencies: The string variable must be wide enough to hold the hex string.

See Also: Convert Float to String (page C-40), Convert Number to Hex String (page C-48), , Convert Number to String (page C-50), Convert Number to String Field (page C-51)

Convert Number to Hex String

String Action

Function: To convert a decimal integer to a hex string.

Typical Uses:
- To send an integer value with a predetermined length to another control engine.
- To print a hex representation of a number or to send it to another device.

Details:
- Does not add leading zeros or spaces.
- If the declared width of the string variable is less than the resulting hex string length, the hex string will be filled with asterisks.
- Upper case is used for all hex characters; for example, 1,000 decimal is represented as 3E8 rather than 3e8.
- A floating point number is first rounded to a whole number, then converted to a hex string.

Arguments:

Argument 1	**Argument 2**
Convert	**Put Result in**
Analog Input	String Variable
Analog Output	
Down Timer Variable	
Float Literal	
Float Variable	
Integer 32 Literal	
Integer 32 Variable	
Integer 64 Literal	
Integer 64 Variable	
Up Timer Variable	

Standard 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

Convert	My_Address	*Integer 32 Variable*
Put Result in	Address_as_Hex	String Variable

OptoScript Example: **NumberToHexString(***Convert, Put Result in***)**

```
NumberToHexString(My_Address, Address_as_Hex);
```

This is a procedure command; it does not return a value.

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use Convert Number to Formatted Hex String when converting floats that require formatting.

Dependencies: The string variable must be wide enough to hold the resulting hex string.

See Also: Convert Number to Formatted Hex String (page C-46), Convert Float to String (page C-40), Convert Number to String (page C-50), Convert Number to String Field (page C-51)

Convert Number to Mistic I/O Hex String

String Action

Function: Converts a float value to an eight-character hex string using the I/O unit engineering units format.

Typical Use: Sending values in engineering units to an analog I/O unit.

Details:
- I/O units use integers to represent all numeric values. Float values are handled using a 16-bit signed integer for the whole number part and a 16-bit unsigned integer for the fractional part. Each count in the fractional part represents 0.000015259. These four bytes become eight bytes when represented in hex.
- Legal range is -32768 +32767.

Arguments:

Argument 1	Argument 2
Number	**Put Result in**
Float Literal	String Variable
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Convert Number to Mistic I/O Hex String

Number	EUNIT_VALUE	*Float Variable*
Put Result in	HEX_VALUE	*String Variable*

OptoScript Example:

NumberToMisticIoHex(*Convert, Put Result in***)**

```
NumberToMisticIoHex(EUNIT_VALUE, HEX_VALUE);
```

This is a procedure command; it does not return a value.

Notes: Use Convert Number to Formatted Hex String when the number represents a count or bit pattern.

See Also: Transmit/Receive Mistic I/O Hex String (page T-18), Convert Mistic I/O Hex String to Float (page C-45), Convert Number to Formatted Hex String (page C-46)

Convert Number to String

String Action

Function: To convert a decimal number to a string.

Typical Use: To print a number or send it to another device.

Details:

- If the declared width of the string variable is less than the resulting string length, the resulting string will be filled with asterisks to alert you to the problem.
- Example: 12345 becomes 12345—Note no change for integers.
- Floats will have an exponential format.

Arguments:

Argument 1	**Argument 2**
Convert	**Put Result in**
Analog Input	String Variable
Analog Output	
Float Literal	
Float Variable	
Integer 32 Literal	
Integer 32 Variable	
Integer 64 Literal	
Integer 64 Variable	

Standard 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

Convert	My_Value	*Float Variable*
Put Result in	Value_as_String	*String Variable*

OptoScript Example: **`NumberToString(`***Convert, Put Result in***`)`**

```
NumberToString(MY_Value, Value_as_String);
```

This is a procedure command; it does not return a value.

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- To avoid scientific notation or to have greater control over format, use Convert Float to String instead.

Dependencies: The string variable must be wide enough to hold the resulting string.

See Also: Convert String to Integer 32 (page C-54), Convert Float to String (page C-40)

Convert Number to String Field

String Action

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:

- 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 string variable is less than the resulting string length, the resulting string is filled with asterisks.
- A value whose length is less than that specified will have leading spaces added as necessary, up to a maximum equal to the string width.
- A value whose length is equal to or greater than the specified length will be sent as is.
- A floating point value will have an exponential format.
- Examples (Quotes are used in OptoScript code, but not in standard ioControl code. They are used here for clarity only):

  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.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Convert	**Length**	**Put Result in**
Analog Input	Integer 32 Literal	String Variable
Analog Output	Integer 32 Variable	
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		

Standard Example:

Convert Number to String Field

Convert	Value	*Integer 32 Variable*
Length	6	*Integer 32 Literal*
Put Result in	Value_as_String	*String Variable*

OptoScript Example:

`NumberToStringField(`*Convert, Length, Put Result in***`)`**

```
NumberToStringField(Value, 6, Value_as_String);
```

This is a procedure command; it does not return a value.

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use Convert Float to String to better control the resulting format, if desired.

Dependencies: The string variable must be wide enough to hold the resulting string.

See Also: Convert Number to Formatted Hex String (page C-46), Convert Float to String (page C-40), Convert Number to String (page C-50), Convert Number to Hex String (page C-48)

Convert String to Float

String Action

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 in OptoScript code, but not in standard ioControl code.
- 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” (exponent). 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.
- If an invalid character is found, the string will be converted to 0. For example, “X 22.2 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-float conversion examples:

STRING	FLOAT
“”	0
“A12”	0
“123P”	0
“123 P”	123
“123.456”	123.456
“22 33 44”	22
“ 22.11”	22.11
“1,234.00”	0
“1234.00”	1234
“1.23e01”	12.3

Arguments:

Argument 1	Argument 2
Convert	**Put Result in**
String Literal	Float Variable
String Variable	

Standard Example:

Convert String to Float

Convert	String_from_Port	*String Variable*
Put Result in	Float_Value	*Float Variable*

OptoScript Example:

StringToFloat(*Convert***)**

```
Float_Value = StringToFloat(String_from_Port);
```

This is a function command; it returns the converted float. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "String Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Convert Float to String (page C-40), Convert String to Integer 32 (page C-54)

Convert String to Integer 32

String Action

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 in OptoScript code, but not in standard ioControl code.
- Valid, convertible characters are 0 to 9. 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 truncated to an integer value. For example, “123.6” is truncated to 123. (To round a float rather than truncating it, do not use this command. Instead, use Convert String to Float and then use Move to move the float to an integer.)
- 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 are ignored. For example, “123 4” and “123.0 X” both convert to 123.
- If an invalid character is found, the string is used up to that character. For example, “X 22 4” becomes 0, since the first character (X) is invalid. “1,234 45” becomes 1, since the comma is invalid.
- The following are string-to-integer conversion examples:

STRING	INTEGER
“”	0
“A12”	0
“123P”	123
“123 P”	123
“123.456”	123
“22 33 44”	22
“ 22.51”	22
“1,234”	1
“1234.00”	1234

Arguments:

Argument 1 **Convert**	Argument 2 **Put Result in**
String Literal	Integer 32 Variable
String Variable	

Standard Example:

Convert String to Integer 32

Convert	String_from_Port	*String Variable*
Put Result in	Int_Value	*Integer 32 Variable*

OptoScript Example:

`StringToInt32(`*Convert***`)`**

```
Int_Value = StringToInt32(String_from_Port);
```

This is a function command; it returns the converted integer. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User’s Guide* for more information.

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Avoid alpha characters. Stick with 0 to 9.
- If you need to convert a string to an integer 64 for use with a 64-point digital-only I/O unit, use the command Convert String to Integer 64.

See Also: Convert String to Float (page C-52), Convert Number to String (page C-50)

Convert String to Integer 64

String Action

Function: To convert a string to an integer 64 value.

Typical Use: Most conversions will be to integer 32 values and use the command Convert String to Integer 32. Use this command to accommodate communications or operator entry strings that must be converted to integer 64 values for use with digital-only 64-point I/O units.

Details:
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- Valid, convertible characters are 0 to 9. 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 truncated to an integer value. For example, "123.6" is truncated to 123. (To round a float rather than truncating it, do not use this command. Instead, use Convert String to Float and then use Move to move the float to an integer.)
- 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 are ignored. For example, "123 4" and "123.0 X" both convert to 123.
- If an invalid character is found, the string is used up to that character. For example, "X 22 4" becomes 0, since the first character (X) is invalid. "1,234 45" becomes 1, since the comma is invalid.
- The following are string-to-integer conversion examples:

String	Integer
""	0
"A12"	0
"123P"	123
"123 P"	123
String	**Integer**
"123.456"	123
"22 33 44"	22
" 22.51"	22
"1,234"	1
"1234.00"	1234

Arguments:

Argument 1	Argument 2
Convert	**Put Result in**
String Literal	Integer 64 Variable
String Variable	

Standard Example:

Convert String to Integer 64

Convert	String_from_Port	*String Variable*
Put Result in	Int_Value	*Integer 64 Variable*

OptoScript Example:

`StringToInt64(`*Convert***`)`**

`Int_Value = StringToInt64(String_from_Port);`

This is a function command; it returns the converted integer. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Avoid alpha characters. Use characters 0 to 9.

See Also: Convert String to Float (page C-52), Convert Number to String (page C-50)

Convert String to Lower Case

String Action

Function: To change any uppercase letters in a string to lower case.

Typical Use: To simplify string matching by making all characters the same case.

Details: Does not affect numbers, blanks, punctuation, etc.

Arguments:

Argument 1
Convert
String Variable

Standard Example:

Convert String to Lower Case

Convert	IO_COMMAND	*String Variable*

OptoScript Example:

`StringToLowerCase(`*Convert***`)`**

`StringToLowerCase(IO_COMMAND);`

This is a procedure command; it does not return a value.

See Also: Convert String to Upper Case (page C-57)

Convert String to Upper Case

String Action

Function: To change any lowercase letters in a string to upper case.

Typical Use: To simplify string matching by making all characters the same case.

Details: Does not affect numbers, blanks, punctuation, etc.

Arguments:

Argument 1
Convert
String Variable

Standard Example:

Convert String to Upper Case

Convert	IO_COMMAND	*String Variable*

OptoScript Example:

StringToUpperCase(*Convert***)**

```
StringToUpperCase(IO_COMMAND);
```

This is a procedure command; it does not return a value.

See Also: Convert String to Lower Case (page C-56)

Copy Current Error to String

Error Handling Action

Function: To copy information about the current error into a string.

Typical Use: To log errors and other information from the message queue.

Details:

- Columns of information from the message queue are put into a string variable with the delimiter you set in Argument 1. Columns are: Error Code, Severity, Chart, Block, Line, Object, Time, and Date. If the information came from a subroutine, the Chart column shows the chart that called the subroutine, and the Block column includes the subroutine name in the format `<Sub Name>.Block`.

  The following sample messages all use a comma as the delimiter:

  ```
  -534,Info,_INIT_IO,-1,0,sio13,17:19:11,01/03/05
  -35,Warning,_INIT_IO,-1,0,ai36_Temp,17:19:21,12/03/04
  -12,Error,Process,TableSub.3,2,strTable,08:46:11,09/24/04
  -15,Error,Powerup,0,1,(null),10:44:42,12/04/04
  User,Warning,Powerup,0,1,custom error,10:39:20,10/19/04
  ```
- If there are no errors in the queue, the string variable will be empty.
- If you are in Minimal Debug rather than Full Debug, the Line column will contain a zero.
- Quotes (“”) are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1	Argument 2
Delimiter	**Put Result in**
Integer 32 Literal	String Variable
Integer 32 Variable	

Standard Example:

Copy Current Error to String

Delimiter	44	*Integer 32 Literal*
Put Result In	strError	*String Variable*

OptoScript Example:

`CurrentErrorToString(`*Delimiter, String***`)`**

```
CurrentErrorToString(44, strError);
```

This is a procedure command; it does not return a value.

Notice that in OptoScript, the integer 32 literal `44` could also be entered as a character constant, in this case: `','`

See Also: Get Error Code of Current Error (page G-52), Clear All Errors (page C-18), Get Error Count (page G-53), Remove Current Error and Point to Next Error (page R-22)

Copy Date to String (DD/MM/YYYY)

Time/Date Action

Function: To read the date from the control engine's real-time clock/calendar and put it into a string variable in the standard European format dd/mm/yyyy, where dd = day (01–31), mm = month (01–12), and yyyy = year (2000–2099).

Typical Use: To date stamp an event in an ioControl program.

Details:

- If the current date is March 1, 2002, this action would place the string "01/03/2002" into the *String* parameter (*Argument 1*).
- The destination string should have a minimum width of ten.

Arguments:

Argument 1
To
String Variable

Standard Example:

Copy Date to String (DD/MM/YYYY)		
To	DATE_STRING	*String Variable*

OptoScript Example:

`DateToStringDDMMYYYY(`*String*`)`

```
DateToStringDDMMYYYY(DATE_STRING);
```

This is a procedure command; it does not return a value.

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.

Queue Error: -44 = String too short.

See Also: Copy Date to String (MM/DD/YYYY) (page C-60), Copy Time to String (page C-61), Set Date (page S-16), Set Time (page S-83)

Copy Date to String (MM/DD/YYYY)

Time/Date Action

Function: To read the date from the control engine's real-time clock/calendar and put it into a string variable in the standard United States format mm/dd/yyyy, where mm = month (01–12), dd = day (01–31), and yyyy = year (2000-2099).

Typical Use: To date stamp an event in an ioControl program.

Details:

- If the current date is March 1, 2002, this action would place the string "03/01/2002" into the *String* parameter (*Argument 1*).
- The destination string should have a minimum width of ten.

Arguments:

Argument 1
To
String Variable

Standard Example:

Copy Date to String (MM/DD/YYYY)

To	DATE_STRING	*String Variable*

OptoScript Example:

`DateToStringMMDDYYYY(`*String***`)`**

```
DateToStringMMDDYYYY(DATE_STRING);
```

This is a procedure command; it does not return a value.

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.

Queue Error: -44 = String too short.

See Also: Copy Date to String (DD/MM/YYYY) (page C-59), Copy Time to String (page C-61), Set Date (page S-16), Set Time (page S-83)

Copy Time to String

Time/Date Action

Function: To read the time from the control engine'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 an ioControl 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 action 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
To
String Variable

Standard Example:

Copy Time to String

To	TIME_STRING	*String Variable*

OptoScript Example:

`TimeToString(`*String***`)`**

```
TimeToString(TIME_STRING);
```

This is a procedure command; it does not return a value.

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.

Queue Error: -44 = String too short.

See Also: Copy Date to String (MM/DD/YYYY) (page C-60), Copy Date to String (MM/DD/YYYY) (page C-60), Set Date (page S-16), Set Time (page S-83)

Cosine

Mathematical Action

Function: To derive the cosine of an angle.

Typical Use: Trigonometric function for computing triangular base of the angle.

Details:

- Calculates the cosine of *Argument 1* and places the result in *Argument 2*.
- *Argument 1* has a theoretical range of -infinity to +infinity, but is limited by the size of the argument you pass.
- The range of *Argument 2* is -1.0 to 1.0, inclusive.
- The following are examples of cosine calculations (rounded to four decimal places):

Radians	Degrees	Result
0.0	0.0	1.0
0.7854	45	0.7071
1.5708	90	0.0
2.3562	135	0.7071
3.1416	180	-1.0
3.9270	225	-0.7071
4.7124	270	0.0
5.4978	315	0.7071
6.2832	360	1.0

Arguments:

Argument 1 **Of**	Argument 2 **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Cosine

Of	RADIANS	*Float Variable*
Put Result in	COSINE	*Float Variable*

OptoScript Example:

`Cosine(`*`Of`*`)`

```
COSINE = Cosine(RADIANS);
```

This is a function command; it returns the cosine. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- To convert units of degrees to units of radians, divide degrees by 57.29578.
- Use Arccosine if the cosine is known and the angle is desired.

See Also: Arccosine (page A-11), Sine (page S-91), Tangent (page T-1)

Decrement Variable

Mathematical Action

Function: To decrease the value specified by 1.

Typical Use: To control countdown 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
[Value]
Float Variable
Integer 32 Variable
Integer 64 Variable

Standard Example:

Decrement Variable
Num_Holes_Left_to_Punch *Integer 32 Variable*

OptoScript Example:

DecrementVariable(*Variable***)**

```
DecrementVariable(Num_Holes_Left_to_Punch);
```

This is a procedure command; it does not return a value. This command is equivalent to the following math expression in OptoScript:

```
Num_Holes_Left_to_Punch = Num_Holes_Left_to_Punch - 1;
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- Executes faster than subtracting 1, both in standard commands and in OptoScript code.

See Also: Increment Variable (page I-1)

Delay (mSec)

Timing Action

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.
Arguments:	**Argument 1** **[Value]** Integer 32 Literal Integer 32 Variable
Standard Example:	Delay (mSec) 1 *Integer 32 Literal*
OptoScript Example:	**`DelayMsec(`***Milliseconds***`)`** `DelayMsec(1);` This is a procedure command; it does not return a value.
Notes:	• For readability, use Delay (Sec) for delays longer than 10 seconds. • When high accuracy is needed, reduce the number of tasks running concurrently.
Queue Errors:	-8 = Value less than zero.
See Also:	Delay (Sec) (page D-3), Start Off-Pulse (page S-96), Stop Timer (page S-102), Pause Timer (page P-1), Continue Timer (page C-39)

Delay (Sec)

Timing Action

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 seconds with millisecond resolution.

Arguments:

Argument 1
[Value]
Float Literal
Float Variable

Standard Example:

Delay (Sec)		
	10.525	*Float Literal*

OptoScript Example:

`DelaySec(`*Seconds***`)`**

```
DelaySec(10.525);
```

This is a procedure command; it does not return a value.

Notes:

- Use Delay (mSec) for delays shorter than 10 seconds.
- When high accuracy is needed, reduce the number of tasks running concurrently.

Queue Errors: -8 = Value less than zero.

See Also: Delay (mSec) (page D-2)

Disable Communication to All I/O Points

Simulation Action

Function: To disable communication between the program in the control engine and all analog and digital points.

Typical Use: To disconnect the program from all analog and digital points for simulation and testing.

To force the program in the control engine to read/write internal values (IVALs) rather than reading/writing to I/O units (XVALs). This command can be used for simulation and for faster processing of program logic in speed-sensitive applications.

Details:

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

Arguments: None

Standard Example: Disable Communication to All I/O Points

OptoScript Example: **`DisableCommunicationToAllIoPoints()`**

```
DisableCommunicationToAllIoPoints();
```

This is a procedure command; it does not return a value.

See Also: Enable Communication to All I/O Points (page E-1)

Disable Communication to All I/O Units

Simulation Action

Function: Changes a flag in the control engine to indicate that all the I/O units are offline. This stops communication from the program to the I/O units.

Typical Use: To force the program in the control engine to read/write internal values (IVALs) rather than reading/writing to I/O units (XVALs). This command can be used for simulation and for faster processing of program logic in speed-sensitive applications.

Details:

- No I/O unit communication errors will be generated by the program while communication to the I/O units is disabled.
- In Debug mode ioControl can still communicate to the I/O units, since it ignores the disabled flag.

Arguments: None.

Standard Example: Disable Communication to All I/O Units

OptoScript Example: **`DisableCommunicationToAllIoUnits()0`**

```
DisableCommunicationToAllIoUnits();
```

This is a procedure command; it does not return a value.

See Also: Enable Communication to All I/O Units (page E-2)

Disable Communication to Event/Reaction

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: To disable communication between the program in the 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 ioControl 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. However, the program in the controller will not be able to read or clear any status bits associated with the event/reaction until it is enabled (see Enable Communication to Event/Reaction).

Arguments:

Argument 1
[Value]
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Disable Communication to Event/Reaction
ESTOP_BUTTON_1 *Digital Event/Reaction*

OptoScript Example:

DisableCommunicationToEventReaction(*Event/Reaction***)**

```
DisableCommunicationToEventReaction(ESTOP_BUTTON);
```

This is a procedure command; it does not return a value.

Notes:

- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- To actually stop an event/reaction, use Disable Scanning 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 Communication to Event/Reaction (page E-3)

Disable Communication to I/O Unit

Simulation Action

Function: To disable communication between the program in the control engine and all points on the I/O unit.

Typical Uses:
- To prohibit the program in the control engine 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 control engine.

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 you change them using Debug mode or special simulation commands).
- Output IVALs will reflect what the program is instructing the outputs to do.
- *Caution:* Any outputs that are on may remain on.

Arguments:

Argument 1
[Value]
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
G4A8R, G4RAX*
G4D16R*
G4D32RS*
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-ENET-S64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-BRS*

* ioControl Professional only

Standard Example:

Disable Communication to I/O Unit
Vapor_Extraction *SNAP-UP1-ADS*

OptoScript Example:

`DisableCommunicationToIoUnit(`*I/O Unit***`)`**

```
DisableCommunicationToIoUnit(Vapor_Extraction);
```

This is a procedure command; it does not return a value.

Notes:
- Communication to I/O units is normally disabled using ioControl.
- If I/O units are disabled to speed logic execution, use the following commands in the order shown:

  1. Move I/O Unit to Numeric Table (with I/O unit still disabled): Copies analog output IVALs updated by program.

2. Get I/O Unit as Binary Value (with I/O unit still disabled): Copies digital output IVALs updated by program.

3. Enable Communication to I/O Unit: Re-establishes communications.

4. Move Numeric Table to I/O Unit: Writes to the table Moved to above. Updates analog outputs.

5. Set Digital-64 I/O Unit from MOMO Masks: writes to the value read above. Updates digital outputs.

6. Move I/O Unit to Numeric Table: Updates analog input IVALs.

7. Get Digital-64 I/O Unit as Binary Value: Updates digital input IVALs.

8. Disable Communication to I/O Unit: Disconnects communications.

9. Program logic . . . (Not for use with commands that access MIN, MAX, AVERAGE, COUNTS, etc.)

Repeat 1 through 9.

See Also: Enable Communication to I/O Unit (page E-4),

Disable Communication to Mistic PID Loop

Simulation Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To disable communication between the program in the control engine 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.
- Because the PID loop runs on the I/O unit, independently of the control engine, this command does not affect the PID in any way. While communication to the PID is disabled, any ioControl command that refers to it by name will not affect it, because the command will have access only to the IVAL.
- No changes can be made to the PID by the program in the control engine while the PID is disabled.

Arguments:

Argument 1
[Value]
PID Loop

Standard Example:

Disable Communication to Mistic PID Loop
HEATER_3 *PID Loop*

OptoScript Example:

DisableCommunicationToMisticPidLoop(*PID Loop***)**

```
DisableCommunicationToMisticPidLoop(HEATER_3);
```

This is a procedure command; it does not return a value.

Notes: To stop updating the PID output, do not use this command. Instead, use Set PID Mode to set the mode to manual.

See Also: Enable Communication to Mistic PID Loop (page E-5)

Disable Communication to PID Loop

Simulation Action

Function: To disable communication between the program in the control engine 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.
- Because the PID loop runs on the I/O unit, independently of the control engine, this command does not affect the PID in any way. Even on a SNAP Ultimate brain, the PID runs on the I/O side, not the control side. While communication to the PID is disabled, any ioControl command that refers to it by name will not affect it, because the command will have access only to the IVAL.
- No changes can be made to the PID by the program in the control engine while the PID is disabled.

Arguments:

Argument 1
[Value]
PID Loop

Standard Example:

Disable Communication to PID Loop		
	HEATER_3	*PID Loop*

OptoScript Example:

`DisableCommunicationToPidLoop(`*PID Loop***`)`**

```
DisableCommunicationToPidLoop(HEATER_3);
```

This is a procedure command; it does not return a value.

Notes: To stop updating the PID output, do not use this command. Instead, use Set PID Mode to set the mode to manual.

See Also: Enable Communication to PID Loop (page E-6), Set PID Mode (page S-70)

Disable Communication to Point

Simulation Action

Function: To disable communication between the program in the control engine and an individual analog or digital point.

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

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

Arguments:

Argument 1
[Value]
Analog Input
Analog Output
Digital Input
Digital Output

Standard Example:

Disable Communication to Point
TANK_LEVEL *Analog Input*

OptoScript Example:

DisableCommunicationToPoint(*Point***)**

```
DisableCommunicationToPoint(TANK_LEVEL);
```

This is a procedure command; it does not return a value.

Notes:
- Use Turn Off instead if the objective is to shut off a digital output.
- Disabling a point is ideal for a startup situation, since the program thinks it is reading an input or updating an output as it normally would.
- Use the IVAL field in Debug mode to change the value of an input.
- Use the XVAL field in Debug mode to change the value of an output.

See Also: Enable Communication to Point (page E-7), I/O Point Communication Enabled? (page I-2)

Disable Event/Reaction Group

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: Changes a flag internal to the controller to indicate that the event/reaction group is offline. This causes communication from the program to the event/reaction group to cease.

Typical Use: To force the program in the controller to read/write internal values (IVALs) rather than reading/writing to I/O units (XVALs). This can be used for simulation.

Details:

- No I/O unit communication errors will be generated by the program while communication to the event/reaction group is disabled.
- In Debug mode ioControl can still communicate to the event/reaction group since it ignores the disabled flag.

Arguments:

Argument 1
[Value]
Event/Reaction Group

Standard Example:

Disable Event/Reaction Group
Event/Reaction Group ER_E_STOP_GROUP_A

OptoScript Example:

`DisableEventReactionGroup(`*E/R Group***`)`**

```
DisableEventReactionGroup(ER_E_STOP_GROUP_A);
```

This is a procedure command; it does not return a value.

Notes: This command has no effect on the operation of the event/reaction group at the I/O unit.

See Also: Enable Event/Reaction Group (page E-8)

Disable I/O Unit Causing Current Error

Error Handling Action

Function: To disable communication between the program in the control engine and all points on the I/O unit if the I/O unit generated the top queue error.

Typical Use: Most I/O unit errors cause the unit to be automatically disabled is posted. This command can be used in an error handling chart to make sure an I/O unit causing an error is disabled.

Details: The control engine generates a error in the message queue 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.

Arguments: None.

Standard Example: Disable I/O Unit Causing Current Error

OptoScript Example: **`DisableIoUnitCausingCurrentError()`**

```
DisableIoUnitCausingCurrentError();
```

This is a procedure command; it does not return a value.

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 message queue is an I/O unit error before using this command, since the error could be caused by something else.
- Always use Remove Current Error and Point to Next Error after using this command.

Dependencies: For this command to have any effect, the top error in the queue must be an error generated by an I/O unit.

Queue Errors: -29 = The current error in the message queue is not an I/O error.

See Also: Enable I/O Unit Causing Current Error (page E-9), Error on I/O Unit? (page E-20)

Disable Mistic PID Output

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To prevent the PID from updating its associated analog output channel.

Typical Use: To allow manual changes to the analog output channel associated with the PID without disturbing the PID and without interference by the PID.

Details:

- A manually set output value will remain unchanged until it is either changed again manually or the PID output is enabled. When the PID output is enabled, any necessary output adjustments will be made to the current value. This is a bumpless operation.
- Clears bit 5 of the PID control word.

Arguments:

Argument 1
Of PID Loop
PID Loop

Standard Example:

Disable Mistic PID Output

Of PID Loop	Extruder_Zone08	*PID Loop*

OptoScript Example:

`DisableMisticPidOutput(`*Of PID Loop***`)`**

```
DisableMisticPidOutput(Extruder_Zone08);
```

This is a procedure command; it does not return a value.

Notes:

- This command is quite useful in presetting a PID output before activation or forcing a PID output to off.
- The PID calculation is ongoing while the PID output is “disabled.” The PID has no knowledge that its connection to the associated analog output channel has been disconnected.

See Also: Enable Mistic PID Output (page E-10)

Disable Mistic PID Output Tracking in Manual Mode

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To prevent the PID output from tracking the PID input while in manual mode.

Typical Use: To put the PID output back to normal mode.

Details:
- Factory default is PID output tracking *disabled.*
- When PID output tracking is disabled the PID output will not track the input while in manual mode. The PID output will remain unchanged by the PID calculation while in manual mode.
- Clears bit 4 of the PID control word.

Arguments:

Argument 1
On PID Loop
PID Loop

Standard Example:

Disable Mistic PID Output Tracking in Manual Mode
On PID Loop Extruder_Zone08 *PID Loop*

OptoScript Example:

DisableMisticPidOutputTrackingInManualMode(*On PID Loop***)**

```
DisableMisticPidOutputTrackingInManualMode(Extruder_Zone08);
```

This is a procedure command; it does not return a value.

Notes:
- This command is best used in the Powerup chart.
- The effects of this command can be stored at the I/O unit permanently by using Write I/O Unit Configuration to EEPROM.

See Also: Enable Mistic PID Output Tracking in Manual Mode (page E-11), Write I/O Unit Configuration to EEPROM (page W-2)

Disable Mistic PID Setpoint Tracking in Manual Mode

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To prevent the PID setpoint from tracking the PID input while in manual mode.

Typical Use: To prevent the setpoint from being altered automatically while in manual mode.

Details:

- Factory default is PID setpoint tracking *enabled.*
- When PID setpoint tracking is disabled the setpoint will not be altered by the PID at the I/O unit. This may be the most desirable state because it does not disturb the setpoint.
- Clears bit 3 of the PID control word.

Arguments:

Argument 1
On PID Loop
PID Loop

Standard Example:

Disable Mistic PID Setpoint Tracking in Manual Mode

On PID Loop	Extruder_Zone08	*PID Loop*

OptoScript Example:

DisableMisticPidSetpointTrackingInManualMode(*On PID Loop***)**

```
DisableMisticPidSetpointTrackingInManualMode(Extruder_Zone08);
```

This is a procedure command; it does not return a value.

Notes:

- This command is best used in the Powerup chart.
- The effects of this command can be stored at the I/O unit permanently by using Write I/O Unit Configuration to EEPROM.

See Also: Enable Mistic PID Setpoint Tracking in Manual Mode (page E-12), Write I/O Unit Configuration to EEPROM (page W-2)

Disable Scanning for All Events

Event/Reaction Action

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
On I/O Unit
B100
B200
B3000 (Analog)
B3000 (Digital)
G4A8R, G4RAX
G4D16R
SNAP-BRS

Standard Example:

Disable Scanning for All Events
On I/O Unit Overtemp_Sensors *G4A8R, G4RAX*

OptoScript Example:

DisableScanningForAllEvents(*On I/O Unit***)**

```
DisableScanningForAllEvents(Overtemp_Sensors);
```

This is a procedure command; it does not return a value.

Notes: To stop a specific event/reaction, use Disable Scanning for Event.

Dependencies: Event/reactions are not supported on simple I/O units.

See Also: Disable Scanning for Event (page D-18), Enable Scanning for Event (page E-14), Enable Scanning for All Events (page E-13)

Disable Scanning for Event

Event/Reaction Action

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
Event/Reaction
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Disable Scanning for Event
Event/Reaction ESTOP_BUTTON_1 *Analog Event/Reaction*

OptoScript Example:

DisableScanningForEvent(*Event/Reaction***)**

```
DisableScanningForEvent(ESTOP_BUTTON_1);
```

This is a procedure command; it does not return a value.

Notes:

- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- To disable all event/reactions, use Disable Scanning for All 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 simple I/O units.

See Also: Disable Scanning for All Events (page D-17), Enable Scanning for Event (page E-14), Enable Scanning for All Events (page E-13)

Disable Scanning of Event/Reaction Group

Event/Reaction Action

NOTE: This command is for mistic I/O units only.

Function: Stops all event/reactions in the specified group.

Typical Use: To stop scanning all event/reactions in the specified group with one command rather than issuing a separate command to stop each one.

Details: There can be up to 16 event/reaction groups, each containing as many as 16 event/reactions. If all related event/reactions are in the same group, this command could be quite useful.

Arguments:

Argument 1
Event/Reaction Group
Event/Reaction Group

Standard Example:

Disable Scanning of Event/Reaction Group
    Event/Reaction Group   ER_E_STOP_GROUP_A

OptoScript Example:

DisableScanningOfEventReactionGroup(*E/R Group***)**

```
DisableScanningOfEventReactionGroup(ER_E_STOP_GROUP_A);
```

This is a procedure command; it does not return a value.

See Also: Enable Scanning of Event/Reaction Group (page E-15)

Divide

Mathematical Action

Function: To divide two numerical values.

Typical Use: To perform a standard division action.

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 -15 (divide by zero) is added to the message queue.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
[Value]	**By**	**Put Result in**
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Divide

	Total_Distance	*Float Variable*
By	2.0	*Float Literal*
Put Result in	Half_Distance	*Float Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `/` operator.

```
Half_Distance = Total_Distance / 2.0;
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*. For more information on mathematical expressions in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- 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 Divide for integer math when the divisor is 2, 4, 8, 16, 32, 64, etc.

Queue Errors: -15 = Divide by zero.

See Also: Modulo (page M-5), Multiply (page M-25), Bit Shift (page B-15)

Down Timer Expired?

Timing Condition

Function: To check if a down timer has expired (reached zero).

Typical Use: Used to measure a time interval with good precision. Better than time delay commands for delays within looping charts.

Details: When a down timer has reached zero, it is considered expired.

Arguments:

Argument 1
Down Timer
Down Timer Variable

Standard Example:

Down Timer Expired?

Down Timer	OVEN_TIMER	*Down Timer Variable*

OptoScript Example:

HasDownTimerExpired(*Down Timer***)**

```
if (HasDownTimerExpired(OVEN_TIMER)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Timing Commands" in Chapter 10 of the *ioControl User's Guide* for more information on using timer commands.

See Also: Start Off-Pulse (page S-96), Stop Timer (page S-102), Continue Timer (page C-39), Pause Timer (page P-1), Set Down Timer Preset Value (page S-21), Delay (Sec) (page D-3), Delay (mSec) (page D-2)

E

Enable Communication to All I/O Points

Simulation Action

Function:	To enable communication between the program in the control engine and all analog and digital points.
Typical Use:	To re-connect the program to all analog and digital points after simulation and testing.
Details:	All analog and digital point communication is enabled by default.
Arguments:	None
Standard Example:	Enable Communication to All I/O Points
OptoScript Example:	**`EnableCommunicationToAllIoPoints()`** `EnableCommunicationToAllIoPoints();` This is a procedure command; it does not return a value.
See Also:	Disable Communication to All I/O Points, I/O Point Communication Enabled?

Enable Communication to All I/O Units

Simulation Action

Function: Changes a flag in the control engine to indicate that all the I/O units are online. This allows normal communication from the program to the I/O units.

Typical Use: To cause the program in the control engine to attempt to read/write to I/O units (XVALs) rather than use internal values (IVALs). Very useful to re-establish communication with all I/O units that have just been turned on without having to specify their name.

Details: Sets the Enabled flag which allows the next program reference to the I/O unit to attempt to communicate with the I/O unit.

Arguments: None.

Standard Example: Enable Communication to All I/O Units

OptoScript Example: **`EnableCommunicationToAllIoUnits()`**

```
EnableCommunicationToAllIoUnits();
```

This is a procedure command; it does not return a value.

Notes:

- Can be used in a chart that executes periodically to automatically bring I/O units that have just been turned on back online.
- Use of this command periodically within a program will prevent the disabling of communication to any point or any I/O unit by any means.

See Also: Disable Communication to All I/O Units

Enable Communication to Event/Reaction

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: To enable communication between the program in the 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
[Value]
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Enable Communication to Event/Reaction
ESTOP_BUTTON_1 *Analog Event/Reaction*

OptoScript Example:

EnableCommunicationToEventReaction(*Event/Reaction***)**

```
EnableCommunicationToEventReaction(ESTOP_BUTTON_1);
```

This is a procedure command; it does not return a value.

Notes:
- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- To enable all event/reactions, use Enable Scanning for All 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: Disable Communication to Event/Reaction, Enable Scanning for All Events

Enable Communication to I/O Unit

Simulation Action

Function: To enable communication between the program in the control engine and all points on the I/O unit.

Typical Use: To re-establish communication between the control engine and the I/O unit after it was automatically or manually disabled.

Details:
- The control engine attempts to communicate with the I/O unit. If the communication succeeds, all points will be configured. 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 will be added to the bottom of the message queue.

Arguments:

Argument 1
[Value]
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
G4A8R, G4RAX*
G4D16R*
G4D32RS*
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-ENET-S64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-PAC-R1
SNAP-PAC-R2
SNAP-BRS*

* ioControl Professional only

Standard Example:

Enable Communication to I/O Unit
Vapor_Extraction *SNAP-UP1-ADS*

OptoScript Example:

EnableCommunicationToIoUnit(*I/O Unit***)**

```
EnableCommunicationToIoUnit(Vapor_Extraction);
```

This is a procedure command; it does not return a value.

Notes: This command is sometimes useful for debugging and/or system startup.

Queue Errors:

-37 = Timeout on lock

-58 = No data received.

See Also: Disable Communication to I/O Unit

Enable Communication to Mistic PID Loop

Simulation Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To enable communication between the program in the control engine 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.
- Because the PID loop runs on the I/O unit, independently of the control engine, this command does not affect the PID in any way. Even on a SNAP Ultimate brain, the PID runs on the I/O side, not the control side. While communication to the PID is enabled, any ioControl command that refers to it by name will have full access.

Arguments:
Argument 1
[Value]
PID Loop

Standard Example:
Enable Communication to Mistic PID Loop
HEATER_3 *PID Loop*

OptoScript Example:
EnableCommunicationToMisticPidLoop(*PID Loop***)**

```
EnableCommunicationToMisticPidLoop(HEATER_3);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to Mistic PID Loop

Enable Communication to PID Loop

Simulation Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To enable communication between the program in the control engine 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.
- Because the PID loop runs on the I/O unit, independently of the control engine, this command does not affect the PID in any way. Even on a SNAP Ultimate brain, the PID runs on the I/O side, not the control side. While communication to the PID is enabled, any ioControl command that refers to it by name will have full access.

Arguments:

Argument 1
[Value]
PID Loop

Standard Example:

Enable Communication to PID Loop
HEATER_3 *PID Loop*

OptoScript Example:

EnableCommunicationToPidLoop(*PID Loop***)**

```
EnableCommunicationToPidLoop(HEATER_3);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to Mistic PID Loop

Enable Communication to Point

Simulation Action

Function: To enable communication between the program in the control engine and an individual analog or digital point.

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

Details:

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

Arguments:

Argument 1
[Value]
Analog Input
Analog Output
Digital Input
Digital Output

Standard Example:

Enable Communication to Point

TANK_LEVEL	*Analog Input*

OptoScript Example:

`EnableCommunicationToPoint(`*`Point`*`)`

```
EnableCommunicationToPoint(TANK_LEVEL);
```

This is a procedure command; it does not return a value.

Notes:

- Use Turn On instead to turn on digital output.
- Use this command to enable an analog or digital point previously disabled by the Disable Communication to Point command.

See Also: Disable Communication to Point, I/O Point Communication Enabled?

Enable Event/Reaction Group

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: Changes a flag internal to the controller to indicate that the event/reaction group is online. This allows normal communication from the program to the event/reaction group in the I/O unit.

Typical Use: To re-enable communication from the program in the controller to the event/reaction group in the I/O unit after it was disabled using Disable Event/Reaction Group.

Details: Sets the event/reaction group Enabled flag which allows the next program reference to anything in that group to attempt to communicate with the I/O unit.

Arguments:

Argument 1
[Value]
Event/Reaction Group

Standard Example:

Enable Event/Reaction Group
ER_E_STOP_GROUP_A

OptoScript Example:

EnableEventReactionGroup(*E/R Group***)**

```
EnableEventReactionGroup(ER_E_STOP_GROUP_A);
```

This is a procedure command; it does not return a value.

Notes: This command has no affect on the operation of the event/reaction group at the I/O unit.

See Also: Disable Event/Reaction Group

Enable I/O Unit Causing Current Error

Error Handling Action

Function: To enable communication between the program in the control engine and all points on the I/O unit if the top queue error was caused by an I/O unit.

Typical Use: To re-establish communication between the control engine and the I/O unit after it was automatically or manually disabled.

Details:
- The control engine generates a queue error 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 will be added to the bottom of the message queue.

Arguments: None.

Standard Example: Enable I/O Unit Causing Current Error

OptoScript Example: **`EnableIoUnitCausingCurrentError()`**

```
EnableIoUnitCausingCurrentError();
```

This is a procedure command; it does not return a value.

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 message queue is an I/O unit error before using this command.
- Always use Remove Current Error and Point to Next Error after using this command.

Dependencies: For this command to have any effect, the top error in the queue must have been caused by an I/O unit.

Queue Errors: -29 = The current error in the message queue is not an I/O error.

-37 = Timeout on lock.

See Also: Disable I/O Unit Causing Current Error, Error on I/O Unit?

Enable Mistic PID Output

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To enable the PID to update its associated analog output channel.

Typical Use: To reconnect the PID with its associated analog output channel after manual changes were made to the analog output channel via program or debugger.

Details:

- A manually set output value will remain unchanged until it is either changed again manually or the PID output is enabled. When the PID output is enabled, any necessary output adjustments will be made to the current value. This is a bumpless operation.
- Sets bit 5 of the PID control word.

Arguments:

Argument 1
On PID Loop
PID Loop

Standard Example:

Enable Mistic PID Output

On PID Loop	EXTRUDER_ZONE08	*PID Loop*

OptoScript Example:

EnableMisticPidOutput(*On PID Loop***)**

```
EnableMisticPidOutput(EXTRUDER_ZONE08);
```

This is a procedure command; it does not return a value.

Notes: The PID calculation is ongoing while the PID output is “disabled.” The PID has no knowledge that its connection to the associated analog output channel has been disconnected.

See Also: Disable Mistic PID Output

Enable Mistic PID Output Tracking in Manual Mode

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To cause the PID output to track the PID input while in manual mode.

Typical Use: As a non-PID related signal converter.

Details:

- Factory default is PID output tracking *disabled.*
- When PID output tracking is enabled the PID output will track the input while in manual mode. This is useful as a signal converter where the input is a temperature sensor for example and the output is 0–10 volts.
- Sets bit 4 of the PID control word.

Arguments:

Argument 1
On PID Loop
PID Loop

Standard Example:

Enable Mistic PID Output Tracking in Manual Mode
On PID Loop EXTRUDER_ZONE08 *PID Loop*

OptoScript Example:

EnableMisticPidOutputTrackingInManualMode(*On PID Loop***)**

```
EnableMisticPidOutputTrackingInManualMode(EXTRUDER_ZONE08);
```

This is a procedure command; it does not return a value.

Notes:

- This command is best used in the Powerup chart.
- The effects of this command can be stored at the I/O unit permanently by using Write I/O Unit Configuration to EEPROM.

See Also: Disable Mistic PID Output Tracking in Manual Mode, Write I/O Unit Configuration to EEPROM

Enable Mistic PID Setpoint Tracking in Manual Mode

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: To cause the PID setpoint to track the PID input while in manual mode.

Typical Use: To prevent a "bump" on the PID output when switching from manual to auto mode.

Details:

- Factory default is PID setpoint tracking *enabled.*
- When PID setpoint tracking is enabled the setpoint will follow the PID input to ensure zero error. Therefore, when switching from manual to auto, the PID output will not change. This is called a "bumpless transfer."
- This may not be the most desirable state because the setpoint is altered, which means the setpoint must be changed back to where it was, which will cause a bump in the PID output.
- Sets bit 3 of the PID control word.

Arguments:

Argument 1
On PID Loop
PID Loop

Standard Example:

Enable Mistic PID Setpoint Tracking in Manual Mode

On PID Loop	EXTRUDER_ZONE08	*PID Loop*

OptoScript Example:

EnableMisticPidSetpointTrackingInManualMode(*On PID Loop***)**

```
EnableMisticPidSetpointTrackingInManualMode(EXTRUDER_ZONE08);
```

This is a procedure command; it does not return a value.

Notes:

- This command is best used in the Powerup chart.
- The effects of this command can be stored at the I/O unit permanently by using Write I/O Unit Configuration to EEPROM.

See Also: Disable Mistic PID Setpoint Tracking in Manual Mode, Write I/O Unit Configuration to EEPROM

Enable Scanning for All Events

Event/Reaction Action

NOTE: This command is for mistic I/O units only.

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
On I/O Unit
B100
B200
B3000 (Analog)
B3000 (Digital)
G4A8R, G4RAX
G4D16R

Standard Example:

Enable Scanning for All Events

On I/O Unit	Overtemp_Sensors	*G4D16R*

OptoScript Example:

EnableScanningForAllEvents(*On I/O Unit***)**

```
EnableScanningForAllEvents(Overtemp_Sensors);
```

This is a procedure command; it does not return a value.

Notes:

- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- To activate a specific event/reaction, use Enable Scanning for Event.
- Normally used after Disable Scanning for All Events.

Dependencies: Event/reactions are not supported on simple I/O units.

See Also: Disable Scanning for Event, Enable Scanning for Event, Disable Scanning for All Events

Enable Scanning for Event

Event/Reaction Action

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
Event/Reaction
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Enable Scanning for Event

Event/Reaction	Acid_Tank_1_High_Level	*Digital Event/Reaction*

OptoScript Example:

EnableScanningForEvent(*Event/Reaction***)**

```
EnableScanningForEvent(Acid_Tank_1_High_Level);
```

This is a procedure command; it does not return a value.

Notes:

- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- To activate all event/reactions, use Enable Scanning for All 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 simple I/O units.

See Also: Enable Scanning for All Events

Enable Scanning of Event/Reaction Group

Event/Reaction Action

NOTE: This command is for mistic I/O units only.

Function: Starts all event/reactions in the specified group.

Typical Use: To start scanning all event/reactions in the specified group with one command rather than issuing a separate command to start each one.

Details: There can be up to 16 event/reaction groups, each containing as many as 16 event/reactions. If all related event/reactions are in the same group, this command could be quite useful.

Arguments:

Argument 1
Event/Reaction Group
Event/Reaction Group

Standard Example:

Enable Scanning of Event/Reaction Group

Event/Reaction Group	ER_E_STOP_GROUP_A

OptoScript Example:

`EnableScanningOfEventReactionGroup()`

```
EnableScanningOfEventReactionGroup(ER_E_STOP_GROUP_A);
```

This is a procedure command; it does not return a value.

See Also: Disable Scanning of Event/Reaction Group

Equal?

Logical Condition

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 Is	Argument 2 To
Analog Input	Analog Input
Analog Output	Analog Output
Digital Input	Digital Input
Digital Output	Digital Output
Down Timer Variable	Down Timer Variable
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
Up Timer Variable	Up Timer Variable

Standard Example:

Equal?	*Is*	BATCH_STEP	*Integer 32 Variable*
	To	4	*Integer 32 Literal*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `==` operator.

```
if (BATCH_STEP == 4) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `==` operator has many uses. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- When testing floats or analog values, use either Greater Than or Equal? or Less Than or Equal? since exact matches are rare.
- Use Within Limits? to test for an approximate match.
- To test for inequality, use either Not Equal? or the False exit.

See Also: Greater? Less? Not Equal? Greater Than or Equal? Less Than or Equal? Within Limits?

Equal to Numeric Table Element?

Logical Condition

Function: To determine if a numeric value is exactly equal to the specified value in a float or 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 float or integer table *Argument 3*). Examples:

Value 1	Value 2	Result
0.0	0.0	True
0.0001	0.0	False
-98.765	-98.765	True
-32768	-32768	True
2222	2222	True

- Evaluates True if both values are exactly the same, False otherwise.

Arguments:

Argument 1 Is	Argument 2 At Index	Argument 3 Of Table
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Down Timer Variable		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard Example:

	Is	THIS_READING	*Float Variable*
Equal to Numeric Table Element?			
	At Index	TABLE_INDEX	*Integer 32 Variable*
	Of Table	TABLE_OF_READINGS	*Float Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `==` operator.

```
if (THIS_READING == TABLE_OF_READINGS[TABLE_INDEX]) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `==` operator has many uses. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- When testing floats or analog values, use either Greater Than or Equal to Numeric Table Element? or Less Than Or Equal To Numeric Table Element? since exact matches are rare.
- To test for inequality, use either Not Equal to Numeric Table Element? or the False exit.

Queue Errors: -12 = Invalid table index value—index was negative or greater than the table size.

See Also: Greater Than or Equal To Numeric Table Element?, Less Than or Equal to Numeric Table Element?

Erase Files in Permanent Storage

Control Engine Action

Function: To delete the files in flash memory.

Typical Use: To delete files in flash memory that are no longer needed.

Details:

- This command deletes ALL files in the brain's or controller's flash memory. However, firmware files, strategy files, and point configuration data are not affected. Files and folders in the file system in RAM are not deleted.
- It is not possible to delete only some files in flash memory.
- To determine what files are in flash memory and RAM, use ioManager. See the instructions in Opto 22 form #1440, the *ioManager User's Guide*.

Arguments:

Argument 1
Put Status In
Integer 32 Variable

Standard Example:

Erase Files in Permanent Storage		
Put Status In	Status	*Integer 32 Variable*

OptoScript Example:

EraseFilesInPermanentStorage()

```
EraseFilesInPermanentStorage()
```

This is a function command; it always returns a zero.

Notes:

- See "Control Engine Commands" in Chapter 10 of the *ioControl User's Guide.*
- This command always returns a zero.

See Also: Save Files To Permanent Storage, Load Files From Permanent Storage

Error?

Error Handling Condition

Function: To determine if there is an error in the message queue.

Typical Use: To determine if further error handling should be performed, for example, in an error handling chart.

Details: Evaluates True if there is an error in the message queue, False otherwise.

Arguments: None.

Standard Example: Error?

OptoScript Example: **`IsErrorPresent()`**

```
if (IsErrorPresent()) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Error Handling Commands" in Chapter 10 of the *ioControl User's Guide.*
- Use Error on I/O Unit? to determine if it is an I/O related error.
- Use Debug mode to view the message queue for detailed information.

See Also: Error on I/O Unit?

Error on I/O Unit?

Error Handling Condition

Function: To determine if the top error in the message queue is an I/O-related error.

Typical Use: To determine if further error handling for I/O units should be performed, for example, in an error handling chart.

Details: Evaluates True if the current error in the message queue is an I/O unit error, False otherwise.

Arguments: None.

Standard Example: Error on I/O Unit?

OptoScript Example:

IsErrorOnIoUnit()

```
if (IsErrorOnioUnit()) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Error Handling Commands" in Chapter 10 of the *ioControl User's Guide.*
- Use Caused an I/O Unit Error? to determine which I/O unit caused the error.
- Use Debug mode to view the message queue for detailed information.

See Also: Caused an I/O Unit Error?, Remove Current Error and Point to Next Error, Error?, Get ID of Block Causing Current Error, Get Line Causing Current Error, Get Name of Chart Causing Current Error, Get Name of I/O Unit Causing Current Error

Event Occurred?

Event/Reaction Condition

NOTE: This command is for mistic I/O units only.

Function: To determine if a specific event has occurred.

Typical Use: To determine which event caused a particular reaction.

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
Has
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Event Occurred?		
Has	Sequence_Finished	*Analog Event/Reaction*

OptoScript Example:

HasEventOccurred(*Event/Reaction***)**

```
if (HasEventOccurred(Sequence_Finished)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- The current state of the event is not relevant to this condition. See 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: Event Occurring? Clear Event Latch

Event Occurring?

Event/Reaction Condition

NOTE: This command is for mistic I/O units only.

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
Is
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Is	Sequence_Finished	*Analog Event/Reaction*

Event Occurring?

OptoScript Example:

IsEventOccurring(*Event/Reaction***)**

```
if (IsEventOccurring(Sequence_Finished)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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: Event Occurred?

Event/Reaction Communication Enabled?

Simulation Condition

NOTE: This command is for mistic I/O units only.

Function: Checks a flag internal to the controller to determine if communication to the specified event/reaction is enabled.

Typical Use: Primarily used in factory QA testing and simulation.

Details: Evaluates True if communication is enabled.

Arguments:

Argument 1
Event/Reaction
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Event/Reaction Communication Enabled?	*Event/Reaction*	ER_E_STOP_1

OptoScript Example:

IsEventReactionCommEnabled(*Event/Reaction***)**

```
if (IsEventReactionCommEnabled(ER_E_STOP_1)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Event/Reaction Group Communication Enabled?

Event/Reaction Group Communication Enabled?

Simulation Condition

NOTE: This command is for mistic I/O units only.

Function: Checks a flag internal to the controller to determine if communication to the specified event/reaction group is enabled.

Typical Use: Primarily used in factory QA testing and simulation.

Details: Evaluates True if communication is enabled.

Arguments:

Argument 1
E/R Group
Event/Reaction Group

Standard Example:

Event/Reaction Group Communication Enabled?	*E/R Group*	ER_E_STOP_GROUP

OptoScript Example:

`IsEventReactionGroupEnabled(`*E/R Group*`)`

```
if (IsEventReactionGroupEnabled(ER_E-STOP_GROUP)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Event/Reaction Communication Enabled?

Event Scanning Disabled?

Event/Reaction Condition

NOTE: This command is for mistic I/O units only.

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
Event/Reaction
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

	Event/Reaction	*Sequence_Finished*
Event Scanning Disabled?		

OptoScript Example:

IsEventScanningDisabled(*Event/Reaction***)**

```
if (IsEventScanningDisabled(Sequence_Finished)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more 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.

Notes: See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Event Scanning Enabled?

Event Scanning Enabled?

Event/Reaction Condition

NOTE: This command is for mistic I/O units only.

Function: To determine if a specific event/reaction is active.

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
Event/Reaction
Analog Event/Reaction
Digital Event/Reaction

Standard Example:

Event Scanning Enabled?	*Event/Reaction*	Sequence_Finished

OptoScript Example:

`IsEventScanningEnabled(`*Event/Reaction***`)`**

```
if (IsEventScanningEnabled(Sequence_Finished)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.

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?

F

Find Character in String

String Action

Function: Locate a character within a string.

Typical Use: When parsing strings to locate delimiters and punctuation characters.

Details:

- The search is case-sensitive.
- The search begins at the location specified so that multiple occurrences of the same character can be found.
- The last parameter will contain an integer specifying the position at which the character is located. Values returned will be from 0 (first position in the string) to the string length.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Find	**Start at Index**	**Of String**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	String Variable	

Standard Example:

Find Character in String

Find	97	*Integer 32 Literal*
Start at Index	0	*Integer 32 Literal*
Of String	MSG_RECEIVED	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

`FindCharacterInString(`*Find, Start at Index, Of String***`)`**

```
POSITION = FindCharacterInString(34, POSITION, MSG_RECEIVED);
```

This is a function command; it returns the position at which the character is located in the string.

Notes:

- When looking for multiple instances of the same character in the string, use the same variable for the 2nd and 4th parameters, and increment the variable after each find so that the same character won't be found again and again.
- The first position in the string is referred to as position 0.

Error Code: -42 = Invalid limit error. Start at Index value is outside of string width range.

-58 = Specified character could not be found.

See Also: Find Substring in String (page F-2)

Find Substring in String

String Action

Function: Locate a string of characters (substring) within a string.

Typical Use: When parsing strings to locate key words.

Details:
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- The search is case-sensitive.
- The search begins at the location specified so that multiple occurrences of the same substring can be found.
- The Put Result In parameter will contain either an integer specifying the position at which the substring starts, or an error code. Values returned will be from 0 (first position in the string) to the string length, or a negative error code.
- Strings that are longer than the specified width for the string variable are truncated and lose characters on the right-hand side.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Find	**Start at Index**	**Of String**	**Put Result in**
String Literal	Integer 32 Literal	String Literal	Integer 32 Variable
String Variable	Integer 32 Variable	String Variable	

Standard Example: This example shows the string in quotes for clarity only; do not use quotes in the standard command:

Find Substring in String

Find	"SHIFT"	*String Literal*
Start at Index	INDEX	*Integer 32 Variable*
Of String	MSG_RECEIVED	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example: **FindSubstringInString(***Find, Start at Index, Of String***)**

```
POSITION = FindSubstringInString("SHIFT", INDEX, MSG_RECEIVED);
```

This is a function command; it returns the position at which the substring starts within the string. Quotes are required in OptoScript code.

Notes: Check for a possible error returned in the Put Result In parameter.

Error Code: -42 = Invalid limit error. Start at Index value was negative or greater than the string length.

-45 = String is empty. Either the string variable searched or the substring is empty.

-57 = Specified substring was not found.

See Also: Find Character in String (page F-1)

Float Valid?

Miscellaneous Condition

Function: To verify that a float variable contains a valid value.

Typical Use: To check float validity after reading a float from an external device, such as a communication handle, a scratch pad location, or an analog point.

Details: This command performs a simple test on the float variable to see if it contains a valid IEEE format float number. If the bit pattern of the float value has at least these bits set, 0x7F800000 (01111111100000000000000000000000), then it is considered invalid and the command returns a false (0).

Arguments:

Argument 1
Is
Float Variable

Standard Example:

Float Valid?

Is	Oil_Pressure	*Float Variable*

OptoScript Example:

IsFloatValid(*Float***)**

```
if (IsFloatValid(Oil_Pressure)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Analog points on an unplugged module return a value of NAN (not a number--an invalid float).

See Also: Move 32 Bits (page M-7), Get I/O Unit Scratch Pad Float Element (page G-70), Read Number from I/O Unit Memory Map (page R-5)

Generate Checksum on String

String Action

Function: Calculate an eight-bit checksum value.

Typical Use: Communication that requires checksum error checking.

Details:

- Checksum type is eight-bit.
- The *Start Value* is also known as the "seed." It is usually zero.
- When calculating the checksum one character at a time (or a group of characters at a time), the *Start Value* must be the result of the calculation on the previous character(s).
- The *On String* can contain as little as one character.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Generate Checksum on String

Start Value	0	*Integer 32 Literal*
On String	MSG_TO_SEND	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

GenerateChecksumOnString(*Start Value, On String***)**

```
POSITION = GenerateChecksumOnString(0, MSG_TO_SEND);
```

This is a function command; it returns the checksum. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: The method used to calculate the checksum is:

1. Take the numerical sum of the ASCII numerical representation of each character in the string.
2. Divide the result by 256.
3. The integer remainder is the eight-bit checksum.

Alternate checksum methods:

- An 8-bit (one byte) checksum for a string can be appended to a string using the Append Character to String command.
- The checksum for an ASCII string can be appended to the string by using the following standard commands:
  1. Convert Number to Formatted Hex String with the length argument set to a value of 2.
  2. Append String to String.

- To calculate the LRC of a string, take the two's complement of the checksum:

  1 Generate checksum on the string.

  2 Subtract the checksum from 255. This is the one's complement of the checksum.

  3 Add one to the result. This is the two's complement of the checksum.

  Example: For a string containing only the capital letter "A", the checksum is 65. To calculate the LRC, subtract the checksum (65) from 255, which equals 190. Add one to this result, resulting in an LRC of 191.

See Also: Verify Checksum on String (page V-3)

Generate Forward CCITT on String

String Action

Function: Calculate a 16-bit CRC value.

Typical Use: Communication that requires CRC error checking.

Details:

- CRC type is 16-bit forward CCITT.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- When calculating the CRC one character at a time (or a group of characters at a time), the *Start Value* must be the result of the calculation on the previous character(s).
- The *On String* can contain as little as one character.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Generate Forward CCITT on String

Start Value	0	*Integer 32 Literal*
On String	MSG_TO_SEND	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

`GenerateForwardCcittOnString(`*Start Value, On String***`)`**

```
POSITION = GenerateForwardCcittOnString(0, MSG_TO_SEND);
```

This is a function command; it returns the forward CCITT. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: The forward CCITT can be appended to the string by using the following commands:

1 Convert Number to Formatted Hex String with the length argument set to a value of 4.

2 Get Substring on first two characters of formatted hex string (index 0, length 2).
Get Substring on next two characters of formatted hex string (index 2, length 2).

3 Convert Hex String to Number on both substrings.

4 Append Character to String on first substring, then second substring to source string.

Result Data: The "Put Result in" argument will contain the Forward CCITT that was calculated.

See Also: Generate Reverse CCITT on String (page G-7), Generate Forward CRC-16 on String (page G-4), Generate Reverse CRC-16 on Table (32 bit) (page G-9)

Generate Forward CRC-16 on String

String Action

Function: Calculate a 16-bit CRC value.

Typical Use: Communication that requires CRC error checking.

Details:

- CRC type is 16-bit forward.
- The *Start Value* is also known as the “seed.” It is usually zero or -1.
- When calculating the CRC one character at a time (or a group of characters at a time), the *Start Value* must be the result of the calculation on the previous character(s).
- The *On String* can contain as little as one character.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Generate Forward CRC-16 on String

Start Value	0	*Integer 32 Literal*
On String	MSG_TO_SEND	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

`GenerateForwardCrc16OnString(`*Start Value, On String***`)`**

```
POSITION = GenerateForwardCrc16OnString(0, MSG_TO_SEND);
```

This is a function command; it returns the forward CRC. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User’s Guide* for more information.

Notes:

- The CRC can be appended to the string one character at a time using Append Character to String. For the first character use Bit Shift -8 on the CRC and append the result. For the second character simply append the original CRC value.
- The CRC can also be appended to the string by using the following commands:
  1. Convert Number to Formatted Hex String with the length argument set to a value of 4.
  2. Get Substring on first two characters of formatted hex string (index 0, length 2).
     Get Substring on next two characters of formatted hex string (index 2, length 2).
  3. Convert Hex String to Number on both substrings.
  4. Append Character to String on first substring, then second substring to source string.

See Also: Generate Reverse CRC-16 on String (page G-8), Generate Forward CCITT on String (page G-3), Generate Reverse CRC-16 on Table (32 bit) (page G-9)

Generate N Pulses

Digital Point Action

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 *Number 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 **On Time (Seconds)**	Argument 2 **Off Time (Seconds)**	Argument 3 **Number of Pulses**	Argument 4 **On Point**
Float Literal	Float Literal	Float Literal	Digital Output
Float Variable	Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable	

Standard Example:

Generate N Pulses

On Time (Seconds)	0.250	*Float Literal*
Off Time (Seconds)	0.500	*Float Literal*
Number of Pulses	Number_of_Pulses	*Float Variable*
On Point	DIG_OUTPUT	*Digital Output*

OptoScript Example:

`GenerateNPulses(`*On Time (Seconds), Off Time (Seconds), Number of Pulses, On Point***`)`**

```
GenerateNPulses(0.250, 0.500, Number_of_Pulses, DIG_OUTPUT);
```

This is a procedure command; it does not return a value.

Notes:

- Pulse trains on mistic brains are cancelled when a Turn Off or Turn On is sent to the output. To cancel a pulse train on an Ethernet brain, use this command with both the on times and off times set to 0.
- 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:

- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Start Continuous Square Wave (page S-94)

Generate Random Number

Mathematical Action

Function: To get a random value between zero and one.

Typical Use: To generate random delay values for retries when multiple clients are requesting the same resource.

Details: Use Seed Random Number before using this command to give the random number generator a random value to start with. Since the sequence of "random" numbers generated for any given seed value is always the same, it is imperative that a random seed value be used to avoid generating the same sequence of numbers every time.

Arguments:

Argument 1
Put in
Float Variable

Standard Example:

Generate Random Number

Put in	LOTTO_SEED	*Float Variable*

OptoScript Example:

`GenerateRandomNumber()`

```
LOTTO_SEED = GenerateRandomNumber();
```

This is a function command; it returns the random number. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: To get a random integer between zero and 99, for example, multiply the float value returned by 99.0 and put the result in an integer.

Dependencies: Use Seed Random Number first.

See Also: Seed Random Number (page S-4)

Generate Reverse CCITT on String

String Action

Function: Calculate a 16-bit CRC value.

Typical Use: Communication that requires CRC error checking.

Details:

- CRC type is 16-bit reverse CCITT.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- When calculating the CRC one character at a time (or a group of characters at a time), the *Start Value* must be the result of the calculation on the previous character(s).
- The *On String* can contain as little as one character.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Generate Reverse CCITT on String

Start Value	0	*Integer 32 Literal*
On String	MSG_TO_SEND	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

`GenerateReverseCcittOnString(`*Start Value, On String***`)`**

```
POSITION = GenerateReversCcittOnString(0, MSG_TO_SEND);
```

This is a function command; it returns the reverse CCITT. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- The reverse CCITT can be appended to the string one character at a time using Append Character to String. For the first character use Bit Shift -8 on the CRC and append the result. For the second character simply append the original CRC value.
- The CCITT can also be appended to the string by using the following commands:
  1. Convert Number to Formatted Hex String using an integer and the length argument set to a value of 4.
  2. Get Substring on first two characters of formatted hex string (index 0, length 2). Get Substring on next two characters of formatted hex string (index 2, length 2).
  3. Convert Hex String to Number on both substrings.
  4. Append Character to String on first substring, then second substring to source string.

See Also: Generate Forward CCITT on String (page G-3), Generate Reverse CRC-16 on String (page G-8), Generate Reverse CRC-16 on Table (32 bit) (page G-9)

Generate Reverse CRC-16 on String

String Action

Function: Calculate a 16-bit CRC value.

Typical Use: Communication that requires CRC error checking.

Details:

- CRC type is 16-bit reverse.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- When calculating the CRC one character at a time (or a group of characters at a time), the *Start Value* must be the result of the calculation on the previous character(s).
- The *On String* can contain as little as one character.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Generate Reverse CRC-16 on String

Start Value	0	*Integer 32 Literal*
On String	MSG_TO _SEND	*String Variable*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

`GenerateReverseCrc16OnString(`*Start Value, On String***`)`**

```
POSITION = GenerateReverseCrc16OnString(0, MSG_TO_SEND);
```

This is a function command; it returns the CRC. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- The CRC can be appended to the string one character at a time using Append Character to String. For the first character use Bit Shift -8 on the CRC and append the result. For the second character simply append the original CRC value.
- The CRC can also be appended to the string by using the following commands:
  1. Convert Number to Formatted Hex String using an integer and the length argument set to a value of 4.
  2. Get Substring on first two characters of formatted hex string (index 0, length 2). Get Substring on next two characters of formatted hex string (index 2, length 2).
  3. Convert Hex String to Number on both substrings.
  4. Append Character to String on first substring, then second substring to source string.

See Also: Generate Forward CRC-16 on String (page G-4), Generate Reverse CCITT on String (page G-7), Generate Reverse CRC-16 on Table (32 bit) (page G-9)

Generate Reverse CRC-16 on Table (32 bit)

Miscellaneous Action

Function: Calculate a 16-bit CRC value.

Typical Use: Communication that requires CRC error checking. The command is a quick and convenient way to verify the integrity of table data transferrred serially.

Details:

- CRC type is 16-bit reverse.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- The table can contain as little as one element.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
Start Value	**Table**	**Starting Element**	**Number of Elements**	**Put Result in**
Integer 32 Literal	Float Table	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Table	Integer 32 Variable	Integer 32 Variable	

Standard Example:

Generate Reverse CRC-16 on Table (32 bit)

Start Value	0	*Integer 32 Literal*
Table	VALUES_TO _SEND	*FloatTable*
Starting Element	1	*Integer 32 Literal*
Number of Elements	31	*Integer 32 Literal*
Put Result in	POSITION	*Integer 32 Variable*

OptoScript Example:

`GenerateReverseCrc16OnTable32(`*Start Value, Table, Starting Element, Number of Elements***`)`**

```
POSITION = GenerateReverseCrc16OnTable32(0, VALUES_TO_SEND, 1, 31);
```

This is a function command; it returns the CRC. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This command is only useful once the data in the table is static.
- The easiest way to check data is to make the table one element longer than necessary, then generate the CRC and move its result to the extra table element. The command Transmit Numeric Table is typically used to transfer table elements, including the CRC value. When the data is received, use this command at the receiving end to generate the CRC again and compare it to the first CRC value. For example, on the control engine sending the data:

  1 Generate Reverse CRC-16 on Table (32 bit) on table elements 1–31.

  2 Use Move to Table Element to move the CRC value to table element 0.

  3 Use Transmit Numeric Table to send all 32 table elements (0–31).

  Then, on the control engine receiving the data:

  1 Receive Numeric Table.

  2 Generate Reverse CRC-16 on Table (32 bit) on table elements 1–31.

  3 Compare the calculated CRC against the value stored in element 0.

See Also: Generate Forward CRC-16 on String (page G-4), Generate Reverse CCITT on String (page G-7), Generate Forward CCITT on String (page G-3)

Get & Clear All HDD Module Off-Latches

High Density Digital Module Action

Function: To read and reset the off-latches for all points on all high-density digital input modules on one I/O unit.

Typical Use: To read and reset off-latches for all high-density digital points on the I/O unit with a single command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places all off-latch data as bitmasks in an integer 32 table at a designated starting index. Argument 2 sets the index number and Argument 3 indicates the table.
- The table that receives the data must contain at least 16 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for point zero is placed in the first specified table element, with other points following in order. If a slot does not contain a high-density digital module, its corresponding table element is zero-filled.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Start Index**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Table	Integer 32 Variable

Standard Example:

Get & Clear All HDD Module Off-Latches

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Start Index	0	*Integer 32 Literal*
Put Result In	Bldg_A_OffL	*Integer 32 Table*
Put Status in	Status_Code	*Integer 32 Variable*

For example, if the I/O unit UIO_A consists of an 8-module rack with an analog module in slot 0 and HDD modules in slots 1–7, table Bldg_A_OffL might be filled as follows:

Index	Value (Bitmask)
0	00000000000000000000000000000000

← (This module is not a HDD module.)

Index	Value (Bitmask)
1	01100001010001110000001010110010
2	00000000000010000100010000000111
3	00100000011000000010010001000100
4	01100001010001110000001010110010
5	00001110000100001100100000001001
6	10000000110000011100000000100100
7	00110000011100001111100000000001
8	00000000000000000000000000000000
↓	↓
15	00000000000000000000000000000000

Each index contains the off-latch data for the HDD module in the corresponding position on the rack. A value of 1 indicates that the off-latch is on (set); a value of 0 indicates that it is off (not set). The least significant bit corresponds to point zero on the module.

In this example, index 2, which contains the off-latch data for all points on the module in slot 2, shows that off-latches for points 0, 1, 2, 10, 14, and 19 are on. All others are off.

The remainder of the table is zero-filled, since there are no more modules.

OptoScript Example:

`GetClearAllHddModuleOffLatches(` *I/O Unit, Start Index, Put Result In* **`)`**

```
Status_Code = GetClearAllHddModuleOffLatches(UIO_A, 0, Bldg_A_OffL);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read and reset the off-latches on only one HDD module, use Get & Clear HDD Module Off-Latches. To read off-latches without clearing them, use Get All HDD Module Off-Latches.
- You can manipulate bits within the table using commands such as Numeric Table Element Bit Test, or move the data in one element to a variable and use commands such as Bit Test.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-3 = Invalid table length.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get & Clear HDD Module Off-Latches (page G-22), Get All HDD Module Off-Latches (page G-32), Numeric Table Element Bit Test (page N-8), Bit Test (page B-17), and other Bit commands.

Get & Clear All HDD Module On-Latches

High Density Digital Module Action

Function: To read and reset on-latches for all points on all high-density digital input modules on an I/O unit.

Typical Use: To read and reset on-latches for all high-density digital points on the I/O unit with a single command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places all on-latch data as bitmasks in an integer 32 table at a designated starting index. Argument 2 sets the index number and Argument 3 indicates the table.
- The table that receives the data must contain at least 16 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for point zero is placed in the first specified table element, with other points following in order. If a slot does not contain a high-density digital module, its corresponding table element is zero-filled.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Start Index**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Table	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable		
SNAP-UP1-ADS			
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get & Clear All HDD Module On-Latches

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Start Index	0	*Integer 32 Literal*
Put Result In	Bldg_A_OnLatches	*Integer 32 Table*
Put Status in	Status_Code	*Integer 32 Variable*

For example, if the I/O unit UIO_A consists of an 8-module rack with an analog module in slot 0 and HDD modules in slots 1–7, table Bldg_A_OnLatches might be filled as follows:

Index	Value (Bitmask)
0	00000000000000000000000000000000
1	01100001010001110000001010110010
2	00000000000010000100010000000111
3	00100000011000000010010001000100
4	01100001010001110000001010110010
5	00001110000100000110010000001001
6	10000000110000011100000000100100
7	00110000011100001111100000000001

(This module is not a HDD module.)

Each index contains the on-latch data for the HDD module in the corresponding position on the rack. A value of 1 indicates that the on-latch is on (set); a value of 0 indicates that it is off (not set). The least significant bit corresponds to point zero on the module.

In this example, index 2, which contains the on-latch data for all points on the module in slot 2, shows that on-latches for points 0, 1, 2, 10, 14, and 19 are on. All others are off.

Index	Value (Bitmask)
8	00000000000000000000000000000000
↓	↓
15	00000000000000000000000000000000

The remainder of the table is zero-filled, since there are no more modules.

OptoScript Example:

`GetClearAllHddModuleOnLatches(`*I/O Unit, Start Index, Put Result In***`)`**

`Status_Code = GetClearAllHddModuleOnLatches(UIO_A, 0, Bldg_A_OnLatches);`

This is a function command; it returns one of the status codes shown below.

Notes:

- To read and reset the on-latches on only one HDD module, use Get & Clear HDD Module On-Latches. To read on-latches without clearing them, use Get All HDD Module On-Latches.
- You can manipulate bits within the table using commands such as Numeric Table Element Bit Test, or move the data in one element to a variable and use commands such as Bit Test.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-3 = Invalid table length.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get & Clear HDD Module On-Latches (page G-24), Get All HDD Module On-Latches (page G-34), Numeric Table Element Bit Test (page N-8), Bit Test (page B-17), and other Bit commands.

Get & Clear Analog Filtered Value

Analog Point Action

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:

- Filtering is used to smooth analog input signals that are erratic or change suddenly. The formula used for filtering is Y = (X - Y)/W + Y, where Y is the filtered value, X is the new unfiltered value, and W is the filter weight.
- 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. The analog input point is sampled 10 times a second with the filtered value stored locally on the I/O unit.
- The unfiltered analog input is still available using standard analog commands.

Arguments:

Argument 1	Argument 2
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get & Clear Analog Filtered Value

From	Temp_Sensor	*Analog Input*
Put in	Filtered_Temp	*Float Variable*

OptoScript Example:

`GetClearAnalogFilteredValue(`*From***`)`**

```
Filtered_Temp = GetClearAnalogFilteredValue(Temp_Sensor);
```

This is a function command; it returns the analog filtered value.The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Do not use this command for frequent reads (one per second or faster) since it continually resets the averaging. Use Get Analog Filtered 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 Debug mode.)

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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

Result Data: 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.

See Also: Get Analog Filtered Value (page G-38), Set Analog Filter Weight (page S-7)

Get & Clear Analog Maximum Value

Analog Point Action

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 value over a given period of time.

Details:

- The current value for each point is regularly read and stored at the I/O unit. Check the specifications for the module and I/O unit 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.

Arguments:

Argument 1	**Argument 2**
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get & Clear Analog Maximum Value

From	Pres_Sensor	*Analog Input*
Put in	MAX_KPA	*Float Variable*

OptoScript Example:

`GetClearAnalogMaxValue(`*From***`)`**

```
MAX_KPA = GetClearAnalogMaxValue(Pres_Sensor);
```

This is a function command; it returns the maximum value of the input since its last reading. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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

Result Data:

- The value returned will be the highest value recorded on this point since the last time the maximum value was cleared, or since the unit was turned on.
- Points 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.

See Also: Get & Clear Analog Minimum Value (page G-16), Get Analog Minimum Value (page G-40)

Get & Clear Analog Minimum Value

Analog Point Action

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 value over a given period of time.

Details:

- The current value for each point is regularly read and stored at the I/O unit. Check the specifications for the module and I/O unit 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.

Arguments:

Argument 1	**Argument 2**
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get & Clear Analog Minimum Value

From	PRES_SENSOR	*Analog Input*
Put in	MIN_KPA	*Float Variable*

OptoScript Example:

`GetClearAnalogMinValue(`*From*`)`

```
MIN_KPA = GetClearAnalogMinValue(Pres_Sensor);
```

This is a function command; it returns the minimum value of the input since its last reading. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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

Result Data:

- The value returned will be the lowest value recorded since the last time the minimum value was reset or since the unit was turned on.
- Points 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.

See Also: Get & Clear Analog Maximum Value (page G-15), Get Analog Maximum Value (page G-39)

Get & Clear Analog Totalizer Value

Analog Point Action

NOTE: This command is for mistic I/O units only.

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 by sampling the input point and storing the total value locally on the I/O unit. This command reads the current total, then clears it to zero.
- The sample rate is set using the Set Analog Totalizer Rate Command.
- Totalizing will be bidirectional if the input range is bidirectional, such as -10 to +10.
- Totalizing will stop when the total reaches a maximum of 3276 seconds.
- Totalizing will resume after using Get & Clear Analog Totalizer Value.
- Totalizing will stop when then input channel becomes under range or disabled. Totalizing will resume when the input signal is back within range.

Arguments:

Argument 1	Argument 2
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get & Clear Analog Totalizer Value

From	Flow_Rate	*Analog Input*
Put in	Total_Barrels	*Float Variable*

OptoScript Example:

`GetClearAnalogTotalizerValue(`*`From`*`)`

```
Total_Barrels = GetClearAnalogTotalizerValue(Flow_Rate);
```

This is a function command; it returns the totalizer value for the analog input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Before using this command, use Set Analog Totalizer Rate once to establish the sampling rate and start the totalizer. Use this command to clear the total before actual readings start.
- Use Get Analog Totalizer Value periodically to simply “watch” the total. When it exceeds 30,000, use Get & Clear Analog Totalizer Value 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 Totalizer Rate must be executed. Otherwise, a value of -32,768 will be returned to indicate an error.

- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

Result Data:
- The value returned will be an integer from -32,768 to 32,767.
- Channels without a module installed will return a value of -32,768 to indicate an error.

See Also: Get Analog Totalizer Value (page G-43), Set Analog Totalizer Rate (page S-12)

Get & Clear Counter

Digital Point Action

Function: To read and clear a standard digital input counter or quadrature counter value.

Typical Use: To count pulses from turbine flow meters, magnetic pickups, encoders, proximity switches, etc. To read incremental encoders for positional or velocity measurement.

Details:
- Standard digital only. For high-density digital, see Get & Clear HDD Module Counter.
- Reads the current value of a digital input counter or quadrature counter and places it in the *Put In* parameter.
- Sets the counter or quadrature counter at the I/O unit to zero. Does not stop the counter or quadrature counter from continuing to count.
- Valid range for a counter is 0 to 2,147,483,647 counts. Valid range for a quadrature counter is -2,147,483,647 to 2,147,483,648 counts.
- On serial (mistic) units, for a quadrature counter, a positive value indicates forward movement (phase B leads phase A), and a negative value indicates reverse movement (phase A leads phase B). On Ethernet-based (MMP) I/O units, the opposite is true (a positive value is returned when phase A leads phase B).
- A quadrature counter occupies two adjacent points. Input module pairs specifically made for quadrature counting must be used. The first point must be an even point number on the 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**
From Point	**Put in**
Counter	Float Variable
Quadrature Counter	Integer 32 Variable

Standard Example:

Get & Clear Counter

From Point	Bottle_Counter	*Counter*
Put in	Number_of_Bottles	*Integer 32 Variable*

OptoScript Example:

`GetClearCounter(`*From Point***`)`**

```
Number_of_Bottles = GetClearCounter(Bottle_Counter);
```

This is a function command; it returns the counter or quadrature counter value from the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- The maximum speed at which a 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.
- For a quadrature counter, 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 Counter once before using this command for the first time.
- Applies only to standard digital inputs configured as a counter or quadrature counter.

See Also: Get & Clear Counter (page G-18), Start Continuous Square Wave (page S-94), Stop Counter (page S-101), Clear Counter (page C-22)

Pro Get & Clear Event Latches

Event/Reaction Action

Function: Gets and clears all event latches in the specified group.

Typical Use: To get and clear all event latches in the specified group with one command rather than issuing a separate command for each one.

Details:
- There can be up to 16 event/reaction groups, each containing as many as 16 event latches. If all related event latches are in the same group, this command could be quite useful.
- The value returned is an integer with the lower 16 bits representing the 16 latches in the group. If the variable has a value greater than zero, one or more latches are set.
- Available on mistic multifunction I/O units. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

Arguments:

Argument 1	**Argument 2**
Event/Reaction Group	**Put in**
Event/Reaction Group	Integer 32 Variable

Standard Example:

Get & Clear Event Latches

Event/Reaction Group	ER_E_STOP_GROUP_A	
Put in	Group_Latch_Status	*Integer 32 Variable*

OptoScript Example:

`GetClearEventLatches(`*E/R Group***`)`**

```
Group_Latch_Status = GetClearEventLatches(ER_E_STOP_GROUP_A);
```

This is a function command; it returns the status of all 16 event latches in the event/reaction group, in the form of an integer with the lower 16 bits representing the latches. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Bit Test could be used to test each of the lower 16 bits numbered 0–15.

See Also: Get Event Latches (page G-54)

Get & Clear HDD Module Counter

High Density Digital Module Action

Function: To read and reset the counter for a specific point on a high-density digital input module.

Typical Use: To read and reset the counter for one point only.

Details:
- Works only on high-density digital input modules, not on standard digital modules.
- Places the counts in an integer 32 variable and then clears the counter.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
I/O Unit	**Module Number**	**Point Number**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable	Integer 32 Variable		
SNAP-UP1-ADS				
SNAP-UP1-M64				
SNAP-ENET-S64				
SNAP-PAC-R1				
SNAP-PAC-R2				

Standard Example:

Get & Clear HDD Module Counter

I/O Unit	Ins_42	*SNAP-ENET-S64*
Module Number	8	*Integer 32 Literal*
Point Number	Meter	*Integer 32 Variable*
Put Result In	Meter_8_Counts	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

OptoScript Example:

`GetClearHddModuleCounter(`*I/O Unit, Module Number, Point Number, Put Result In***`)`**

```
Status_Code = GetClearHddModuleCounter(Ins_42, 8, Meter, Meter_8_Counts);
```

This is a function command; it returns one of the status codes shown below.

Notes:
- To read and clear all counters on a module, use Get & Clear HDD Module Counters. To read counters without clearing them, use Get HDD Module Counters.
- See “High Density Digital Module Commands” in Chapter 10 of the *ioControl User’s Guide*, and see form #1547, the *SNAP High-Density Digital Module User’s Guide*.
- Counters with values of more than 2 billion may appear as negative numbers.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get & Clear HDD Module Counters (page G-21), Get HDD Module Counters (page G-57)

Get & Clear HDD Module Counters

High Density Digital Module Action

Function: To read and reset the counters for all points on a high-density digital input module.

Typical Use: To read and reset all counters on a module in one command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places counter data for all points in the module in an integer 32 table at a designated starting index, and then clears all counters. Argument 3 sets the index number and Argument 4 indicates the table.
- The table that receives the data must contain at least 32 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for point zero is placed in the first specified table element, with other points following in order.

Arguments:

Argument 1 I/O Unit	Argument 2 Module Number	Argument 3 Start Index	Argument 4 Put Result In	Argument 5 Put Status In
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Table	Integer 32 Variable

Standard Example:

Get & Clear HDD Module Counters

I/O Unit	In_42	*SNAP-ENET-S64*
Module Number	Section	*Integer 32 Variable*
Start Table Index	Index	*Integer 32 Variable*
Put Result In	Meter_Ct	*Integer 32 Table*
Put Status in	Status_Code	*Integer 32 Variable*

For example, if the value of the variable Index is zero, the first four elements of the Meter_Counts table might be filled as follows:

Index	Counter Value	
0	61	← Counter data for point 0
1	85	← Counter data for point 1
2	102	← Counter data for point 2
3	42	← Counter data for point 3

OptoScript Example:

`GetClearHddModuleCounters(`*I/O Unit, Module Number, Start Index, Put Result In***`)`**

```
Status_Code = GetClearHddModuleCounters(In_42, Section, Index, Meter_Ct);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read and clear just one counter on a module, use Get & Clear HDD Module Counter. To read counters without clearing them, use Get HDD Module Counters.

- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.
- Counters with values of more than 2 billion may appear as negative numbers.

Status Codes:

0 = Success

-3 = Invalid table length. Table must contain at least 32 elements.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get & Clear HDD Module Counter (page G-20), Get HDD Module Counters (page G-57)

Get & Clear HDD Module Off-Latches

High Density Digital Module Action

Function: To read and reset the off-latches of all points on a high-density digital input module.

Typical Use: To read and clear off-latches on a module in one command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places a bitmask in an integer 32 variable showing the state of off-latches for all points on the module, and resets the latches. The least significant bit in the mask corresponds to point 0. A value of 1 in a bit means the off-latch is on (set); a value of 0 in the bit means the off-latch is off (not set).

Arguments:

Argument 1 I/O Unit	Argument 2 Module Number	Argument 3 Put Result In	Argument 4 Put Status In
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Standard Example:

Get & Clear HDD Module Off-Latches

I/O Unit	Bldg_A	*SNAP-B3000-ENET, SNAP-ENET-RTC*
Module Number	9	*Integer 32 Literal*
Put Result In	Fan_OffLatches	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

An example of the result is illustrated below. Only the first 8 and last 8 off-latches are shown.

Bitmask	Hex	9				3				→	B				2			
	Binary	1	0	0	1	0	0	1	1	→	1	0	1	1	0	0	1	0
	Off-latch	on	off	off	on	off	off	on	on		on	off	on	on	off	off	on	off
	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0

OptoScript Example:

`GetClearHddModuleOffLatches(`*I/O Unit, Module Number, Put Result In***`)`**

```
Status_Code = GetClearHddModuleOffLatches(Bldg_A, 9, Fan_OffLatches);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read off-latches without clearing them, use Get HDD Module Off-Latches.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get HDD Module Off-Latches (page G-58), Get & Clear All HDD Module Off-Latches (page G-10), Get All HDD Module On-Latches (page G-34)

Get & Clear HDD Module On-Latches

High Density Digital Module Action

Function: To read and reset the on-latches of all points on a high-density digital input module.

Typical Use: To read and reset all on-latches on a module in one command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places a bitmask in an integer 32 variable that indicates the state of on-latches for all points on the module, and resets the latches. The least significant bit corresponds to point 0. A value of 1 in a bit means the on-latch is on (set); a value of 0 in the bit means the on-latch is off (not set).

Arguments:

Argument 1 I/O Unit	Argument 2 Module Number	Argument 3 Put Result In	Argument 4 Put Status In
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Standard Example:

Get & Clear HDD Module On-Latches

I/O Unit	Bldg_A	*SNAP-B3000-ENET, SNAP-ENET-RTC*
Module Number	9	*Integer 32 Literal*
Put Result In	Fan_OnLatches	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

An example of the result is illustrated below. Only the first 8 and last 8 on-latches are shown.

Bitmask	Hex	9				3				→	B				2			
	Binary	1	0	0	1	0	0	1	1	→	1	0	1	1	0	0	1	0
	On-latch	on	off	off	on	off	off	on	on		on	off	on	on	off	off	on	off
	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0

OptoScript Example:

`GetClearHddModuleOnLatches(`*I/O Unit, Module Number, Put Result In***`)`**

```
Status_Code = GetClearHddModuleOnLatches(Bldg_A, 9, Fan_OnLatches);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read on-latches without clearing them, use Get HDD Module On-Latches.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes: 0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get HDD Module On-Latches (page G-60), Get & Clear All HDD Module On-Latches (page G-12), Get All HDD Module Off-Latches (page G-32)

Get & Clear Off-Latch

Digital Point Action

Function: To read and re-arm a high-speed off-latch associated with a standard digital input.

Typical Use: To ensure detection of an extremely brief on-to-off transition of a digital input.

Details:

- Standard digital only. For high-density digital, see Get & Clear HDD Module Off-Latches.
- Reads and re-arms the off-latch of a single digital input.
- The next time the input point 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 control engine to detect, since they are processed by the I/O unit.
- If *Argument 2* is a digital output and the latch is not set, the output will turn off. If the latch is set, the output will turn on.

Arguments:

Argument 1 **From Point**	**Argument 2** **Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get & Clear Off-Latch

From Point	BUTTON_3_LATCH	*Digital Input*
Put in	ALARM_HORN	*Digital Output*

OptoScript Example:

GetClearOffLatch(*From Point***)**

```
ALARM_HORN = GetClearOffLatch(BUTTON_3_Latch);
```

This is a function command; it returns a value of true (non-zero) or false (0) indicating whether the off latch has been set. The returned value can be consumed by a digital output (as in the example shown) or by a variable, control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: The ability of the I/O units 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 standard digital inputs.

See Also: Get Off-Latch (page G-102), Clear Off-Latch (page C-26), Clear All Latches (page C-20)

Get & Clear On-Latch

Digital Point Action

Function: To read and re-arm a high-speed on-latch associated with a standard digital input.

Typical Use: To ensure detection of an extremely brief off-to-on transition of a digital input.

Details:

- Standard digital only. For high-density digital, see Get & Clear HDD Module On-Latches.
- Reads and re-arms the on-latch of a single digital input.
- The next time the input point changes from off to on, the on-latch will be set.
- Off-latches detect on-off-on input transitions that would otherwise occur too fast for the control engine to detect, since they are processed by the I/O unit.
- The value read is placed in the argument specified by the *Put In* 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 non-zero (True).

Arguments:

Argument 1	Argument 2
From Point	**Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get & Clear On-Latch

From Point	E_STOP_BUTTON	*Digital Input*
Put in	LATCH_VAR	*Integer 32 Variable*

OptoScript Example:

GetClearOnLatch(*From Point***)**

```
LATCH_VAR = GetClearOffLatch(E_STOP_BUTTON);
```

This is a function command; it returns a value of true (non-zero) or false (0) indicating whether the on latch has been set. The returned value can be consumed by a variable (as in the example shown) or by a digital output, control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: The ability of the 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 standard digital inputs.

See Also: Get On-Latch (page G-106), Clear On-Latch (page C-27), Clear All Latches (page C-20)

Get & Restart Off-Pulse Measurement

Digital Point Action

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**
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Standard Example:

Get & Restart Off-Pulse Measurement

From Point	STANDBY_SWITCH	*Off Pulse*
Put in	OFF_TIME	*Float Variable*

OptoScript Example:

`GetRestartOffPulseMeasurement(`*From Point***`)`**

```
OFF_TIME = GetRestartOffPulseMeasurement(STANDBY_SWITCH);
```

This is a function command; it returns the duration of the first complete off-pulse. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use Get Off-Pulse Measurement Complete Status 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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get Off-Pulse Measurement (page G-103), Get Off-Pulse Measurement Complete Status (page G-104)

Get & Restart Off-Time Totalizer

Digital Point Action

NOTE: This command is for mistic I/O units only.

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
From Point	**Put in**
Off Totalizer	Float Variable
	Integer 32 Variable

Standard Example:

Get & Restart Off-Time Totalizer

From Point	Power_Status	*Off Totalizer*
Put in	System_Down_Time	*Integer 32 Variable*

OptoScript Example:

`GetRestartOffTimeTotalizer(`*From Point***`)`**

```
System_Down_Time = GetRestartOffTimeTotalizer(Power_Status);
```

This is a function command; it returns the total off-time of the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 Off-Time Totalizer to read the totalized value without resetting it.

Dependencies:

- Applies only to inputs configured with the totalize-off feature.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get Off-Time Totalizer (page G-105)

Get & Restart On-Pulse Measurement

Digital Point Action

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
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Standard Example:

Get & Restart On-Pulse Measurement

From Point	Standby_Switch	*On Pulse*
Put in	On_Time	*Float Variable*

OptoScript Example:

GetRestartOnPulseMeasurement(*From Point***)**

```
On_Time = GetRestartOnPulseMeasurement(Standby_Switch);
```

This is a function command; it returns the duration of the first on-pulse. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use Get On-Pulse Measurement Complete Status 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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get On-Pulse Measurement (page G-107), Get On-Pulse Measurement Complete Status (page G-108)

Get & Restart On-Time Totalizer

Digital Point Action

NOTE: This command is for mistic I/O units only.

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
From Point	**Put in**
On Totalizer	Float Variable
	Integer 32 Variable

Standard Example:

Get & Restart On-Time Totalizer

From Point	Circ_Motor_Pwr	*On Totalize*
Put in	Motor_Runtime	*Integer 32 Variable*

OptoScript Example:

`GetRestartOnTimeTotalizer(`*From Point***`)`**

```
Motor_Runtime = GetRestartOnTimeTotalizer(Circ_Motor_Pwr);
```

This is a function command; it returns the total on-time of the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 On-Time Totalizer to read the totalized value without resetting it.

Dependencies:
- Applies only to inputs configured with the totalize-on feature.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get On-Time Totalizer (page G-109)

Pro Get & Restart Period

Digital Point Action

NOTE: This command is for mistic I/O units only.

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 *Put In* 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
From Point	**Put in**
Period	Float Variable
	Integer 32 Variable

Standard Example:

Get & Restart Period

From Point	SHAFT_INPUT	*Period*
Put in	SHAFT_CYCLE	*Integer 32 Variable*

OptoScript Example:

GetRestartPeriod(*From Point***)**

```
SHAFT_CYCLE = GetRestartPeriod(SHAFT_INPUT);
```

This is a function command; it returns the period. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This command should be used to start the period measurement.
- This command measures the first complete period only and restarts.
- 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.
- Available on mistic multifunction I/O units. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get Period (page G-110)

Get All HDD Module Off-Latches

High Density Digital Module Action

Function: To read the off-latches for all points on all high-density digital input modules on one I/O unit.

Typical Use: To get off-latches for all high-density digital points on the I/O unit with a single command, without clearing the latches.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places all off-latch data as bitmasks in an integer 32 table at a designated starting index. Argument 2 sets the index number and Argument 3 indicates the table.
- The table that receives the data must contain at least 16 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for the module in position zero is placed in the first specified table element, with other modules following in order. If a slot does not contain a high-density digital module, its corresponding table element is zero-filled.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Start Index**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Table	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable		
SNAP-UP1-ADS			
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get All HDD Module Off-Latches

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Start Index	0	*Integer 32 Literal*
Put Result In	Bldg_A_OffLatches	*Integer 32 Table*
Put Status in	Status_Code	*Integer 32 Variable*

For example, if the I/O unit UIO_A consists of an 8-module rack with an analog module in slot 0 and HDD modules in slots 1–7, table Bldg_A_OffLatches might be filled as follows:

Index	Value (Bitmask)
0	00000000000000000000000000000000
1	01100001010001110000001010110010
2	00000000000010000100010000000111
3	00100000011000000010010001000100
4	01100001010001110000001010110010
5	00001110000100001100100000001001
6	10000000110000011100000000100100
7	00110000011100001111100000000001

(This module is not a HDD module.)

Each index contains the off-latch data for the HDD module in the corresponding position on the rack. A value of 1 indicates that the off-latch is on (set); a value of 0 indicates that it is off (not set). The least significant bit corresponds to point zero on the module.

In this example, index 2, which contains the off-latch data for all points on the module in slot 2, shows that off-latches for points 0, 1,2, 10, 14, and 19 are on. All others are off.

Index	Value (Bitmask)
8	00000000000000000000000000000000
↓	↓
15	00000000000000000000000000000000

The remainder of the table is zero-filled, since there are no more modules.

OptoScript Example:

`GetAllHddModuleOffLatches(`*I/O Unit, Start Index, Put Result In***`)`**

`Status_Code = GetAllHddModuleOffLatches(UIO_A, 0, Bldg_A_OffLatches);`

This is a function command; it returns one of the status codes shown below.

Notes:

- To read the off-latches on only one HDD module, use Get HDD Module Off-Latches. To read and clear off-latches, use Get & Clear All HDD Module Off-Latches.
- You can manipulate bits within the table using commands such as Numeric Table Element Bit Test, or move the data in one element to a variable and use commands such as Bit Test.
- See “High Density Digital Module Commands” in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-3 = Invalid table length.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get HDD Module Off-Latches (page G-58), Get & Clear All HDD Module Off-Latches (page G-10), Numeric Table Element Bit Test (page N-8), Bit Test (page B-17), and other Bit commands.

Get All HDD Module On-Latches

High Density Digital Module Action

Function: To read the on-latches for all points on all high-density digital input modules on one I/O unit.

Typical Use: To get on-latches for all high-density digital points on the I/O unit with a single command, without clearing the latches.

Details:

- Works only on high-density digital input modules, not on standard digital modules.
- Places all on-latch data as bitmasks in an integer 32 table at a designated starting index. Argument 2 sets the index number and Argument 3 indicates the table.
- The table that receives the data must contain at least 16 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for the module in position zero is placed in the first specified table element, with other modules following in order. If a slot does not contain a high-density digital module, its corresponding table element is zero-filled.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Start Index**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Table	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable		
SNAP-UP1-ADS			
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get All HDD Module On-Latches

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Start Index	0	*Integer 32 Literal*
Put Result In	Bldg_A_OnLatches	*Integer 32 Table*
Put Status in	Status_Code	*Integer 32 Variable*

For example, if the I/O unit UIO_A consists of an 8-module rack with an analog module in slot 0 and HDD modules in slots 1–7, table Bldg_A_OnLatches might be filled as follows:

Index	Value (Bitmask)
0	00000000000000000000000000000000
1	01100001010001110000001010110010
2	00000000000010000100010000000111
3	00100000011000000010010001000100
4	01100001010001110000001010110010
5	00001110000100001100100000001001
6	10000000110000011100000000100100
7	00110000011100001111100000000001

(This module is not a HDD module.)

Each index contains the on-latch data for the HDD module in the corresponding position on the rack. A value of 1 indicates that the on-latch is on (set); a value of 0 indicates that it is off (not set). The least significant bit corresponds to point zero on the module.

In this example, index 2, which contains the on-latch data for all points on the module in slot 2, shows that on-latches for points 0, 1, 2, 10, 14, and 19 are on. All others are off.

Index	Value (Bitmask)
8	00000000000000000000000000000000
↓	↓
15	00000000000000000000000000000000

The remainder of the table is zero-filled, since there are no more modules.

OptoScript Example:

`GetAllHddModuleOnLatches(`*I/O Unit, Start Index, Put Result In***`)`**

```
Status_Code = GetAllHddModuleOnLatches(UIO_A, 0, Bldg_A_OnLatches);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read the on-latches on only one HDD module, use Get HDD Module On-Latches. To read and clear on-latches, use Get & Clear All HDD Module On-Latches.
- You can manipulate bits within the table using commands such as Numeric Table Element Bit Test, or move the data in one element to a variable and use commands such as Bit Test.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-3 = Invalid table length.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get HDD Module On-Latches (page G-60), Get & Clear All HDD Module On-Latches (page G-12), Numeric Table Element Bit Test (page N-8), Bit Test (page B-17), and other Bit commands.

Get All HDD Module States

High Density Digital Module Action

Function: To read the states of all points on all high-density digital input or output modules on one I/O unit.

Typical Use: To get the states for all high-density digital points on the I/O unit with a single command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places all status data as bitmasks in an integer 32 table at a designated starting index. Argument 2 sets the index number and Argument 3 indicates the table.
- The table that receives the data must contain at least 16 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for the module in position zero is placed in the first specified table element, with other modules following in order. If a slot does not contain a high-density digital module, its corresponding table element is zero-filled.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Start Index**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Table	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable		
SNAP-UP1-ADS			
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get All HDD Module States

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Start Index	0	*Integer 32 Variable*
Put Result In	Bldg_A_Status	*Integer 32 Table*
Put Status in	Status_Code	*Integer 32 Variable*

For example, if the I/O unit UIO_A consists of an 8-module rack with an analog module in slot 0 and HDD modules in slots 1–7, table Bldg_A_Status might be filled as follows:

Index	Value (Bitmask)
0	00000000000000000000000000000000
1	01100001010001110000001010110010
2	00000000000010000100010000000111
3	00100000011000000001001000100100
4	01100001010001110000001010110010
5	00001110000100001100100000001001
6	10000000110000001110000000100100
7	00110000001110000011111000000001

(This module is not a HDD module.)

Each index contains the status data for the HDD module in the corresponding position on the rack. A value of 1 indicates that the point is on; a value of 0 indicates that it is off. The least significant bit in the mask corresponds to point zero on the module.

In this example, index 2, which contains the status of all points on the module in slot 2, shows that points 0, 1, 2, 10, 14, and 19 are on. All other points on the module are off.

Index	Value (Bitmask)
8	00000000000000000000000000000000
↓	↓
15	00000000000000000000000000000000

The remainder of the table is zero-filled, since there are no more modules.

OptoScript Example:

`GetAllHddModuleStates(`*I/O Unit, Start Index, Put Result In***`)`**

```
Status_Code = GetAllHddModuleStates(UIO_A, 0, Bldg_A_Status);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read the points on only one HDD module, use Get HDD Module States.
- You can manipulate bits within the table using commands such as Numeric Table Element Bit Test, or move the data in one element to a variable and use commands such as Bit Test.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-3 = Invalid table length.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get HDD Module States (page G-61), Set HDD Module from MOMO Masks (page S-23), Numeric Table Element Bit Test (page N-8), Bit Test (page B-17), and other Bit commands.

Pro Get Analog Filtered Value

Analog Point Action

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. The analog input point is sampled 10 times a second with the filtered value stored locally on the I/O unit.
- The unfiltered analog input is still available using standard analog commands.

Arguments:

Argument 1	Argument 2
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get Analog Filtered Value

From	TEMP_SENSOR	*Analog Input*
Put in	FILTERED_TEMP	*Float Variable*

OptoScript Example:

`GetAnalogFilteredValue(`*From***`)`**

```
FILTERED_TEMP = GetAnalogFilteredValue(TEMP_SENSOR);
```

This is a function command; it returns the filtered value of the analog input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 Debug mode.)

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.

Result Data: 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.

See Also: Get & Clear Analog Filtered Value (page G-14), Set Analog Filter Weight (page S-7)

Get Analog Maximum Value

Analog Point Action

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 point is regularly read and stored at the I/O unit. Check the specifications for the module and I/O unit 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.

Arguments:

Argument 1	Argument 2
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get Analog Maximum Value

From	PRES_SENSOR	*Analog Input*
Put in	MAX_KPA	*Float Variable*

OptoScript Example:

GetAnalogMaxValue(*From***)**

```
MAX_KPA = GetAnalogMaxValue(PRES_SENSOR);
```

This is a function command; it returns the maximum value of the analog input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use Get & Clear Analog Maximum Value to clear the max value before actual readings commence.
- The value returned will be the highest value recorded on this point since the last time the maximum value was cleared, or since the unit was turned on.
- Points 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.

See Also: Get & Clear Analog Maximum Value (page G-15), Get & Clear Analog Minimum Value (page G-16), Get Analog Minimum Value (page G-40)

Get Analog Minimum Value

Analog Point Action

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 point is regularly read and stored at the I/O unit. Check the specifications for the module and I/O unit 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.

Arguments:

Argument 1	**Argument 2**
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get Analog Minimum Value

From	PRES_SENSOR	*Analog Input*
Put in	MIN_KPA	*Float Variable*

OptoScript Example:

`GetAnalogMinValue(`*From***`)`**

```
MIN_KPA = GetAnalogMinValue(PRES_SENSOR);
```

This is a function command; it returns the minimum value of the analog input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use Get & Clear Analog Minimum Value to clear the min value before actual readings commence.
- The value returned will be the lowest value recorded since the last time the minimum value was reset or since the unit was turned on.
- Points 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.

See Also: Get & Clear Analog Minimum Value (page G-16), Get & Clear Analog Maximum Value (page G-15), Get Analog Maximum Value (page G-39)

Get Analog Square Root Filtered Value

Analog Point Action

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. The input point is sampled 10 times a second.
- The unfiltered analog input is still available using standard analog commands.

Arguments:

Argument 1	Argument 2
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get Analog Square Root Filtered Value

From	DP_FLOW_XMTR	*Analog Input*
Put in	Filtered_Flow	*Float Variable*

OptoScript Example:

`GetAnalogSquareRootFilteredValue(`*From*`)`

```
Filtered_Flow = GetAnalogSquareRootFilteredValue(DP_FLOW_XMTR);
```

This is a function command; it returns the square root of the filtered value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 Debug mode.)
- 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.

Result Data: Channels without a module installed will return a value of -32,768 to indicate an error.

See Also: Get Analog Square Root Value (page G-42), Set Analog Filter Weight (page S-7)

Get Analog Square Root Value

Analog Point Action

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.

Arguments:

Argument 1	**Argument 2**
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get Analog Square Root Value

From	DP_FLOW_XMTR	*Analog Input*
Put in	FLOW_RATE	*Float Variable*

OptoScript Example:

`GetAnalogSquareRootValue(`*From*`)`

```
FLOW_RATE = GetAnalogSquareRootValue(DP_FLOW_XMTR);
```

This is a function command; it returns the square root of the value from the analog input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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.

Result Data: Channels without a module installed will return a value of -32,768 to indicate an error.

See Also: Get Analog Square Root Filtered Value (page G-41)

Get Analog Totalizer Value

Analog Point Action

NOTE: This command is for mistic I/O units only.

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 by sampling the input point and storing the total value locally on the I/O unit.
- The sample rate is set using the Set Analog Totalizer Rate Command.
- 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 Get & Clear Analog Totalizer Value.
- Totalizing will stop when an input channel is too far under range. Totalizing will resume when the input signal is back within range.

Arguments:

Argument 1	Argument 2
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Standard Example:

Get Analog Totalizer Value

From	Flow_Rate	*Analog Input*
Put in	Total_Barrels	*Float Variable*

OptoScript Example:

GetAnalogTotalizerValue(*From***)**

```
Total_Barrels = GetAnalogTotalizerValue(Flow_Rate);
```

This is a function command; it returns the totalized value of the analog input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- See Notes for Set Analog Totalizer Rate before using this command.
- Use Get & Clear Analog Totalizer Value to clear the total before actual readings commence.
- Use this command periodically to simply "watch" the total. When it exceeds 30,000, use Get & Clear Analog Totalizer Value to capture the total to a float variable and reset it to zero.

Dependencies: Before using this command, Set Analog Totalizer Rate must be executed. Otherwise, a value of -32,768 will be returned to indicate an error.

Result Data:
- The value returned will be an integer from -32,768 to 32,767.
- Channels without a module installed will return a value of -32,768 to indicate an error.

See Also: Get & Clear Analog Totalizer Value (page G-17), Set Analog Totalizer Rate (page S-12)

Get Available File Space

Control Engine Action

Function: To determine how much file space is currently available in the file system.

Typical Use: To make sure there is sufficient file space available before writing data to a file.

Details:

- In *Argument 1*, show whether file space data should be returned in bytes or megabytes:
  - 0 = return units of bytes
  - 1 = return units of megabytes (1048576 bytes)
- The maximum number of files is limited only by available memory. Each file uses 516 bytes of overhead plus its number of bytes rounded up to the nearest multiple of 516 bytes.
- The maximum amount of memory available in the control engine's file system is approximately 2 MB on a SNAP-PAC-R controller or SNAP Ultimate brain, 2.5 on a SNAP-PAC-S controller, and 1 MB on a SNAP-LCE controller (varies slightly depending on the control engine firmware version).

Arguments:

Argument 1	Argument 2
File System Type	**Put result in**
Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	

Standard Example:

Get Available File Space

File System Type	0	*Integer 32 Literal*
Put result in	File_Space	*Integer 32 Variable*

OptoScript Example:

`GetAvailableFileSpace(`*FileSystemType***`)`**

```
File_Space = GetAvailableFileSpace(0);
```

This is a function command; it returns the size of file space available (a positive value, in units specified by *Argument 1*), or it returns a status code (a negative value, as shown below). The returned value can be consumed by a variable as shown in the example or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Using the Control Engine's File System" in the Communication Commands section of Chapter 10 in the *ioControl User's Guide*.
- Use Get & Clear Analog Minimum Value to clear the min value before actual readings commence.
- For a quick check of the available file space on your device in Debug mode, you don't need to use this command. Instead, double-click the control engine's name in the Strategy Tree and look at File Space Avail. in the Inspect dialog box.

Status Codes: -36 = Feature not implemented (file system type not supported with the type of hardware in use).

See Also: Erase Files in Permanent Storage (page E-18), Load Files From Permanent Storage (page L-7), Get Number of Characters Waiting (page G-101)

Get Chart Status

Chart Action

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. Applicable 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	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Standard Example:

Get Chart Status

Chart	CHART_A	*Chart*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`GetChartStatus(`*Chart***`)`**

```
STATUS = GetChartStatus(CHART_A);
```

This is a function command; it returns the status of the chart. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 float variable, since the bits cannot be tested.

See Also: Chart Stopped? (page C-10), Chart Running? (page C-9), Bit Test (page B-17)

Get Communication Handle Value

Communication Action

Function: Returns a string that is the current value (the parameters) of the communication handle.

Typical Use: To find out the current communication parameters for a communication handle.

Arguments:

Argument 1	**Argument 2**
From	To
Communication Handle	String Variable

Standard Example:

Get Communication Handle Value

From	COMM_B	*Communication Handle*
To	COMM_VALUE	*String Variable*

OptoScript Example:

`GetCommunicationHandleValue(`*From, To***`)`**

```
GetCommunicationHandleValue(COMM_B, COMM_VALUE);
```

This is a procedure command; it does not return a value.

See Also: Set Communication Handle Value (page S-15)

Get Control Engine Address

Control Engine Action

Function: Returns the address of the control engine as a string.

Typical Use: To identify a specific control engine. Used in peer-to-peer communication to identify the origin of a message.

Details:

- For control engines that use IP addressing, the address will be in the format `10.20.30.40`
- The returned address is typically preprended to a string or used as the first element of a table.

Arguments:

Argument 1
Put in
String Variable

Standard Example:

Get Control Engine Address

Put in	Address_Code	*String Variable*

OptoScript Example:

`GetControlEngineAddress()`

```
GetControlEngineAddress(Address_Code);
```

This is a procedure command; it does not return a value.

See Also: Get Control Engine Type (page G-47), Get Firmware Version (page G-55)

Get Control Engine Type

Control Engine Action

Function: Returns a numeric code unique to the control engine type.

Typical Use: In programs that must configure themselves according to the control engine type in which they are running.

Details:
- Primarily used in factory QA testing.
- Returns 402 for all types of SNAP Ultimate I/O brains.
- Returns 403 for SNAP-LCE controllers.
- Returns 512 for SNAP-PAC-S controllers.

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Control Engine Type

Put in	TYPE_CODE	*Integer 32 Variable*

OptoScript Example:

`GetEngineType()`

```
TYPE_CODE = GetEngineType();
```

This is a function command; it returns a value indicating the control engine type. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Get Control Engine Address (page G-46), Get Firmware Version (page G-55)

Get Counter

Digital Point Action

Function: To read a standard digital input counter or quadrature counter value.

Typical Use: To count pulses from turbine flow meters, magnetic pickups, encoders, proximity switches, etc.

Details:

- Standard digital only. For high-density digital, see Get HDD Module Counters.
- Reads the current value of a digital input counter or quadrature counter and places it in the *Put In* parameter.
- Does *not* reset the counter or quadrature counter at the I/O unit to zero.
- Does *not* stop the counter or quadrature counter from continuing to count.
- Valid range for a counter is 0 to 2,147,483,647 counts.
- Valid range for a quadrature counter is -2,147,483,647 to 2,147,483,648 counts.

Arguments:

Argument 1 **From Point**	**Argument 2** **Put in**
Counter	Float Variable
Quadrature Counter	Integer 32 Variable

Standard Example:

Get Counter

From Point	Bottle_Counter	*Counter*
Put in	Number_of_Bottles	*Float Variable*

OptoScript Example:

`GetCounter(`*From Point***`)`**

```
Number_of_Bottles = GetCounter(Bottle_Counter);
```

This is a function command; it returns the counter or quadrature counter value of the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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.

Dependencies:

- Always use Start Counter once before using this command for the first time.
- Applies only to standard digital inputs configured with the counter or quadrature counter feature.

See Also: Get & Clear Counter (page G-18), Start Continuous Square Wave (page S-94), Stop Counter (page S-101), Clear Counter (page C-22)

Get Day

Time/Date Action

Function: To read the day of the month (1 through 31) from the control engine's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in an ioControl 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, 2002, this action would place the value 2 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Day

Put In	Day_of_Month	*Integer 32 Variable*

OptoScript Example:

`GetDay()`

```
Day_of_Month = GetDay();
```

This is a function command; it returns the numerical day of the month. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 LAST_DAY_OF_MONTH just before executing Get Day, then compare DAY_OF_MONTH with LAST_DAY_OF_MONTH using Not Equal? When they are not equal, midnight has just occurred.

See Also: Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Day (page S-17), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Get Day of Week

Time/Date Action

Function: To read the number of the day of the week (0 through 6) from the control engine's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in an ioControl 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
  Monday = 1
  Tuesday = 2
  Wednesday = 3
  Thursday = 4
  Friday = 5
  Saturday = 6
- If the current day is a Wednesday, this action would place the value 3 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Day of Week		
Put In	Day_of_Week	*Integer 32 Variable*

OptoScript Example:

`GetDayOfWeek()`

```
Day_of_Week = GetDayOfWeek();
```

This is a function command; it returns a number indicating the day of the week. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This is a one-time read of the day of the week. If the day changes, you will need to execute this command again to get the current day of the week.
- It is advisable to use this action once in the Powerup chart and once after midnight rollover thereafter. See Notes for Get Day.

See Also: Get Day (page G-49), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Day (page S-17),, Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Get End-Of-Message Terminator

Communication Action

Function: To find out the end-of-message (EOM) character currently set for a specific communication handle.

Typical Use: To make sure the communication handle's EOM character is set as needed.

Details:

- The communication handle must already be opened for the command to take effect. Use the command Open Outgoing Communication to open the handle.
- The character is represented by an ASCII value (see the ASCII table under "String Commands" in Chapter 10 of the *ioControl User's Guide*). For example, a space is a character 32 and a "1" is a character 49.
- The default end-of-message character is 13 (carriage return).

Arguments:

Argument 1	**Argument 2**
Communication Handle	**Put Status In**
Communication Handle	Integer 32 Variable

Standard Example:

Get End-Of-Message Terminator

Communication Handle	UIO_A	*Communication Handle*
Put Status In	EOM_Term	*Integer 32 Variable*

OptoScript Example:

GetEndOfMessageTerminator(*Communication Handle, Put Status In***)**

```
EOM_Term = GetEndOfMessageTerminator(UIO_A);
```

This is a function command; it returns the current EOM character or a status code of -52, if the communication handle has not been opened. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Status Codes: -52 = Invalid connection—not opened.

See Also: Set End-Of-Message Terminator (page S-22), Open Outgoing Communication (page O-4)

Get Error Code of Current Error

Error Handling Action

Function: To return the oldest error code in the message 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 Remove Current Error and Point to Next Error is used first.
- The message queue holds a total of 1000 errors and messages.
- See the Errors Appendix in the *ioControl User's Guide* for a list of errors that may appear in the message queue.

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Error Code of Current Error

Put in	ERROR_CODE	*Integer 32 Variable*

OptoScript Example:

`GetErrorCodeOfCurrentError()`

```
ERROR_CODE = GetErrorCodeOfCurrentError();
```

This is a function command; it returns the code for the oldest error in the message queue. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- Use Remove Current Error and Point to Next Error to drop the oldest error from the queue so the next error can be evaluated.
- For detailed information, use Control Engine→Inspect in Debug mode to view the message queue.

See Also: Clear All Errors (page C-18), Get Error Count (page G-53), Remove Current Error and Point to Next Error (page R-22)

Get Error Count

Error Handling Action

Function: To determine the number of errors in the message queue.

Typical Use: To allow an error handling chart to determine that there are no more errors to process.

Details:
- The queue holds a total of 1000 errors and messages.
- Returns a zero if the queue is empty.

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Error Count
Put in ERROR_COUNT *Integer 32 Variable*

OptoScript Example:

`GetErrorCount()`

```
ERROR_COUNT = GetErrorCount();
```

This is a function command; it returns the number of errors in the message queue. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- To eliminate all errors from the queue, use Clear All Errors.
- Use Debug mode to view the message queue for detailed information.

See Also: Clear All Errors (page C-18), Get Error Code of Current Error (page G-52), Remove Current Error and Point to Next Error (page R-22)

Get Event Latches

Event/Reaction Action

Function: Gets all event latches in the specified group.

Typical Use: To get all event latches in the specified group with one command rather than issuing a separate command for each one.

Details:

- There can be up to 16 event/reaction groups, each containing as many as 16 event latches. If all related event latches are in the same group, this command could be quite useful.
- The value returned is an integer with the lower 16 bits representing the 16 latches in the group. If the variable has a value greater than zero, one or more latches are set.

Arguments:

Argument 1	Argument 2
Event/Reaction Group	**Put in**
Event/Reaction Group	Integer 32 Variable

Standard Example:

Get Event Latches

Event/Reaction Group	ER_E_STOP_GROUP_A	
Put in	Group_Latch_Status	*Integer 32 Variable*

OptoScript Example:

GetEventLatches(*E/R Group***)**

```
Group_Latch_Status = GetEventLatches(ER_E_STOP_GROUP_A);
```

This is a function command; it returns a bitmask representing the status of event latches in the event/reaction group. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Bit Test could be used to test each of the lower 16 bits numbered 0–15.

See Also: Get & Clear Event Latches (page G-19)

Get Firmware Version

Control Engine Action

Function: Returns a string containing the firmware (kernel) version.

Typical Use: In programs that must configure themselves according to the firmware version under which they are running.

Details: The returned string will be in the format `R1.0a`.

Arguments:

Argument 1
Put in
String Variable

Standard Example:

Get Firmware Version

Put in	REV_CODE	*String Variable*

OptoScript Example:

`GetFirmwareVersion(`*Put in*`)`

```
GetFirmwareVersion(REV_CODE);
```

This is a procedure command; it does not return a value.

See Also: Get Control Engine Type (page G-47)

Get Frequency

Digital Point Action

NOTE: This command is for mistic I/O units only.

Function: To read digital input frequency value.

Typical Use: To read the speed of rotating machinery, velocity encoders, etc.

Details:
- Reads the current frequency unit of a digital input and places it in the *Put In* parameter.
- Returns an integer value from 0 to 65,535 (see Notes below).
- The default unit is 1 Hertz (see Notes below).
- Not available on SNAP Ethernet brains.

Arguments:

Argument 1	**Argument 2**
From Point	**Put in**
Frequency	Float Variable
	Integer 32 Variable

Standard Example:

Get Frequency

From Point	SHAFT_PICKUP	*Frequency*
Put in	MOTOR_SPEED	*Integer 32 Variable*

OptoScript Example:

`GetFrequency(`*From Point***`)`**

```
MOTOR_SPEED = GetFrequency(SHAFT_PICKUP);
```

This is a function command; it returns the frequency units value of the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- Since the the default 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.
- If you change the unit to 10 Hertz, you will need to multiply the returned value by 10.

  Example 1: Unit is set to 1Hz units. You call this command, and receive a value of 57. The frequency = 57 x 1Hz = 57Hz

  Example 2: Unit is set to 10Hz units. You call this command, and receive a value of 57. The frequency = 57 x 10Hz = 570Hz
- 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.

Get HDD Module Counters

High Density Digital Module Action

Function: To read the counters for all points on a high-density digital input module.

Typical Use: To get counts without clearing them.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Places counter data for all points in the module in an integer 32 table at a designated starting index. Argument 3 sets the index number and Argument 4 indicates the table.
- The table that receives the data must contain at least 32 elements after the starting index. (If the table is not large enough, an error -3 is returned.) Data for point zero is placed in the first specified table element, with other points following in order.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Module Number**	Argument 3 **Start Table Index**	Argument 4 **Put Result In**	Argument 5 **Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Table	Integer 32 Variable

Standard Example:

Get HDD Module Counters

I/O Unit	Installation_42	*SNAP-ENET-S64*
Module Number	10	*Integer 32 Literal*
Start Table Index	0	*Integer 32 Literal*
Put Result In	Rotations	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

For example, the first four elements of the Rotations table might be filled as follows:

Index	Counter Value	
0	25678	← Counter data for point 0
1	25678	← Counter data for point 1
2	30946747	← Counter data for point 2
3	42	← Counter data for point 3

OptoScript Example:

`GetHddModuleCounters(`*I/O Unit, Module Number, Start Table Index, Put Result In***`)`**

```
Status_Code = GetHddModuleCounters(Installation_42, 10, 0, Rotations);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read and clear counters, use Get & Clear HDD Module Counter (one counter) or Get & Clear HDD Module Counters (all counters on a module).
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.
- Counters with values of more than 2 billion may appear as negative numbers.

Status Codes: 0 = Success

-3 = Invalid table length. Table must contain at least 32 elements.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get & Clear HDD Module Counter (page G-20), Get & Clear HDD Module Counters (page G-21)

Get HDD Module Off-Latches

High Density Digital Module Action

Function: To read the off-latches of all points on a high-density digital input module.

Typical Use: To read off-latches without clearing latches.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Uses a bitmask to indicate the state of off-latches for all points on the module. The least significant bit corresponds to point 0. A value of 1 in a bit means the off-latch is on (set); a value of 0 in the bit means the off-latch is off (not set).

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Module Number**	Argument 3 **Put Result In**	Argument 4 **Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Standard Example:

Get HDD Module Off-Latches

I/O Unit	Bldg_A	*SNAP-B3000-ENET, SNAP-ENET-RTC*
Module Number	9	*Integer 32 Literal*
Put Result In	Fan_OffLatches	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

An example of the result is illustrated below. Only the first 8 and last 8 off-latches are shown.

Bitmask	Hex	9				3				⟶	B				2			
	Binary	1	0	0	1	0	0	1	1	⟶	1	0	1	1	0	0	1	0

Off-latch	on	off	off	on	off	off	on	on		on	off	on	on	off	off	on	off
Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0

OptoScript Example:

`GetHddModuleOffLatches(`*I/O Unit, Module Number, Put Result In***`)`**

```
Status_Code = GetHddModuleOffLatches(Bldg_A, 9, Fan_OffLatches);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read all off-latches for all HDD modules on one I/O unit, use Get All HDD Module Off-Latches.
- See “High Density Digital Module Commands” in Chapter 10 of the *ioControl User’s Guide*, and see form #1547, the *SNAP High-Density Digital Module User’s Guide*.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get HDD Module On-Latches (page G-60), Get All HDD Module Off-Latches (page G-32), Get & Clear HDD Module Off-Latches (page G-22), Get & Clear All HDD Module Off-Latches (page G-10)

Get HDD Module On-Latches

High Density Digital Module Action

Function: To read the on-latches of all points on a high-density digital input module.

Typical Use: To read on-latches without clearing latches.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Uses a bitmask to indicate the state of on-latches for all points on the module. The least significant bit corresponds to point 0. A value of 1 in a bit means the on-latch is on (set); a value of 0 in the bit means the on-latch is off (not set).

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Module Number**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC, SNAP-UP1-ADS, SNAP-UP1-M64, SNAP-ENET-S64, SNAP-PAC-R1, SNAP-PAC-R2	Integer 32 Literal, Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Standard Example:

Get HDD Module On-Latches

I/O Unit	Bldg_A	*SNAP-B3000-ENET, SNAP-ENET-RTC*
Module Number	9	*Integer 32 Literal*
Put Result In	Fan_OnLatches	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

An example of the result is illustrated below. Only the first 8 and last 8 on-latches are shown.

Bitmask	Hex	9				3				→	B				2			
	Binary	1	0	0	1	0	0	1	1	→	1	0	1	1	0	0	1	0
	On-latch	on	off	off	on	off	off	on	on		on	off	on	on	off	off	on	off
	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0

OptoScript Example:

`GetHddModuleOnLatches(`*I/O Unit, Module Number, Put Result In***`)`**

```
Status_Code = GetHddModuleOnLatches(Bldg_A, 9, Fan_OnLatches);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read all on-latches for all HDD modules on one I/O unit, use Get All HDD Module On-Latches.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes: 0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get HDD Module Off-Latches (page G-58), Get All HDD Module On-Latches (page G-34), Get & Clear HDD Module On-Latches (page G-24), Get & Clear All HDD Module On-Latches (page G-12)

Get HDD Module States

High Density Digital Module Action

Function: To read the states of all points on a high-density digital input or output module.

Typical Use: To get information about all points on one module in one command.

Details:

- Works only on high-density digital modules, not on standard digital modules.
- Uses a bitmask to indicate the state of each point on the module. The least significant bit corresponds to point 0. A value of 1 in a bit means the point is on; a value of 0 in the bit means the point is off.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
I/O Unit	**Module Number**	**Put Result In**	**Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC, SNAP-UP1-ADS, SNAP-UP1-M64, SNAP-ENET-S64, SNAP-PAC-R1, SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Standard Example:

Get HDD Module States

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Module Number	12	*Integer 32 Literal*
Put Result In	Fan_Status	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

An example of the result is illustrated below. Only the first 8 and last 8 points are shown.

Bitmask	Hex	9				3				→	B				2			
	Binary	1	0	0	1	0	0	1	1	→	1	0	1	1	0	0	1	0
	State	on	off	off	on	off	off	on	on		on	off	on	on	off	off	on	off
	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0

OptoScript Example:

`GetHddModuleStates(`*I/O Unit, Module Number, Put Result In***`)`**

```
Status_Code = GetHddModuleStates(UIO_A, 12, Fan_Status);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To read the points on all HDD modules on one I/O unit, use Get All HDD Module States.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get All HDD Module States (page G-36), Set HDD Module from MOMO Masks (page S-23), Turn On HDD Module Point (page T-28), Turn Off HDD Module Point (page T-26)

Get High Bits of Integer 64

Logical Action

Function:

To read only the upper 32 bits of a 64-bit integer and place them in a 32-bit integer.

Typical Use:

To convert half of a 64-bit integer into a 32-bit integer for faster manipulation. Often used when only part of a 64-point digital rack is populated with points.

Details:

- Returns the upper 32 bits, which represent the upper 32 points on a 64-point digital-only rack, to the numeric variable specified.
- The least significant bit corresponds to point 32; the most significant bit corresponds to point 63.

Arguments:

Argument 1	**Argument 2**
High Bits From	**Put in**
Integer 64 Variable	Integer 32 Variable

Standard Example:

Get High Bits of Integer 64

High Bits From	INPUT_BOARD_2	*Integer 64 Variable*
Put in	IN_BD2_HIGH	*Integer 32 Variable*

OptoScript Example:

`GetHighBitsOfInt64(`*High Bits From***`)`**

```
IN_BD2_HIGH = GetHighBitsOfInt64(INPUT_BOARD_2);
```

This is a function command; it returns the upper 32 bits of a 64-bit integer. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: This command is useful if you want to get information from a digital-only SNAP-ENET-D64 or SNAP-UP1-D64 brain, which use “integer 64” commands, into a program that doesn’t directly support 64-bit integers. Such programs include ioDisplay and third-party products.

See Also: Get Low Bits of Integer 64 (page G-85), Make Integer 64 (page M-1)

Get Hours

Time/Date Action

Function: To read the hour (0 through 23) from the control engine’s real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in an ioControl 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 action would place the value 14 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Hours

Put In	HOURS	*Integer 32 Variable*

OptoScript Example:

`GetHours()`

```
HOURS = GetHours();
```

This is a function command; it returns the hour of the day (0 through 23) from the control engine’s real-time clock. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User’s Guide* for more information.

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 (page G-49), Get Day of Week (page G-50), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Day (page S-17), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Get ID of Block Causing Current Error

Error Handling Action

Function: Gets the ID number of the block that caused the top queue error.

Typical Use: In an error handling chart to build a history of errors in a string table.

Details: Only works when the top queue error is *not* an I/O unit error.

Arguments:

Argument 1
Put in
Integer 32 Variable

Standard Example:

Get Id of Block Causing Current Error

Put in	Error_Block_ID	*Integer 32 Variable*

OptoScript Example:

`GetIdOfBlockCausingCurrentError()`

```
Error_Block_ID = GetIdOfBlockCausingCurrentError();
```

This is a function command; it returns the ID number of the block that caused the top error in the message queue. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Blocks are numbered starting with zero.

Dependencies: The top queue error must *not* be an I/O unit error.

See Also: Get Name of Chart Causing Current Error (page G-98), Get Name of I/O Unit Causing Current Error (page G-99)

Get I/O Unit as Binary Value

I/O Unit Action

Function: To read the current on/off status of all digital points on the I/O unit.

Typical Use: To efficiently read the status of all digital points on a single I/O unit with one command.

Details:

- Reads the current on/off status of all digital points on the I/O unit specified and updates the IVALs and XVALs for all points. Reads outputs as well as inputs.
- Returns status (a 32-bit or 64-bit integer) to the numeric variable specified.
- If a point is on, there will be a “1” in the respective bit. If the point is off, there will be a “0” in the respective bit. The least significant bit corresponds to point zero.
- An analog, serial, or PID point on a mixed I/O unit will appear as a “0”.
- If a specific point is disabled, it will not be read. If the entire I/O unit is disabled, none of the points will be read.

Arguments:

Argument 1 **From**	**Argument 2** **Put in**
B100*	Integer 32 Variable
B3000 (Digital)*	Integer 64 Variable
G4 Digital Local Simple I/O Unit*	
G4D16R*	
G4D32RS*	
SNAP-ENET-D64	
SNAP-UP1-D64	
SNAP-UP1-M64	
SNAP-ENET-S64	
SNAP-B3000-ENET, SNAP-ENET-RTC	
SNAP-UP1-ADS	
SNAP-PAC-R1	
SNAP-PAC-R2	
SNAP-BRS*	

* ioControl Professional only

Standard Example:

Get I/O Unit as Binary Value

From	INPUT_BOARD_2	*SNAP-ENET-D64*
Put in	IN_BD2_STATUS	*Integer 64 Variable*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Bit mask	Binary	0	1	1	0	1	1	0	0	→	0	1	0	0	0	0	1	0
	Hex	6				C				→	4				2			

To save space, the example shows only the first eight points and the last eight points on the 64-point I/O unit. Points with a value of 1 are on; points with a value of 0 are off.

OptoScript Example: **`GetIoUnitAsBinaryValue(`*I/O Unit*`)`**

```
IN_BD2_STATUS = GetUnitAsBinaryValue(Input_Board_2);
```

This is a function command; it returns the current on/off status of all digital points, in the form of a bitmask. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Use Bit Test to examine individual bits.

See Also: Set Digital-64 I/O Unit from MOMO Masks (page S-19), Set Mixed I/O Unit from MOMO Masks (page S-55)

Get I/O Unit Event Message State

I/O Unit—Event Message Action

Function: To determine the current state of an event message on a SNAP Ultimate or Ethernet I/O unit.

Typical Use: To find out whether an e-mail, SNMP, or other kind of event message has been sent.

Details: Possible states are: 0 = Inactive, 1 = Active, or 2 = Acknowledged.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Event Message Number**	**Put Result in**	**Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Variable	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable		
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get I/O Unit Event Message State

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Event Message Number	0	*Integer 32 Literal*
Put Result in	State	*Integer 32 Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

`GetIoUnitEventMsgState(`*I/O Unit, Event Message #, Put Result in***`)`**

```
Status = GetIoUnitEventMsgState(UIO_A, 0, State);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- See "I/O Unit—Event Message Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use ioManager to configure the types, intervals, and text of event messages. You can configure up to 128 messages for each I/O unit.
- To find out the text of the message, use Get I/O Unit Event Message Text.
- To send the message, use Set I/O Unit Event Message State.

Status Codes:

0 = success

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Event Message Text (page G-68), Set I/O Unit Event Message State (page S-26), Set I/O Unit Event Message Text (page S-27)

Get I/O Unit Event Message Text

I/O Unit—Event Message Action

Function: To read the text of an event message on a SNAP Ultimate or Ethernet I/O unit.

Typical Use: To read the text of an e-mail, SNMP, or other kind of message sent as a response to an event that occurs within strategy logic.

Details: The message text is returned in *Argument 3*. The string variable for *Argument 3* should be 128 characters long to hold the message text.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Event Message Number**	**Put Result in**	**Put Status in**
SNAP-ENET-D64	Integer 32 Literal	String Variable	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable		
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get I/O Unit Event Message Text

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Event Message Number	0	*Integer 32 Literal*
Put Result in	Msg_0	*String Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

`GetIoUnitEventMsgText(`*I/O Unit, Event Message #, Put Result in***`)`**

```
Status = GetIoUnitEventMsgText(UIO_A, 0, Msg_0);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- See “I/O Unit—Event Message Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Use ioManager to configure the types, intervals, and text of event messages. You can configure up to 128 messages for each I/O unit.
- If the variable in *Argument 3* is shorter than 128 characters, as many characters as fit are placed in it and an error -23 is returned.

Status Codes:

0 = success

-12 = Invalid index. Event message number is less than 0 or greater than 127.

-23 = String too short. String variable in *Argument 3* must be 128 characters long.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Event Message State (page G-67), Set I/O Unit Event Message State (page S-26), Set I/O Unit Event Message Text (page S-27)

Get I/O Unit Scratch Pad Bits

I/O Unit—Scratch Pad Action

Function: To read a bit in the Scratch Pad area of a SNAP Ultimate or Ethernet brain.

Typical Use: For peer-to-peer communication. Strategy data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- Use Set I/O Unit Scratch Pad Bits from MOMO Mask to store the data in the Scratch Pad area, and then use this command to retrieve it.
- The entire Scratch Pad Bits area is returned to the variable named in *Argument 2*.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
I/O Unit	**Put Result in**	**Put Status in**
SNAP-ENET-D64	Integer 64 Variable	Integer 32 Variable
SNAP-UP1-D64		
SNAP-B3000-ENET,		
SNAP-ENET-RTC		
SNAP-UP1-ADS		
SNAP-PAC-R1		
SNAP-PAC-R2		

Standard Example:

Get I/O Unit Scratch Pad Bits

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Put Result in	MyInt64Var	*Integer 64 Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

GetIoUnitScratchPadBits(*I/O Unit, Put Result in***)**

```
Status = GetIoUnitScratchPadBits(UIO_B, MyInt64Var);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To find out the value of a specific bit in the returned data, use Bit Test. See other logical commands for other ways to work with the returned data.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = success

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Float Table (page G-72), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Get I/O Unit Scratch Pad Float Element

I/O Unit—Scratch Pad Action

Function: To read a float in the Scratch Pad area of a remote or local SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use Set I/O Unit Scratch Pad Float Element to store the variable data in the Scratch Pad area, and then use this command to retrieve it.
- The float area of the Scratch Pad is a table containing 10240 elements (index numbers 0–10,239). Enter the index number of the element you want to read in *Argument 2*. The float value is returned to the float variable named in *Argument 3*.

Arguments:

Argument 1 **I/O Unit**	**Argument 2** **Index**	**Argument 3** **Put Result in**	**Argument 4** **Put Status in**
SNAP-ENET-D64 SNAP-UP1-D64 SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Float Variable	Integer 32 Variable

Standard Example:

Get I/O Unit Scratch Pad Float Element

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Index	26	*Integer 32 Literal*
Put Result in	MyFloatVar	*Float Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

GetIoUnitScratchPadFloatElement(*I/O Unit, Index, Put Result in***)**

```
Status = GetIoUnitScratchPadFloatElement(UIO_B, 26, MyFloatVar);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To retrieve more than one float value in a single command, use Get I/O Unit Scratch Pad Float Table.

- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Scratch Pad Float Element (page S-32), Get I/O Unit Scratch Pad Bits (page G-69), Get I/O Unit Scratch Pad Float Table (page G-72), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Get I/O Unit Scratch Pad Float Table

I/O Unit—Scratch Pad Action

Function: To read a series of float values in the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use Set I/O Unit Scratch Pad Float Element more than once, or use Set I/O Unit Scratch Pad Float Table, to store the variable data in the Scratch Pad area. Use this command to retrieve the float values and place them in a table defined in the peer's strategy.
- The float area of the Scratch Pad is a table containing 10240 elements (index numbers 0–10239). Enter the number of elements you want to read in *Argument 2* and the index number of the starting element in *Argument 3*.
- The float values are returned to the float table named in *Argument 5*, starting at the index shown in *Argument 4*.
- *Argument 3*, From Index, is the start index of the source table.
- *Argument 4*, To Index, is the start index of the destination table that data will be written to.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Length**	**From Index**	**To Index**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Literal	Integer 32 Literal
SNAP-UP1-D64	Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Argument 5	Argument 6
To Table	**Put Status in**
Float Table	Integer 32 Variable

Standard Example:

Get I/O Unit Scratch Pad Float Table

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Length	64	*Integer 32 Literal*
From Index	0	*Integer 32 Literal*
To Index	0	*Integer 32 Literal*
To Table	MyFloatTable	*Float Table*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

GetIoUnitScratchPadFloatTable(*I/O Unit, Length, From Index, To Index, To Table***)**

```
Status = GetIoUnitScratchPadFloatTable(UIO_B, 64, 0, 0, MyFloatTable);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To retrieve a single float value, use Get I/O Unit Scratch Pad Float Element.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = success

-3 = Invalid length. *Argument 2* (Length) less than 0 or greater than 10240.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Set I/O Unit Scratch Pad Float Table (page S-34), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Bits (page G-69), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Get I/O Unit Scratch Pad Integer 32 Element

I/O Unit—Scratch Pad Action

Function: To read an integer 32 in the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use Set I/O Unit Scratch Pad Integer 32 Element to store the variable data in the Scratch Pad area, and then use this command to retrieve it.
- The integer 32 area of the Scratch Pad is a table containing 10,240 elements (index numbers 0–10239). Enter the index number of the element you want to read in *Argument 2*. The integer 32 value is returned to the integer 32 variable named in *Argument 3*.
- The integer 32 value is returned to the integer 32 variable named in *Argument 3*.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
I/O Unit	**Index**	**Put Result in**	**Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Variable	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable		
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get I/O Unit Scratch Pad Integer 32 Element

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Index	26	*Integer 32 Literal*
Put Result in	MyInt32Var	*Integer 32 Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

GetIoUnitScratchPadInt32Element(*I/O Unit, Index, Put Result in***)**

```
Status = GetIoUnitScratchPadInt32Element(UIO_B, 26, MyInt32Var);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To retrieve more than one integer 32 value in a single command, use Get I/O Unit Scratch Pad Integer 32 Table.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Get I/O Unit Scratch Pad Bits (page G-69), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Float Table (page G-72), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Get I/O Unit Scratch Pad Integer 32 Table

I/O Unit—Scratch Pad Action

Function: To read a series of integer 32 values in the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use Set I/O Unit Scratch Pad Integer 32 Element more than once, or use Set I/O Unit Scratch Pad Integer 32 Table, to store the variable data in the Scratch Pad area. Use this command to retrieve the integer values in one step and place them in a table defined in the peer's strategy.
- The integer 32 area of the Scratch Pad is a table containing 10,240 elements (index numbers 0–10239). Enter the number of elements you want to read in *Argument 2* and the index number of the starting element in *Argument 3*.
- The integer values are returned to the integer 32 table named in *Argument 5*, starting at the index shown in *Argument 4*.

Arguments:

Argument 1 **I/O Unit**	**Argument 2** **Length**	**Argument 3** **From Index**	**Argument 4** **To Index**
SNAP-ENET-D64 SNAP-UP1-D64 SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable

Argument 5 **To Table**	**Argument 6** **Put Status in**
Integer 32 Table	Integer 32 Variable

Standard Example:

Get I/O Unit Scratch Pad Integer 32 Table

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Length	64	*Integer 32 Literal*
From Index	0	*Integer 32 Literal*
To Index	0	*Integer 32 Literal*
To Table	MyInt32Table	*Integer 32 Table*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

GetIoUnitScratchPadInt32Table(*I/O Unit, Length, From Index, To Index, To Table***)**

```
Status = GetIoUnitScratchPadInt32Table(UIO_B,64, 0, 0, MyInt32Table);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To retrieve a single integer 32 value, use Get I/O Unit Scratch Pad Integer 32 Element.

- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-3 = Invalid length. *Argument 2* (Length) less than 0 or greater than 3072.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Float Table (page G-72), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Bits (page G-69), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Get I/O Unit Scratch Pad String Element

I/O Unit—Scratch Pad Action

Function: To read a string in the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use Set I/O Unit Scratch Pad String Element to store the variable data in the Scratch Pad area, and then use this command to retrieve it.
- The string area of the Scratch Pad is a table containing 64 elements (index numbers 0–63). Each string element can hold 128 characters or 128 bytes of binary data. Enter the index number of the element you want to read in *Argument 2*. The string is returned to the string variable named in *Argument 3*.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Index**	Argument 3 **Put Result in**	Argument 4 **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	String Variable	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable		
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Get I/O Unit Scratch Pad String Element

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Index	26	*Integer 32 Literal*
Put Result in	MyStringVar	*String Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

GetIoUnitScratchPadStringElement(*I/O Unit, Index, Put Result in***)**

```
Status = GetIoUnitScratchPadStringElement(UIO_B,26, MyStringVar);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To retrieve more than one string in a single command, use Get I/O Unit Scratch Pad String Table.
- If the destination string width is smaller than the received string, as many characters as possible are placed in the string and a -23 error is returned.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.

- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-23 = String too short. Destination string width is smaller than received string. (As many characters as possible are placed in the string.)

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Scratch Pad String Element (page S-40), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Float Table (page G-72), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad Bits (page G-69), Get I/O Unit Scratch Pad String Table (page G-80)

Get I/O Unit Scratch Pad String Table

I/O Unit—Scratch Pad Action

Function: To read a series of strings in the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use Set I/O Unit Scratch Pad String Element or Set I/O Unit Scratch Pad String Table to store the variable data in the Scratch Pad area, and then use this command to retrieve it.
- The string area of the Scratch Pad is a table containing 64 elements (index numbers 0–63). Each string element can hold 128 characters or 128 bytes of binary data. Enter the number of elements you want to read in *Argument 2* and the index number of the starting element in *Argument 3*.
- The string values are returned to the string table named in *Argument 5*, starting at the index shown in *Argument 4*.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
I/O Unit	**Length**	**From Index**	**To Index**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Literal	Integer 32 Literal
SNAP-UP1-D64	Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Argument 5	**Argument 6**
To Table	**Put Status in**
String Table	Integer 32 Variable

Standard Example:

Get I/O Unit Scratch Pad String Table

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Length	8	*Integer 32 Literal*
From Index	0	*Integer 32 Literal*
To Index	0	*Integer 32 Literal*
To Table	MyStringTable	*String Table*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

`GetIoUnitScratchPadStringTable(`*I/O Unit, Length, From Index, To Index, To Table***`)`**

```
Status = GetIoUnitScratchPadStringTable(UIO_B, 8, 0, 0, MyStringTable);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To retrieve a single string, use Get I/O Unit Scratch Pad String Element.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can

connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.

- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-23 = String too short. Destination string width is smaller than received string. (As many characters as possible are placed in the string.)

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Scratch Pad String Table (page S-41), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Float Table (page G-72), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad Bits (page G-69)

Get Julian Day

Time/Date Action

Function: Gets the number of days starting with January 1 up to and including today's date.

Typical Use: Wherever Julian dates are required.

Details: Value returned will be from 1 to 366. For example, January 1 will always be Julian day 1. December 31 will be Julian day 365 (or 366 in a leap year).

Arguments:

Argument 1
Put in
Integer 32 Variable

Standard Example:

Get Julian Day		
Put in	Todays_Julian_Day	*Integer 32 Variable*

OptoScript Example:

`GetJulianDay()`

```
Todays_Julian_Day = GetJulianDay();
```

This is a function command; it returns the number of the current day, computed since the beginning of the year. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Copy Date to String (MM/DD/YYYY) (page C-60)

Get Length of Table

Miscellanous Action

Function: To obtain the declared length (size) of a float, integer, string, or pointer table.

Typical Use: To determine the last index when reading or writing to a table.

Details: A size of 10, for example, means there are 10 elements numbered 0–9.

Arguments:

Argument 1	Argument 2
Table	**Put in**
Float Table	Float Variable
Integer 32 Table	Integer 32 Variable
Integer 64 Table	
Pointer Table	
String Table	

Standard Example:

Get Length of Table

Table	Config_Data	*Integer 32 Table*
Put in	Config_Data_Size	*Integer 32 Variable*

OptoScript Example:

`GetLengthOfTable(`*Table***`)`**

```
Config_Data_Size = GetLengthOfTable(Config_Data);
```

This is a function command; it returns the length of the table. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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.

Get Line Causing Current Error

Error Handling Action

Function: Gets the line within a flowchart block that caused the top queue error.

Typical Use: In an error-handling chart to build a history of errors.

Details:
- Works only when the top queue error is *not* an I/O unit error.
- The strategy must have been loaded to the control engine in full debug mode for this command to work. If the strategy is in minimal debug mode, the command returns a zero.

Arguments:

Argument 1
Put in
Integer 32 Variable

Standard Example:

Get Line Causing Current Error

Put in	Error_Block_ID	*Integer 32 Variable*

OptoScript Example:

`GetLineCausingCurrentError()`

```
Error_Block_ID = GetLineCausingCurrentError();
```

This is a function command; it returns the line that caused the top error in the message queue. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Dependencies: The top queue error must *not* be an I/O unit error.

See Also: Get ID of Block Causing Current Error (page G-64), Get Name of Chart Causing Current Error (page G-98), Get Name of I/O Unit Causing Current Error (page G-99)

Get Low Bits of Integer 64

Logical Action

Function: To read only the lower 32 bits of a 64-bit integer and place them in a 32-bit integer.

Typical Use: To convert half of a 64-bit integer into a 32-bit integer for faster manipulation. Often used when only part of a 64-point digital rack is populated with points.

Details:

- Returns the lower 32 bits, which represent the lower 32 points on a 64-point digital-only rack, to the numeric variable specified.
- The least significant bit corresponds to point zero; the most significant bit corresponds to point 32.

Arguments:

Argument 1	**Argument 2**
Low Bits From	**Put in**
Integer 64 Variable	Integer 32 Variable

Standard Example:

Get Low Bits of Integer 64

Low Bits From	INPUT_BOARD_2	*Integer 64 Variable*
Put in	IN_BD2_LOW	*Integer 32 Variable*

OptoScript Example:

`GetLowBitsOfInt64(`*Integer 64***`)`**

```
IN_BD2_LOW = GetLowBitsOfInt64(INPUT_BOARD_2);
```

This is a function command; it returns the lower 32 bits of a 64-bit integer. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: This command is useful if you want to get information from a digital-only SNAP-ENET-D64 or SNAP-UP1-D64 brain, which use "integer 64" commands, into a program that doesn't directly support 64-bit integers. Such programs include ioDisplay and third-party products.

See Also: Get High Bits of Integer 64 (page G-62), Make Integer 64 (page M-1)

Get Minutes

Time/Date Action

Function: To read the minute (0 through 59) from the control engine's real-time clock/calendar and put it into a numeric variable.

Typical Use: To trigger an event in an ioControl 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 action would place the value 35 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Minutes		
Put In	MINUTES	*Integer 32 Variable*

OptoScript Example:

`GetMinutes()`

```
MINUTES = GetMinutes();
```

This is a function command; it returns the current minute (0 through 59) from the control engine's real-time clock. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Day (page S-17), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Get Mistic PID Control Word

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: Reads the bits that represent the PID configuration.

Typical Use: To verify the PID configuration when troubleshooting.

Details: Bit assignments:

11 1 = Use SqRt value from input channel. 0 = Use actual input value.
10 1 = Setpoint was above high clamp. Write zero to clear.
9 1 = Setpoint was below low clamp. Write zero to clear.
8 1 = Input channel under-range. Write zero to clear.
7 1 = Loop active. 0 = Loop reset (stopped).
6 1 = Loop in auto mode. 0 = Loop in manual mode.
5 1 = Output enabled. 0 = Output disabled (disconnected).
4 1 = Output tracks input in manual mode. 0 = no action.
3 1 = Setpoint tracks input in manual mode. 0 = no action.
2 1 = Input from host. 0 = Input from channel.
1 1 = Setpoint from channel. 0 = Setpoint from host.
0 1 = Use filtered value from input channel. Must have filtering active on the input channel.
0 = Use current value of input channel.

Arguments:

Argument 1	Argument 2
From PID Loop	**Put in**
PID Loop	Integer 32 Variable

Standard Example:

Get Mistic PID Control Word

From PID Loop	Extruder_Zone08	*PID Loop*
Put in	PID_CTRL_WORD	*Integer 32 Variable*

OptoScript Example:

GetMisticPidControlWord(*From PID Loop***)**

```
PID_CTRL_WORD = GetMisticPidControlWord(Extruder_Zone08);
```

This is a function command; it returns the bits that represent the PID configuration. The returned value can be consumed by a variable (as in the example shown) or by a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: The PID Control Word is actually a 16-bit number. The four most significant bits are reserved.

See Also: Set Mistic PID Control Word (page S-44)

Get Mistic PID D Term

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: Reads the derivative value from the PID.

Typical Use: To store "as found" PID parameters for later use.

Details:

- Reads the derivative value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	**Argument 2**
From PID Loop	**Put in**
PID Loop	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID D Term

From PID Loop	Extruder_Zone08	*PID Loop*
Put in	Zone08_DTerm	*Float Variable*

OptoScript Example:

`GetMisticPidDTerm(`*From PID Loop***`)`**

```
Zone08_DTerm = GetMisticPidDTerm(Extruder_Zone08);
```

This is a function command; it returns the derivative value from the PID loop. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Always use a float variable to store the result.

See Also: Set Mistic PID D Term (page S-45)

Get Mistic PID I Term

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: Reads the Integral value from the PID.

Typical Use: To store “as found” PID parameters for later use.

Details:
- Reads the Integral value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	Argument 2
From PID Loop	**Put in**
PID Loop	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID I Term

From PID Loop	Extruder_Zone08	*PID Loop*
Put in	Zone08_ITerm	*Float Variable*

OptoScript Example:

GetMisticPidITerm(*From PID Loop***)**

```
Zone08_ITerm = GetMisticPidITerm(Extruder_Zone08);
```

This is a function command; it returns the integral value from the PID loop. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Always use a float variable to store the result.

See Also: Set Mistic PID I Term (page S-46)

Get Mistic PID Input

PID—Mistic Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the input value (also known as the process variable) of the PID.

Typical Use: To find out the PID input value at the time of the most recent scan.

Details:

- The value read has the same engineering units as the specified PID input channel.
- This command retrieves the input value from the most recent scan. To find out the value right now, independent of scan time, use Get PID Current Input.
- The input can be an analog channel or a PID output (for cascaded PIDs), or it can be determined by the strategy in the control engine using Set PID Input.

Arguments:

Argument 1	Argument 2
PID Loop	**Input**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID Input

PID Loop	HEATER_3	*PID Loop*
Input	PID_INPUT_VALUE	*Float Variable*

OptoScript Example:

GetMisticPidInput(*PID Loop***)**

```
PID_INPUT_VALUE = GetMisticPidInput(HEATER_3);
```

This is a function command; it returns the input value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- Use to detect bad or out-of-range PID input values. When such a value is found, use the Set PID Output 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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Set Mistic PID Input (page S-47)

Get Mistic PID Mode

PID—Mistic Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read whether the PID is in auto or manual mode.

Typical Use: To store current PID parameters for later use.

Details:

- Reads auto/manual mode from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).
- Checks bit 6 of the PID control word. Returns a 1 (logical True) if in auto, otherwise a zero (logical False) is returned.
- Returns a zero if in auto mode or a 1 if in manual mode.

Arguments:

Argument 1	Argument 2
PID Loop	**Mode**
PID Loop	Integer 32 Variable

Standard Example:

Get Mistic PID Mode

PID Loop	Extruder_Zone08	*PID Loop*
Mode	ZONE08_MODE	*Integer 32 Variable*

OptoScript Example:

`GetMisticPidMode(`*PID Loop*`)`

```
ZONE08_MODE = GetMisticPidMode(Extruder_Zone08);
```

This is a function command; it returns a zero (auto mode) or a 1 (manual mode). The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Get Mistic PID Output

PID—Mistic Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the output value of the PID.

Typical Use: To read the current PID output and store it for future use.

Details: The value read has the same engineering units as the specified PID output channel.

Arguments:

Argument 1	Argument 2
PID Loop	**Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID Output

PID Loop	HEATER_3	*PID Loop*
Output	TPO_OUTPUT	*Analog Output*

OptoScript Example:

GetMisticPidOutput(*PID Loop***)**

```
TPO_OUTPUT = GetMisticPidOutput(HEATER_3);
```

This is a function command; it returns the output value of the PID loop. The returned value can be consumed by an analog output (as in the example shown) or by a variable, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- This command can also be used to detect when the PID output is updated (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.

See Also: Enable Communication to Mistic PID Loop (page E-5)

Get Mistic PID Output Rate of Change

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

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 percent per scan period. For example, if the PID output channel is 4–20 mA, 16.00 would be returned by default, representing 100 percent of the span.

Arguments:

Argument 1	Argument 2
From PID Loop	**Put in**
PID Loop	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID Output Rate of Change

From PID Loop	HEATER_3	*PID Loop*
Put in	PID_RATE_LIMIT	*Float Variable*

OptoScript Example:

GetMisticPidOutputRateOfChange(*From PID Loop***)**

```
PID_RATE_LIMIT = GetMisticPidOutputRateOfChange(HEATER_3);
```

This is a function command; it returns the output rate-of-change limit of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- Many additional PID loop control features are available. See the *Mistic Analog and Digital Commands Manual* (Opto 22 form 270) or consult Opto 22 Product Support.

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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Set Mistic PID Output Rate of Change (page S-50), Set Mistic PID Scan Rate (page S-52)

Get Mistic PID P Term

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: Reads the gain value from the PID.

Typical Use: To store "as found" PID parameters for later use.

Details: Reads the gain value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	Argument 2
From PID Loop	**Put in**
PID Loop	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID P Term

From PID Loop	Extruder_Zone08	*PID Loop*
Put in	Zone08_PTerm	*Float Variable*

OptoScript Example:

GetMisticPidPTerm(*From PID Loop***)**

```
Zone08_PTerm = GetMisticPidPTerm(Extruder_Zone08);
```

This is a function command; it returns the gain value from the PID. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Always use a float variable to store the result.

See Also: Set Mistic PID P Term (page S-51)

Get Mistic PID Scan Rate

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O or the SNAP-PID-V module.

Function: Gets the PID calculation interval.

Typical Use: To store "as found" PID parameters for later use.

Details: Reads the Scan Rate value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	Argument 2
From PID Loop	**Put in**
PID Loop	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID Scan Rate

From PID Loop	Extruder_Zone08	*PID Loop*
Put in	Zone08_Scan_Rate	*Float Variable*

OptoScript Example:

`GetMisticPidScanRate(`*From PID Loop***`)`**

```
Zone08_Scan_Rate = GetMisticPidScanRate(Extruder_Zone08);
```

This is a function command; it returns the PID calculation interval (scan rate) for the PID loop. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Always use a float variable to store the result.

See Also: Set Mistic PID Scan Rate (page S-52)

Get Mistic PID Setpoint

PID—Mistic Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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.
- This command retrieves the setpoint value from the most recent scan. To find out the value right now, independent of scan time, use Get PID Current Setpoint.
- The setpoint can be an analog channel or a PID output (for cascaded PIDs), or it can be determined by the strategy in the control engine using Set PID Setpoint.

Arguments:

Argument 1	Argument 2
PID Loop	**Setpoint**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get Mistic PID Setpoint

PID Loop	Heater_3	*PID Loop*
Setpoint	Pid_Setpoint_Value	*Float Variable*

OptoScript Example:

GetMisticPidSetpoint(*PID Loop***)**

```
PID_Setpoint_Value = GetMisticPidSetpoint(Heater_3);
```

This is a function command; it returns the setpoint value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- 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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Get PID Current Setpoint (page G-114), Set Mistic PID Setpoint (page S-53)

Get Month

Time/Date Action

Function: To read the month value (1 through 12) from the control engine'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, 2002, this action would place the value 3 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Month

Put In	MONTH	*Integer 32 Variable*

OptoScript Example:

`GetMonth()`

```
MONTH = GetMonth();
```

This is a function command; it returns a value representing the current month (1 through 12). The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

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 (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Seconds (page G-135), Get Year (page G-145), Set Day (page S-17), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Get Name of Chart Causing Current Error

Error Handling Action

Function: Gets the name of the chart that caused the top queue error.

Typical Use: In an error handling chart to build a history of errors.

Details: Only works when the top queue error is *not* an I/O unit error.

Arguments:

Argument 1
Put in
String Variable

Standard Example:

Get Name of Chart Causing Current Error

Put in	CHART_NAME	*String Variable*

OptoScript Example:

`GetNameOfChartCausingCurrentError(`*Put in***`)`**

```
(GetNameOfChartCausingCurrentError(CHART_NAME);
```

This is a procedure command; it does not return a value.

Notes: String length for name should be at least 50.

Dependencies: The top queue error must *not* be an I/O unit error.

See Also: Get ID of Block Causing Current Error (page G-64) Get Line Causing Current Error (page G-84), Get Name of I/O Unit Causing Current Error (page G-99)

Get Name of I/O Unit Causing Current Error

Error Handling Action

Function: Gets the name of the I/O unit that caused the top queue error.

Typical Use: In an error handling chart to build a history of errors.

Details: Only works when the top queue error is an I/O unit error.

Arguments:

Argument 1
Put in
String Variable

Standard Example:

Get Name of I/O Unit Causing Current Error

Put in	IO_UNIT_NAME	*String Variable*

OptoScript Example:

GetNameOfIoUnitCausingCurrentError(*Put in***)**

```
GetNameOfIoUnitCausingCurrentError(IO_UNIT_NAME);
```

This is a procedure command; it does not return a value.

Notes: String length for name should be at least 50.

Dependencies: The top queue error must be an I/O unit error.

See Also: Get Name of Chart Causing Current Error (page G-98), Get ID of Block Causing Current Error (page G-64) Get Line Causing Current Error (page G-84)

Get Nth Character

String Action

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 in OptoScript code, but not in standard ioControl code.
- Valid range for the *Index* parameter (*Argument 2*) is 0 to the string length.
- A negative result (-12) indicates an error in the value of the Index parameter used.

Arguments:

Argument 1	Argument 2	Argument 3
From String	**Index**	**Put Result in**
String Literal	Integer 32 Literal	Float Variable
String Variable	Integer 32 Variable	Integer 32 Variable

Standard Example: The following example gets the decimal ASCII value for a character in the string “ABC.” If the *Index* is 0, the returned value will be 65 (the decimal ASCII value for “A”). Quotes are shown in the example for clarity only; do not use quotes in standard commands.

Get Nth Character

From String	“ABC”	*String Literal*
Index	INDEX	*Integer 32 Variable*
Put Result in	ASCII_VALUE	*Integer 32 Variable*

OptoScript Example: **GetNthCharacter(***From String, Index***)**

```
ASCII_VALUE = GetNthCharacter("ABC”, INDEX);
```

This is a function command; it returns the ASCII value for a character within a string. Quotes are required in OptoScript code. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure. See Chapter 11 of the *ioControl User’s Guide* for more information.

Notes:

- See “String Commands” in Chapter 10 of the *ioControl User’s Guide*.
- 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.

Status Codes: -12 = Invalid index.

See Also: Get Substring (page G-139), Append Character to String (page A-9), Get String Length (page G-138)

Get Number of Characters Waiting

Communication Action

Function: To get the number of characters available to be received from a communication handle and put it into a numeric variable.

Typical Use: To determine if there are any characters or a particular number of characters to be received before actually receiving them, or to determine the size of a file that's just been opened.

Details:

- A value of 0 means there are no characters to be received. A negative value indicates an error.
- Each character counts as one regardless of what it is.
- For Ethernet, the maximum number of characters that can be buffered is 8760, and any value greater than zero indicates the actual number of characters waiting in the receive buffer.
- When using the file communication handle, this command returns the size of the file (if just opened) or the number of characters after the current position (if some characters have already been read or received, or the position has been moved).
- This command cannot be used with an FTP communication handle.

Arguments:

Argument 1	Argument 2
Communication Handle	**Put In**
Communication Handle	Float Variable
	Integer 32 Variable

Standard Example:

Get Number of Characters Waiting

Communication Handle	UIO_A	*Communication Handle*
Put in	CHAR_COUNT	*Integer 32 Variable*

OptoScript Example:

`GetNumCharsWaiting(`*Communication Handle***`)`**

```
CHAR_COUNT = GenNumCharsWaiting(UIO_A);
```

This is a function command; it returns the number of characters available to be received. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use to determine if the number of characters expected equals the number of characters actually ready to be received.
- If result > 0, there are characters available to be received.
- If result = 0, there are no characters to be received.
- If result < 0, there was an error executing this command. For example, the communication handle may not be opened (use Open Outgoing Communication).

Queue Errors: -36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-37 = Lock port timeout.

-39 = Receive timeout.

-52 = Invalid connection—not opened.

-53 = Connection number not valid.

See Also: Send Communication Handle Command (page S-2), especially the getpos and setpos commands

Get Off-Latch

Digital Point Action

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:

- Standard digital only. For high-density digital, see Get HDD Module Off-Latches.
- 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 control engine to detect, since they are processed locally by the I/O unit.
- Places the value read into the argument specified by the *Put In* parameter. The argument will contain a non-zero value (True) if the latch is set and 0 (False) if the latch is not set.

Arguments:

Argument 1 **From Point**	Argument 2 **Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get Off-Latch

From Point	START_BUTTON	*Digital Input*
Put in	RELEASED	*Float Variable*

OptoScript Example:

GetOffLatch(*On Point***)**

```
if (GetOffLatch(START_BUTTON)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

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 standard digital inputs.

See Also: Get & Clear Off-Latch (page G-25), Clear Off-Latch (page C-26), Clear All Latches (page C-20), Off-Latch Set? (page O-2)

Pro Get Off-Pulse Measurement

Digital Point Action

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
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Standard Example:

Get Off-Pulse Measurement

From Point	Overheat_Switch	*Off Pulse*
Put in	OFF_TIME	*Float Variable*

OptoScript Example:

GetOffPulseMeasurement(*From Point***)**

```
OFF_TIME = GetOffPulseMeasurement(Overheat_Switch);
```

This is a function command; it returns the duration of the first off-pulse for the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use Get Off-Pulse Measurement Complete Status 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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart Off-Pulse Measurement (page G-27), Get Off-Pulse Measurement Complete Status (page G-104)

Get Off-Pulse Measurement Complete Status

Digital Point Action

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 *Put In* parameter. The argument will contain a non-zero value (True) if the measurement is complete or a 0 (False) if it is incomplete.

Arguments:

Argument 1	Argument 2
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Standard Example:

Get Off-Pulse Measurement Complete Status

From Point	Overheat_Switch	*Off Pulse*
Put in	Pulse_Complete	*Integer 32 Variable*

OptoScript Example:

`GetOffPulseMeasurementCompleteStatus(`*From Point***`)`**

```
Pulse_Complete = GetOffPulseMeasurementCompleteStatus(Overheat_Switch);
```

This is a function command; it returns a value of true (-1) or false (0), indicating whether a complete measurement has occurred. The returned value can be consumed by a variable (as in the example shown) or by a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

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 Measurement or Get & Restart Off-Pulse Measurement to read the value.

Dependencies:

- Applies only to inputs configured with the off-pulse measurement feature.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get Off-Pulse Measurement (page G-103), Get & Restart Off-Pulse Measurement (page G-27)

Get Off-Time Totalizer

Digital Point Action

NOTE: This command is for mistic I/O units only.

Function: To read digital input total off time.

Typical Use: To accumulate the total off time of a device to possibly indicate downtime.

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
From Point	**Put in**
Off Totalizer	Float Variable
	Integer 32 Variable

Standard Example:

Get Off-Time Totalizer

From Point	Heater_Output	*Off Totalizer*
Put in	Heater_Down_Time	*Float Variable*

OptoScript Example:

`GetOffTimeTotalizer(`*From Point***`)`**

```
Heater_Down_Time = GetOffTimeTotalizer(Heater_Output);
```

This is a function command; it returns the total time the digital input was off. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- To ensure the totalizer is cleared at start-up, use Get & Restart Off-Time Totalizer 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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart Off-Time Totalizer (page G-28)

Get On-Latch

Digital Point Action

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:

- Standard digital only. For high-density digital, see Get HDD Module On-Latches.
- 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 control engine to detect, since they are processed locally by the I/O unit.
- Places the value read into the argument specified by the *Put In* parameter. The argument will contain a non-zero value (True) if the latch is set and 0 (False) if the latch is not set.

Arguments:

Argument 1	**Argument 2**
From Point	**Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get On-Latch

From Point	ESTOP_BUTTON	*Smart Digital Input*
Put in	EMERGENCY_STOP	*Float Variable*

OptoScript Example:

`GetOnLatch(`*On Point***`)`**

```
if (GetOnLatch(ESTOP_BUTTON)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

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 standard digital inputs.

See Also: Get & Clear On-Latch (page G-26), Clear On-Latch (page C-27), Clear All Latches (page C-20), On-Latch Set? (page O-4)

Get On-Pulse Measurement

Digital Point Action

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**
From Point	**Put in**
On Pulse	Float Variable
	Integer 32 Variable

Standard Example:

Get On-Pulse Measurement

From Point	Overspeed_Switch	*On Pulse*
Put in	On_Time	*Float Variable*

OptoScript Example:

`GetOnPulseMeasurement(`*From Point***`)`**

```
On_Time = GetOnPulseMeasurement(Overspeed_Switch);
```

This is a function command; it returns the duration of the first on-pulse for the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- Use Get On-Pulse Measurement Complete Status 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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart On-Pulse Measurement (page G-29), Get On-Pulse Measurement Complete Status (page G-108)

Get On-Pulse Measurement Complete Status

Digital Point Action

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 *Put In* parameter. The argument will contain a non-zero value (True) if the measurement is complete or a 0 (False) if it is incomplete.

Arguments:

Argument 1	Argument 2
From Point	**Put in**
On Pulse	Float Variable
	Integer 32 Variable

Standard Example:

Get On-Pulse Measurement Complete Status

From Point	Pressure_Switch	*On Pulse*
Put in	Pulse_Complete	*Integer 32 Variable*

OptoScript Example:

GetOnPulseMeasurementCompleteStatus(*From Point***)**

```
Pulse_Complete = GetOnPulseMeasurementCompleteStatus(Pressure_Switch);
```

This is a function command; it returns a value of true (-1) or false (0), indicating whether a complete measurement has occurred. The returned value can be consumed by a variable (as in the example shown) or by a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

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 Measurement or Get & Restart On-Pulse Measurement to read the value.

Dependencies:

- Applies only to inputs configured with the on-pulse measurement feature.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart On-Pulse Measurement (page G-29), Get On-Pulse Measurement (page G-107)

Get On-Time Totalizer

Digital Point Action

NOTE: This command is for mistic I/O units only.

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**
From Point	**Put in**
On Totalizer	Float Variable
	Integer 32 Variable

Standard Example:

Get On-Time Totalizer

From Point	Pump_Power	*On Totalizer*
Put in	Pump_Runtime	*Float Variable*

OptoScript Example:

GetOnTimeTotalizer(*From Point***)**

```
Pump_Runtime = GetOnTimeTotalizer(Pump_Power);
```

This is a function command; it returns the total time the digital input was on. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- To ensure the totalizer is cleared at start-up, use Get & Restart On-Time Totalizer 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.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart On-Time Totalizer (page G-30)

Get Period

Digital Point Action

NOTE: This command is for mistic I/O units only.

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.
- Not available on SNAP Ethernet brains.

Arguments:

Argument 1 **From Point**	Argument 2 **Put in**
Period	Float Variable Integer 32 Variable

Standard Example:

Get Period

From Point	SHAFT_INPUT	*Period*
Put in	SHAFT_CYCLE	*Float Variable*

OptoScript Example:

GetPeriod(*From Point***)**

```
SHAFT_CYCLE = GetPeriod(SHAFT_INPUT);
```

This is a function command; it returns the period for the digital input. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:
- This command measures the first complete period only. No period measurement is performed after the first measurement until the Get & Restart Period 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 & Restart Period command must be used to start the measurement.
- Applies only to inputs configured with the period feature.
- Available on mistic multifunction I/O units. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart Period (page G-31)

Get Period Measurement Complete Status

Digital Point Action

NOTE: This command is for mistic I/O units only.

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 *Put In* parameter. The argument will contain a non-zero value (True) if the measurement is complete or a 0 (False) if it is incomplete.
- Not available on SNAP Ethernet brains.

Arguments:

Argument 1	Argument 2
From Point	**Put in**
Period	Float Variable
	Integer 32 Variable

Standard Example:

Get Period Measurement Complete Status

From Point	Pressure_Switch	*Period*
Put in	Period_Complete	*Integer 32 Variable*

OptoScript Example:

`GetPeriodMeasurementCompleteStatus(`*From Point***`)`**

```
Period_Complete = GetPeriodMeasurementCompleteStatus(Pressure_Switch);
```

This is a function command; it returns a value of true (-1) or false (0), indicating whether a complete measurement has occurred. The returned value can be consumed by a variable (as in the example shown) or by a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

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 & Restart Period to read the value.

Dependencies:

- Applies only to inputs configured with the period measurement feature.
- Available on mistic multifunction I/O units. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Get & Restart Period (page G-31), Get Period (page G-110)

Get PID Configuration Flags

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the current PID configuration options.

Typical Use: To find out current configuration options.

Details: PID configuration options can be set when you initially configure the PID loop in ioManager or ioControl, or in strategy logic using the command Set PID Configuration Flags.

Configuration options are returned as a 32-bit integer. One or multiple options can be chosen. Possible values (in hex) are:

- 0x00000000 = Standard; no special flags.
- 0x00000001 = Square root of input is enabled.
- 0x00000002 = If input goes out of range, output will be forced to a predetermined value.
- 0x00000004 = If input goes out of range, PID will switch to manual; if input returns to normal range, PID will switch back to automatic.

Arguments:

Argument 1	Argument 2
PID Loop	**Configuration Flags**
PID Loop	Integer 32 Variable

Standard Example:

Get PID Configuration Flags

PID Loop	HEATER_3	*PID Loop*
Configuration Flags	PID_CONFIG_FLAGS	*Integer 32 Variable*

OptoScript Example:

GetPidConfigFlags(*PID Loop***)**

```
PID_CONFIG_FLAGS = GetPidConfigFlags(HEATER_3);
```

This is a function command; it returns an integer 32 containing the PID configuration flags from the Ultimate I/O brain's memory map (see Details, above). The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Set PID Configuration Flags (page S-59)

Get PID Current Input

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the input value (also known as the process variable) of the PID at a specific point in time.

Typical Use: To find out current PID values.

Details:

- This command is similar to Get PID Input; however, Get PID Input retrieves the input value from the most recent scan. Since values may fluctuate between scan times, Get PID Current Input retrieves the value right now, independent of scan time.
- The value read has the same engineering units as the specified PID input channel.
- The input can be an analog channel or a PID output (for cascaded PIDs), or it can be determined by the strategy in the control engine using Set PID Input.

Arguments:

Argument 1	Argument 2
PID Loop	**Input**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Current Input

PID Loop	HEATER_3	*PID Loop*
Input	PID_INPUT_VALUE	*Float Variable*

OptoScript Example:

GetPidCurrentInput(*PID Loop***)**

```
PID_INPUT_VALUE = GetPidCurrentInput(HEATER_3);
```

This is a function command; it returns the current input value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- Use to detect bad or out-of-range PID input values. When such a value is found, use the Set PID Output 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.

See Also: Enable Communication to PID Loop (page E-6), Get PID Input (page G-120), Set PID Input (page S-65)

Get PID Current Setpoint

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the setpoint value of the PID at a specific point in time.

Typical Use: To verify that the setpoint of the PID is as expected.

Details:

- This command is similar to Get PID Setpoint; however, Get PID Setpoint retrieves the input value from the most recent scan. Since values may fluctuate between scan times, Get PID Current Setpoint retrieves the value right now, independent of scan time.
- The value read has the same engineering units as the specified PID setpoint.
- The setpoint can be an analog channel or a PID output (for cascaded PIDs), or it can be determined by the strategy in the control engine using Set PID Setpoint.

Arguments:

Argument 1	Argument 2
PID Loop	**Setpoint**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Current Setpoint

PID Loop	Heater_3	*PID Loop*
Setpoint	Pid_Setpoint_Value	*Float Variable*

OptoScript Example:

`GetPidCurrentSetpoint(`*PID Loop***`)`**

```
PID_Setpoint_Value = GetPidCurrentSetpoint(Heater_3);
```

This is a function command; it returns the current setpoint value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- 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.

See Also: Enable Communication to PID Loop (page E-6), Get PID Setpoint (page G-130), Set PID Setpoint (page S-75)

Get PID Feed Forward

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the PID feed forward value for applications requiring feed forward control.

Typical Use: To determine current PID values.

Details: For all four PID algorithms, Feed Forward and Feed Forward Gain values are multiplied and then added to the output; therefore, a value of 0 in either field results in no change to the output.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Feed Forward**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Feed Forward

PID Loop	HEATER_3	*PID Loop*
Feed Forward	PID_FEED_FORWARD	*Float Variable*

OptoScript Example:

GetPidFeedForward(*PID Loop***)**

```
PID_FEED_FORWARD = GetPidFeedForward(HEATER_3);
```

This is a function command; it returns the feed forward value for the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Set PID Feed Forward (page S-60)

Get PID Feed Forward Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the feed forward gain value of the PID output for applications requiring feed forward control.

Typical Use: To determine current PID values.

Details: For all four PID algorithms, Feed Forward and Feed Forward Gain values are multiplied and then added to the output; therefore, a value of 0 in either field results in no change to the output.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Feed Fwd Gain**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Feed Forward Gain

PID Loop	HEATER_3	*PID Loop*
Feed Fwd Gain	PID_FEED_FD_GAIN	*Float Variable*

OptoScript Example:

GetPidFeedForwardGain(*PID Loop***)**

```
PID_FEED_FD_GAIN = GetPidFeedForwardGain(HEATER_3);
```

This is a function command; it returns the feed forward gain value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Set PID Feed Forward Gain (page S-61)

Get PID Forced Output When Input Over Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the forced value that will be sent to the PID output when the input is over the established range.

Typical Use: To determine current PID values.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Forced Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Forced Output When Input Over Range

PID Loop	HEATER_3	*PID Loop*
Forced Output	PID_OUTPUT_OVER_RANGE	*Float Variable*

OptoScript Example:

`GetPidForcedOutputWhenInputOverRange(`*PID Loop***`)`**

```
PID_OUTPUT_OVER_RANGE = GetPidForcedOutputWhenInputOverRange(HEATER_3);
```

This is a function command; it returns the output that will be forced if the input is over the normal range. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Get PID Forced Output When Input Under Range (page G-118), Set PID Forced Output When Input Over Range (page S-62)

Get PID Forced Output When Input Under Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the forced value that will be sent to the PID output when the input is under the established range.

Typical Use: To determine current PID values.

Arguments:

Argument 1	Argument 2
PID Loop	**Forced Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Forced Output When Input Under Range

PID Loop	HEATER_3	*PID Loop*
Forced Output	PID_OUTPUT_UNDER_RANGE	*Float Variable*

OptoScript Example:

`GetPidForcedOutputWhenInputUnderRange(`*PID Loop***`)`**

```
PID_OUTPUT_UNDER_RANGE = GetPidForcedOutputWhenInputUnderRange(HEATER_3);
```

This is a function command; it returns the output that will be forced if the input is under the normal range. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Get PID Forced Output When Input Over Range (page G-117), Set PID Forced Output When Input Under Range (page S-63)

Get PID Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: Reads the gain value from the PID.

Typical Use: To store PID parameters for later use.

Details: Reads the gain value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	Argument 2
PID Loop	**Gain**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Gain

PID Loop	Extruder_Zone08	*PID Loop*
Gain	Zone08_Gain	*Float Variable*

OptoScript Example:

GetPidGain(*PID Loop***)**

```
Zone08_Gain = GetPidGain(Extruder_Zone08);
```

This is a function command; it returns the gain value from the PID. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- To store the result, always use a float variable.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Set PID Gain (page S-64)

Get PID Input

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the input value (also known as the process variable) of the PID.

Typical Use: To find out the PID input value at the time of the most recent scan.

Details:
- The value read has the same engineering units as the specified PID input channel.
- This command retrieves the input value from the most recent scan. To find out the value right now, independent of scan time, use Get PID Current Input.
- The input can be an analog channel or a PID output (for cascaded PIDs), or it can be determined by the strategy in the control engine using Set PID Input.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Input**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Input

PID Loop	HEATER_3	*PID Loop*
Input	PID_INPUT_VALUE	*Float Variable*

OptoScript Example:

`GetPidInput(`*PID Loop***`)`**

```
PID_INPUT_VALUE = GetPidInput(HEATER_3);
```

This is a function command; it returns the input value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:
- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- Use to detect bad or out-of-range PID input values. When such a value is found, use the Set PID Output 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.

See Also: Enable Communication to PID Loop (page E-6), Get PID Current Input (page G-113), Set PID Input (page S-65)

Get PID Input High Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the highest expected value from the PID's input.

Typical Use: To determine current PID configuration.

Arguments:

Argument 1	**Argument 2**
PID Loop	**High Range**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Input High Range

PID Loop	HEATER_3	*PID Loop*
High Range	PID_High_Range	*Float Variable*

OptoScript Example:

GetPidInputHighRange(*PID Loop***)**

```
PID_HIGH_RANGE = GetPidInputHighRange(HEATER_3);
```

This is a function command; it returns the highest valid input of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Input Low Range (page G-122), Set PID Input High Range (page S-66)

Get PID Input Low Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the lowest expected value from the PID's input.

Typical Use: To determine current PID configuration.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Low Range**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Input Low Range

PID Loop	HEATER_3	*PID Loop*
Low Range	PID_LOW_RANGE	*Float Variable*

OptoScript Example:

GetPidInputLowRange(*PID Loop***)**

```
PID_LOW_RANGE = GetPidInputLowRange(HEATER_3);
```

This is a function command; it returns the lowest valid input of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Input High Range (page G-121), Set PID Input Low Range (page S-67)

Get PID Max Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the maximum output change limit of the PID.

Typical Use: To find out current PID parameters and save them for future use.

Details:

- The max output change value defines the maximum amount that the PID output is allowed to change per scan period. This value makes sure the output will ramp up, for example, rather than increasing too quickly. The units are the same as those defined for the PID output point.
- The default value is the range of the output point. This allows the PID output to move as much as 100 percent per scan period. For example, if the PID output point is 4–20 mA, 16.00 would be returned by default, representing 100 percent of the range.
- Note that the max output change limits the PID algorithm and may slow it down.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Max Change**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Max Output Change

PID Loop	HEATER_3	*PID Loop*
Max Change	PID_MAX_LIMIT	*Float Variable*

OptoScript Example:

`GetPidMaxOutputChange(`*PID Loop*`)`

```
PID_MAX_LIMIT = GetPidMaxOutputChange(HEATER_3);
```

This is a function command; it returns the maximum possible change in the output of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Max Output Change (page G-123), Set PID Scan Time (page S-74)

Get PID Min Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the minimum amount of change that must occur before the PID output will change.

Typical Use: To find out current PID parameters and save them for future use.

Details:

- The min output change value defines how much the PID output must change for the change to be applied. A minimum value avoids constant changing, which might wear out valve linkage, for example. The units are the same as those defined for the PID output channel.
- The default value is zero (no minimum). The value must be a positive number.
- The change is applied when it exceeds the minimum in either direction (up or down).

Arguments:

Argument 1	**Argument 2**
PID Loop	**Min Change**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Min Output Change

PID Loop	HEATER_3	*PID Loop*
Min Change	PID_MIN_LIMIT	*Float Variable*

OptoScript Example:

GetPidMinOutputChange(*PID Loop***)**

```
PID_MIN_LIMIT = GetPidMinOutputChange(HEATER_3);
```

This is a function command; it returns the minimum possible change in the output of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Max Output Change (page G-123), Set PID Min Output Change (page S-69), Set PID Scan Time (page S-74)

Get PID Mode

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read whether the PID is in auto or manual mode.

Typical Use: To store current PID parameters for later use.

Details:
- Reads auto/manual mode from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).
- Returns a zero if in auto mode or a 1 if in manual mode.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Mode**
PID Loop	Integer 32 Variable

Standard Example:

Get PID Mode

PID Loop	Extruder_Zone08	*PID Loop*
Mode	ZONE08_MODE	*Integer 32 Variable*

OptoScript Example:

`GetPidMode(`*PID Loop***`)`**

```
ZONE08_MODE = GetPidMode(Extruder_Zone08);
```

This is a function command; it returns a zero (auto mode) or a 1 (manual mode). The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

See Also: Set PID Mode (page S-70)

Get PID Output

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the output value of the PID.

Typical Use: To read the current PID output and store it for future use.

Details: The value read has the same engineering units as the specified PID output channel.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Output

PID Loop	HEATER_3	*PID Loop*
Output	TPO_OUTPUT	*Analog Output*

OptoScript Example:

GetPidOutput(*PID Loop***)**

```
TPO_OUTPUT = GetPidOutput(HEATER_3);
```

This is a function command; it returns the output value of the PID loop. The returned value can be consumed by an analog output (as in the example shown) or by a variable, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- This command can also be used to detect when the PID output is updated (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.

See Also: Enable Communication to PID Loop (page E-6), Set PID Output (page S-71)

Get PID Output High Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the high clamp value currently set for the PID output.

Typical Use: To determine current PID values.

Details: The output low clamp and high clamp values define the range of output for this PID loop.

Arguments:

Argument 1	Argument 2
PID Loop	**High Clamp**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Output High Clamp

PID Loop	HEATER_3	*PID Loop*
High Clamp	PID_HIGH_CLAMP	*Float Variable*

OptoScript Example:

`GetPidOutputHighClamp(`*PID Loop***`)`**

```
PID_HIGH_CLAMP = GetPidOutputHighClamp(HEATER_3);
```

This is a function command; it returns the highest possible value for the output of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Output Low Clamp (page G-128), Set PID Output High Clamp (page S-72)

Get PID Output Low Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the low clamp value currently set for the PID output.

Typical Use: To determine current PID values.

Details: The output low clamp and high clamp values define the range of output for this PID loop.

Arguments:

Argument 1	Argument 2
PID Loop	**Low Clamp**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Output Low Clamp

PID Loop	HEATER_3	*PID Loop*
Low Clamp	PID_LOW_CLAMP	*Float Variable*

OptoScript Example:

GetPidOutputLowClamp(*PID Loop***)**

```
PID_LOW_CLAMP = GetPidOutputLowClamp(HEATER_3);
```

This is a function command; it returns the lowest possible value for the output of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Output High Clamp (page G-127), Set PID Output Low Clamp (page S-73)

Get PID Scan Time

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: Gets the PID calculation interval (the scan time).

Typical Use: To store current PID parameters for later use.

Details: Reads the Scan Time value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	Argument 2
PID Loop	**Scan Time (sec)**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Scan Time

PID Loop	Extruder_Zone08	*PID Loop*
Scan Time (sec)	Zone08_Scan_Time	*Float Variable*

OptoScript Example:

`GetPidScanTime(`*PID Loop***`)`**

```
Zone08_Scan_Time = GetPidScanTime(Extruder_Zone08);
```

This is a function command; it returns the PID calculation interval (scan time) for the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- To store the result, always use a float variable.

See Also: Set PID Scan Time (page S-74)

Get PID Setpoint

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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.
- This command retrieves the setpoint value from the most recent scan. To find out the value right now, independent of scan time, use Get PID Current Setpoint.
- The setpoint can be an analog channel or a PID output (for cascaded PIDs), or it can be determined by the strategy in the control engine using Set PID Setpoint.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Setpoint**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Setpoint

PID Loop	Heater_3	*PID Loop*
Setpoint	Pid_Setpoint_Value	*Float Variable*

OptoScript Example:

`GetPidSetpoint(`*PID Loop***`)`**

```
PID_Setpoint_Value = GetPidSetpoint(Heater_3);
```

This is a function command; it returns the setpoint value of the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- 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.

See Also: Enable Communication to PID Loop (page E-6), Get PID Current Setpoint (page G-114), Set PID Setpoint (page S-75)

Get PID Status Flags

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To read the current state of PID flags.

Typical Use: To determine whether input is below or above normal range and whether the output is being forced.

Details: Returns a bit mask that indicates current PID status data. More than one flag can be set at a time. Use bitwise commands to get each flag. Flag values are:

- 0x00000001 = Input is below input low range
- 0x00000002 = Input is above input high range
- 0x00000004 = Input was out of range and output is being forced to a predetermined value set during PID configuration

Arguments:

Argument 1	Argument 2
PID Loop	**Status Flags**
PID Loop	Integer 32 Variable

Standard Example:

Get PID Status Flags

PID Loop	HEATER_3	*PID Loop*
Status Flags	PID_STATUS_FLAGS	*Integer 32 Variable*

OptoScript Example:

`GetPidStatusFlags(`*PID Loop*`)`

```
PID_STATUS_FLAGS = GetPidStatusFlags(HEATER_3);
```

This is a function command; it returns an integer 32 containing the PID status flags from the Ultimate I/O brain's memory map. Possible values are listed above.

The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes: See "PID Commands" in Chapter 10 of the ioControl User's Guide.

Dependencies: Communication to the PID must be enabled for this command to read the actual value from the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Configuration Flags (page G-112), Set PID Configuration Flags (page S-59)

Get PID Tune Derivative

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: Reads the derivative tuning value from the PID.

Typical Use: To store current PID parameters for later use.

Details: Reads the derivative value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	Argument 2
PID Loop	**Tune Derivative**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Tune Derivative

PID Loop	Extruder_Zone08	*PID Loop*
Tune Derivative	Zone08_Derivative	*Float Variable*

OptoScript Example:

GetPidTuneDerivative(*PID Loop***)**

```
Zone08_Derivative = GetPidTuneDerivative(Extruder_Zone08);
```

This is a function command; it returns the derivative value from the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- To store the result, always use a float variable.

See Also: Set PID Tune Derivative (page S-76)

Get PID Tune Integral

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: Reads the Integral tuning value from the PID.

Typical Use: To store current PID parameters for later use.

Details: Reads the Integral value from the PID in the I/O unit. If the PID is disabled or the I/O unit is disabled, the last known value will be returned instead (the IVAL).

Arguments:

Argument 1	**Argument 2**
PID Loop	**Tune Integral**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Standard Example:

Get PID Tune Integral

PID Loop	Extruder_Zone08	*PID Loop*
Tune Integral	Zone08_Integral	*Float Variable*

OptoScript Example:

GetPidTuneIntegral(*From PID Loop***)**

```
Zone08_Integral = GetPidTuneIntegral(Extruder_Zone08);
```

This is a function command; it returns the integral value from the PID loop. The returned value can be consumed by a variable (as in the example shown) or by an analog point, a mathematical expression, etc. See Chapter 11 of the ioControl User's Guide for more information.

Notes:

- See "PID Commands" in Chapter 10 of the ioControl User's Guide.
- To store the result, always use a float variable.

See Also: Set PID Tune Integral (page S-77)

Get Pointer From Name

Pointers Action

Function: To assign an object to a pointer based on the object's name.

Typical Use: To help process requests from peers when the object needed may change dynamically.

Details:

- If a variable of the specified name is not found, the pointer is set to null.
- The variable name must match the pointer's type. For example, if the pointer is a float pointer, the variable name must be for a float variable.
- The variable name is case sensitive.

Arguments:

Argument 1	**Argument 2**
Name	**Pointer**
String Literal	Pointer Variable
String Variable	

Standard Example:

Get Pointer From Name

Name	"My_Integer"	*String Literal*
Pointer	pInteger	*Pointer Variable*

OptoScript Example:

`GetPointerFromName(`*Name*, *Pointer***`)`**

```
GetPointerFromName("My_Integer", pInteger);
```

This is a procedure command; it does not return a value.

Notes: For more information on peer-to-peer communication, see "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Move to Pointer (page M-19)

Get Seconds

Time/Date Action

Function: To read the seconds (0 through 59) from the control engine's real-time clock/calendar and put it into a numeric variable.

Typical Use: To use seconds information in an ioControl 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 action would place the value 26 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Seconds

Put In	SECONDS	*Integer 32 Variable*

OptoScript Example:

GetSeconds()

```
SECONDS = GetSeconds();
```

This is a function command; it returns the second (0 through 59) from the control engine's real-time clock. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- This is a one-time read of the second. 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 Seconds Since Midnight (page G-136), Get Day (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), , Get Year (page G-145), Set Day (page S-17), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Get Seconds Since Midnight

Time/Date Action

Function: Gets the number of seconds since midnight.

Typical Use: In place of timers to determine time between events or to time stamp an event with a number rather than a string.

Details: Value returned is an integer from 0 to 86,399.

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Seconds Since Midnight

Put in	TIME_IN_SECONDS	*Integer 32 Variable*

OptoScript Example:

`GetSecondsSinceMidnight()`

```
TIME_IN_SECONDS = GetSecondsSinceMidnight();
```

This is a function command; it returns the number of seconds since midnight. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: To find elapsed time in HOURS, MINUTES, SECONDS since midnight using standard commands:

Move the seconds to an integer 32 variable: *TEMP_VAR*
Divide *TEMP_VAR* by: 3600 and move to: *HOURS*
MODULO *TEMP_VAR* by: 3600 and move to: *TEMP_VAR*
Divide *TEMP_VAR* by: 60 and move to: *MINUTES*
MODULO *TEMP_VAR* by: 60 and move to: *SECONDS.*

To find the same thing using OptoScript code:

```
TEMP_VAR = GetSecondsSinceMidnight();
HOURS = TEMP_VAR / 3600;
MINUTES = (TEMP_VAR % 3600 / 60;
SECONDS = (TEMP_VAR % 3600) % 60;
```

See Also: Get Seconds (page G-135)

Get Severity of Current Error

Error Handling Action

Function: To read the severity of the oldest error in the message queue.

Typical Use: To allow a chart to perform error handling.

Details:

- Valid severity values are:

  0 = Queue is empty
  4 = Info
  8 = Warning
  16 = Error
- The same error is read each time unless Remove Current Error and Point to Next Error is used first.
- The message queue can hold up to 1000 errors.

Arguments:

Argument 1
Put In
Float Variable
Integer 32 Variable

Standard Example:

Get Severity of Current Error		
Put In	nCurrentError	*Integer 32 Variable*

OptoScript Example:

`GetSeverityOfCurrentError()`

```
nCurrentError = GetSeverityOfCurrentError();
```

This is a function command; it returns the severity value of the error. The returned value can be consumed by a variable (as shown) or by another item, such as a control structure.

See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: For detailed information on errors, use Control Engine Inspect in Debug mode to view the message queue.

See Also: Get Error Code of Current Error (page G-52), Clear All Errors (page C-18), Get Error Count (page G-53), Remove Current Error and Point to Next Error (page R-22)

Get String Length

String Action

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 in OptoScript code, but not in standard ioControl code.
- 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 ioControl 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**
Of String	**Put Result in**
String Literal	Float Variable
String Variable	Integer 32 Variable

Standard 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

Of String	MY_STRING	*String Literal*
Put Result in	STRING_LEN	*Integer 32 Variable*

OptoScript Example: **`GetStringLength(`***Of String***`)`**

```
STRING_LEN = GetStringLength(MY_STRING);
```

This is a function command; it returns the length of the string. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use before Get Nth Character to stay within the string length.

See Also: Get Nth Character (page G-100)

Get Substring

String Action

Function: To copy a portion of a string.

Typical Uses: To parse or extract data from a string, to skip leading or trailing characters, or to extract data from strings that may contain starting and ending character sequences generated by barcode readers or scales.

Details:

- Quotes (“”) are used in OptoScript code, but not in standard ioControl code.
- Valid range for Start At Index (*Argument 2*) is 0 to the string length minus one. If it is less than 0 or longer than the From String parameter, a null string is copied to the substring.
- If the combination of the Start At Index (*Argument 2*) and Num. Characters (*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 “MONTUEWEDTHUFRI”:

Start At	Number of Characters	Substring Returned
0	3	“MON”
3	3	“TUE”
0	4	“MONT”
13	3	“RI”
15	5	“”

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
From String	**Start at Index**	**Num. Characters**	**Put Result in**
String Literal	Integer 32 Literal	Integer 32 Literal	String Variable
String Variable	Integer 32 Variable	Integer 32 Variable	

Standard Example: The following example gets a single day from the string “MONTUEWEDTHUFRI”; quotes are shown here for clarity only. Do not use them in standard commands.

Get Substring

From String	“MONTUEWEDTHUFRI”	*String Literal*
Start at Index	INDEX	*Integer 32 Variable*
Num. Characters	3	*Integer 32 Literal*
Put Result in	STRING	*String Variable*

OptoScript Example: **GetSubstring(***From String, Start at Index, Num. Characters, Put Result in***)**

```
GetSubstring("MONTUEWEDTHUFRI", INDEX, 3, STRING);
```

This is a procedure command; it does not return a value. Quotes are required in OptoScript code.

Notes:

- See “String Commands” in Chapter 10 of the *ioControl User’s Guide*.
- 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 (page G-100)

Get System Time

Time/Date Action

Function: Gets the number of seconds since the control engine has been turned on.

Typical Use: Accumulate "up-time."

Details: Value returned is an integer.

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get System Time

Put in	TIME_IN_SECONDS	*Integer 32 Variable*

OptoScript Example:

`GetSystemTime()`

```
TIME_IN_SECONDS = GetSystemTime();
```

This is a function command; it returns the number of seconds since the control engine was last turned on. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Get Seconds Since Midnight (page G-136)

Get Target Address State

I/O Unit Action

Function: To determine which target addresses on an I/O unit in a redundant system are enabled and which address is active.

Typical Use: To determine which networks in a redundant system are enabled and which network is active.

Details:

- A target address is the IP address of an Ethernet interface on an I/O unit.
- In a redundant network architecture, you can assign two target addresses to an I/O unit. In ioControl these are called the Primary Address and the Secondary Address. By default, the Primary Address is used, but the server will switch to the Secondary Address if the primary address is not available.
- Each target address has an *enabled* state and an *active* state. If both target addresses are enabled, they are available to be used. However, only one address can be used at a given time, so there can only be one active address.
- This command returns an Enable Mask value and an Active Mask value for a given I/O unit.
- The Enable Mask indicates which target addresses are active as follows:
  0=No addresses are enabled
  1=Only the Primary Address is enabled
  2=Only the Secondary Address is enabled
  3=Both addresses are enabled.
- The Active Mask indicates which address is active as follows:
  0=No address is active
  1=Primary Address is active
  2=Secondary Address is active

Arguments:

Argument 1	Argument 2	Argument 2
Enable Mask	**Active Mask**	**I/O Unit**
Integer 32 Variable	Integer 32 Variable	SNAP-ENET-D64 SNAP-UP1-D64 SNAP-ENET-S64 SNAP-UP1-M64 SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-PAC-R1 SNAP-PAC-R2

Standard Example:

Get Target Address State

Enable Mask	ENABLE_MASK	*Integer 32 Variable*
Active Mask	ACTIVE_MASK	*Integer 32 Variable*
I/O Unit	UNIT	*SNAP-UP1-ADS*

OptoScript Example:

GetTargetAddressState(*Enable Mask, Active Mask, I/O Unit***)**

```
GetTargetAddressState(ENABLE_MASK, ACTIVE_MASK, UNIT);
```

This is a procedure command; it does not return a value.

Notes:
- A fully redundant system may also include ioDisplay clients and OptoOPCServers. These commands only deal with the control engine communicating with I/O units. ioDisplay and OptoOPCServer have their own mechanism for controlling their use of the network.

See Also: Set All Target Address States (page S-5), Set Target Address State (page S-81)

Get Type From Name

Miscellaneous Action

Function: To find out the data type (string, floating point, etc.) of a variable in the strategy.

Typical Use: Used with the command Get Value From Name, to find out the data type of a variable and pass it to another software application or device that knows only the variable's name.

Details:
- This command does not handle pointers. If the variable is a pointer, a zero will be returned.
- Reads the data type of the variable named in *Argument 1* and places a bitmask in *Argument 2* representing the data type. Possible values (in hex) are as follows:

Value in Hex	Data Type
00020002	Digital I/O Point
00020010	Analog I/O Point
00400005	Mixed I/O Unit
00400006	Digital 64 I/O Unit
00400007	Mixed 64 I/O Unit
00800000	Integer 32
00800001	Integer 64
00800002	Float
00800003	Down Timer

Value in Hex	Data Type
00800004	Up Timer
00810000	Integer 32 Table
00810001	Integer 64 Table
00810002	Float Table
01000000	String
01010000	String Table
02000000	Chart
09000000	Communication Handle

If a variable is persistent, the first digit in hex will be a 4 (bit 30 is set). Examples:

00800001 40800001	Integer 64 Persistent Integer 64

01010000 41010000	String Table Persistent String Table

If a variable is local to a subroutine, the first digit in hex will be a 1 (bit 28 is set). Examples:

10800000	Local Integer 23

10800001	Local Integer 64

Arguments:

Argument 1 **Name**	**Argument 2** **Put in**
String Literal String Variable	Integer 32 Variable

Standard Example:	Get Type From Name		
	Name	Variable_Name	*String Literal*
	Put in	DATA_TYPE	*Integer 32 Variable*

OptoScript Example:

```
GetTypeFromName(Name)
DATA_TYPE = GetTypeFromName(Variable_Name);
```

This is a function command; it returns the data type of the variable in the form of a bitmask. The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Get Value From Name (page G-143)

Get Value From Name

Miscellaneous Action

Function: To find out the value of a variable named in the strategy.

Typical Use: To pass the value of a variable to another software application or device that knows the variable's name. In a subroutine, to find out the current value of a global variable whose name is known.

Details:

- Gets the value of the variable named in *Argument 1* and places that value in the form of a string into *Argument 2*.
- The value of the variable named in *Argument 1* can be of various types; it will automatically be converted into a string.
- The string variable in *Argument 2* must be wide enough to fit any value (converted into a string) that may go there.
- This command can be used with most non-pointer types. It won't work with parameters passed into a subroutine, but can be used with local subroutine variables.
- Types supported include: string and numeric table elements, strings, communication handles, numeric variables, points, and boards.
- If used in a subroutine to find out the current value of a global variable, the subroutine must know the variable's name. The name can be passed in via a string or a string table.
- To get the value of an element in a table, follow the name of the variable with the desired index in square brackets. For example, MyTable[2] would return the value of the third element in MyTable as a string (Argument 2).

Arguments:

Argument 1	Argument 2	Argument 3
Name	**Put Result In**	**Put Status In**
String Literal	String Variable	Integer 32 Variable
String Variable		

Standard Example:	Get Value From Name		
	Name	Item_Count	*String Variable*

Put Result In	Production	*String Variable*
Put Status In	Status	*Integer 32 Variable*

OptoScript Example:

`GetValueFromName(`*Name, Put Result In***`)`**

```
Status = GetValueFromName(Item_Count, Production);
```

This is a function command; it returns the value of the variable in the form of a string. The returned value can be consumed by a variable (as shown in Status Codes below) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Miscellaneous Commands" in Chapter 10 of the *ioControl User's Guide*.
- If you need to know the data type of the variable named in Argument 1, use the command Get Type From Name.
- If you need to use the variable's value in a mathematical computation, convert the string to the data type you need using one of the Convert commands.

Status Codes:

0 = Success

-28 = Object not found. Variable doesn't exist or is spelled incorrectly (name is case-sensitive), or the variable is a pointer or other unsupported type.

-36 = Feature not implemented. The type of the object passed is not yet supported.

-69 = Variable named in Argument 1 not found. Check the name and case.

See Also:

Get Type From Name (page G-142), Convert String to Float (page C-52), Convert String to Integer 32 (page C-54), Convert String to Integer 64 (page C-55)

Get Year

Time/Date Action

Function: To read the year value (2000 through 2099) from the control engine's real-time clock/calendar and put it into a numeric variable.

Typical Use: To use year information in an ioControl program.

Details:

- The destination variable can be an integer or a float, although an integer is preferred.
- If the current date is March 2, 2002, this action would place the value 2002 into the *Put In* parameter (*Argument 1*).

Arguments:

Argument 1
Put in
Float Variable
Integer 32 Variable

Standard Example:

Get Year

Put In	YEAR	*Integer 32 Variable*

OptoScript Example:

`GetYear()`

```
YEAR = GetYear();
```

This is a function command; it returns the four digits of the year (2000 through 2099). The returned value can be consumed by a variable (as in the example shown) or by a mathematical expression, a control structure, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

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 (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Set Day (page S-17), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Greater?

Logical Condition

Function: To determine if one numeric value is greater than another.

Typical Use: To determine if a timer has reached a 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 (non-zero) if *Argument 1* is greater than *Argument 2*, False (zero) otherwise.

Arguments:

Argument 1 Is	Argument 2 Than
Analog Input	Analog Input
Analog Output	Analog Output
Digital Input	Digital Input
Digital Output	Digital Output
Down Timer Variable	Down Timer Variable
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
Up Timer Variable	Up Timer Variable

Standard Example:

Is	CALCULATED_VALUE	*Integer 32 Variable*
Greater?		
Than	1000	*Integer 32 Literal*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `>` operator.

```
if (CALCULATED_VALUE > 1000) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use Within Limits? to test for an approximate match. To test for less than or equal, use either Less Than or Equal? or the false exit.

See Also: Less? (page L-1) Not Equal? (page N-4) Greater Than or Equal? (page G-148) Less Than or Equal? (page L-3) Within Limits? (page W-1)

Greater Than Numeric Table Element?

Logical Condition

Function: To determine if a numeric value is greater than a specified value in a float or 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 float or integer table *Argument 3*). Examples:

Value 1	Value 2	Result
0.0	0.0	False
0.0001	0.0	True
-98.765	-98.765	False
1	0	True
22221	2222	True

- Evaluates True (non-zero) if the first value is greater than the second, False (zero) otherwise.

Arguments:

Argument 1 Is	Argument 2 At Index	Argument 3 Of Table
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Down Timer Variable		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard Example:

Is	THIS_READING	*Float Variable*
Greater Than Numeric Table Element?		
At Index	TABLE_INDEX	*Integer 32 Variable*
Of Table	TABLE_OF_READINGS	*Float Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `>` operator.

```
if (THIS_READING > TABLE_OF_READINGS[TABLE_INDEX]) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To test for less than or equal to, use either Less Than or Equal to Numeric Table Element? or the False exit.

Queue Errors: -12 = Invalid table index.

See Also: Less Than Numeric Table Element? (page L-2) Not Equal to Numeric Table Element? (page N-5) Greater Than or Equal To Numeric Table Element? (page G-149) Less Than or Equal to Numeric Table Element? (page L-4)

Greater Than or Equal?

Logical Condition

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 (non-zero) if the first value is greater than or equal to the second, False (zero) otherwise.

Arguments:

Argument 1 Is	Argument 2 To
Analog Input	Analog Input
Analog Output	Analog Output
Digital Input	Digital Input
Digital Output	Digital Output
Down Timer Variable	Down Timer Variable
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
Up Timer Variable	Up Timer Variable

Standard Example:

Greater Than or Equal?

Is	ROOM_TEMP	*Analog Input*
To	78.5000	*Float Literal*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `>=` operator.

```
if (ROOM_TEMP >= 78.5000) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use Within Limits? to test for an approximate match. To test for less than, use either Less? or the False exit.
- 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? (page L-1) Not Equal? (page N-4) Less Than or Equal? (page L-3) Within Limits? (page W-1)

Greater Than or Equal To Numeric Table Element?

Logical Condition

Function: To determine if a numeric value is greater than or equal to a specified value in a float or 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 float or integer table *Argument 3*). Examples:

Value 1	Value 2	Result
0.0	0.0	True
0.0001	0.0	True
22.22	22.222	False
-32768	-32767	False
22221	2222	True

- Evaluates True (non-zero) if the first value is greater than or equal to the second, False (zero) otherwise.

Arguments:

Argument 1 **Is**	Argument 2 **At Index**	Argument 3 **Of Table**
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Down Timer Variable		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard Example:

Is	THIS_READING	*Float Variable*

Greater Than or Equal to Numeric Table Element?

At Index	TABLE_INDEX	*Integer 32 Variable*
Of Table	TABLE_OF_READINGS	*Float Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `>=` operator.

```
if (THIS_READING >= TABLE_OF_READINGS[TABLE_INDEX]) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. For more on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To test for less than, use either Less Than Table Element? or the False exit.

Queue Errors: -12 = Invalid table index.

See Also: Less Than Numeric Table Element? (page L-2) Not Equal to Numeric Table Element? (page N-5) Greater Than Numeric Table Element? (page G-147) Less Than or Equal to Numeric Table Element? (page L-4)

Hyperbolic Cosine

Mathematical Action

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*.

Arguments:

Argument 1 **Of**	Argument 2 **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Hyperbolic Cosine

Of	2.0	*Float Literal*
Put Result in	ANSWER	*Float Variable*

OptoScript Example:

`HyperbolicCosine(`*`Of`*`)`

```
ANSWER = HyperbolicCosine(2.0);
```

This is a function command; it returns the hyperbolic cosine of the value. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Queue Errors: -13 = Overflow error—result too large.

See Also: Hyperbolic Sine (page H-2), Hyperbolic Tangent (page H-3)

Hyperbolic Sine

Mathematical Action

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*.

Arguments:

Argument 1	**Argument 2**
Of	**Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Hyperbolic Sine

Of	2.0	*Float Literal*
Put Result in	ANSWER	*Float Variable*

OptoScript Example:

`HyperbolicSine(`*`Of`*`)`

```
ANSWER = HyperbolicSine(2.0);
```

This is a function command; it returns the hyperbolic sine of the value. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Queue Errors: -13 = Overflow error—result too large.

See Also: Hyperbolic Cosine (page H-1), Hyperbolic Tangent (page H-3)

Hyperbolic Tangent

Mathematical Action

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*.
- The result is a value ranging from -1.0 to 1.0.

Arguments:

Argument 1 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Hyperbolic Tangent

Of	2.0	*Float Literal*
Put Result in	ANSWER	*Float Variable*

OptoScript Example:

`HyperbolicTangent(`*`Of`*`)`

```
ANSWER = HyperbolicTangent(2.0);
```

This is a function command; it returns the hyperbolic tangent of the value. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Queue Errors: -13 = Overflow error—result too large.

See Also: Hyperbolic Cosine (page H-1), Hyperbolic Sine (page H-2)

Increment Variable

Mathematical Action

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** **[Value]** Float Variable Integer 32 Variable Integer 64 Variable
Standard Example:	Increment Variable LOOP_COUNTER *Integer 32 Variable*
OptoScript Example:	**`IncrementVariable(`***Variable***`)`** `IncrementVariable(LOOP_COUNTER);` This is a procedure command; it does not return a value.
Notes:	• See “Mathematical Commands” in Chapter 10 of the *ioControl User’s Guide*. • Executes faster than adding 1.
See Also:	Decrement Variable (page D-1)

I/O Point Communication Enabled?

Simulation Condition

Function: Checks a flag internal to the control engine to determine if communication to the specified I/O point is enabled.

Typical Use: Primarily used in factory QA testing and simulation.

Details: Evaluates True if communication is enabled.

Arguments:

Argument 1
I/O Point
Analog Input
Analog Output
Digital Input
Digital Output

Standard Example:

I/O Point Communication Enabled?

I/O Point	PUMP_3_STATUS	*Analog Input*

OptoScript Example:

`IsIoPointCommEnabled(`*I/O Point***`)`**

```
if (IsIoPointCommEnabled(PUMP_3_STATUS)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Enable Communication to Point (page E-7), Disable Communication to Point (page D-11), I/O Unit Communication Enabled? (page I-3)

I/O Unit Communication Enabled?

Simulation Condition

Function: Checks a flag internal to the control engine to determine if communication to the specified I/O unit is enabled.

Typical Use: Primarily used in factory QA testing and simulation, and in error handling charts.

Details: Evaluates True if communication is enabled.

Arguments:

Argument 1
I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
G4A8R, G4RAX*
G4D16R*
G4D32RS*
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-ENET-S64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-PAC-R1
SNAP-PAC-R2
SNAP-BRS*

* ioControl Professional only

Standard Example:

I/O Unit	PUMP_HOUSE	*SNAP-ENET-D64*

I/O Unit Communication Enabled?

OptoScript Example:

IsIoUnitCommEnabled(*I/O Unit***)**

```
if (IsIoUnitCommEnabled(PUMP_HOUSE)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Enable Communication to I/O Unit (page E-4), Disable Communication to I/O Unit (page D-7), I/O Point Communication Enabled? (page I-2), I/O Unit Ready? (page I-4)

I/O Unit Ready?

I/O Unit Condition

Function: Tests communication with the specified I/O unit.

Typical Use: To determine if the I/O unit is operational and that communication with it is functional.

Details: The control engine tests communication with the I/O unit by reading the I/O unit's type from the status read area of the memory map and making sure no error is returned. If communication is successful (regardless of whether the I/O unit is enabled or disabled), the condition evaluates True.

Arguments:

Argument 1
Is
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
G4A8R, G4RAX*
G4D16R*
G4D32RS*
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-ENET-S64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-PAC-R1
SNAP-PAC-R2
SNAP-BRS*

* ioControl Professional only

Standard Example:

I/O Unit Ready?		
Is	PUMP_HOUSE	*SNAP-ENET-D64*

OptoScript Example: **IsIoUnitReady(***I/O Unit***)**

```
if (IsIoUnitReady(PUMP_House)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Ideal for determining "System Ready" status.

See Also: I/O Point Communication Enabled? (page I-2), I/O Unit Communication Enabled? (page I-3)

IVAL Move Numeric Table to I/O Unit

I/O Unit Action

Function: Writes to the internal value (IVAL) of all analog points on the I/O unit.

Typical Use: Simulation, testing, and certification where communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows all IVALs to be modified as if they were being changed by real I/O.

Arguments:

Argument 1 **Start at Index**	Argument 2 **Of Table**	Argument 3 **Move to**
Integer 32 Literal	Float Table	B100*
Integer 32 Variable	Integer 32 Table	B200*
		B3000 (Analog)*
		B3000 (Digital)*
		G4A8R, G4RAX*
		G4D16R*
		G4D32RS*
		SNAP-ENET-D64
		SNAP-UP1-D64
		SNAP-UP1-M64
		SNAP-ENET-S64
		SNAP-B3000-ENET, SNAP-ENET-RTC
		SNAP-UP1-ADS
		SNAP-PAC-R1
		SNAP-PAC-R2
		SNAP-BRS*

* ioControl Professional only

Standard Example:

IVAL Set Analog from Table

Start at Index	0	*Integer 32 Literal*
Of Table	TEST_TABLE	*Float Table*
Move to	AI_101	*SNAP-B3000-ENET, SNAP-ENET-RTC*

OptoScript Example:

`IvalSetAnalogFromTable(`*Start at Index, Of Table, Move to***`)`**

```
IvalSetAnalogFromTable(0, TEST_TABLE, AI_101);
```

This is a procedure command; it does not return a value.

Notes: Primarily used to write to inputs.

See Also: IVAL Set Analog Point (page I-6), Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Analog Point

Simulation Action

Function: Writes to the internal value (IVAL) of an analog input or output.

Typical Use: Simulation, testing, and certification where communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	**Argument 2**
To	**On Point**
Float Literal	Analog Input
Float Variable	Analog Output
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

VAL Set Analog Point

To	5.63	*Float Literal*
On Point	PROCESS_PH	*Analog Input*

OptoScript Example:

`IvalSetAnalogPoint(`*To, On Point***`)`**

```
IvalSetAnalogPoint(5.63, PROCESS_PH);
```

This is a procedure command; it does not return a value.

Notes: Primarily used to write to inputs. May be used to test when an output is updated by a change of value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Counter

Simulation Action

Function: Writes to the internal value (IVAL) of a counter or quadrature counter digital input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details:
- The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.
- Valid range for quadrature counters is 0 to 2,147,483,647.

Arguments:

Argument 1	Argument 2
To	**On Point**
Integer 32 Literal	Counter
Integer 32 Variable	Quadrature Counter

Standard Example:

IVAL Set Counter

To	2484	*Integer 32 Literal*
On Point	PROCESS_FLOW_TOTAL	*Counter*

OptoScript Example:

IvalSetCounter(*To, On Point* **)**

```
IvalSetCounter(2484, PROCESS_FLOW_TOTAL);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Digital Binary

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use IVAL Set I/O Unit from MOMO Masks (page I-11) instead.

Function: Writes to the internal value (IVAL) of all 16 digital outputs on the specified I/O unit.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1 **On Mask**	Argument 2 **Off Mask**	Argument 3 **On I/O Unit**
Integer 32 Literal	Integer 32 Literal	B100
Integer 32 Variable	Integer 32 Variable	B3000 (Digital)
		B3000 SNAP Mixed I/O
		G4 Digital Local Simple I/O Unit
		G4D16R
		G4D32RS
		SNAP-BRS

Standard Example:

IVAL Set Digital Binary

On Mask	PUMPS_ON_MASK	*Integer 32 Variable*
Off Mask	0	*Integer 32 Literal*
On I/O Unit	PUMP_CTRL	*B3000 (Digital)*

OptoScript Example:

IvalSetDigitalBinary(*On Mask, Off Mask, On I/O Unit***)**

```
Ival SetDigitalBinary(PUMPS_ON_MASK, 0, PUMP_CTRL);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Digital-64 I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use IVAL Set I/O Unit from MOMO Masks (page I-11) instead.

Function: Writes to the internal values (IVALs) of all points on a digital 64 I/O unit.

Typical Use: For simulation and testing, to assign specific values from a must-on, must-off mask to points.

Details:

- The program will use IVALs exclusively when communication to the I/O unit is disabled. This command allows the IVALs to be modified as if they were being changed by real I/O.
- This command is 64 times faster than using Turn On or Turn Off 64 times. It updates the IVALs for all 64 points. It affects only selected output points and does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2.
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	Argument 2	Argument 3
Must-on Mask	**Must-off Mask**	Digital 64 I/O Unit
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64
Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

IVAL Set Digital-64 I/O Unit from MOMO Masks

Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Digital-64 I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-D64*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example: **IvalSetDigital64IoUnitFromMomo(** *Must-On Mask, Must-Off Mask, Digital-64 I/O Unit* **)**

```
IvalSetDigital64IoUnitFromMomo(0x060003C0000000C2i64,
                                0xB0F240010308A020i64, PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value. (Note that Integer 64 literals in OptoScript code take an `i64` suffix.)

Notes: Primarily used to write to inputs.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7), IVAL Set Mixed I/O Unit from MOMO Masks (page I-16), IVAL Set Mixed 64 I/O Unit from MOMO Masks (page I-15)

Pro IVAL Set Frequency

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: Writes to the internal value (IVAL) of a digital frequency input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	**Argument 2**
To	**On Point**
Integer 32 Literal	Frequency
Integer 32 Variable	

Standard Example:

IVAL Set Frequency

To	400	*Integer 32 Literal*
On Point	Process_Flow_Rate	*Frequency*

OptoScript Example: **IvalSetFrequency(** *To, On Point* **)**

```
IvalSetFrequency(400, Process_Flow_Rate);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0–65535.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set I/O Unit from MOMO Masks

Simulation Action

Function: Writes to the internal value (IVAL) of all digital outputs on the specified I/O unit.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details:

- The program will use IVALs exclusively when communication to the specified I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.
- This command updates the IVALs for all selected output points. It does not affect input points.
- To turn on a point, set the respective bit in the data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in both arguments 1 and 2.
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
On Mask	**Off Mask**	**On I/O Unit**
Integer 32 Literal	Integer 32 Literal	B100
Integer 32 Variable	Integer 32 Variable	B3000 (Digital)
Integer 64 Literal	Integer 64 Literal	B3000 SNAP Mixed I/O
Integer 64 Variable	Integer 64 Variable	G4 Digital Local Simple I/O Unit
		G4D16R
		G4D32RS
		SNAP-BRS
		SNAP-B3000-ENET, SNAP-ENET-RTC
		SNAP-ENET-S64
		SNAP-PAC-R1
		SNAP-PAC-R2
		SNAP-UP1-ADS
		SNAP-UP1-M64

Standard Example:

IVAL Set I/O Unit from MOMO Masks

On Mask	0x060003C0000000C2	*Integer 64 Literal*
Off Mask	0xB0F240010308A020	*Integer 64 Literal*
On I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-M64*

The effect of this command is illustrated below::

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			

Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`IvalSetiOUnitfromMOMO(`*On Mask, Off Mask, On I/O Unit***`)`**

```
IvalSetiOUnitfromMOMO(0x060003C0000000C2i64, 0xB0F240010308A020i64,
PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Mistic PID Control Word

Simulation Action

Function: Writes to the internal value (IVAL) of the bits that represent the PID configuration.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: Bit assignments:

11 1 = Use SqRt value from input channel.
10 1 = Setpoint was above high clamp. Write zero to clear.
9 1 = Setpoint was below low clamp. Write zero to clear.
8 1 = Input channel under-range. Write zero to clear.
7 1 = Loop active. 0 = Loop stopped.
6 1 = Loop in auto mode. 0 = Loop in manual mode.
5 1 = Output active. 0 = Output disconnected.
4 1 = Output tracks input in manual mode. 0 = no action.
3 1 = Setpoint tracks input in manual mode. 0 = no action.
2 1 = Input from host. 0 = Input from channel.
1 1 = Setpoint from channel. 0 = Setpoint from host.
0 1 = Use filtered value from input channel. Must have filtering active on the input channel.
0 = Use current value of input channel.

To set any bit(s) put a 1 for each bit to set in the MOMO On parameter. To clear any bit(s) put a 1 for each bit to clear in the MOMO Off parameter. All MOMO bit positions with zeros will leave the corresponding PID control word bit unchanged.

Arguments:

Argument 1	Argument 2	Argument 3
On Mask	**Off Mask**	**For PID Loop**
Integer 32 Literal	Integer 32 Literal	PID Loop
Integer 32 Variable	Integer 32 Variable	

Standard Example:

IVAL Set Mistic PID Control Word

On Mask	PID_CTRL_SET	*Integer 32 Variable*
Off Mask	PID_CTRL_CLEAR	*Integer 32 Variable*
For PID Loop	EXTRUDER_ZONE08	*PID Loop*

OptoScript Example:

IvalSetMisticPidControlWord(*On Mask, Off Mask, For PID Loop***)**

```
IvalSetMisticPidControlWord(PID_CTRL_SET, PID_CTRL_CLEAR,
EXTRUDER_ZONE08);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

Pro IVAL Set Mistic PID Process Term

Simulation Action

Function: Writes to the internal value (IVAL) of a PID input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	**Argument 2**
To	**On PID Loop**
Float Literal	PID Loop
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set Mistic PID Process Term

To	1500	*Integer 32 Literal*
On PID Loop	Influent_Flow_Controller	*PID Loop*

OptoScript Example:

IvalSetMisticPidProcessTerm(*To, On PID Loop***)**

```
IvalSetMisticPidProcessTerm(1500, Influent_Flow_Controller);
```

This is a procedure command; it does not return a value.

Notes: Valid range is equal to the scaling of the PID input channel.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Mixed 64 I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use IVAL Set I/O Unit from MOMO Masks (page I-11) instead.

Function: Writes to the internal values (IVALs) of all digital points on a mixed 64 I/O unit (an I/O unit with a SNAP-UP1-M64 brain).

Typical Use: For simulation and testing, to assign specific values from a must-on, must-off mask to digital points.

Details:

- The program will use IVALs exclusively when communication to the I/O unit is disabled. This command allows the IVALs to be modified as if they were being changed by real I/O.
- This command is 64 times faster than using Turn On or Turn Off 64 times. It updates the IVALs for all 64 points. It affects only selected output points and does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of “1.”To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of “1.” To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2.
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Must-on Mask	**Must-off Mask**	Mixed 64 I/O Unit
Integer 32 Literal	Integer 32 Literal	SNAP-UP1-M64
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

IVAL Set Mixed 64 I/O Unit from MOMO Masks

Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Mixed 64 I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-M64*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example: **`IvalSetMixed64IoUnitFromMomo(`***Must-On Mask, Must-Off Mask, Mixed 64 I/O Unit***`)`**

```
IvalSetMixed64IoUnitFromMomo(0x060003C0000000C2i64,
                             0xB0F240010308A020i64, PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value. (Note that Integer 64 literals in OptoScript code take an `i64` suffix.)

Notes: Primarily used to write to inputs.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7), IVAL Set Mixed I/O Unit from MOMO Masks (page I-16), IVAL Set Digital-64 I/O Unit from MOMO Masks (page I-9)

IVAL Set Mixed I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use IVAL Set I/O Unit from MOMO Masks (page I-11) instead.

Function: Writes to the internal values (IVALs) of all digital points on a mixed I/O unit.

Typical Use: For simulation and testing, to assign specific values from a must-on, must-off mask to digital points.

Details:

- The program will use IVALs exclusively when communication to the I/O unit is disabled. This command allows the IVALs to be modified as if they were being changed by real I/O.
- This command is 64 times faster than using Turn On or Turn Off 64 times. It updates the IVALs for all 64 points. It affects only selected output points and does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2.
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	Argument 2	Argument 3
Must-on Mask	**Must-off Mask**	Mixed I/O Unit
Integer 32 Literal	Integer 32 Literal	SNAP-B3000-ENET,
Integer 32 Variable	Integer 32 Variable	SNAP-ENET-RTC
		SNAP-UP1-ADS

Standard Example:

IVAL Set Mixed I/O Unit from MOMO Masks

Must On Mask	0x0600C0C2	*Integer 32 Variable*
Must Off Mask	0xB001A020	*Integer 32 Literal*
Mixed I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-ADS*

The effect of this command is illustrated below:

	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight and the last eight digital points on the rack. For the points shown, points 26, 25, 7, 6, and 1 will be turned on. Points 31, 29, 28, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`IvalSetMixedIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Mixed I/O Unit***`)`**

```
IvalSetMixedIoUnitFromMomo(PUMPS_ON_MASK, 0xB001A020, PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value.

Notes: Primarily used to write to inputs.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7), IVAL Set Digital-64 I/O Unit from MOMO Masks (page I-9), IVAL Set Mixed 64 I/O Unit from MOMO Masks (page I-15)

IVAL Set Off-Latch

Simulation Action

Function: Writes to the internal value (IVAL) of a digital latch input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details:

- The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.
- Any non-zero value sets the latch; zero clears the latch.

Arguments:

Argument 1	Argument 2
To	**On Point**
Integer 32 Literal	Digital Input
Integer 32 Variable	

Standard Example:

IVAL Set Off-Latch

To	-1	*Integer 32 Literal*
On Point	Process_Stop_Button	*Digital Input*

OptoScript Example:

`IvalSetOffLatch(`*To, On Point***`)`**

```
IvalSetOffLatch(-1, Process_Stop_Button);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Off-Pulse

Simulation Action

Function: Writes to the internal value (IVAL) of a digital pulse input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details:

- The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.
- This command applies to SNAP-UP1-ADS and SNAP-B3000-ENET I/O units as well as to *mistic* I/O units.

Arguments:

Argument 1	**Argument 2**
To	**On Point**
Float Literal	Off Pulse
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set Off-Pulse

To	150000	*Integer 32 Literal*
On Point	TIME_PULSE_INPUT	*Off Pulse*

OptoScript Example:

`IvalSetOffPulse(` *To, On Point* **`)`**

```
IvalSetOffPulse(150000, TIME_PULSE_INPUT);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0–2 billion in units of 100 microseconds.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Off-Totalizer

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: Writes to the internal value (IVAL) of a digital totalizer input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	**Argument 2**
To	**On Point**
Float Literal	Off Totalizer
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set Off-Totalizer

To	36000000	*Integer 32 Literal*
On Point	PUMP_OFF_TIME	*Totalizer Off*

OptoScript Example:

IvalSetOffTotalizer(*To, On Point***)**

```
IvalSetOffTotalizer(36000000, PUMP_OFF_TIME);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0–2 billion in units of 100 microseconds.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set On-Latch

Simulation Action

Function: Writes to the internal value (IVAL) of a digital latch input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details:
- The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.
- Any non-zero value sets the latch; zero clears the latch.

Arguments:

Argument 1	**Argument 2**
To	**On Point**
Integer 32 Literal	Digital Input
Integer 32 Variable	

Standard Example:

IVAL Set On-Latch

To	0	*Integer 32 Literal*
On Point	Process_Start_Button	*Digital Input*

OptoScript Example:

IvalSetOnLatch(*To, On Point***)**

```
IvalSetOnLatch(0, Process_Start_Button);
```

This is a procedure command; it does not return a value.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set On-Pulse

Simulation Action

Function: Writes to the internal value (IVAL) of a digital pulse input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	Argument 2
To	**On Point**
Float Literal	On Pulse
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set On-Pulse

To	133300	*Integer 32 Literal*
On Point	TIME_PULSE_INPUT	*On Pulse*

OptoScript Example:

IvalSetOnPulse(*To, On Point***)**

```
IvalSetOnPulse(133300, TIME_PULSE_INPUT);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0–2 billion in units of 100 microseconds.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set On-Totalizer

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: Writes to the internal value (IVAL) of a digital totalizer input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1 **To**	Argument 2 **On Point**
Float Literal	On Totalizer
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set On-Totalizer

To	72000000	*Integer 32 Literal*
On Point	PUMP_ON_TIME	*On Totalizer*

OptoScript Example:

IvalSetOnTotalizer(*To, On Point***)**

```
IvalSetOnTotalizer(72000000, PUMP_ON_TIME);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0–2 billion in units of 100 microseconds.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Period

Simulation Action

NOTE: This command is for mistic I/O units only.

Function: Writes to the internal value (IVAL) of a digital input configured to measure a time period.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	Argument 2
To	**On Point**
Float Literal	Period
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set Period

To	5.63	*Float Literal*
On Point	Pump_On_Time	*Period*

OptoScript Example:

IvalSetPeriod(*To, On Point***)**

```
IvalSetPeriod(5.63, Pump_On_Time);
```

This is a procedure command; it does not return a value.

Notes: Value to write is in seconds.

See Also: Get Period (page G-110), Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set Simple 64 I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use IVAL Set I/O Unit from MOMO Masks (page I-11) instead.

Function: Writes to the internal values (IVALs) of all digital points on a SNAP Simple 64-point I/O unit (an I/O unit with a SNAP-ENET-S64 brain).

Typical Use: For simulation and testing, to assign specific values from a must-on, must-off mask to digital points.

Details:

- The program will use IVALs exclusively when communication to the I/O unit is disabled. This command allows the IVALs to be modified as if they were being changed by real I/O.
- This command is 64 times faster than using Turn On or Turn Off 64 times. It updates the IVALs for all 64 points. It affects only selected output points and does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2.
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	Argument 2	Argument 3
Must-on Mask	**Must-off Mask**	Simple 64 I/O Unit
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-S64
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

IVAL Set Simple 64 I/O Unit from MOMO Masks

Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Simple 64 I/O Unit	PUMP_CTRL_UNIT	*SNAP-ENET-S64*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example: **`IvalSetSimple64IoUnitFromMomo(`***Must-On Mask, Must-Off Mask, Simple 64 I/O Unit***`)`**

```
IvalSetSimple64IoUnitFromMomo(0x060003C0000000C2i64,
                              0xB0F240010308A020i64, PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value. (Note that Integer 64 literals in OptoScript code take an `i64` suffix.)

Notes: Primarily used to write to inputs.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7), IVAL Set Digital-64 I/O Unit from MOMO Masks (page I-9), IVAL Set Mixed 64 I/O Unit from MOMO Masks (page I-15)

IVAL Set TPO Percent

Simulation Action

Function: Writes to the internal value (IVAL) of a digital TPO output.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1 **To**	**Argument 2** **On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set TPO Percent

To	43.66	*Float Literal*
On Point	ZONE_3_HEATER	*TPO*

OptoScript Example: **`IvalSetTpoPercent(`***To, On Point***`)`**

```
IvalSetTpoPercent(43.66, ZONE_3_HEATER);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0.0 to 100.0.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Set TPO Period

Simulation Action

Function: Writes to the internal value (IVAL) of a digital TPO period.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1	Argument 2
Value	**To**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

IVAL Set TPO Period

Value	1.00	*Float Literal*
To	ZONE_3_HEATER	*TPO*

OptoScript Example:

IvalSetTpoPeriod(*Value, On Point***)**

```
IvalSetTpoPeriod(1.00, ZONE_3_HEATER);
```

This is a procedure command; it does not return a value.

Notes: Valid range is 0.1 to 429,496.7 seconds with resolution to 100 microseconds.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Turn Off

Simulation Action

Function: Writes to the internal value (IVAL) of a digital input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1
[Value]
Digital Input
Digital Output

Standard Example:

IVAL Turn Off

Process_Start_Button	*Digital Input*

OptoScript Example:

`IvalTurnOff(`*Point***`)`**

```
IvalTurnOff(Process_Start_Button);
```

This is a procedure command; it does not return a value.

Notes: Turns Off the IVAL for the specified point.

See Also: Disable Communication to All I/O Units (page D-5), Disable Communication to I/O Unit (page D-7)

IVAL Turn On

Simulation Action

Function: Writes to the internal value (IVAL) of a digital input.

Typical Use: Simulation, testing, and certification where either there are no I/O units or communication to the I/O units is disabled.

Details: The program will use IVALs exclusively when communication to the specified point or I/O unit is disabled. This command allows the IVAL to be modified as if it were being changed by real I/O.

Arguments:

Argument 1
[Value]
Digital Input
Digital Output

Standard Example:

IVAL Turn On

PROCESS_START_BUTTON	*Digital Input*

OptoScript Example:

`IvalTurnOn(`*Point***`)`**

```
IvalTurnOn(Process_Start_Button);
```

This is a procedure command; it does not return a value.

Notes: Turns On the IVAL for the specified point.

Less?

Logical Condition

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 Is	Argument 2 Than
Analog Input	Analog Input
Analog Output	Analog Output
Digital Input	Digital Input
Digital Output	Digital Output
Down Timer Variable	Down Timer Variable
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
Up Timer Variable	Up Timer Variable

Standard Example:

Less?

Is	TANK_LEVEL	*Analog Input*
Than	FILL_SETPOINT	*Float Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<` operator.

```
if (TANK_LEVEL < FILL_SETPOINT) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `<` operator. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use Within Limits? to test for an approximate match.
- To test for greater than or equal to, use either Greater Than or Equal? or the False exit.

See Also: Greater? (page G-146) Not Equal? (page N-4) Equal? (page E-16) Greater Than or Equal? (page G-148)

Less Than Numeric Table Element?

Logical Condition

Function: To determine if a numeric value is less than a specified value in a float or 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 float or integer table *Argument 3*). Examples:

Value 1	Value 2	Result
0.0	0.0	False
0.0001	0.0	False
-98.766	-98.765	True
-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**
Is	**At Index**	**Of Table**
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Down Timer Variable		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard Example:

Is	THIS_READING	*Float Variable*
Less Than Numeric Table Element?		
At Index	TABLE_INDEX	*Integer 32 Variable*
Of Table	TABLE_OF_READINGS	*Float Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<` operator.

```
if (THIS_READING < TABLE_OF_READINGS[TABLE_INDEX]) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `<` operator. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To test for greater than or equal to, use either Greater Than or Equal to Table Element? or the False exit.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to table size.

See Also: Greater Than or Equal To Numeric Table Element? (page G-149)

Less Than or Equal?

Logical Condition

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**
Is	**To**
Analog Input	Analog Input
Analog Output	Analog Output
Digital Input	Digital Input
Digital Output	Digital Output
Down Timer Variable	Down Timer Variable
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
Up Timer Variable	Up Timer Variable

Standard Example:

Less Than or Equal?		
Is	TEMPERATURE	*Float Variable*
To	98.60	*Float Literal*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<=` operator.

```
if (TEMPERATURE <= 98.60) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `<=` operator. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use Within Limits? to test for an approximate match.
- To test for greater than, use either the Greater? condition or the False exit.

See Also: Greater? (page G-146), Not Equal? (page N-4), Greater Than or Equal? (page G-148)

Less Than or Equal to Numeric Table Element?

Logical Condition

Function: To determine if a numeric value is less than or equal to a specified value in a float or 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 float or integer table *Argument 3*). Examples:

Value 1	Value 2	Result
0.0	0.0	True
0.0001	0.0	False
22.22	22.222	True
-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 Is	Argument 2 At Index	Argument 3 Of Table
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Down Timer Variable		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard Example:

Is	THIS_READING	*Float Variable*
Less Than or Equal to Numeric Table Element?		
At Index	TABLE_INDEX	*Integer 32 Variable*
Of Table	TABLE_OF_READINGS	*Float Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<=` operator.

```
if (THIS_READING <= TABLE_OF_READINGS[TABLE_INDEX]) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `<=` operator. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To test for greater than, use either Greater Than Table Element? or the False exit.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Greater Than Numeric Table Element? (page G-147) Not Equal to Numeric Table Element? (page N-5) Equal to Numeric Table Element? (page E-17) Greater Than or Equal To Numeric Table Element? (page G-149)

Listen for Incoming Communication

Communication Action

Function: In TCP/IP communication, to start listening for incoming open communication requests. (In this case the control engine acts as the slave, and the session is opened by the master.)

Typical Use: To listen for an incoming request to open communication.

Details:

- Applies to communication via TCP communication handles only.
- When configuring the communication handle, be careful to choose a port that is not used by other, unrelated devices on the network.

Arguments:

Argument 1	Argument 2
Communication Handle	**Put Status In**
Communication Handle	Integer 32 Variable

Standard Example:

Listen for Incoming Communication

Communication Handle	Ultimate_A	*Communication Handle*
Put Status In	STATUS	*Integer 32 Variable*

OptoScript Example:

`ListenForIncomingCommunication(`*Communication Handle***`)`**

```
STATUS = ListenForIncomingCommunication(Ultimate_A);
```

This is a procedure command; it returns one of the status codes listed below. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide.*
- After using this command, use Accept Incoming Communication to complete the connection.
- It is only necessary to use this command once per port, even if you use the Accept command several times.
- To determine whether the connection is still open, use Get Number of Characters Waiting or Communication Open?
- Using TCP, this command will return a true (non-zero) if there are still characters to be received, even if the other side has closed. This situation is called a "half open" connection. Make sure the characters are received so that sessions aren't used up by a half-open state.
- If you use this command repeatedly with a different port number, eventually the command will return an error. The maximum successful calls to the command and error number returned vary based on the firmware and user application as far as the number of Ethernet communication handles already in use.
- In currently available firmware for SNAP-UP1-ADS, SNAP-UP1-M64, SNAP-UP1-S64, and SNAP-LCE, the maximum is 64 with error -49.
- The SNAP-PAC-S1 can open up to about 100 listening sessions and the SNAP-PAC-R1 about 75. Both will then return -438. The number of sessions is subject to available memory.
- Keep in mind system resources are shared by both listening sessions and active open sessions.

Status Codes: 0 = Success

-10 = Invalid port.

-36 = Invalid command. Use this command only with a TCP communication handle; for other communication handles, use Open Outgoing Communication instead.

-47 = Open failed. Handle has already been opened.

-49 = No more connections are available. Maximum number of connections already in use.

-203 = Driver not found.

-438 = Could not create socket

See Also: Accept Incoming Communication (page A-2), Get Number of Characters Waiting (page G-101), Communication Open? (page C-32) Open Outgoing Communication (page O-4)

Load Files From Permanent Storage

Control Engine Action

Function: To read the files in flash memory and store them to its file system in RAM, thereby replacing files previously in the root directory of the file system.

Typical Use: To retrieve files previously saved into flash memory.

Details:

- Copies all files currently in flash memory to its file system in RAM. Replaces all files in the root directory of the file system. Folders in the root directory and files within folders are not replaced.
- This command does not affect point and function configurations, the ioControl strategy, or the brain's or controller's memory map.
- To determine what files are in flash memory and in RAM, use ioManager. Follow the instructions in Opto 22 form #1440, the *ioManager's User's Guide*.

Arguments:

Argument 1
Put Status In
Integer 32 Variable

Standard Example:

Load Files From Permanent Storage

Put Status In	STATUS	*Integer 32 Variable*

OptoScript Example:

LoadFilesFromPermanentStorage()

```
STATUS = LoadFilesFromPermanentStorage();
```

This is a function command; it returns a zero (indicating success) or an error (indicating failure). The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Control Engine Commands" in Chapter 10 of the *ioControl User's Guide.*
- The equivalent of this command happens automatically when the controller is turned on. However, when the controller is turned *off* or loses power, all files and folders in its file system are deleted; only the files saved to flash memory can be loaded back into RAM when the controller is turned on again.

Status Codes:

0 = Success

-408 = Error during file access. No files are currently saved in flash memory.

See Also: Erase Files in Permanent Storage (page E-18), Save Files To Permanent Storage (page S-1)

Make Integer 64

Logical Action

Function: To combine two 32-bit integers into a single 64-bit integer.

Typical Use: To put the two halves of a 64-bit integer back together after separating them for faster individual manipulation.

Details:

- Places one 32-bit integer in the upper half of a 64-bit integer and the other 32-bit integer in the lower half.
- When the integer 64 is made, the least significant bit corresponds to point zero and the most significant bit corresponds to point 64 on a 64-point digital rack, when *Argument 3* is an I/O unit.

Arguments:

Argument 1 **High Integer**	Argument 2 **Low Integer**	Argument 3 **Put in**
Integer 32 Literal	Integer 32 Literal	Integer 64 Variable
Integer 32 Variable	Integer 32 Variable	SNAP-ENET-D64*
		SNAP-UP1-D64*
		SNAP-UP1-M64*
		SNAP-ENET-S64*

* Standard commands only

Standard Example:

Make Integer 64

High Integer	IN_BD2_HIGH	*Integer 32 Variable*
Low Integer	IN_BD2_LOW	*Integer 32 Variable*
Put in	IN_BD2_STATUS	*Integer 64 Variable*

OptoScript Example:

MakeInt64(*High Integer, Low Integer***)**

```
IN_BD2_STATUS = MakeInt64(IN_BD2_HIGH, IN_BD2_LOW);
```

This is a function command; it returns the 64-bit integer. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. It cannot be consumed by an I/O unit, however. See Chapter 11 of the *ioControl User's Guide* for more information on OptoScript.

Although the returned value cannot be consumed by an I/O unit, you can accomplish the same thing by using OptoScript code such as the following:

```
nnTemp1 = MakeInt64(nHiPart, nLoPart);
SetDigital64IoUnitFromMomo(nnTemp1, bitnot nnTemp1, MyDig64);
```

Notes: This command is useful if you want to get information from a program that doesn't directly support 64-bit integers, such as ioDisplay and third-party products, and use that information in a digital-only SNAP D64 Ultimate or Ethernet I/O unit.

See Also: Get High Bits of Integer 64 (page G-62), Get Low Bits of Integer 64 (page G-85)

Maximum

Mathematical Action

Function: To select the greater of two values.

Typical Use: To select the higher pressure or temperature reading.

Details: The greater of the two values is selected.

Arguments:

Argument 1 **Compare**	Argument 2 **Withp**	Argument 3 **Put Maximum in**
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Maximum

Compare	Pressure_A	*Analog Input*
With	Pressure_B	*Analog Input*
Put Maximum in	Highest_Pressure	*Float Variable*

OptoScript Example:

Max(*Compare, With***)**

```
Highest_Pressure = Max(Pressure_A, Pressure_B);
```

This is a function command; it returns the greater of the two values. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Minimum (page M-3)

Minimum

Mathematical Action

Function: To select the lesser of two values.

Typical Use: To select the lower pressure or temperature reading.

Details: The lesser of the two values is selected.

Arguments:

Argument 1 Compare	**Argument 2** With	**Argument 3** Put Minimum in
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Minimum

Compare	Pressure_A	*Analog Input*
With	Pressure_B	*Analog Input*
Put Minimum in	Lowest_Pressure	*Float Variable*

OptoScript Example:

`Min(`*Compare, With***`)`**

```
Lowest_Pressure = Min(Pressure_A, Pressure_B);
```

This is a function command; it returns the lesser value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Maximum (page M-2)

Mistic PID Loop Communication Enabled?

Simulation Condition

Function: Checks a flag internal to the controller to determine if communication to the specified PID loop is enabled.

Typical Use: Primarily used in factory QA testing and simulation.

Details: Evaluates True if communication is enabled.

Arguments:

Argument 1
PID Loop
PID Loop

Standard Example:

PID Loop	FACTORY_HEAT_2BA

Mistic PID Loop Communication Enabled?

OptoScript Example:

IsMisticPidLoopCommEnabled(*PID Loop***)**

```
if (IsMisticPidLoopCommEnabled(FACTORY_HEAT_2BA)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: I/O Point Communication Enabled? (page I-2)

Modulo

Mathematical Action

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**
[Value]	**By**	**Put Result in**
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Modulo

	Num_Parts_Produced	*Integer 32 Variable*
By	Minutes_Elapsed	*Integer 32 Variable*
Put Result in	Productivity_Remainder	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `%` operator.

```
Productivity_Remainder = Num_Parts_Produced % Minutes_Elapsed;
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `%` operator can be used in several ways. For more information on mathematical expressions in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.

See Also: Divide (page D-20), Multiply (page M-25)

Move

Miscellaneous Action

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: ioControl 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 non-zero is returned for on, 0 for off.
- *From Latch:* A value of non-zero 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 point, the output will go to the nearest range limit, either zero or full scale.
- *From Integer 32 to Integer 64:* Integer values are moved into the high or upper half. For conversions from integer 32 to integer 64 (or vice versa), use the commands Make Integer 64, Get High Bits of Integer 64, and Get Low Bits of Integer 64.

Arguments:

Argument 1	**Argument 2**
From	**To**
Analog Input	Analog Output
Analog Output	Digital Output
Digital Input	Down Timer Variable
Digital Output	Float Variable
Down Timer Variable	Integer 32 Variable
Float Literal	Integer 64 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Integer 64 Literal	
Integer 64 Variable	
Up Timer Variable	

Standard Example:

Move

From	DIG1	*Digital Input*
To	DIG1_STATUS	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the = operator.

```
DIG1_STATUS = DIG1;
```

Notes:
- In OptoScript code, simply make assignments where you would use the Move command.
- In standard commands, you can use Move with timers as the equivalent of two other commands (in OptoScript code, the = operator has the same effect):
  - With up timers, Move is the same as using Set Up Timer Target Value and Start Timer. The value moved is the target value, and it overwrites any target value already in place. The up timer starts immediately from zero.
  - With down timers, Move is the same as using Set Down Timer Preset Value and Start Timer. The value moved is the preset value the timer will start from, and it overwrites any preset value previously set. The timer starts immediately from the preset value.

Queue Errors: -13 = Overflow error—integer or float value was too large.

See Also: Move String (page M-16), Move to Numeric Table Element (page M-17) and other Move to Table commands, Move from Numeric Table Element (page M-8) and other Move from Table commands.

Move 32 Bits

Logical Action

Function: To move the internal bit pattern of an integer 32 into a float, or to move a float into an integer 32.

Typical Use: To help parse or create binary data when communicating with other devices.

Arguments:

Argument 1 **From**	Argument 2 **To**
Float Literal	Float Variable
Float Variable	Integer 32 Variable
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Move 32 Bits

From	Source_Data	*Integer 32 Variable*
To	Float	*Float Variable*

OptoScript Example:

`Move32Bits(`*From, To***`)`**

```
Move32Bits(Source_Data, Float);
```

This is a procedure command; it does not return a value.

Notes: See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.

Move from Numeric Table Element

Miscellaneous Action

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 minus 1 (size – 1).

Arguments:

Argument 1 **From Index**	Argument 2 **Of Table**	Argument 3 **To**
Integer 32 Literal	Float Table	Analog Output
Integer 32 Variable	Integer 32 Table	Digital Output
	Integer 64 Table	Float Variable
		Integer 32 Variable
		Integer 64 Variable

Standard Example:

Move from Numeric Table Element

From Index	0	*Integer 32 Literal*
Of Table	LOOK_UP_TABLE	*Float Table*
To	PRESS_OUT	*Analog Output*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `=` operator.

```
PRESS_OUT = LOOK_UP_TABLE[0];
```

Notes: In OptoScript code, simply make an assignment from the table element.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

-13 = Overflow—integer or float value was too large.

See Also: Move Numeric Table Element to Numeric Table (page M-13), Move to Numeric Table Element (page M-17), Shift Numeric Table Elements (page S-90)

Move from Pointer Table Element

Pointers Action

Function: To move an object from a pointer table to a pointer variable.

Typical Use: To retrieve objects from pointer tables.

Details: This command allows you to retrieve objects from a pointer table and place them into pointer variables of the same type.

Arguments:

Argument 1 **Index**	Argument 2 **Of Table**	Argument 3 **To Pointer**
Integer 32 Literal Integer 32 Variable	Pointer Table	Pointer Variable

Standard Example:

Move From Pointer Table Element

Index	CURRENT_INDEX	*Integer 32 Variable*
Of Table	IO_POINTERS	*Pointer Table*
To Pointer	TANK_SWITCH_POINTER	*Pointer Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the = operator.

```
TANK_SWITCH_POINTER = IO_POINTERS[CURRENT_INDEX];
```

Notes:

- In OptoScript code, simply make an assignment from the table element.
- Be sure to move the object from the table into a pointer of the same type. If the types are different, an error will be posted to the message queue.

Queue Errors: -30 = Pointer was not initialized. Use Move to Pointer Table Element to initialize the table entry.

-69 = Invalid parameter (null pointer) passed to driver.

See Also: Move to Pointer (page M-19), Move to Pointer Table Element (page M-21),

Move from String Table Element

String Action

Function: To copy a string from a string table.

Typical Uses:

- To create a numeric-to-string lookup table, or to retrieve strings from a table for further processing.

Details:

- Quotes (“”) are used in OptoScript code, but not in standard ioControl code.
- Valid range for *Index* (*Argument 1*) is zero to the table length minus 1 (size – 1).
- If the string moved from the table is longer than the string variable width (*Argument 3*), it is truncated to fit.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
From Index	**Of Table**	**To**
Integer 32 Literal	String Table	String Variable
Integer 32 Variable		

Standard 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 *From Index.*

Move from String Table Element

From Index	INDEX	*Integer 32 Variable*
Of Table	STRING_TABLE	*String Table*
To	STRING	*String Variable*

The results of this command are as follows:

Index	**String**
0	“SUN”
1	“MON”
2	“TUE”
3	“WED”
4	“THU”
5	“FRI”
6	“SAT”

OptoScript Example: OptoScript doesn’t use a command; the function is built in. Use the = operator. Remember that quotes are required in OptoScript code.

```
STRING = STRING_TABLE[INDEX];
```

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, simply make an assignment to the string.
- A string table is a good way to correlate a number to a string.
- Use Move to String Table 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.

Queue Errors: -12 = Invalid table index—index was negative or greater than or equal to the table size.

See Also: Move to String Table Element (page M-23), String Equal to String Table Element? (page S-104), Get Substring (page G-139), Get Length of Table (page G-83)

Move I/O Unit to Numeric Table

I/O Unit Action

Function: To read current on/off status of all digital points and current values of all analog points on an I/O unit and move the returned values to a numeric table.

Typical Use: To efficiently read all points of data on a single I/O unit with one command.

Details:

- This command is much faster than using Move several times.
- Reads both inputs and outputs. Updates the IVALs and XVALs for all points.
- Point zero corresponds to the first specified table element. The command returns status to the table beginning at the index specified in Argument 2. If there are more points than table elements from the specified index to the end of the table, no data will be written to the table and a -12 will be placed in the message queue. For an Ultimate or Ethernet I/O unit, 64 table elements are required.
- For digital points, if the point is on, there will be a non-zero in the respective table element. If the point is off, there will be a zero in the respective table element.
- For analog points, the current value of the point in engineering units will appear in the respective table element.
- Points that are not configured will return a value of 0.0.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be read.

Arguments:

Argument 1 **From**	Argument 2 **Starting Index**	Argument 3 **Of Table**
B100*	Integer 32 Literal	Float Table
B200*	Integer 32 Variable	Integer 32 Table
B3000 (Analog)*		
B3000 (Digital)*		
G4A8R, G4RAX*		
G4D16R*		
G4D32RS*		
SNAP-ENET-D64		
SNAP-UP1-D64		
SNAP-UP1-M64		
SNAP-ENET-S64		
SNAP-B3000-ENET, SNAP-ENET-RTC		
SNAP-UP1-ADS		
SNAP-PAC-R1		
SNAP-PAC-R2		
SNAP-BRS*		

* ioControl Professional only

Standard Example:

Move I/O Unit to Numeric Table

From	UNIT_255	*SNAP-UP1-ADS*
Starting Index	0	*Integer 32 Literal*
Of Table	DATA_TABLE	*Float Table*

OptoScript Example: **MoveIoUnitToNumTable(***I/O Unit, Starting Index, Of Table***)**

```
MoveIoUnitToNumTable(UNIT_255, 0, DATA_TABLE);
```

This is a procedure command; it does not return a value.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Move Numeric Table to I/O Unit (page M-14)

Move Numeric Table Element to Numeric Table

Miscellaneous Action

Function: To copy a single value from one table to another or from one table element to another table element within the same table.

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 table, different types, or the same type.
- Any value sent to an invalid index is discarded, and an error -12 is added to the message queue.
- The valid range for each index is zero to the table length minus 1 (size – 1).

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
From Index	**Of Table**	**To Index**	**Of Table**
Integer 32 Literal	Float Table	Integer 32 Literal	Float Table
Integer 32 Variable	Integer 32 Table	Integer 32 Variable	Integer 32 Table
	Integer 64 Table		Integer 64 Table

Standard Example:

Move Numeric Table Element to Numeric Table

From Index	17	*Integer 32 Literal*
Of Table	I/O_STATUS_TABLE	*Integer 32 Table*
To Index	27	*Integer 32 Literal*
Of Table	I/O_STATUS_TABLE	*Integer 32 Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `=` operator.

```
I/O_STATUS_TABLE[27] = I/O_STATUS_TABLE[17];
```

Notes:

- In OptoScript code, simply make an assignment to the table element.
- To move several values, put this command in a loop using variables for both indexes.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

-13 = Overflow—integer or float value was too large.

See Also: Move to Numeric Table Element (page M-17)

Move Numeric Table to I/O Unit

I/O Unit Action

Function: To control multiple analog and digital output points on the same I/O unit simultaneously with a single command.

Typical Use: To efficiently control a selected group of analog and digital outputs with one command.

Details:

- This command is much faster than using Turn On, Turn Off, or Move for each point.
- Updates the IVALs and XVALs for all 64 points. Affects all output points. Does not affect input points.
- The first specified table element corresponds to point zero.
- A digital point is turned off by setting the respective table element to 0. A digital point is turned on by setting the respective table element to non-zero.
- An analog point is set by the value in the respective table element.
- If a specific point is disabled, only its internal value (IVAL) will be written to. If the entire I/O unit is disabled, only the internal values (IVALS) on all 64 points will be written to.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Start at Index	**Of Table**	**Move to**
Integer 32 Literal	Float Table	B100*
Integer 32 Variable	Integer 32 Table	B200*
		B3000 (Analog)*
		B3000 (Digital)*
		G4A8R, G4RAX*
		G4D16R*
		G4D32RS*
		SNAP-ENET-D64
		SNAP-UP1-D64
		SNAP-UP1-M64
		SNAP-ENET-S64
		SNAP-B3000-ENET, SNAP-ENET-RTC
		SNAP-UP1-ADS
		SNAP-PAC-R1
		SNAP-PAC-R2
		SNAP-BRS*

* ioControl Professional only

Standard Example:

Move Numeric Table to I/O Unit

Start at Index	4	*Integer 32 Variable*
Of Table	IO_STATUS_TABLE	*Integer 32 Table*
Move to	VALVE_CONTROL	*SNAP-UP1-ADS*

OptoScript Example:

`MoveNumTableToIoUnit(`*Start at Index, Of Table, Move to***`)`**

```
(4, IO_STATUS_TABLE, VALVE_CONTROL);
```

This is a procedure command; it does not return a value.

Notes: In the above example, index 4 of the table will map to point 0 of the I/O unit, index 5 will map to point 1 of the I/O unit, and so on.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Move I/O Unit to Numeric Table (page M-12)

Move Numeric Table to Numeric Table

Miscellaneous Action

Function: To copy values from one table to another.

Typical Use: To copy temporary values to a final location.

Details:

- The two tables must be of the same type and must be different tables. They can be different sizes, but make sure the Length parameter is not too long for either table.
- The valid range for each table index is zero to the table length - 1 (size - 1).

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**	**Argument 5**
From Table	**From Index**	**To Table**	**To Index**	**Length**
Float Table	Integer 32 Literal	Float Table	Integer 32 Literal	Integer 32 Literal
Integer 32 Table	Integer 32 Variable	Integer 32 Table	Integer 32 Variable	Integer 32 Variable
Integer 64 Table		Integer 64 Table		

Standard Example:

Move Numeric Table to Numeric Table

From Table	Temp_Table	*Integer 32 Table*
From Index	0	*Integer 32 Literal*
To Table	Status_Table	*Integer 32 Table*
To Index	16	*Integer 32 Literal*
Length	8	*Integer 32 Literal*

OptoScript Example:

`MoveNumTableToNumTable(`*From Table, From Index, To Table, To Index, Length***`)`**

```
MoveNumTableToNumTable(Temp_Table, 0, Status_Table, 16, 8);
```

This is a procedure command; it does not return a value.

Queue Errors: -6 = Data field error. Source and destination tables must be different.

-12 = Invalid table index or length

-13 = Overflow—integer or float value was too large.

-29 = Wrong object type. Arguments 1 and 3 must both be tables and of the same type.

See Also: Move to Numeric Table Element (page M-17)

Move String

String Action

Function: To copy the contents of one string to another.

Typical Use: To save, initialize, or clear strings.

Details:
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- If the width of the destination string variable 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**
Move String	**To**
String Literal	String Variable
String Variable	

Standard Example: The following example initializes a string variable to "Hello"; quotes are shown for clarity only; do not use them in standard commands.

Move String	"Hello"	*String Literal*
To	HELLO_STRING	*String Variable*

The following example clears a string variable; again, quotes are shown for clarity, but do not use them.

Move String

From	""	*String Literal*
Move to	MY_STRING	*String Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the = operator. Remember that quotes are required in OptoScript code.

```
HELLO_STRING = "Hello";
MY_STRING = "";
```

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, simply make an assignment to the string.

Dependencies: The destination string variable should be wide enough to hold the source string. If it is not, the source string will be truncated.

See Also: Append String to String (page A-10), Copy Time to String (page C-61)

Move to Numeric Table Element

Miscellaneous Action

Function: To copy a value from virtually any source to a table element.

Typical Use: To create a list of various values in a table.

Details:

- All numeric type conversions are automatically handled according to the rules detailed for the Move command.
- Any value sent to an invalid index is discarded, and an error -12 is added to the message queue.
- The valid range for each index is zero to the table length minus 1 (size – 1).

Arguments:

Argument 1 **From**	Argument 2 **To Index**	Argument 3 **Of Table**
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		

Standard Example:

Move to Numeric Table Element

From	0	*Integer 32 Literal*
To Index	27	*Integer 32 Literal*
Of Table	IO_STATUS_TABLE	*Integer 32 Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the = operator.

```
IO_STATUS_TABLE[27] = 0;
```

Notes:

- In OptoScript code, simply make an assignment to the table element.
- To move the same value to several table elements, put this command in a loop using a variable for the index.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

-13 = Overflow—integer or float value was too large.

See Also: Move from Numeric Table Element (page M-8), Move to Numeric Table Elements (page M-18)

Move to Numeric Table Elements

Miscellaneous Action

Function: To set a given value to a range of table elements within the same table.

Typical Use: To initialize elements within a table to the same value.

Details:

- All numeric type conversions are automatically handled according to the rules detailed for the Move command.
- Any value sent to an invalid index is discarded, and an error -12 is added to the message queue.
- The valid range for each index is zero to the table length minus 1 (size – 1). However, if you need to set a value to the entire table and don't know the table's size, you can use a starting index of 0 and an ending index of -1.

Arguments:

Argument 1 **From**	Argument 2 **Start Index**	Argument 3 **End Index**	Argument 4 **Of Table**
Float Literal	Integer 32 Literal	Integer 32 Literal	Float Table
Float Variable	Integer 32 Variable	Integer 32 Variable	Integer 32 Table
Integer 32 Literal			Integer 64 Table
Integer 32 Variable			
Integer 64 Literal			
Integer 64 Variable			

Standard Example:

Move to Numeric Table Elements

From	0	*Integer 32 Literal*
Start Index	4	*Integer 32 Literal*
End Index	10	*Integer 32 Literal*
Of Table	IO_STATUS_TABLE	*Integer 32 Table*

OptoScript Example:

`MoveToNumTableElements(`*From, Start Index, End Index, Of Table***`)`**

```
MoveToNumTableElements(0, 4, 10, IO_STATUS_TABLE);
```

This is a procedure command; it does not return a value.

Notes: Compared to other methods such as loops, this command initializes table elements very quickly.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

-13 = Overflow—integer or float value was too large.

See Also: Move from Numeric Table Element (page M-8), Move to Numeric Table Element (page M-17)

Move to Pointer

Pointers Action

Function: To assign an object to a pointer.

Typical Use: To initialize a pointer.

Details: The pointer will point to the object specified. Any operation that can be performed on the object can likewise be performed on the pointer. When you perform an operation on a pointer, you are actually performing the operation on the object.

Arguments:

Argument 1	**Argument 2**
Object	**Pointer**
Analog Event/Reaction*	Pointer Variable
Analog Input	
Analog Output	
B100*	
B200*	
B3000 (Analog)*	
B3000 (Digital)*	
Chart	
Communication Handle?	
Digital Event/Reaction*	
Digital Input	
Digital Output	
Down Timer Variable	
Event/Reaction Group*	
Float Table	
Float Variable	
G4A8R, G4RAX*	
G4D16R*	
G4D32RS*	
Integer 32 Table	
Integer 32 Variable	
Integer 64 Table	
Integer 64 Variable	
PID Loop	
Pointer Variable	
SNAP-ENET-D64	
SNAP-UP1-D64	
SNAP-UP1-M64	
SNAP-ENET-S64	
SNAP-B3000-ENET, SNAP-ENET-RTC	
SNAP-UP1-ADS	
SNAP-PAC-R1	
SNAP-PAC-R2	
SNAP-BRS*	
String Table	
String Variable	
Up Timer Variable	

* ioControl Professional only

Standard Example:

Move To Pointer

Object	PUMP_VALVE	*Digital Output*
Pointer	IO_POINTER	*Pointer Variable*

OptoScript Example:

OptoScript doesn't use a command; the function is built in. Use the `&` operator to get the address of the object and use the `=` operator to make the assignment:

```
IO_POINTER = &PUMP_VALVE;
```

Notes:

- In OptoScript code, simply make an assignment to the pointer.
- For standard commands, the Move To Pointer command will be validated when the OK button in the Add Instruction dialog box is pressed. For OptoScript code, the type will be validated by the compiler.

See Also:

Clear Pointer (page C-28), Pointer Equal to NULL? (page P-3)

Move to Pointer Table Element

Pointers Action

Function: To assign an object to a pointer table element.

Typical Use: To initialize a pointer table with objects of various types.

Details:

- This command takes the pointer for the object being pointed to and moves it to the table element.

Arguments:

Argument 1 **Object**	Argument 2 **Index**	Argument 3 **Of Table**
Analog Event/Reaction*	Integer 32 Literal	Pointer Table
Analog Input	Integer 32 Variable	
Analog Output		
B100*		
B200*		
B3000 (Analog)*		
B3000 (Digital)*		
Chart		
Communication Handle		
Digital Event/Reaction*		
Digital Input		
Digital Output		
Down Timer Variable		
Event/Reaction Group*		
Float Table		
Float Variable		
G4A8R, G4RAX*		
G4D16R*		
G4D32RS*		
Integer 32 Table		
Integer 32 Variable		
Integer 64 Table		
Integer 64 Variable		
SNAP-ENET-D64		
SNAP-UP1-D64		
SNAP-UP1-M64		
SNAP-ENET-S64		
SNAP-B3000-ENET, SNAP-ENET-RTC		
SNAP-UP1-ADS		
SNAP-PAC-R1		
SNAP-PAC-R2		
SNAP-BRS*		
String Table		
String Variable		
Up Timer Variable		

* ioControl Professional only

Standard Example:

Move to Pointer Table Element

Object	Valve_One	*Integer 32 Variable*

Index Of Table	Current_Index Digital_Outputs	*Integer 32 Variable Pointer Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `&` operator to get the address of the object and use the `=` operator to make the assignment:

```
Digital_Outputs[Current_Index] = &Valve_One;
```

Notes: In OptoScript code, simply make an assignment to the pointer table.

See Also: Move from Pointer Table Element (page M-9), Pointer Table Element Equal to NULL? (page P-4)

Move to String Table Element

String Action

Function: To put a string into a string table.

Typical Use: To load strings into a table for later retrieval.

Details:
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- Valid range for *Index* (*Argument 2*) is zero to the table length minus 1 (size – 1).
- Strings with a length greater than the width of the table will be truncated to fit.

Arguments:

Argument 1	Argument 2	Argument 3
From	**To Index**	**Of Table**
String Literal	Integer 32 Literal	String Table
String Variable	Integer 32 Variable	

Standard Example: In the following example, quotes are shown for clarity only. Do not use them in standard commands.

Move to String Table Element

From	"MON"	*String Literal*
To Index	INDEX	*Integer 32 Variable*
Of Table	STRING_TABLE	*String Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the = operator. Remember that quotes are required in OptoScript code.

```
STRING_TABLE[INDEX] = "MON";
```

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, simply make an assignment to the table element.
- 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." An integer variable would also be required to "remember" the next available index (where the next entry goes).

Queue Errors: -12 = Invalid table index—index was negative or greater than or equal to the table size.

See Also: Move from String Table Element (page M-10), Get Length of Table (page G-83), Move to String Table Elements (page M-24)

Move to String Table Elements

String Action

Function: To put a given string into a range of table elements within the same table.

Typical Use: To initialize elements within a table to the same string.

Details:
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- Valid range for *Index* (*Argument 2*) is zero to the table length minus 1 (size – 1). However, if you need to set a value to the entire table and don't know the table's size, you can use a starting index of 0 and an ending index of -1.
- Strings with a length greater than the width of the table will be truncated to fit.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
From	**Start Index**	**End Index**	**Of Table**
String Literal	Integer 32 Literal	Integer 32 Literal	String Table
String Variable	Integer 32 Variable	Integer 32 Variable	

Standard Example: In the following example, quotes are shown for clarity only. Do not use them in standard commands.

Move to String Table Elements

From	"MON"	*String Literal*
Start Index	0	*Integer 32 Literal*
End Index	6	*Integer 32 Literal*
Of Table	DAYS	*String Table*

OptoScript Example: **`MoveToStrTableElements(`***From, Start Index, End Index, Of Table***`)`**

```
MoveToStrTableElements("MON", 0, 6, DAYS);
```

This is a procedure command; it does not return a value. Remember that quotes are required in OptoScript code.

Notes:
- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Compared to other methods such as loops, this command initializes table elements very quickly.

Queue Errors: -12 = Invalid table index—index was negative or greater than or equal to the table size.

See Also: Move from String Table Element (page M-10), Get Length of Table (page G-83), Move to String Table Element (page M-23)

Multiply

Mathematical Action

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 *Argument 1* and *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.

Arguments:

Argument 1 [Value]	**Argument 2** Times	**Argument 3** Put Result in
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Multiply

	Ingredient_1_Weight	*Analog Input*
Times	Temperature_Adjust	*Float Variable*
Put Result in	Corrected_Weight	*Analog Output*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `*` operator.

```
Corrected_Weight = Ingredient_1_Weight * Temperature_Adjust;
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `*` operator can be used in many ways. For more information on mathematical expressions in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- *Speed Tip:* Use Bit Shift instead for integer math where the multiplier is 2, 4, 8, 16, 32, 64, and so on.

Queue Errors: -13 = Overflow error—result too large.

See Also: Divide (page D-20), Bit Shift (page B-15)

N

Natural Log

Mathematical Action

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 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Natural Log

Of	Fermentation_Rate	*Float Variable*
Put Result in	Rate_Calculation	*Float Variable*

OptoScript Example:

`NaturalLog(`*`Of`*`)`

```
Rate_Calculation = NaturalLog(Fermentation_Rate);
```

This is a function command; it returns the natural log of the value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: ioControl only implements a natural logarithm command. However, there is a simple way to compute logarithms for bases other than base e. Divide the natural log of the number by the natural log of the base:

$$\mathrm{Log}_{BASE}(\mathrm{number}) = \frac{\ln(\mathrm{number})}{\ln(\mathrm{base})}$$

For example:

$$\mathrm{Log}_{10}(100) = \frac{\ln(100)}{\ln(10)} = 2$$

Just remember that the range of the logarithm argument is a number greater than zero. A control engine error will be flagged if the argument is less than or equal to zero.

To get a $\log_{10}$, divide the result of this command by 2.302585, which is ln(10).

Number	LOG_e	LOG_{10}
1	0	0
10	2.302585	1
100	4.605170	2
1000	6.907755	3

Queue Errors: -13 = Overflow error—result too large.

-14 = Invalid number.

See Also: Raise to Power (page R-2)

NOT

Logical Action

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 a copy of *Argument 1* and puts result in *Argument 2*. Examples:

Argument 1	Argument 2
0	1
-1	0
22	0

- If *Argument 1* is True (non-zero), the result will be False (0). If *Argument 1* is False (0), the result will be True (non-zero).

Arguments:

Argument 1 [Value]	Argument 2 Put Result in
Digital Input	Digital Output
Digital Output	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Integer 64 Variable
Integer 32 Literal	
Integer 32 Variable	
Integer 64 Literal	
Integer 64 Variable	

Standard Example:

NOT

	Current_State	*Integer 32 Variable*
Put Result in	DOUT1	Digital Output

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `not` operator.

```
DOUT1 = not Current_State;
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `not` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Integers or digital points are best for this command. For other types, consider using Test Within Limits, Test Greater, and Test Less.
- To invert the True/False state of *Argument 1*, make both arguments the same.
- To toggle all 32 or 64 bits of an integer, use Bit NOT.

See Also: Bit NOT (page B-5), Test Within Limits (page T-9), Test Greater (page T-4), Test Less (page T-6)

NOT?

Logical Condition

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
Is
Digital Input
Digital Output
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Standard Example:

NOT? *Is* CURRENT_STATE *Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `not` operator.

```
if (not Current_State) then
```

Notes:
- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `not` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Integers or digital points are best for this command. For other types, consider using Within Limits? Greater? or Less?
- To determine whether a value is True (non-zero), use either Variable True? or the False exit.

See Also: AND? (page A-8) OR? (page O-7) XOR? (page X-2) Variable True? (page V-2) Within Limits? (page W-1) Greater? (page G-146) Less? (page L-1)

Not Equal?

Logical Condition

Function: To determine if two values are different.

Typical Use: To perform reverse logic.

Details:

- Determines if *Argument 1* is different from *Argument 2*. Evaluates True if the two values are different, False otherwise. Examples:

Argument 2	Result
0	False
0	True
65280	True
22.22	False

Arguments:

Argument 1	**Argument 2**
Is	**To**
Analog Input	Analog Input
Analog Output	Analog Output
Digital Input	Digital Input
Digital Output	Digital Output
Down Timer Variable	Down Timer Variable
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable
Up Timer Variable	Up Timer Variable

Standard Example:

Is	BATCH_STEP	*Integer 32 Variable*
Not Equal?		
To	4	*Integer 32 Literal*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<>` operator.

```
if (BATCH_STEP <> 4) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. In OptoScript code, the `<>` operator can be used in several ways. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- Use Within Limits? to test for an approximate match (recommended for non-integers). To test for equality, use either Equal? or the False exit.

See Also: Greater? (page G-146), Less? (page L-1), Less Than or Equal? (page L-3), Greater Than or Equal? (page G-148), Equal? (page E-16), Within Limits? (page W-1)

Not Equal to Numeric Table Element?

Logical Condition

Function: To determine if a numeric value is different from a specified value in a float or 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 float or integer table *Argument 3*). Examples:

Value 1	Value 2	Result
0.0	0.0	False
0.0001	0.0	True
-98.765	-98.765	False
-32768	-32768	False
2222	2222	False

- Evaluates True if the two values are different, False otherwise.

Arguments:

Argument 1 **Is**	Argument 2 **At Index**	Argument 3 **Of Table**
Analog Input	Integer 32 Literal	Float Table
Analog Output	Integer 32 Variable	Integer 32 Table
Digital Input		Integer 64 Table
Digital Output		
Down Timer Variable		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard Example:

Is	This_Reading	*Float Variable*
Not Equal to Numeric Table Element?		
At Index	Table_Index	*Integer 32 Variable*
Of Table	Table_of_Readings	*Float Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `<>` operator.

```
if (This_Reading <> Table_of_Readings[Table_Index]) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `<>` operator can be used in several ways. For more information on comparison operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- To test for equality, use either Equal to Table Element? or the False exit.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to table size.

See Also: Equal to Numeric Table Element? (page E-17), Greater Than Numeric Table Element? (page G-147), Greater Than or Equal To Numeric Table Element? (page G-149), Less Than Numeric Table Element? (page L-2), Less Than or Equal to Numeric Table Element? (page L-4)

Numeric Table Element Bit Clear

Logical Action

Function: To clear a specific bit (set it to 0) at the specified index in an integer table.

Typical Use: To clear a bit in an integer table that is used as a flag.

Details:

- Valid range for the bit to clear is 0–31.
- Table indexes are zero through table length minus one.

Arguments:

Argument 1	Argument 2	Argument 3
Element Index	**Of Integer Table**	**Bit To Clear**
Integer 32 Literal	Integer 32 Table	Integer 32 Literal
Integer 32 Variable		Integer 32 Variable

Standard Example:

Numeric Table Element Bit Clear

Element Index	4	*Integer 32 Literal*
Of Integer Table	PUMP_CTRL_BITS	*Integer 32 Table*
Bit To Clear	15	*Integer 32 Literal*

OptoScript Example:

`NumTableElementBitClear(`*Element Index, Of Integer Table, Bit to Clear***`)`**

```
NumTableElementBitClear(4, PUMP_CTRL_BITS, 15);
```

This is a procedure command; it does not return a value.

Queue Errors: -12 = Invalid table index value—index was negative or greater than the table size.

See Also: Bit Clear (page B-4), Numeric Table Element Bit Set (page N-7), Numeric Table Element Bit Test (page N-8)

Numeric Table Element Bit Set

Logical Action

Function: To set a specific bit (set it to 1) at the specified index in an integer table.

Typical Use: To set a bit in an integer table that is used as a flag.

Details:
- Valid range for the bit to set is 0–31.
- Table indexes are zero through table length minus one.

Arguments:

Argument 1	Argument 2	Argument 3
Element Index	**Of Integer Table**	**Bit to Set**
Integer 32 Literal	Integer 32 Table	Integer 32 Literal
Integer 32 Variable		Integer 32 Variable

Standard Example:

Numeric Table Element Bit Set

Element Index	4	*Integer 32 Literal*
Of Integer Table	PUMP_CTRL_BITS	*Integer 32 Table*
Bit to Set	15	*Integer 32 Literal*

OptoScript Example:

`NumTableElementBitSet(`*Element Index, Of Integer Table, Bit to Set***`)`**

```
NumTableElementBitSet(4, PUMP_CTRL_BITS, 15);
```

This is a procedure command; it does not return a value.

Queue Errors: -12 = Invalid table index value—index was negative or greater than the table size.

See Also: Bit Set (page B-14), Numeric Table Element Bit Clear (page N-6), Numeric Table Element Bit Test (page N-8)

Numeric Table Element Bit Test

Logical Action

Function: To test a specific bit at the specified index in an integer table to see if it is set or not.

Typical Use: To test a bit in an integer table that is used as a flag.

Details:
- A logical True (non-zero) is returned if the bit is set, otherwise a logical False (0) is returned.
- Valid range for the bit to test is 0–31 for Integer 32 tables, or 0–63 for Integer 64 tables.
- Table indexes are zero through table length minus one.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Element Index	**Of Integer Table**	**Bit to Test**	**Put Result in**
Integer 32 Literal	Integer 32 Table	Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 64 Table	Integer 32 Variable	Float Variable
			Integer 32 Variable

Standard Example:

Numeric Table Element Bit Test

Element Index	4	*Integer 32 Literal*
Of Integer Table	Pump_Ctrl_Bits	*Integer 32 Table*
Bit to Test	15	*Integer 32 Literal*
Put Result in	Result	*Integer 32 Variable*

OptoScript Example:

`NumTableElementBitTest(`*Element Index, Of Integer Table, Bit to Test***`)`**

```
Result = NumTableElementBitTest(4, Pump_Ctrl_Bits, 15);
```

This is a function command; it returns the status of the bit, either set (non-zero) or not set (0). The returned value can be consumed by a variable (as in the example shown) or by a control structure, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: The value returned is the bit status.

Queue Errors: -12 = Invalid table index value—index was negative or greater than the table size.

See Also: Numeric Table Element Bit Set (page N-7), Numeric Table Element Bit Clear (page N-6)

Off?

Digital Point Condition

Function: To determine if a digital input or output is off.

Typical Use: To determine the status of a digital input or output point.

Details:
- Evaluates True if the specified point is off, False if the point is on.
- *Speed Tip:* Use Get Digital I/O Unit as Binary Value to get the state of all points at once. Then use Bit Test to determine the state of individual points.

Arguments:

Argument 1
Is
Digital Input
Digital Output

Standard Example:

Is	Safety_Interlock	*Digital Input*
	Off?	

OptoScript Example:

IsOff(*Point***)**

```
if (IsOff(Safety_Interlock)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: May be used with either input or output points.

Dependencies: Applies to all digital inputs and outputs.

See Also: On? (page O-3)

Off-Latch Set?

Digital Point Condition

Function: Checks the status of the specified off latch.

Typical Use: To determine if a button was pressed or an object passed by a sensor.

Details: Evaluates True if the latch is set, which indicates that the specified input changed from on to off.

Arguments:

Argument 1
On Point
Digital Input

Standard Example:

Off-Latch Set?

On Point	PUMP3_STOP_BUTTON

OptoScript Example:

`IsOffLatchSet(`*On Point***`)`**

```
if (IsOffLatchSet(PUMP3_STOP_BUTTON)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Use Clear Off-Latch if true to reset the latch for next time.

See Also: On-Latch Set? (page O-4)

On?

Digital Point Condition

Function: To determine if a digital input or output is on.

Typical Use: To determine the status of a digital input or output point.

Details: Evaluates True if the specified point is on, False if the point is off.

Arguments:

Argument 1
Is
Digital Input
Digital Output

Standard Example:

On?	*Is*	Motor_Power	*Digital Input*

OptoScript Example:

`IsOn(`*Point***`)`**

```
if (IsOn(Motor_Power)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- May be used with either input or output points.
- *Speed Tip:* Use Get I/O Unit as Binary Value to get the state of all digital points at once. Then use Bit Test to determine the state of individual points.

Dependencies: Applies to all digital inputs and outputs.

See Also: Off? (page O-1)

On-Latch Set?

Digital Point Condition

Function: Checks the status of the specified on latch.

Typical Use: To determine if a button was pressed or an object passed by a sensor.

Details: Evaluates True if the latch is set, which indicates that the specified input changed from off to on.

Arguments:

Argument 1
On Point
Digital Input

Standard Example:

On Point	Clip_Missing_Prox

On-Latch Set?

OptoScript Example:

`IsOnLatchSet(`*On Point***`)`**

```
if (IsOnLatchSet(Clip_Missing_Prox)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Use Clear On-Latch if true to reset the latch for next time.

See Also: Off-Latch Set? (page O-2)

Open Outgoing Communication

Communication Action

Function: To establish communication with another device or entity. Once the connection is established, communication can go both ways (incoming and outgoing).

Typical Use: To communicate with other devices on the network via TCP/IP, UDP, or a serial connection; to FTP data from the brain to a file on another device; or to work with files in the brain's file structure.

Arguments:

Argument 1	Argument 2
Communication Handle	**Put Result in**
Communication Handle	Integer 32 Variable

Standard Example:

Open Outgoing Communication

Communication Handle	TANK_CONTROL	*Communication Handle*
Put Result in	COMM_STATUS	*Integer 32 Variable*

OptoScript Example:

`OpenOutgoingCommunication(`*Communication Handle***`)`**

```
COMM_STATUS = OpenOutgoingCommunication(TANK_CONTROL);
```

This is a function command; it returns a status code as defined below.

Notes:
- For TCP communication, depending on network traffic and the network arrangement, you may need to add a delay to the chart to make sure the session is open. The amount of delay needed depends on your network. (Distant connections might even take more than one second.) If you add a delay to the chart, then check the status of the session using Get Number of Characters Waiting.
- See “Communication Commands” in the *ioControl User’s Guide*, Chapter 10.

Status Codes:

0 = Success

-10 = Serial: Invalid port number.

-20 = Serial: Device busy. May be in use by another user or application. Use ioManager to check communication port control configuration; make sure device is not being used by PPP or M2M.

-46 = Invalid string. Check communication handle value (must have no spaces, be lowercase).

-47 = Open failed. Handle has already been opened.

-49 = No more connections are available. Maximum number of connections of this type already in use.

-50 = Open connection timeout. Could not establish connection within the timeout period.

-78 = No destination given. When sending a file via FTP, use Send Communication Handle Command to specify the name of the file on the remote server.

-203 = Driver could not be found or loaded. Make sure the communication handle designator (tcp, ftp, file, etc.) is in lowercase letters and correctly spelled.

-412 = TCP/IP: Cannot connect error. Make sure the device is on.

-417 = Cannot open file. Check filename; verify that the file exists.

-446 = FTP: Login failed. Check user name, password, and maximum number of logins on server.

-447 = FTP: Connection failed. Check IP address and port.

-448 = FTP: Could not create session. Check IP address and port.

See Also: Close Communication (page C-29), Communication Open? (page C-32)

OR

Logical Action

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 *Argument 1* and *Argument 2* and puts result in *Argument 3*. The result is non-zero (True) if either value is non-zero, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument3
0	0	0
-1	0	1
0	-1	1
-1	-1	1

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 **[Value]**	**Argument 2** **With**	**Argument 3** **Put Result in**
Digital Input	Digital Input	Digital Output
Digital Output	Digital Output	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

OR		
	LIMIT_SWITCH1	*Digital Input*
With	LIMIT_SWITCH2	*Digital Output*
Put Result in	MOTOR1_OUTPUT	*Digital Output*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `or` operator.

```
MOTOR1_OUTPUT = LIMIT_SWITCH1 or LIMIT_SWITCH2;
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `or` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- It is advisable to use only integers or digital points with this command.
- In OptoScript code, you can combine logical operators and OR multiple variables, for example: `x = a or b or c or d;`
- In standard ioControl code, 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 (page B-10)

OR?

Logical Condition

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 Is	Argument 2 [Value]
Digital Input	Digital Input
Digital Output	Digital Output
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable

Standard Example:

OR?	*Is*	LIMIT_SWITCH1	*Digital Input*
		LIMIT_SWITCH2	*Digital Input*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `or` operator.

```
if (LIMIT_SWITCH1 or LIMIT_SWITCH2) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `or` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- It is advisable to use only integers or digital points with this command.
- To determine whether both values are False (zero, off), use either Variable False? or the False exit.
- Multiple uses of OR? within a condition block result in the OR? pairs being AND?ed.

See Also: NOT (page N-2), AND? (page A-8) XOR? (page X-2)

P

Pause Timer

Timing Action

Function: To pause a timer variable.

Typical Use: Used with the Continue Timer command to trade on or off time of a variable or I/O point.

Details:

- The timer must have been started with either the Start Timer or Move commands.
- To start a paused timer again from the value at which it was paused, use the command Continue Timer.

Arguments:

Argument 1
Timer
Down Timer Variable
Up Timer Variable

Standard Example:

Pause Timer

Timer	OVEN_TIMER	*Down Timer Variable*

OptoScript Example:

PauseTimer(*Timer***)**

```
PauseTimer(OVEN_TIMER);
```

This is a procedure command; it does not return a value.

Notes: See "Timing Commands" in Chapter 10 of the *ioControl User's Guide* for more information on using timers.

See Also: Start Off-Pulse (page S-96), Stop Timer (page S-102), Continue Timer (page C-39), Set Down Timer Preset Value (page S-21), Set Up Timer Target Value (page S-86)

PID Loop Communication Enabled?

Simulation Condition

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: Checks a flag in the control engine to determine whether communication to the specified PID loop is enabled.

Typical Use: Primarily used in factory QA testing and simulation.

Details:

- Evaluates True if communication is enabled.
- Because the PID runs on the I/O unit, not in the control engine, any ioControl command referring to a PID loop by name will not affect the PID while communication to it is disabled. Even on a SNAP Ultimate brain, the PID loop runs on the I/O side, not the control side.
- No changes can be made to the PID by the program in the control engine while the PID is disabled.

Arguments:

Argument 1
PID Loop
PID Loop

Standard Example:

PID Loop	*FACTORY_HEAT_2BA*

PID Loop Communication Enabled?

OptoScript Example:

`IsPidLoopCommEnabled(`*PID Loop***`)`**

```
if (IsPidLoopCommEnabled(FACTORY_HEAT_2BA)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the ioControl User's Guide for more information.

See Also: Enable Communication to PID Loop (page E-6); Disable Communication to Mistic PID Loop (page D-9); I/O Point Communication Enabled? (page I-2)

Pointer Equal to NULL?

Pointers Condition

Function: To determine if a pointer is pointing to an object.

Typical Use: To verify that a pointer is pointing to an object (to prevent an undefined pointer).

Details: Evaluates False if the pointer is pointing to an object, True otherwise.

Arguments:

Argument 1
Pointer
Pointer Variable

Standard Example:

Pointer Equal to NULL?	*Pointer*	IO_Pointer	*Pointer Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `==` and `null` operators.

```
if (IO_Pointer == null) then
```

Notes:

- The example shown is only one way to use these operators. For more information on operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- If you try to perform an operation on a NULL pointer, an error will be posted in the message queue.

See Also: Clear Pointer (page C-28), Move to Pointer (page M-19)

Pointer Table Element Equal to NULL?

Pointers Condition

Function: To determine if a specific element of a pointer table points to an object.

Typical Use: To verify that an element in a pointer table is pointing to an object (to prevent an undefined pointer).

Details: Evaluates False if the specified element is pointing to an object, True otherwise.

Arguments:

Argument 1	**Argument 2**
Index	**Of Table**
Integer 32 Literal	Pointer Table
Integer 32 Variable	

Standard Example:

Index	Current_Index	*Integer 32 Variable*
Pointer Table Element Equal to NULL?		
Of Table	IO_Table	*Pointer Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `==` and `null` operators.

```
if (IO_Table[Current_Index] == null) then
```

Notes:

- The example shown is only one way to use these operators. For more information on operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- If you try to perform an operation on a NULL pointer, an error will be posted in the message queue.

See Also: Clear Pointer Table Element (page C-28), Move to Pointer Table Element (page M-21)

Raise e to Power

Mathematical Action

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.

Arguments:

Argument 1	Argument 2
Exponent	**Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Raise e to Power

Exponent	Gas_Pressure	*Analog Input*
Put Result in	Pressure_Calculation	*Float Variable*

OptoScript Example:

`RaiseEToPower(`*Exponent***`)`**

```
Pressure_Calculation = RaiseEToPower(Gas_Pressure);
```

This is a function command; it returns the result of the mathematical computation. The returned value can be consumed by a variable (as shown) or by another item, such as a math expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.

Queue Errors: -13 = Overflow error—result too large.

See Also: Natural Log (page N-1), Raise to Power (page R-2)

Raise to Power

Mathematical Action

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**
Raise	**To the**	**Put Result in**
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Up Timer Variable
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Raise to Power

Raise	10	*Integer 32 Literal*
To the	2	*Integer 32 Literal*
Put Result in	TEN_SQUARED	*Integer 32 Variable*

OptoScript Example:

`Power(`*Raise, To the***`)`**

```
= Power(10, 2);
```

This is a function command; it returns the result of the mathematical computation. The returned value can be consumed by a variable (as shown) or by another item, such as a math expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- Multiplying a number by itself is faster than raising a number to the power of 2.

Queue Errors:

-13 =Overflow error—result too large.

-14 = Invalid number.

See Also: Raise e to Power (page R-1), Square Root (page S-92)

Ramp Analog Output

Analog Point Action

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.
- This command applies to SNAP-UP1-ADS, SNAP-B3000-ENET, and SNAP-UP1-M64 I/O units as well as to *mistic* I/O units."
- Ramping starts from the current output value and proceeds toward the specified endpoint value.
- The ramp rate is specified in engineering units per second. This rate should be a positive number. A rate of zero or less will cause error -42 (Invalid limit) to appear in the message queue.
- 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
Ramp Endpoint	**Units/Sec**	**Point to Ramp**
Float Literal	Float Literal	Analog Output
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Ramp Analog Output

Ramp Endpoint	SOAK_TEMP	*Float Variable*
Units/Sec	RAMP_RATE	*Float Variable*
Point to Ramp	TEMP_CONTROL	*Analog Output*

OptoScript Example:

RampAnalogOutput(*Ramp Endpoint, Units/Sec, Point to Ramp***)**

```
RampAnalogOutput(SOAK_TEMP, RAMP_RATE, TEMP_CONTROL);
```

This is a procedure command; it does not return a value.

Notes:

- To stop the ramp on a mistic I/O unit at any time, use Move (or an assignment in OptoScript code) to send the desired “static” value to the analog output channel. To achieve the same result on any type of brain, send a new Ramp analog Output command with the desired “static” value as the endpoint and a very fast rate.
- 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.

Queue Errors: -42 = Invalid limit. (The ramp rate was less than or equal to zero.)

Read Event/Reaction Hold Buffer

Event/Reaction Action

NOTE: This command is for mistic I/O units only.

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
Event/Reaction	**Put in**
Analog Event/Reaction	Float Variable
Digital Event/Reaction	Integer 32 Variable

Standard Example:

Read Event/Reaction Hold Buffer

Event/Reaction	Sequence_Finished	*Analog Event/Reaction*
Put in	Counter_Value	*Integer 32 Variable*

OptoScript Example:

`ReadEventReactionHoldBuffer(`*Event/Reaction***`)`**

```
Counter_Value = ReadEventReactionHoldBuffer(Sequence_Finished);
```

This is a function command; it returns the value in the event/reaction hold buffer. The returned value can be consumed by a variable (as shown) or by another item, such as a math expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Event/Reaction Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use 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.

Read Number from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a value from an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map and store that value in an integer or float variable.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioControl or ioManager. The control engine must be connected to the I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use Scratch Pad commands instead (Get I/O Unit Scratch Pad Float Element and related commands).
- *Argument 4*, Mem address, includes only the last eight digits of the memory map address (the lower 32 bits).
- A maximum of 256 32-bit numeric or eight 128-byte string entries can be read at once.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Mem address**	Argument 3 **To**	Argument 4 **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Float Variable	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	Integer 32 Variable	
SNAP-UP1-M64		Integer 64 Variable	
SNAP-ENET-S64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Read Number from I/O Unit Memory Map

I/O Unit	MYIOUNIT	*SNAP-UP1-ADS*
Mem address	0xFFFFFFFF	*Integer 32 Literal*
To	MYINTVAR	*Integer 32 Variable*
Put Status In	STATUS	*Integer 32 Variable*

OptoScript Example:

`ReadNumFromIoUnitMemMap(`*I/O Unit, Mem address, To***`)`**

```
STATUS = ReadNumFromIoUnitMemMap(MYIOUNIT, 0xFFFFFFFF, MYINTVAR);
```

This is a function command; it returns a status code as listed below.

Notes:

- In Action blocks, use hex integer display for easy entering of memory map addresses.
- The control engine does not convert the variable type to match the area of memory map being read. The control engine has no knowledge of which memory map areas are integers and which are floats. You must write the correct type of data to the specified memory map address.

For example, unpredictable results would occur if you try to read an integer 32 variable from the analog point area of the memory map. A float variable should be used instead. See the *SNAP Ethernet-Based I/O Units Protocols and Programming Guide* (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

- If *Argument 3* is an Integer 64 variable, 64 bits of data will be read. For example, if you read the address 0xF0300020 (the first integer for unit type in the Status Read area), you will also receive the I/O unit hardware revision (month), which starts at 0xF0300024.

Status Codes:

0 = success

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-56 = Invalid memory map address.

-58 = No data received. Make sure I/O unit has power.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Read Numeric Table from I/O Unit Memory Map (page R-7), Write Numeric Table to I/O Unit Memory Map (page W-4), Write Number to I/O Unit Memory Map (page W-3), Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad Float Element (page G-70), Get I/O Unit Scratch Pad Float Table (page G-72)

Read Numeric Table from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a range of values from an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map and store them into an integer 32 or float table.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioControl or ioManager. The control engine must be connected to the I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use Scratch Pad commands instead (Get I/O Unit Scratch Pad Integer 32 Table and related commands).
- *Argument 1*, Length, is the length of data in the memory map in quads (groups of four bytes) and also the number of table elements. Maximum length is 64 quadlets (256 bytes).
- *Argument 4*, Mem address, includes only the last eight digits of the memory map address (the lower 32 bits).

Arguments:

Argument 1 **Length**	**Argument 2** **Start Index**	**Argument 3** **I/O Unit**	**Argument 4** **Mem address**
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64	Integer 32 Variable
		SNAP-UP1-M64	
		SNAP-ENET-S64	
		SNAP-B3000-ENET,	
		SNAP-ENET-RTC	
		SNAP-UP1-ADS	
		SNAP-PAC-R1	
		SNAP-PAC-R2	

Argument 5 **To**	**Argument 6** **Put Status in**
Float Table	Integer 32 Variable
Integer 32 Table	

Standard Example:

Read Numeric Table from I/O Unit Memory Map

Length	0x10	*Integer 32 Literal*
Start Index	0x5	*Integer 32 Literal*
I/O Unit	MYIOUNIT	*SNAP-UP1-D64*
Mem address	0xFFFFFFFF	*Integer 32 Literal*
To	MYINTTABLE	*Integer 32 Table*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

ReadNumTableFromIoUnitMemMap(*Length, Start Index, I/O Unit, Mem address, To***)**

```
STATUS = ReadNumTableFromIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYINTTABLE);
```

This is a function command; it returns a status code as listed below.

In OptoScript code, you can use hex in some arguments and another format in others, for example:

```
STATUS = ReadNumTableFromIoUnitMemMap(16, 5, MYIOUNIT, 0xFFFFFFFF,
MYINTTABLE);
```

Notes:

- In Action blocks, use hex integer display for easy entering of memory map addresses. When you display integers in hex, note that the length of data and start index arguments are also in hex.
- The control engine does not convert the table type to match the area of the memory map being read. The control engine has no knowledge of which memory map areas are integers and which are floats. You must write the correct type of data to the specified memory map address.

  For example, unpredictable results would occur if you try to read an integer 32 table from the analog bank area of the memory map. A float table should be used instead. See the *SNAP Ethernet-Based I/O Units Protocols and Programming Guide* (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-56 = Invalid memory map address.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Read Number from I/O Unit Memory Map (page R-5), Write Numeric Table to I/O Unit Memory Map (page W-4), Write Number to I/O Unit Memory Map (page W-3), Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Get I/O Unit Scratch Pad Float Table (page G-72)

Read String from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a value from an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map and store that value in a string variable.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioControl or ioManager. The control engine must be connected to the I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use Scratch Pad commands instead (Get I/O Unit Scratch Pad String Element and related commands).
- *Argument 3*, Mem address, includes only the last eight digits of the memory map address (the lower 32 bits).

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
Length	**I/O Unit**	**Mem address**	**To**	**Put Status in**
Integer 32 Literal	SNAP-ENET-D64	Integer 32 Literal	String Variable	Integer 32 Variable
Integer 32 Variable	SNAP-UP1-D64	Integer 32 Variable		
	SNAP-UP1-M64			
	SNAP-ENET-S64			
	SNAP-B3000-ENET,			
	SNAP-ENET-RTC			
	SNAP-UP1-ADS			
	SNAP-PAC-R1			
	SNAP-PAC-R2			

Standard Example:

Read String from I/O Unit Memory Map

Length	20	*Integer 32 Literal*
I/O Unit	MYIOUNIT	*SNAP-B3000-ENET, SNAP-ENET-RTC*
Mem address	0xFFFFFFFF	*Integer 32 Literal*
To	MYSTRINGVAR	*String Variable*
Put Status In	STATUS	*Integer 32 Variable*

OptoScript Example:

`ReadStrFromIoUnitMemMap(`*Length, I/O Unit, Mem address, To***`)`**

```
STATUS = ReadStrFromIoUnitMemMap(20, MYIOUNIT, 0xFFFFFFFF, MYSTRINGVAR);
```

This is a function command; it returns a status code as listed below.

Notes:

- In Action blocks, use hex integer display for easy entering of memory map addresses.
- The control engine does not convert the variable type to match the area of memory map being read. The control engine doesn't know which memory map areas are strings and which are other formats. You must read the correct type of data from the specified memory map address.

  For example, unpredictable results would occur if you try to read a string variable from the analog point area of the memory map. A float variable should be used instead. See the

SNAP Ethernet-Based I/O Units Protocols and Programming Guide (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

Status Codes: 0 = Success

-3 = Invalid length. Length must be greater than zero.

-12 = Invalid table index value—index was negative or greater than the table size.

-23 = Destination string too short.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-56 = Invalid memory map address.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Read String Table from I/O Unit Memory Map (page R-11), Write String Table to I/O Unit Memory Map (page W-7), Write String to I/O Unit Memory Map (page W-9), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Read String Table from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a range of values from an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map and store them in a string table.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O units that have been configured in ioControl or ioManager. The control engine must be connected to the I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use Scratch Pad commands instead (Get I/O Unit Scratch Pad String Table and related commands).
- *Argument 1*, Length, is the number of bytes to read in the memory map. Data is read in block sizes that are multiples of four.
- *Argument 4*, Mem address, includes only the last eight digits of the memory map address (the lower 32 bits).

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Length	**Start Index**	**I/O Unit**	**Mem address**
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64	Integer 32 Variable
		SNAP-UP1-M64	
		SNAP-ENET-S64	
		SNAP-B3000-ENET,	
		SNAP-ENET-RTC	
		SNAP-UP1-ADS	
		SNAP-PAC-R1	
		SNAP-PAC-R2	

Argument 5	Argument 6
To	**Put Status in**
String Table	Integer 32 Variable

Standard Example:

Read String Table from I/O Unit Memory Map

Length	0x10	*Integer 32 Literal*
Start Index	0x5	*Integer 32 Literal*
I/O Unit	MYIOUNIT	*SNAP-UP1-ADS*
Mem address	0xFFFFFFFF	*Integer 32 Literal*
To	MYSTRINGTABLE	*String Table*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`ReadStrTableFromIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, To***`)`**

```
STATUS = ReadStrTableFromIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYSTRINGTABLE);
```

This is a function command; it returns a status code as listed below.

In OptoScript, you can use hex in one argument but not in others, for example:

```
STATUS = ReadStrTableFromIoUnitMemMap(16, 5, MYIOUNIT, 0xFFFFFFFF,
MYSTRINGTABLE);
```

Notes:

- In Action blocks, use hex integer display for easy entering of memory map addresses. When you display integers in hex, note that the length of data and start index arguments are also in hex.
- The control engine does not convert the table type to match the area of the memory map being read. The control engine has no knowledge of which memory map areas are strings and which are other formats. You must read the correct type of data from the specified memory map address.

  For example, unpredictable results would occur if you try to read a string table from the analog bank area of the memory map. A float table should be used instead. See the *SNAP Ethernet-Based I/O Units Protocols and Programming Guide* (Opto 22 form 1465) to determine the data types for specific areas of the memory map.
- The string table width needs to be at least 4. If not, a -23 error is returned.

Status Codes:

0 = Success

-3 = Invalid length. Length must be greater than zero.

-12 = Invalid table index value—index was negative or greater than the table size.

-23 = Destination string too short.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-56 = Invalid memory map address.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Read String from I/O Unit Memory Map (page R-9), Write String Table to I/O Unit Memory Map (page W-7), Write String to I/O Unit Memory Map (page W-9), Get I/O Unit Scratch Pad String Element (page G-78), Get I/O Unit Scratch Pad String Table (page G-80)

Receive Character

Communication Action

Function: To get a single character from a communication handle and move it to a numeric variable.

Typical Use: To get a message from another device or file one character at a time. Use Append Character to String (or a + in OptoScript) to append these characters (selectively if desired) to a string variable.

Details:

- Receives the next character. For example, receives the oldest character from the receive buffer for a TCP communication handle, or receives the next character in a file. Character values will be 0–255.
- If there are no characters to receive, a negative error code number (for example, -58) is returned. To avoid this problem, use Get Number of Characters Waiting before using this command.
- 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 and a 32 will appear as a space.

Arguments:

Argument 1	Argument 2
Communication Handle	**Put in**
Communication Handle	Float Variable
	Integer 32 Variable

Standard Example:

Receive Character

Communication Handle	UNIT_2	*Communication Handle*
Put in	CHAR	*Integer 32 Variable*

OptoScript Example:

ReceiveChar(*Communication Handle***)**

```
CHAR = ReceiveChar(UNIT_2);
```

This is a function command; it returns the next character available for the communication handle. The returned value can be consumed by a variable (as shown) or by another item, such as a math expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*. For an ASCII table, see "String Commands" in the same chapter.
- Always use command Get Number of Characters Waiting before this command to avoid unnecessary timeout errors.
- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Status Codes: -36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-58 = Character not found.

-76 = At end of file.

See Also: Append Character to String (page A-9), Get Number of Characters Waiting (page G-101), Receive N Characters (page R-14), Send Communication Handle Command (page S-2)

Receive N Characters

Communication Action

Function: Gets a specified number of characters from a communication handle.

Typical Use: Can be used to receive the message a piece at a time, especially when the message is longer than a single string can hold.

Details:

- If N is greater than the number of characters ready to be received, all the characters will be returned along with an error, often -39.
- If no characters are in the receive buffer, a -58 error will be returned.
- If N is greater than the string length, as many characters as will fit will be returned along with a String Too Short error (-23).

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Put in	**Number of Characters**	**Communication Handle**	**Put Status in**
String Variable	Integer 32 Literal	Communication Handle	Float Variable
	Integer 32 Variable		Integer 32 Variable

Standard Example:

Receive N Characters

Put in	RECV_MSG	*String Variable*
Number of Characters	QTY_CHARS	*Integer 32 Variable*
Communication Handle	UNIT_2	*Communication Handle*
Put Status in	RECV_STATUS	*Integer 32 Variable*

OptoScript Example:

`ReceiveNChars(`*Put in, Number of Characters, Communication Handle***`)`**

```
RECV_STATUS = ReceiveNChars(RECV_MSG, QTY_CHARS, UNIT_2);
```

This is a function command; it returns a zero if successful, or one of the status codes listed below.

Notes:

- The length of the string variable should be a few characters greater than the longest expected string.
- Use Receive String to get end-of-message character-delimited pieces of the message in the receive buffer.
- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Dependencies:

- Must have previously used Open Outgoing Communication to establish a session, or (for a TCP communication handle) Accept Incoming Communication to accept a session initiated by a TCP/IP peer.
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Status Codes: -36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-37 = Lock port timeout.

-39 = Timeout on receive.

-44 = String too short.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-58 = Character not found.

-69 = Invalid parameter (null pointer) passed. Make sure communication handle is open.

-76 = At end of file.

See Also: Receive Character (page R-13), Get Number of Characters Waiting (page G-101), Set End-Of-Message Terminator (page S-22), Get End-Of-Message Terminator (page G-51), Transfer N Characters (page T-11)

Receive Numeric Table

Communication Action

Function: Moves a specific number of elements from the device or file specified in the communication handle to an integer or float numeric table.

Typical Use: Efficient method of numeric data transfer from one entity to another.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
Length	**Start at Index**	**Of Table**	**Communication Handle**	**Put Status in**
Integer 32 Literal	Integer 32 Literal	Float Table	Communication Handle	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	Integer 32 Table		
		Integer 64 Table		

Standard Example:

Receive Numeric Table

Length	64	*Integer 32 Literal*
Start at Index	0	*Integer 32 Literal*
Of Table	PEER_DATA_TABLE	*Float Table*
Communication Handle	UNIT_2	*Communication Handle*
Put Status in	RECV_STATUS	*Integer 32 Variable*

OptoScript Example:

`ReceiveNumTable(`*Length, Start at Index, Of Table, Communication Handle***`)`**

```
RECV_STATUS = ReceiveNumTable(64, 0, PEER_DATA_TABLE, UNIT_2);
```

This is a function command; it returns one of the status codes listed below.

Note

- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Dependencies:

- Must have previously used Open Outgoing Communication, or (for TCP communication handles) Listen for Incoming Communication and Accept Incoming Communication to accept a session initiated by a TCP/IP peer. See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide* for more information.
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Status Codes:

0 = Success.

-36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-37 = Lock port timeout.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-58 = No data received. Make sure I/O unit has power.

-69 = Invalid parameter (null pointer) passed to command. Make sure communication handle is open.

-76 = At end of file.

Queue Error: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Receive String (page R-19), Receive String Table (page R-21), Receive Pointer Table (page R-17), Transmit Numeric Table (page T-15), Transmit String Table (page T-23), Transmit Pointer Table (page T-16)

Receive Pointer Table

Communication Action

Function: Moves data from the device or file specified in the communication handle into the variables pointed to by a pointer table.

Typical Use: Efficient method of data transfer from one entity to another (for example, two SNAP Ultimate I/O systems), especially when transferring both strings and numbers.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**	**Argument 5**
Length	**Start at Index**	**Of Table**	**Communication Handle**	**Put Status in**
Integer 32 Literal	Integer 32 Literal	Pointer Table	Communication Handle	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable			

Standard Example:

Receive Pointer Table

Length	64	*Integer 32 Literal*
Start at Index	0	*Integer 32 Literal*
Of Table	PEER_DATA_TABLE	*Pointer Table*
Communication Handle	UNIT_2	*Communication Handle*
Put Status in	RECV_STATUS	*Integer 32 Variable*

OptoScript Example:

ReceivePtrTable(*Length, Start at Index, Of Table, Communication Handle***)**

```
RECV_STATUS = ReceivePtrTable(64, 0, PEER_DATA_TABLE, UNIT_2);
```

This is a function command; it returns one of the status codes listed below.

Dependencies:

- Must have previously used Open Outgoing Communication, or (for TCP communication handles) Listen for Incoming Communication and Accept Incoming Communication to accept a session initiated by a TCP/IP peer. See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide* for more information.
- Pointers in the table cannot point to another table.
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Notes:

- Make sure that the tables used on both ends of the communication point to the same types and sizes of data. For example, if you transmit a table with pointers to a float, an integer, and a string with width 10, the table on the receiving end must be exactly the same.

- Check errors using the status codes returned by these commands. If you are using a communication handle (like TCP) that buffers data and you have an error, use the Clear Receive Buffer command to make sure the buffer does not fill up.
- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Status Codes: 0 = Success.

-29 = Wrong object type. Pointers in the table must point to strings, integers, or floats.

-36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-37 = Lock port timeout.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-58 = No data received. Make sure I/O unit has power.

-69 = Invalid parameter (null pointer). Make sure communication handle is open and pointer points to something.

Queue Error: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Receive String (page R-19), Receive String Table (page R-21), Receive Numeric Table (page R-16), Transmit Numeric Table (page T-15), Transmit String Table (page T-23), Transmit Pointer Table (page T-16)

Receive String

Communication Action

Function: Gets the first end-of-message (EOM) character-delimited string from the device or file specified in the communication handle.

Typical Use: To parse data which contains EOM-delimited strings.

Details:

- All characters up to the first EOM are read or moved to the string. The EOM is discarded. If there is no EOM to be received, the control engine waits for the communication variable's timeout period for an EOM to arrive. If no EOM is received within the timeout period, error code -39 is put in the status variable.
- If the EOM-delimited string is longer than the destination string length, a -23 error is returned and as many characters as fit in the destination string are placed there. To see how many characters were received, use a Get Length command for the destination string. The characters remaining, minus the data just received, may be retrieved by a subsequent call to Receive String.

Arguments:

Argument 1	Argument 2	Argument 3
Put in	**Communication Handle**	**Put Status in**
String Variable	Communication Handle	Float Variable
		Integer 32 Variable

Standard Example:

Receive String

Put in	RECV_MSG	*String Variable*
Communication Handle	UNIT_2	*Communication Handle*
Put Status in	RECV_STATUS	*Integer 32 Variable*

OptoScript Example:

ReceiveString(*Put in, Communication Handle***)**

```
RECV_STATUS = ReceiveString(RECV_MSG, UNIT_2);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- This command is not recommended for receiving binary messages, since EOM characters may occur within the binary message. Use Receive N Characters instead.
- The length of the string variable should be a few characters greater than the longest expected string.
- All messages in the Ethernet receive buffer are 16-bit CRC error checked.
- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.
- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Dependencies:

- Must have previously used Open Outgoing Communication, or (for TCP communication handles) Accept Incoming Communication to accept a session initiated by a TCP/IP peer.

- After using Open Outgoing Communication, use the Set End-Of-Message Terminator command to change the EOM from the default of 13 (carriage return) if necessary.
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Status Codes: 0 = Success

-23 = Destination string too short.

-36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-37 = Lock port timeout.

-39 = Timeout on receive.

-44 = String too short.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-57 = String not found. No EOM found.

-58 = No data received. Make sure I/O unit has power.

-69 = Invalid parameter (null pointer) passed to command. Make sure communication handle is open.

See Also: Receive Numeric Table (page R-16), Transmit Numeric Table (page T-15), Transmit String (page T-22), Open Outgoing Communication (page O-4), Set End-Of-Message Terminator (page S-22), Get End-Of-Message Terminator (page G-51)

Receive String Table

Communication Action

Function: Moves a specific number of elements from the device or file specified in the communication handle to a string table.

Typical Use: Efficient method of reading a delimited file into a table.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
Length	**Start at Index**	**Of Table**	**Communication Handle**	**Put Status in**
Integer 32 Literal	Integer 32 Literal	String Table	Communication Handle	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable			

Standard Example:

Receive String Table

Length	64	*Integer 32 Literal*
Start at Index	0	*Integer 32 Literal*
Of Table	PEER_DATA_TABLE	*String Table*
Communication Handle	UNIT_2	*Communication Handle*
Put Status in	RECV_STATUS	*Integer 32 Variable*

OptoScript Example:

ReceiveStrTable(*Length, Start at Index, Of Table, Communication Handle***)**

```
RECV_STATUS = ReceiveStrTable(64, 0, PEER_DATA_TABLE, UNIT_2);
```

This is a function command; it returns one of the status codes listed below.

Note:

- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Dependencies:

- Must have previously used Open Outgoing Communication to establish a session, or (for TCP communication handles) Listen for Incoming Communication and Accept Incoming Communication to accept a session initiated by a TCP/*IP* peer. See "Communication Commands" in Chapter 10 of the *ioControl User's Guide* for more information.
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Status Codes:

0 = Success.

-3 = Buffer overrun or invalid length. Length (Argument 1) is greater than the number of elements in the destination table.

-12 = Invalid table index value—index was negative or greater than or equal to the table size.

-36 = Invalid command or feature not implemented. A firmware upgrade may be required to use this feature on this type of communication handle.

-37 = Lock port timeout.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-58 = No data received. Make sure I/O unit has power.

-59 = Could not receive data. Command may not apply to the type of communication handle used.

-69 = Invalid parameter (null pointer). Make sure communication handle is open.

See Also: Receive String (page R-19), Receive Numeric Table (page R-16), Receive Pointer Table (page R-17), Transmit Numeric Table (page T-15), Transmit String Table (page T-23), Transmit Pointer Table (page T-16), Transfer N Characters (page T-11)

Remove Current Error and Point to Next Error

Error Handling Action

Function: To drop the oldest error from the message queue and bring the next error to the top of the queue.

Typical Use: To access items in the message queue during error handling within the ioControl 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.

Standard Example: Remove Current Error and Point to Next Error

OptoScript Example: **`RemoveCurrentError()`**

```
RemoveCurrentError();
```

This is a procedure command; it does not return a value.

Notes:

- You can use the condition Error? to determine if there are errors in the queue before using this command.
- Use Debug mode to view the message queue for detailed information.

See Also: Error? (page E-19) Get Error Count (page G-53), Get Error Code of Current Error (page G-52), Get Name of Chart Causing Current Error (page G-98), Get Name of I/O Unit Causing Current Error (page G-99)

Retrieve Strategy CRC

Control Engine Action

Function: Returns the 16-bit CRC originally calculated on the program in RAM during the last download.

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
Put in
Integer 32 Variable

Standard Example:

Retrieve Strategy CRC

Put in	ORIGINAL_CRC	*Integer 32 Variable*

OptoScript Example:

`RetrieveStrategyCrc()`

```
ORIGINAL_CRC = RetrieveStrategyCrc();
```

This is a function command; it returns the CRC. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

See Also: Calculate Strategy CRC (page C-3)

Round

Mathematical Action

Function: To round up or down to the nearest integer value.

Typical Use: To discard a fractional part of a number that isn't meaningful while still keeping the number as a float type.

Details: Fractional values less than 0.5 cause no change to the whole number. Fractional values of 0.5 and greater cause the whole number to be incremented by 1.

Arguments:

Argument 1	**Argument 2**
[Value]	**Put Result in**
Float Literal	Float Variable
Float Variable	Integer 32 Variable
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Round

	Boiler_Avg_Temp	*Float Variable*
Put Result in	Boiler_Working_Temp	*Float Variable*

OptoScript Example:

Round (*Value* **)**

```
Boiler_Working_Temp = Round(Boiler_Avg_Temp);
```

This is a function command; it returns the rounded integer value. The returned value can be consumed by a variable (as shown) or by another item, such as a mathematical expression or a control structure. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: Using Move (or an assignment in OptoScript code) to copy a float value to an integer variable will round automatically.

See Also: Truncate (page T-24)

Save Files To Permanent Storage

Control Engine Action

Function: To save the files that are in the root directory of a SNAP Ultimate brain's or SNAP-LCE controller's file system into flash memory.

Typical Use: To avoid losing files if the brain or controller is turned off.

Details:

- All files in the root directory of the file system are saved to its flash memory, replacing files currently in flash memory. (Firmware files, strategy files, and point configuration data are not affected.)
- Files that are not in the root directory but inside folders cannot be saved to flash, nor can folders be saved. To save a file to flash, put it in the root.
- Flash memory in the SNAP Ultimate brain or the SNAP-LCE controller can contain a maximum of 393,216 bytes for file storage. A SNAP-PAC-S controller can contain a maximum of 4 MB. However, each file stored in flash memory requires 72 bytes of overhead.
- **CAUTION:** If you use this command in a strategy, make certain it is not in a loop. You can literally wear out the hardware if you write to flash too many times.

Arguments:

Argument 1
Put Status In
Integer 32 Variable

Standard Example:

Save Files To Permanent Storage

Put Status In	STATUS	*Integer 32 Variable*

OptoScript Example:

`SaveFilesToPermanentStorage()`

```
SaveFilesToPermanentStorage();
```

This is a function command; it always returns a zero.

Notes:

- See "Control Engine Commands" in Chapter 10 of the *ioControl User's Guide*.
- This command always returns a zero. However, the command could fail if files in the root directory of the file system are too large for flash memory, or if there are no files in the root.
- To determine what files are in the file system before using this command and to find out file sizes, you can use ioManager. Follow the instructions in Opto 22 form #1440, the *ioManager User's Guide*.
- To determine the size of a file in the file system, open the file using a File communication handle in read mode, and then use the command Get Number of Characters Waiting. See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.

See Also: Erase Files in Permanent Storage (page E-18), Save Files To Permanent Storage (page S-1), Get Number of Characters Waiting (page G-101)

Send Communication Handle Command

Communication Action

Function: To send a command that accomplishes a specific purpose for the type of communication handle you are using.

Typical Use: To work with files on the SNAP Ultimate brain or SNAP-LCE controller, or to change or specify a remote filename when using an FTP communication handle.

Details: The following commands are available for the communication handles shown:

Comm Handle Type	Commands Available	Description
ftp	dest:<filename>	Used for appending data to an existing file. Specifies the destination (the name of the remote file) on the device (specified in the communication handle) that will be used with a Transfer or Transmit communication handle command, or with the *delete* command (below).
	send:<local filename>,<remote filename>	Sends a whole file to the device specified in the communication handle, where it will have the name indicated. If the local filename already exists, the file is overwritten.
	get:<remote filename>,<local filename>	Retrieves the specified remote file and places it locally under the name indicated. If the local filename already exists, the file is overwritten.
	delete	Removes the file named in the *dest:* command (above) on the remote ftp server. Before using this command, use Open Outgoing Communication first to open the handle, then use the *dest:* command, then use this command.
	dir<:optional directory name>	Retrieves a directory listing for the specified directory name (or the root, if omitted). Returns an integer that indicates the number of entries retrieved. Use commands like Receive String and Receive String Table to read in the listings.
file	delete	Removes the file named in the communication handle and closes the handle. Before using this command, use Open Outgoing Communication first to open the handle.
	getpos	Returns an integer that indicates the current position in the file.
	setpos:<position>	Jumps to the specified position within the file.
	find:<mystring>	(Strings only) Searches for the string within the file and returns its location as an offset from the current position in the file. File must have been opened in r (read) mode.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Communication Handle	**Command**	**Put Status In**
Communication Handle	String Literal	Integer 32 Variable
	String Variable	

Standard Example:

Send Communication Handle Command

Communication Handle	Log_File	*Communication Handle*
Command	delete	*String Literal*
Put Status In	Status_Variable	*Integer 32 Variable*

OptoScript Example:

`SendCommunicationHandleCommand(`*Communication Handle, Command***`)`**

```
Status_Variable = SendCommunicationHandleCommand(Log_File, "delete");
```

This is a function command; it returns one of the status codes listed below. Quotes are required for strings in OptoScript.

Notes: For information on communication handles, see "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = Success.

-11 = Could not send data.

-36 = Feature not implemented (syntax error in command, or command not supported with the type of communication handle in use).

-44 = String too short. (File communication handle) String looked for was empty.

-46 = Invalid string. Check format of command (missing colon, etc.).

-52 = Invalid connection—not opened.

-58 = No data received. If using a file communication handle *find*, make sure file was opened in r (read) mode.

-76 = End of file error. (File communication handle) Didn't find the string you were looking for.

-497 = The remote filename used for an ftp *get* doesn't exist.

See Also: Open Outgoing Communication (page O-4), Get Communication Handle Value (page G-46), Close Communication (page C-29)

Seed Random Number

Mathematical Action

Function: To set a random starting point for the random number generator.

Typical Use:
- To ensure the random number generator does not generate the same sequence of numbers each time it is started.
- To switch random number sequences on-the-fly by "re-seeding" the random number generator.

Details:
- This command seeds the random number generator with a value that should be unique each time the command is issued.
- This command is typically used once at the beginning of a strategy, or occasionally within a strategy. Do not use it too often, as very frequent use could cause the numbers generated to be less random.

Arguments: None.

Standard Example: **Seed Random Number**

OptoScript Example: **`SeedRandomNumber()`**

```
SeedRandomNumber();
```

This is a procedure command; it does not return a value.

See Also: Generate Random Number (page G-6)

Set All Target Address States

I/O Unit Action

Function: To control which target addresses in a redundant system should be enabled on all I/O units.

Typical Use: To control which network is used in a redundant system.

Details:

- A target address is the IP address of an Ethernet interface on an I/O unit.
- In a redundant network architecture, you can assign two target addresses to an I/O unit. In ioControl these are called the Primary Address and the Secondary Address. By default, the Primary Address is used, but the server will switch to the Secondary Address if the primary address is not available.
- Each target address has an *enabled* state and an *active* state. If both target addresses are enabled, they are available to be used. However, only one address can be used at a given time, so there can only be one active address.
- Use Argument 1 to enable one or both addresses.
- Use Argument 2 to disable one or both addresses.
- Use Argument 3 to make one address active.
- Only the last bit of the 32-bit data field is used. Therefore, for arguments 1, 2, and 3 you can use the integers 0, 1, 2, and 3 to indicate the following:
  0=No change
  1=Primary Target Address
  2=Secondary Target Address
  3=Primary and Secondary Target Addresses

Arguments:

Argument 1	Argument 2	Argument 3
Must On Mask	**Must Off Mask**	**Active Mask**
Integer 32 Literal	Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Standard Example: This example assumes that there are redundant networks. It enables the secondary network, disables the primary network, and makes the secondary network active.

Set All Target Address States

Must On Mask	2	*Integer 32 Literal*
Must Off Mask	1	*Integer 32 Literal*
Active Mask	2	*Integer 32 Literal*

OptoScript Example: **`SetAllTargetAddressStates(`***Must-On Mask, Must-Off Mask, Active Mask***`)`**

```
SetAllTargetAddressStates(2, 1, 2);
```

This is a procedure command; it does not return a value.

Notes:

- See "I/O Unit Commands " in Chapter 10 of the *ioControl User's Guide*.
- Arguments 1 and 2 (the Must On Mask and the Must Off Mask) together comprise the enable mask. You can use the enable mask in the following combinations:

To do this:	Must On Mask:	Must Off Mask:
Enable both addresses	3	0
Enable Primary	1	0
Enable Secondary	2	0
Enable *only* Primary	1	2
Enable *only* Secondary	2	1
Disable Primary	0	1
Disable Secondary	0	2
Disable both addresses	0	3

- Argument 3 makes one address active or both addresses inactive as follows:

To do this:	Active Mask:
Make both addresses inactive	0
Activate Primary	1
Activate Secondary	2

- A fully redundant system may also include ioDisplay clients and OptoOPCServers. These commands only deal with the control engine communicating with I/O units. ioDisplay and OptoOPCServer have their own mechanism for controlling their use of the network.

See Also: Set Target Address State (page S-81), Get Target Address State (page G-141)

Set Analog Filter Weight

Analog Point Action

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

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 each time the brain's analog I/O scanner scans the analog point.
- A zero disables filtering. A larger value increases filtering.
- For more information on how analog filter weight works, see Opto 22 form #1440, the *ioManager User's Guide*.

Arguments:

Argument 1	Argument 2
To	**On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Analog Filter Weight

To	FILTER_WEIGHT	*Integer 32 Variable*
On Point	TEMP_IN1	*Analog Input*

OptoScript Example:

`SetAnalogFilterWeight(`*To, On Point***`)`**

```
SetAnalogFilterWeight(FILTER_WEIGHT, TEMP_IN1);
```

This is a procedure command; it does not return a value.

Notes: 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 Debug mode.)

Set Analog Gain

Analog Point Action

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

Typical Uses: To improve calibration on a temperature input, or to rescale an input from one range (for example, 25–50 percent) to a range of 0–100 percent.

Details:

- Always use Set Analog Offset before using this command.
- The default gain value is 1.0. The valid range for gain is any floating point value. A gain of 4.0 will cause a 25 percent input value to read 100 percent (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 **To**	**Argument 2** **On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Analog Gain

To	GAIN_COEFFICIENT	*Float Variable*
On Point	PRESS_IN	*Analog Input*

OptoScript Example:

`SetAnalogGain(`*To, On Point***`)`**

```
SetAnalogGain(GAIN_COEFFICIENT, PRESS_IN);
```

This is a procedure command; it does not return a value.

Notes:

- Instead of using this command, it is recommended that you calibrate inputs when configuring I/O points in ioManager. See Opto 22 form 1440, the *ioManager User's Guide*, for instructions. 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 flash memory at the I/O unit. (You can do so through Debug mode or in ioManager.)

Dependencies: Must use Set Analog Offset first.

See Also: Set Analog Offset (page S-11), Calculate & Set Analog Gain (page C-1)

Set Analog Load Cell Fast Settle Level

Analog Point Action

Function: To set the fast settle level on a SNAP-AILC load cell analog input module.

Typical Use: To get filtered readings faster.

Details:

- Use with the filter weight command (see Set Analog Load Cell Filter Weight) to get filtered readings faster.
- The effects of this command are greater when there are large changes in the load cell output (such as when you first put something heavy on a scale), and a large filter weight is used. Filtered readings are returned noticeably faster.
- Values for the fast settle level range from 0 to 32767. A value of 0 turns the fast settle feature off. Setting the filter weight value to 0, 1, or 32767 also turns this feature off.
- Setting the fast settle level too low causes this feature to start too soon, and results in no reduction in the time it takes to get the filtered value.
- The filtered reading is on channel 2 of the SNAP-AILC module.
- To see how the fast settle level works:
  1. Use the Set Analog Load Cell Filter Weight command to set the filter weight to 255.
  2. Use the Set Analog Load Cell Fast Settle Level command to set the fast settle level to 0 (shuts off fast settle feature).
  3. Use ioDisplay to display Supertrends of the unfiltered channel 1, and the filtered channel 2.
  4. Cause a large change in the load cell output, and observe the difference in the unfiltered and filtered trends. Note the time it takes for the filtered reading to settle.
  5. Now set the fast settle level to 1 and make a large change to the load cell output. This causes fast settling to be applied too soon, and the trend for the filtered reading will show erratic spikes.
  6. As you increase the fast settle level, the trend will smooth out and return readings faster than when the fast settle level is not applied. By experimenting, you will find the ideal fast settle value to use in your application.

Arguments:

Argument 1 **To**	**Argument 2** **On Point**
Integer 32 Literal Integer 32 Variable	Analog Input

Standard Example: This example sets the fast settle level of the analog point to 5.

Set Analog Load Cell Fast Settle Level

To	5	*Integer 32 Literal*
On Point	Load_Cell_A	*Analog Input*

OptoScript Example: **SetAnalogLoadCellFastSettleLevel(***To, On Point***)**

```
SetAnalogLoadCellFastSettleLevel(5, Load_Cell_A);
```

This is a procedure command; it does not return a value.

Notes:
- To ensure that the value will always be correct, store this and other changeable I/O unit values in flash memory at the I/O unit. (You can do so through Debug mode or in ioManager.)

Dependencies: This command is valid only when used on a properly configured SNAP-AILC module.

See Also: Set Analog Load Cell Filter Weight (page S-10)

Set Analog Load Cell Filter Weight

Analog Point Action

Function: To set the filter weight on a SNAP-AILC load cell analog input module.

Typical Use: To smooth load cell input signals that are erratic or change suddenly.

Details:
- Initially, this command copies the current inut value to the filtered value to initialize it. Thereafter, a percentage of the diference between the current input value and the last filtered value is added each time the module scans th load cell point.
- The filter weight range of values is 0 to 255. A 0 or 1 disables filtering. A larger value increases filtering, and the default filter weight value is 128.
- Use with the fast settle level (see Set Analog Load Cell Settle Level) to get the filtered reading faster.
- The filtered reading is on channel 2 of the SNAP-AILC module.

Arguments:

Argument 1	Argument 2
To	**On Point**
Integer 32 Literal	Analog Input
Integer 32 Variable	

Standard Example: This example sets the filter weight to 25.

Set Analog Load Cell Filter Weight

To	25	*Integer 32 Literal*
On Point	Load_Cell_A	*Analog Input*

OptoScript Example: **SetAnalogLoadCellFilterWeight(***To, On Point***)**

```
SetAnalogLoadCellFilterWeight(25, Load_Cell_A);
```

This is a procedure command; it does not return a value.

Notes:
- To ensure that the value will always be correct, store this and other changeable I/O unit values in flash memory at the I/O unit. (You can do so through Debug mode or in ioManager.)
- The filtered weight is reduced when the difference between the adc data and the filtered data is greater than the fast settle level.

Dependencies: This command is valid only when used on a properly configured SNAP-AILC module.

See Also: Set Analog Load Cell Fast Settle Level (page S-9)

Set Analog Offset

Analog Point Action

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 (for example, 25–50 percent) to a range of 0–100 percent.

Details:
- Always use Set Analog Gain after using this command.
- The default offset value is 0.
- An offset of -1,024 will cause a 25 percent input value to read 0 percent (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
To	**On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Analog Offset

To	OFFSET	*Integer 32 Variable*
On Point	PRESS_IN	*Analog Input*

OptoScript Example:

SetAnalogOffset(*To, On Point***)**

```
SetAnalogOffset(OFFSET, PRESS_IN);
```

This is a procedure command; it does not return a value.

Notes:
- Instead of using this command, it is recommended that you calibrate inputs when configuring I/O points in ioManager. See Opto 22 form 1440, the *ioManager User's Guide*, for instructions. 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 flash memory at the I/O unit. (You can do so through Debug mode or in ioManager.)

See Also: Set Analog Gain (page S-8), Calculate & Set Analog Offset (page C-2)

Set Analog Totalizer Rate

Analog Point Action

NOTE: This command is for mistic I/O units only.

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 point 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.
- Totalizing will be bidirectional if the input range is bidirectional, such as -10 to +10.

Arguments:

Argument 1 **To (Seconds)**	**Argument 2** **On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Analog Totalizer Rate

To (Seconds)	TOTALIZE_RATE	*Float Variable*
On Point	FUEL_FLOW	*Analog Input*

OptoScript Example:

SetAnalogTotalizerRate(*To Seconds, On Point***)**

```
SetAnalogTotalizerRate(TOTALIZE_RATE, FUEL_FLOW);
```

This is a procedure command; it does not return a value.

Notes:

- Use Get Analog Totalizer 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 Get & Clear Analog Totalizer Value 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. The following table uses a sampling rate of 1.0 seconds. (For other sample rates, divide these numbers by the sample rate.)

Flow Rate Units	Divisor (Float Literal)
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 0–1,000 liters per minute.

  Get & Clear Analog Totalizer Value

From	FLOW_RATE	*Analog Input*
Put in	TEMP_FLOAT1	*Float Variable*

Divide Temp_Float1

By	60.0	
Put Result in	TEMP_FLOAT1	*Float Variable*

Do Add Temp_Float1

Plus	LITERS	
Put Result in	LITERS	*Float Variable*

See Also: Get Analog Totalizer Value (page G-43), Get & Clear Analog Totalizer Value (page G-17)

Set Analog TPO Period

Analog Point Action

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

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

Details:

- Analog points will not function as TPOs until this command is issued.
- For a SNAP-AOD-29 module, TPO periods are multiples of 0.251 seconds, ranging from 0.251 to 64.25 seconds. If the value entered is not an exact multiple of the period, it is rounded to the nearest period value.
- The time proportion period specifies the total time the output is varied.
- Use Move to set the percent of on time by moving a value from 0–100 to the analog output point.
- Always use 0–100 for the analog TPO scaling.

Arguments:

Argument 1 **To (Seconds)**	**Argument 2** **On Point**
Float Literal	Analog Output
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example: This example sets the period for the TPO point named TPO OUTPUT to 5.02 seconds (the value 5.0 is rounded automatically to the nearest period value, 5.02). If Move is used to set a 50 percent duty cycle (by Moving 50.0 to TPO OUTPUT), then the analog output will repeatedly cycle on for 2.51 seconds and off for 2.51 seconds.

Set Analog TPO Period

To (Seconds)	5.0	*Float Literal*
On Point	TPO_OUTPUT	*Analog Input*

OptoScript Example: **`SetAnalogTpoPeriod(`***To, On Point***`)`**

```
SetAnalogTpoPeriod(5.0, TPO_OUTPUT);
```

This is a procedure command; it does not return a value.

Notes:

- To ensure that the TPO period will always be correct, store this and other changeable I/O unit values in flash memory (EEPROM) at the I/O unit using the Debug mode in ioControl. For more information, see the *ioControl User's Guide*.
- If the TPO period is not stored in permanent memory at the I/O unit, use Set Analog TPO Period immediately before Moving 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 time proportional output module.

Set Communication Handle Value

Communication Action

Function: Sends a string to change the current value of the communication handle.

Typical Use: To set the current communication parameters for a communication handle before using an Open Outgoing Communication command.

Arguments:

Argument 1	**Argument 2**
From	**To**
Communication Handle	Communication Handle
String Literal	
String Variable	

Standard Example:

Set Communication Handle Value

From	tcp:10.22.30.40:22005	*String Literal*
To	COMM_Y	*Communication Handle*

OptoScript Example:

`SetCommunicationHandleValue(`*Value, Communication Handle***`)`**

```
SetCommunicationHandleValue("tcp:10.22.30.40:22005", COMM_Y);
```

This is a procedure command; it does not return a value. Quotes are required for strings in OptoScript.

Notes:

- The example shown above is for outgoing communication using a TCP communication handle. See "Communication Commands" in Chapter 10 of the *ioControl User's Guide* for details about all communication handle types and values.
- If you use a string literal in *Argument 1*, make sure the communication handle type (for example, `tcp`, `ftp`, `file`) is in lowercase letters.
- If the communication handle is currently open, the value will be changed *but will not affect the connection*. To avoid confusion, use Communication Open? before using this command, to determine whether the handle is already open.

See Also: Get Communication Handle Value (page G-46) Open Outgoing Communication (page O-4), Communication Open? (page C-32)

Set Date

Time/Date Action

Function: To set the date in the control engine's real-time clock/calendar to the value contained in a string variable or string literal, using the standard United States format mm/dd/yyyy, where mm = month (01–12), dd = day (01–31), and yyyy = year (2000–2099).

Typical Use: To set the date from an ioControl program.

Details:

- Uses the standard
- If the desired date to set is March 1, 2002, the *To* parameter (*Argument 1*) should contain the string "03/01/2002".
- Executing this command would set the control engine's real-time clock/calendar to March 1, 2002.
- Updates day of week also.
- All erroneous date strings are ignored.

Arguments:

Argument 1
To
String Literal
String Variable

Standard Example:

Set Date		
To	US_DATE_STRING	*String Variable*

OptoScript Example:

SetDate(*To*)

```
SetDate(US_DATE_STRING);
```

This is a procedure command; it does not return a value.

Notes:

- An easier way to update the time and date on the control engine is to click the Sync to PC's Time/Date button when inspecting the control engine in ioControl Debug mode or in ioTerminal.
- To change the date, use an integer variable 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 control engine'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 (DD/MM/YYYY) (page C-59), Copy Date to String (MM/DD/YYYY) (page C-60), Copy Time to String (page C-61)

Set Day

Time/Date Action

Function: To set the day of the month (1 through 31) in the control engine's real-time clock/calendar.

Typical Use: To set the day of the month from an ioControl 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, 2002, the *To* parameter (*Argument 1*) should contain the value 2.
- Executing this command would then set the day of the month in the control engine's real-time clock/calendar.
- Updates day of week also.
- All erroneous day values are ignored.

Arguments:

Argument 1
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Standard Example:

Set Day
To DAY_OF_MONTH *Integer 32 Variable*

OptoScript Example:

SetDay(*To*)

```
SetDay(DAY_OF_MONTH);
```

This is a procedure command; it does not return a value.

Note: Do not issue this command continuously.

See Also: Get Day (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Hours (page S-25), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Set Digital I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use Set I/O Unit from MOMO Masks (page S-29) instead.

Function: To control multiple digital output points 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 points. Affects only selected output points. Does not affect input points.
- Uses only the lowest (least significant) 16 bits of the integer. The least significant bit corresponds to point zero.
- A point is selected for activation by setting the respective bit in the 16-bit data field of argument 1 (the must-on bit mask) to a value of "1." A point is selected for deactivation by setting the respective bit in the 16-bit data field of argument 2 (the must-off bit mask) to a value of "1." Any bits set to a value of 0 in *both* arguments 1 and 2 will leave those points unaffected.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	Argument 2	Argument 3
Must On Mask	**Must Off Mask**	**Digital I/O Unit**
Integer 32 Literal	Integer 32 Literal	B100
Integer 32 Variable	Integer 32 Variable	B3000 (Digital)
		B3000 SNAP Mixed I/O
		G4 Digital Local Simple I/O Unit
		G4D16R
		G4D32RS
		SNAP-BRS

Standard Example:

Set Digital I/O Unit from MOMO Masks

Must On Mask	PUMPS_ON_MASK	*Integer 32 Variable*
Must Off Mask	3840	*Integer 32 Literal*
Digital I/O Unit	PUMP_CTRL	*B3000 (Digital)*

The effect of this command is illustrated below:

	Point Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
	Hex	0				0				F				0			
Must-off Bit Mask	Binary	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
	Hex	0				F				0				0			

In this example, points 4, 5, 6, and 7 will be turned on. Points 8, 9, 10, and 11 will be turned off. Points 0, 1, 2, 3, 12, 13, 14, and 15 are not changed.

OptoScript Example: **`SetDigitalIoUnitFromMomo(`***Must-On Mask, Must-Off Mask, Digital I/O Unit***`)`**

```
SetDigitalIoUnitFromMomo(PUMPS_ON_MASK, 3840, PUMP_CTRL);
```

This is a procedure command; it does not return a value.

Notes:

- For a 64-point digital-only rack, use the command Set Digital-64 I/O Unit from MOMO Masks.
- Use Bit Set or Bit Clear to change individual bits in an integer variable.

Set Digital-64 I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use Set I/O Unit from MOMO Masks (page S-29) instead.

Function: To control multiple digital output points on the same 64-point digital-only I/O unit simultaneously with a single command.

Typical Use: To efficiently control all digital outputs on a 64-point digital rack with one command.

Details:

- This command is 64 times faster than using Turn On or Turn Off 64 times.
- Updates the IVALs and XVALs for all 64 points. Affects only selected output points. Does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2. (Check for conflicts; if the same bit is set to 1 in both masks, the point is turned off.)
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Must On Mask	**Must Off Mask**	**Digital-64 I/O Unit**
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64 Unit
Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64 Unit
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

Set Digital-64 I/O Unit from MOMO Masks

Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Digital-64 I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-D64*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`SetDigital64IoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Digital-64 I/O Unit***`)`**

```
SetDigital64IoUnitFromMomo(0x060003C0000000C2i64, 0xB0F240010308A020i64,
PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value. (Note that Integer 64 literals in OptoScript code take an `i64` suffix.)

Notes: Use Bit Set or Bit Clear to change individual bits in an integer variable.

See Also: Get I/O Unit as Binary Value (page G-65)

Set Down Timer Preset Value

Timing Action

Function: To set the value from which a down timer counts down.

Typical Use: To initialize a down timer.

Details:

- This command sets the value from which a down timer counts down, but it *does not start the timer*. To start the timer counting down, use the command Start Timer.
- The preset value will be persistent between calls to Start Timer.
- *Argument 1* must be a positive number in seconds.

Arguments:

Argument 1	**Argument 2**
Target Value	**Down Timer**
Float Literal	Down Timer Variable
Float Variable	

Standard Example:

Set Down Timer Preset Value

Target Value	60.0	*Float Literal*
Down Timer	OVEN_TIMER	*Down Timer Variable*

OptoScript Example:

`SetDownTimerPreset(`*Target Value, Down Timer***`)`**

```
SetDownTimerPreset(60.0, OVEN_TIMER);
```

This is a procedure command; it does not return a value.

Notes:

- See “Timing Commands” in Chapter 10 of the *ioControl User’s Guide* for more information on using timers.
- To set the preset value and start the timer in one step, use the Move command to move the preset value to the timer. The timer will immediately start counting down from the value moved to it. Using Move overwrites any preset value previously set, so subsequent Start Timer commands will start from the value most recently moved.

See Also: Start Off-Pulse (page S-96), Stop Timer (page S-102), Continue Timer (page C-39), Pause Timer (page P-1), Down Timer Expired? (page D-21)

Set End-Of-Message Terminator

Communication Action

Function: To set the end-of-message (EOM) character for a specific communication handle.

Typical Use: To parse delimited strings when using one of the following commands: Receive String, Receive String Table, Transmit/Receive String, Transmit String, Transmit String Table.

Details:

- The communication handle must already be opened for this command to take effect. Use the command Open Outgoing Communication to open the handle.
- The character is represented by an ASCII value (see the ASCII table under "String Commands" in Chapter 10 of the *ioControl User's Guide*). For example, a space is a character 32 and a "1" is a character 49. Commonly used delimiters include a comma (character 44) and a colon (character 58).
- The default EOM is 13 (carriage return).

Arguments:

Argument 1	**Argument 2**
Communication Handle	**To Character**
Communication Handle	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set End-Of-Message Terminator

Communication Handle	UIO_A	*Communication Handle*
To Character	EOM_Term	*Integer 32 Variable*

OptoScript Example:

`SetEndOfMessageTerminator(`*Communication Handle, To Character***`)`**

```
SetEndOfMessageTerminator(UIO_A, EOM_Term);
```

This is a procedure command; it does not return a value.

Queue Errors: -52 = Invalid connection—not opened.

See Also: Get End-Of-Message Terminator (page G-51), Open Outgoing Communication (page O-4), Receive String Table (page R-21), Transmit String Table (page T-23)

Set HDD Module from MOMO Masks

High Density Digital Module Action

Function: To control multiple points on the same high-density digital output module simultaneously with a single command.

Typical Use: To efficiently control multiple digital outputs on one module with one command.

Details:

- If setting all 32 points, this command is about 32 times faster than using Turn On HDD Module Point or Turn Off HDD Module Point 32 times.
- To turn on a point, set the respective bit in the 32-bit data field of argument 1 (the must-on bit mask) to a value of 1. To turn off a point, set the respective bit in the 32-bit data field of argument 2 (the must-off bit mask) to a value of 1. To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2. (Check for conflicts; if the same bit is set to 1 in both masks, the point is turned off.)
- The least significant bit corresponds to point zero.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**	**Argument 5**
I/O Unit	**Module Number**	**Must On Mask**	**Must Off Mask**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable	Integer 32 Variable	Integer 32 Variable	
SNAP-UP1-ADS				
SNAP-UP1-M64				
SNAP-ENET-S64				
SNAP-PAC-R1				
SNAP-PAC-R2				

Standard Example:

Set HDD Module from MOMO Masks

I/O Unit	Bldg_A	*SNAP-UP1-M64*
Module Number	3	*Integer 32 Literal*
Must On Mask	0x060000C2	*Integer 32 Variable*
Must Off Mask	0xB0000020	*Integer 32 Literal*
Put Status In	Status_Code	*SNAP-UP1-ADS*

The effect of this command is illustrated below:

	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight and the last eight digital points on the rack. For the points shown, points 26, 25, 7, 6, and 1 will be turned on. Points 31, 29, 28, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`SetHddModuleFromMomo(`*I/O Unit, Module Number, Must-On Mask, Must-Off Mask***`)`**

```
Status_Code = SetHDDModuleFromMomo(Bldg_A, 3, 0x060000C2, 0xB0000020);
```

This is a function command; it returns one of the status codes shown below.

Status Codes: 0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Turn On HDD Module Point (page T-28), Turn Off HDD Module Point (page T-26)

Set Hours

Time/Date Action

Function: To set the hours value (0 through 23) in the control engine's real-time clock/calendar.

Typical Use: To set the hours value from an ioControl 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 control engine's real-time clock/calendar.
- The control engine'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
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Standard Example:

Set Hours		
To	HOURS	Integer 32 Variable

OptoScript Example:

SetHours(*To*)

```
SetHours(HOURS);
```

This is a procedure command; it does not return a value.

Note: Do not issue this command continuously.

See Also: Get Day (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Day (page S-17), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Set I/O Unit Event Message State

I/O Unit—Event Message Action

Function: To activate or deactivate a SNAP Ultimate or Ethernet I/O unit event message, or to acknowledge an SNMP message.

Typical Use: To send an e-mail, SNMP, or other kind of event message.

Details:

- Use ioManager to configure the types, intervals, and text of event messages. You can configure up to 128 messages for each I/O unit.
- To start sending the message as it is configured, set the state to 1 = Active.
- SNMP messages must be acknowledged in order to inactivate them. To do so, set the state to 2 = Acknowledged.
- To stop sending the message or return it to a non-triggered state, set it to 0 = Inactive. A delay is not needed between activating and inactivating the message, as the commands are put into a queue and processed in order.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Event Message Number**	Argument 3 **State**	Argument 4 **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Literal	Integer 32 Vairable
SNAP-UP1-D64	Integer 32 Vairable	Integer 32 Vairable	
SNAP-UP1-M64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Set I/O Unit Event Message State

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Event Message Number	5	*Integer 32 Literal*
State	1	*Integer 32 Literal*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

`SetIoUnitEventMsgState(`*I/O Unit, Event Message Number, State***`)`**

```
Status = SetIoUnitEventMsgState(UIO_A, 5, 1);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- See "I/O Unit—Event Message Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use Get I/O Unit Event Message State to check the current state of the message, for example, to see if the message is already active before activating it.
- If you are using one event message for several situations, use Set I/O Unit Event Message Text to change the text of the message being sent.

Status Codes: 0 = success

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Event Message Text (page S-27), Get I/O Unit Event Message State (page G-67), Get I/O Unit Event Message Text (page G-68)

Set I/O Unit Event Message Text

I/O Unit—Event Message Action

Function: To change the text of an event message on a SNAP Ultimate or Ethernet I/O unit.

Typical Use: To "recycle" a message if all 128 messages on an I/O unit are already used, to create dynamic message content.

Details:

- Use ioManager to configure the types, intervals, and text of event messages. You can configure up to 128 messages for each I/O unit.
- Use caution with this command. Change text only when necessary, and use Get I/O Unit Event Message State to check the state of the message before changing it.

Arguments:

Argument 1 **I/O Unit**	**Argument 2** **Event Message Number**	**Argument 3** **Message Text**	**Argument 4** **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	String Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	String Variable	
SNAP-UP1-M64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Set I/O Unit Event Message Text

I/O Unit	UIO_A	*SNAP-UP1-ADS*
Event Message Number	5	*Integer 32 Literal*
Message Text	Machine failure	*String Literal*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`SetIoUnitEventMsgText(`*I/O Unit, Event Message Number, Message Text***`)`**

```
STATUS = SetIoUnitEventMsgText(UIO_A, 5, "Machine failure");
```

This is a function command; it returns one of the status codes listed below. Note that quotes must be used for strings in OptoScript.

Notes:

- See "I/O Unit—Event Message Commands" in Chapter 10 of the *ioControl User's Guide*.

- This command should be used when all 128 messages are already in use. If you need to use the same message with different text, it is best to double up on messages that are mutually exclusive, for example, “Tank level too high” and “Tank level too low”.
- This command can also be used to create dynamic message content, for example to send a message reporting a changing pressure level.
- Before using this command, check the current state of the message using Get I/O Unit Event Message State, to avoid sending the wrong message.
- Message text is limited to 127 characters. If it is longer than 127 characters, the first 127 characters are sent and an error -23 is returned.

Status Codes: 0 = success

-12 = Invalid index. Event message number is less than 0 or greater than 127.

-23 = Destination string too short. Message text is longer than 127 characters. The first 127 characters are sent.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Set I/O Unit Event Message State (page S-26), Get I/O Unit Event Message State (page G-67), Get I/O Unit Event Message Text (page G-68)

Set I/O Unit from MOMO Masks

I/O Unit Action

Function: To control multiple digital output points 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:

- Updates the IVALs and XVALs for all selected output points. Does not affect input points. Does not affect analog points in any position on the rack.
- To turn on a point, set the respective bit in the data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in both arguments 1 and 2. (Check for conflicts; if the same bit is set to 1 in both masks, the point is turned off.)
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Must On Mask	**Must Off Mask**	**Digital I/O Unit**
Integer 32 Literal	Integer 32 Literal	B100
Integer 32 Variable	Integer 32 Variable	B3000 (Digital)
Integer 64 Literal	Integer 64 Literal	B3000 SNAP Mixed I/O
Integer 64 Variable	Integer 64 Variable	G4 Digital Local Simple I/O Unit
		G4D16R
		G4D32RS
		SNAP-BRS
		SNAP-B3000-ENET, SNAP-ENET-RTC
		SNAP-ENET-S64
		SNAP-PAC-R1
		SNAP-PAC-R2
		SNAP-UP1-ADS
		SNAP-UP1-M64

Standard Example:

Set Digital I/O Unit from MOMO Masks

Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Digital I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-M64*

The effect of this command is illustrated below::

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`SetIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Digital I/O Unit***`)`**

```
SetIoUnitFromMomo(0x060003C0000000C2i64, 0xB0F240010308A020i64,
PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value.

Notes:

- Use Bit Set or Bit Clear to change individual bits in an integer variable.

Set I/O Unit Scratch Pad Bits from MOMO Mask

I/O Unit—Scratch Pad Action

Function: To write bits to the Scratch Pad area of a local or remote SNAP Ultimate or Ethernet brain.

Typical Use: For peer-to-peer communication. Strategy data can be stored in the Scratch Pad area and retrieved by a peer on the network.

Details:

- Use this command to store the data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad Bits to retrieve it.
- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Must-on Mask**	Argument 3 **Must-off Mask**	Argument 4 **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	Integer 32 Variable	
SNAP-UP1-M64	Integer 64 Literal	Integer 64 Literal	
SNAP-B3000-ENET,	Integer 64 Variable	Integer 64 Variable	
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Set I/O Unit Scratch Pad Bits from MOMO Mask

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Must-on Mask	MyOnMask	*Integer 64 Variable*
Must-off Mask	MyOffMask	*Integer 64 Variable*
Put Status in	Status	*Integer 32Variable*

OptoScript Example:

`SetIoUnitScratchPadBitsFromMomo(`*I/O Unit, Must-on Mask, Must-off Mask***`)`**

```
Status = SetIoUnitScratchPadBitsFromMomo(UIO_B, MyOnMask, MyOffMask);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- It is best to use 64-bit values for *Argument 2* and *Argument 3*. ioControl and OptoScript will convert a 32-bit value to 64 bits and then use the 64-bit value. Because both integer 32 and integer 64 values are signed integers, an integer 32 value of 0xAAAAAAAA will be converted to 0xFFFFFFFFAAAAAAAA.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = success

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Scratch Pad Bits (page G-69), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Float Table (page S-34), Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad String Table (page S-41)

Set I/O Unit Scratch Pad Float Element

I/O Unit—Scratch Pad Action

Function: To write a float to the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use this command to store the variable data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad Float Element or Get I/O Unit Scratch Pad Float Table to retrieve it.
- The float area of the Scratch Pad is a table containing 10,240 elements (index numbers 0–10239). Enter the index number of the element you want to set in *Argument 2*.

Arguments:

Argument 1 **I/O Unit**	**Argument 2** **Index**	**Argument 3** **From**	**Argument 4** **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Float Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	Float Variable	
SNAP-UP1-M64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Set I/O Unit Scratch Pad Float Element

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Index	26	*Integer 32 Literal*
From	1.2	*Float Literal*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

SetIoUnitScratchPadFloatElement(*I/O Unit, Index, From***)**

```
Status = SetIoUnitScratchPadFloatElement(UIO_B, 26, 1.2);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To write more than one float value to the Scratch Pad area in a single command, use Set I/O Unit Scratch Pad Float Table.

- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Scratch Pad Float Element (page G-70), Set I/O Unit Scratch Pad Float Table (page S-34), Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad String Table (page S-41), Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31)

Set I/O Unit Scratch Pad Float Table

I/O Unit—Scratch Pad Action

Function: To write a series of float values to the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use this command to place variable data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad Float Element or Get I/O Unit Scratch Pad Float Table to retrieve it.
- The float area of the Scratch Pad is a table containing 10,240 elements (index numbers 0–10239). Enter the number of elements you want to set in the Scratch Pad area in *Argument 2* and the index number of the starting element in *Argument 3*. In *Argument 4* enter the starting index of the table you are writing from; in *Argument 5* enter the name of the table.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Length**	Argument 3 **To Index**	Argument 4 **From Index**
SNAP-ENET-D64 SNAP-UP1-D64 SNAP-UP1-M64 SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable

Argument 5 **From Table**	Argument 6 **Put Status in**
Float Table	Integer 32 Variable

Standard Example:

Set I/O Unit Scratch Pad Float Table

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Length	64	*Integer 32 Literal*
To Index	0	*Integer 32 Literal*
From Index	0	*Integer 32 Literal*
From Table	MyFloatTable	*Float Table*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

`SetIoUnitScratchPadFloatTable(`*I/O Unit, Length, To Index, From Index, From Table***`)`**

```
Status = SetIoUnitScratchPadFloatTable(UIO_B, 64, 0, 0, MyFloatTable);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To write a single float value to the Scratch Pad area, use Set I/O Unit Scratch Pad Float Element.

- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = success

-3 = Invalid length. *Argument 3* (Length) less than 0 or greater than 3072.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get I/O Unit Scratch Pad Float Table (page G-72), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31), Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad String Table (page S-41)

Set I/O Unit Scratch Pad Integer 32 Element

I/O Unit—Scratch Pad Action

Function: To write an integer 32 to the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use this command to store the variable data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad Integer 32 Element to retrieve it.
- The integer 32 area of the Scratch Pad is a table containing 10,240 elements (index numbers 0–10239). Enter the index number of the element you want to set in *Argument 2*.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
I/O Unit	**Index**	**From**	**Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	Integer 32 Variable	
SNAP-UP1-M64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Set I/O Unit Scratch Pad Integer 32 Element

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Index	26	*Integer 32 Literal*
From	99	*Integer 32 Literal*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

SetIoUnitScratchPadInt32Element(*I/O Unit, Index, From***)**

```
Status = SetIoUnitScratchPadInt32Element(UIO_B, 26, 99);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To write more than one integer 32 value to the Scratch Pad area in a single command, use Set I/O Unit Scratch Pad Integer 32 Table.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Scratch Pad Integer 32 Element (page G-74), Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Float Table (page S-34), , Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad String Table (page S-41), Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31)

Set I/O Unit Scratch Pad Integer 32 Table

I/O Unit—Scratch Pad Action

Function: To write a series of integer 32 values to the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use this command to store the variable data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad Integer 32 Element or Get I/O Unit Scratch Pad Integer 32 Table to retrieve it.
- The integer 32 area of the Scratch Pad is a table containing 10,240 elements (index numbers 0–10239). Enter the number of elements you want to set in *Argument 2* and the index number of the starting element in *Argument 3*.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Length**	Argument 3 **To Index**	Argument 4 **From Index**
SNAP-ENET-D64	Integer 32 Literal	Integer 32 Literal	Integer 32 Literal
SNAP-UP1-D64	Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
SNAP-UP1-M64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Argument 5 **From Table**	Argument 6 **Put Status in**
Integer 32 Table	Integer 32 Variable

Standard Example:

Set I/O Unit Scratch Pad Integer 32 Table

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Length	64	*Integer 32 Literal*
To Index	0	*Integer 32 Literal*
From Index	0	*Integer 32 Literal*
From Table	MyInt32Table	*Integer 32 Table*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

`SetIoUnitScratchPadInt32Table(`*I/O Unit, Length, To Index, From Index, From Table***`)`**

```
Status = SetIoUnitScratchPadInt32Table(UIO_B, 64, 0, 0, MyInt32Table);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To write a single integer 32 value to the Scratch Pad area, use Set I/O Unit Scratch Pad Integer 32 Element.

- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-3 = Invalid length. *Argument 3* (Length) less than 0 or greater than 3072.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Scratch Pad Integer 32 Table (page G-76), Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Float Table (page S-34), , Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad String Table (page S-41), Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31)

Set I/O Unit Scratch Pad String Element

I/O Unit—Scratch Pad Action

Function: To write a string to the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use this command to store the variable data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad String Element to retrieve it.
- The string area of the Scratch Pad is a table containing 64 elements (index numbers 0–63). Enter the index number of the element you want to set in *Argument 2*.
- Each string element can hold 128 characters or 128 bytes of binary data.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Index**	**From**	**Put Status in**
SNAP-ENET-D64	Integer 32 Literal	String Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	String Variable	
SNAP-UP1-M64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Set I/O Unit Scratch Pad String Element

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Index	26	*Integer 32 Literal*
From	MyStringVar	*String Variable*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

SetIoUnitScratchPadStringElement(*I/O Unit, Index, From***)**

```
Status = SetIoUnitScratchPadStringElement(UIO_B, 26, MyStringVar);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To write more than one string value to the Scratch Pad area in a single command, use Set I/O Unit Scratch Pad String Table.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes: 0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Get I/O Unit Scratch Pad String Element (page G-78), Set I/O Unit Scratch Pad String Table (page S-41), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Float Table (page S-34), Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31)

Set I/O Unit Scratch Pad String Table

I/O Unit—Scratch Pad Action

Function: To write series of strings to the Scratch Pad area of a local or remote SNAP Ultimate brain.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the brain's Scratch Pad area and retrieved by a peer on the network.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- You can use this command to store the variable data in the Scratch Pad area, and then use Get I/O Unit Scratch Pad String Table to retrieve it.
- The string area of the Scratch Pad is a table containing 64 elements (index numbers 0–63). Enter the number of elements you want to set in *Argument 2* and the index number of the starting element in *Argument 3*.
- Each string element can hold 128 characters or 128 bytes of binary data.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Length**	Argument 3 **To Index**	Argument 4 **From Index**
SNAP-ENET-D64 SNAP-UP1-D64 SNAP-UP1-M64 SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable

Argument 5 **From Table**	Argument 6 **Put Status in**
String Table	Integer 32 Variable

Standard Example:

Set I/O Unit Scratch Pad String Table

I/O Unit	UIO_B	*SNAP-UP1-ADS*
Length	8	*Integer 32 Literal*
To Index	0	*Integer 32 Literal*
From Index	0	*Integer 32 Literal*
From Table	MyStringTable	*String Table*
Put Status in	Status	*Integer 32 Variable*

OptoScript Example:

SetIoUnitScratchPadStringTable(*I/O Unit, Length, To Index, From Index, From Table***)**

```
Status = SetIoUnitScratchPadStringTable(UIO_B, 8, 0, 0, MyStringTable);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- To write a single string value to the Scratch Pad area, use Set I/O Unit Scratch Pad String Element.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another Ultimate I/O unit or an application running on a PC) that can connect to the I/O unit's command processor port (usually port 2001). Be aware of all devices that have access to the area, and make sure that their reads and writes are synchronized so that correct data is available to all devices when needed.
- Since this command accesses a table on an I/O unit, it requires communication to that unit, so it will take more time than just moving data between tables in a strategy.
- See "I/O Unit—Scratch Pad Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-58 = No data received. I/O unit may be turned off or unreachable.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Get I/O Unit Scratch Pad String Table (page G-80), Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Float Table (page S-34), Set I/O Unit Scratch Pad Integer 32 Element (page S-36), Set I/O Unit Scratch Pad Integer 32 Table (page S-38), Set I/O Unit Scratch Pad Bits from MOMO Mask (page S-31)

Set Minutes

Time/Date Action

Function: To set the minutes value (0 through 59) in the control engine's real-time clock/calendar.

Typical Use: To set the minutes value from an ioControl program.

Detail:

- 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 control engine's real-time clock/calendar.
- The control engine'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
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Standard Example:

Set Minutes

To	MINUTES	*Integer 32 Variable*

OptoScript Example:

`SetMinutes(`*`To`*`)`

```
SetMinutes(MINUTES);
```

This is a procedure command; it does not return a value.

Note: Do not issue this command continuously.

See Also: Get Day (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Hours (page S-25), Set Day (page S-17), Set Month (page S-57), Set Seconds (page S-78), Set Year (page S-89)

Set Mistic PID Control Word

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

Function: Change the bits that control the PID operation.

Typical Use: To alter the PID configuration.

Details:

- Bit assignments:

11 1 = Use SqRt value from input point.
10 1 = Setpoint was above high clamp. Write zero to clear.
9 1 = Setpoint was below low clamp. Write zero to clear.
8 1 = Input point under-range. Write zero to clear.
7 1 = Loop active. 0 = Loop stopped.
6 1 = Loop in auto mode. 0 = Loop in manual mode.
5 1 = Output active. 0 = Output disconnected.
4 1 = Output tracks input in manual mode. 0 = no action.
3 1 = Setpoint tracks input in manual mode. 0 = no action.
2 1 = Input from host. 0 = Input from point.
1 1 = Setpoint from point. 0 = Setpoint from host.
0 1 = Use filtered value from input point. Must have filtering active on the input point.
0 = Use current value of input point.

- To set any bit(s) put a 1 for each bit to set in the On Mask parameter. To clear any bit(s) put a 1 for each bit to clear in the Off Mask parameter. All mask bit positions with zeros will leave the corresponding PID control word bit unchanged.

Arguments:

Argument 1	Argument 2	Argument 3
On Mask	**Off Mask**	**For PID Loop**
Integer 32 Literal	Integer 32 Literal	PID Loop
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Set Mistic PID Control Word

On Mask	PID_CTRL_SET	*Integer 32 Variable*
Off Mask	PID_CTRL_CLEAR	*Integer 32 Variable*
For PID Loop	EXTRUDER_ZONE08	*PID Loop*

OptoScript Example:

SetMisticPidControlWord(*On-Mask, Off-Mask, For PID Loop***)**

```
SetMisticPidControlWord(PID_CTRL_SET, PID_CTRL_CLEAR, EXTRUDER_ZONE08);
```

This is a procedure command; it does not return a value.

Note: The PID Control Word is actually a 16-bit number. The four most significant bits are reserved.

See Also: Get Mistic PID Control Word (page G-87)

Set Mistic PID D Term

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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 **To**	**Argument 2** **On PID Loop**
Float Literal Float Variable Integer 32 Literal Integer 32 Variable	PID Loop

Standard Example:

Set Mistic PID D Term

To	D_TERM_VALUE	*Float Variable*
On PID Loop	HEATER_3	*PID Loop*

OptoScript Example:

`SetMisticPidDTerm(`*To, On PID Loop***`)`**

```
SetMisticPidDTerm(D_TERM_VALUE, HEATER_3);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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.

See Also: Enable Communication to Mistic PID Loop (page E-5)

Set Mistic PID I Term

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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
To	**On PID Loop**
Float Literal	PID Loop
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Mistic PID I Term

To	I_TERM_VALUE	*Float Variable*
On PID Loop	HEATER_3	*PID Loop*

OptoScript Example:

SetMisticPidITerm(*To, On PID Loop***)**

```
SetMisticPidITerm(I_TERM_VALUE, HEATER_3);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- 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.

See Also: Enable Communication to Mistic PID Loop (page E-5)

Set Mistic PID Input

PID—Mistic Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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 point 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 0.8 seconds to 1.0 seconds should be adequate).

Arguments:

Argument 1	Argument 2
PID Loop	**Input**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set Mistic PID Input

PID Loop	HEATER_3	*PID Loop*
Input	PID_INPUT_VALUE	*Float Variable*

OptoScript Example:

SetMisticPidInput(*PID Loop, Input***)**

```
SetMisticPidInput(HEATER_3, PID_INPUT_VALUE);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- Do not send the input value to the PID less frequently than the PID scan rate, as it will adversely affect the PID performance.

Dependencies:

- You 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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Get Mistic PID Input (page G-90)

Set Mistic PID Mode to Auto

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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
On PID Loop
PID Loop

Standard Example:

Set Mistic PID Mode to Auto

On PID Loop	HEATER_3	*PID Loop*

OptoScript Example:

`SetMisticPidModeToAuto(`*On PID Loop***`)`**

```
SetMisticPidModeToAuto(HEATER_3);
```

This is a procedure command; it does not return a value.

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, Debug mode, or ioDisplay to write directly to the PID output analog point. 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 presetting 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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Set Mistic PID Mode to Manual (page S-49)

Set Mistic PID Mode to Manual

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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, wait at least 10 milliseconds; then use the Move command, Debug mode, or ioDisplay to write directly to the PID output analog point. 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 Opto 22 Product Support.

Arguments:

Argument 1
On PID Loop
PID Loop

Standard Example:

Set Mistic PID Mode to Manual

On PID Loop	HEATER_3	*PID Loop*

OptoScript Example:

SetMisticPidModeToManual(*On PID Loop*)

```
SetMisticPidModeToManual(HEATER_3);
```

This is a procedure command; it does not return a value.

Notes: Use Get PID Setpoint first to save the PID setpoint to a float variable.

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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Set Mistic PID Mode to Auto (page S-48)

Set Mistic PID Output Rate of Change

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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 point.
- The default value is the span of the output point. This allows the PID output to move as much as 100 percent per scan period. For example, if the PID output point is 4–20 mA, 16.00 would be returned by default, representing 100 percent of the span.

Arguments:

Argument 1	Argument 2
To	**On PID Loop**
Float Literal	PID Loop
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Mistic PID Output Rate of Change

To	PID_RATE_LIMIT	*Float Variable*
On PID Loop	HEATER_3	*PID Loop*

OptoScript Example:

`SetMisticPidOutputRateOfChange(`*To, On PID Loop***`)`**

```
SetMisticPidOutputRateOfChange(PID_RATE_LIMIT, HEATER_3);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Tune the loop before reducing the output rate-of-change.
- Set the output rate-of-change back to 100 percent before retuning the PID.
- Many additional PID loop control features are available. See the *Mistic Analog and Digital Commands Manual* (Opto 22 form 270).

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.

See Also: Enable Communication to Mistic PID Loop (page E-5), Get Mistic PID Output Rate of Change (page G-93), Set Mistic PID Scan Rate (page S-52)

Set Mistic PID P Term

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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 **To**	**Argument 2** **On PID Loop**
Float Literal Float Variable Integer 32 Literal Integer 32 Variable	PID Loop

Standard Example:

Set Mistic PID P Term

To	GAIN	*Float Variable*
On PID Loop	HEATER_3	*PID Loop*

OptoScript Example:

`SetMisticPidPTerm(` *To, On PID Loop* **`)`**

```
SetMisticPidPTerm(GAIN, HEATER_3);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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.

See Also: Enable Communication to Mistic PID Loop (page E-5)

Set Mistic PID Scan Rate

PID—Mistic Action

NOTE: This command is not for use with SNAP Ethernet I/O units or SNAP-PID-V modules.

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 **To**	Argument 2 **On PID Loop**
Float Literal	PID Loop
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set Mistic PID Scan Rate

To	Scan_Rate	*Float Variable*
On PID Loop	Heater_3	*PID Loop*

OptoScript Example:

SetMisticPidScanRate(*To, On PID Loop***)**

```
SetMisticPidScanRate(Scan_Rate, Heater_3);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- 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.

See Also: Enable Communication to Mistic PID Loop (page E-5)

Set Mistic PID Setpoint

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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:
- To use this command, the setpoint must be configured to come from Host.
- The value you send has the same engineering units as the PID input.
- The setpoint can be an analog point, even from another I/O unit.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Setpoint**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set Mistic PID Setpoint

PID Loop	Heater_3	*PID Loop*
Setpoint	Pid_Setpoint_Value	*Float Variable*

OptoScript Example:

`SetMisticPidSetpoint(`*PID Loop, Setpoint***`)`**

```
SetMisticPidSetpoint(Heater_3, PID_Setpoint_Value);
```

This is a procedure command; it does not return a value.

Notes:
- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Send a new setpoint value to the PID only when necessary.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to Mistic PID Loop (page E-5), Get Mistic PID Setpoint (page G-96)

Set Mixed 64 I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use *Set I/O Unit from MOMO Masks (page S-29)* *instead.*

Function: To control multiple digital output points on the same 64-point mixed I/O unit simultaneously with a single command (applies to I/O units with a SNAP-UP1-M64 brain only).

Typical Use: To efficiently control all digital outputs on a mixed 64-point rack with one command.

Details:

- This command is 64 times faster than using Turn On or Turn Off 64 times.
- Updates the IVALs and XVALs for all 64 points. Affects only selected output points. Does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2. (Check for conflicts; if the same bit is set to 1 in both masks, the point is turned off.)
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Must On Mask	**Must Off Mask**	**Mixed 64 I/O Unit**
Integer 32 Literal	Integer 32 Literal	SNAP-UP1-M64
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

Set Mixed 64 I/O Unit from MOMO Masks

Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Mixed 64 I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-M64*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:	**`SetMixed64IoUnitFromMomo(`***Must-On Mask, Must-Off Mask, Mixed 64 I/O Unit***`)`**

```
SetMixed64IoUnitFromMomo(0x060003C0000000C2i64, 0xB0F240010308A020i64,
PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value. (Note that Integer 64 literals in OptoScript code take an `i64` suffix.)

Notes: Use Bit Set or Bit Clear to change individual bits in an integer variable.

See Also: Get I/O Unit as Binary Value (page G-65)

Set Mixed I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use Set I/O Unit from MOMO Masks (page S-29) instead.

Function: To control multiple digital output points on the same mixed I/O unit simultaneously with a single command.

Typical Use: To efficiently control all digital outputs on a mixed I/O rack with one command.

Details:

- This command is 32 times faster than using Turn On or Turn Off 32 times.
- Updates the IVALs and XVALs for all 32 digital points. Affects only selected digital output points. Does not affect digital input points. Does not affect analog points in any position on the rack.
- To turn on a point, set the respective bit in the 32-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 32-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2. (Check for conflicts; if the same bit is set to 1 in both masks, the point is turned off.)
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	Argument 2	Argument 3
Must On Mask	**Must Off Mask**	**Mixed I/O Unit**
Integer 32 Literal	Integer 32 Literal	SNAP-B3000-ENET,
Integer 32 Variable	Integer 32 Variable	SNAP-ENET-RTC
		SNAP-UP1-ADS

Standard Example:

Set Mixed I/O Unit from MOMO Masks

Must On Mask	0x0600C0C2	*Integer 32 Variable*
Must Off Mask	0xB001A020	*Integer 32 Literal*
Mixed I/O Unit	PUMP_CTRL_UNIT	*SNAP-UP1-ADS*

The effect of this command is illustrated below:

	Point Number	31	30	29	28	27	26	25	24		7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight and the last eight digital points on the rack. For the points shown, points 26, 25, 7, 6, and 1 will be turned on. Points 31, 29, 28, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`SetMixedIoUnitFromMomo(`*Must-On Mask, Must-Off Mask, Mixed I/O Unit***`)`**

```
SetMixedIoUnitFromMomo(PUMPS_ON_MASK, 0xB001A020, PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value.

Notes: Use Bit Set or Bit Clear to change individual bits in an integer variable.

See Also: Get I/O Unit as Binary Value (page G-65)

Set Month

Time/Date Action

Function: To set the month value (1 through 12) in the control engine's real-time clock/calendar.

Typical Use: To set the month from an ioControl 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 control engine's real-time clock/calendar.
- The control engine'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
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Standard Example:

Set Month		
To	MONTH	*Integer 32 Variable*

OptoScript Example:

SetMonth(*To*)

```
SetMonth(MONTH);
```

This is a procedure command; it does not return a value.

Note: Do not issue this command continuously.

See Also: Get Day (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Hours (page S-25), Set Day (page S-17), Set Minutes (page S-43), Set Seconds (page S-78), Set Year (page S-89)

Set Nth Character

String Action

Function: Changes a character within a string.

Typical Use: When building communication strings prior to sending.

Details:

- The character can be written to any position from 0 up to the current string length.
- Valid range for the character is 0–255.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
To	**In String**	**At Index**	**Put Status In**
Integer 32 Literal	String Variable	Integer 32 Literal	Float Variable
Integer 32 Variable		Integer 32 Variable	Integer 32 Variable

Standard Example:

Set Nth Character

To	62	*Integer 32 Literal*
In String	MSG_RECEIVED	*String Variable*
At Index	POSITION	*Integer 32 Variable*
Put Status In	STATUS	*Integer 32 Variable*

OptoScript Example:

SetNthCharacter(*To, In String, At Index***)**

```
STATUS = SetNthCharacter(62, MSG_RECEIVED, POSITION);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- A status of zero indicates success.
- The string could initially be filled with nulls or spaces up to its declared width to avoid null string errors.

Status Codes:

0 = Success

-42 = Invalid index value. The index was negative or greater than the string length, or the character value is outside the range 0-255.

-45 = Null/empty string. The string being written to is empty.

See Also: Find Character in String (page F-1), Get Nth Character (page G-100)

Set PID Configuration Flags

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change PID configuration options within strategy logic.

Typical Use: To force output to a predetermined value or change it to manual if input goes out of range.

Details:

- PID configuration options can be set using this command or when you initially configure the PID loop in ioManager or ioControl.
- Configuration options are sent as a 32-bit integer (a mask). One or multiple options can be chosen. Option values (in hex) are:
  - 0x00000000 = Standard; no special flags.
  - 0x00000001 = Enable square root of input.
  - 0x00000002 = If input goes out of range, force output to a predetermined value. (Set the predetermined value when you initially configure the PID loop.)
  - 0x00000004 = If input goes out of range, switch PID to manual. (if input returns to normal range, PID will switch back to automatic.)

Arguments:

Argument 1	Argument 2
PID Loop	**Configuration Flags**
PID Loop	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Configuration Flags

PID Loop	HEATER_3	*PID Loop*
Configuration Flags	PID_CONFIG_FLAGS	*Integer 32 Variable*

OptoScript Example:

`SetPidConfigFlags(`*PID Loop, Configuration Flags***`)`**

```
SetPidConfigFlags(HEATER_3, PID_CONFIG_FLAGS);
```

This is a procedure command; it does not return a value.

Notes: See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.

Dependencies: Communication to the PID must be enabled for this command to take effect.

See Also: Enable Communication to PID Loop (page E-6), Get PID Configuration Flags (page G-112)

Set PID Feed Forward

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the feed forward value for the PID loop.

Typical Use: To set the value of the PID feed forward for applications requiring feed forward control.

Details:

- The initial value is normally set when the PID is configured and tuned.
- For all four PID algorithms, the Feed Forward and the Feed Forward Gain values are multiplied and then added to the output; therefore, a value of 0 for either results in no change to the output.

Arguments:

Argument 1	Argument 2
PID Loop	**Feed Forward**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Feed Forward

PID Loop	HEATER_3	*PID Loop*
Feed Forward	PID_FEED_FORWARD	*Float Variable*

OptoScript Example:

`SetPidFeedForward(`*PID Loop, Feed Forward***`)`**

```
SetPidFeedForward(HEATER_3, PID_FEED_FORWARD);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the ioControl User’s Guide.
- Feed forward is added before output clamping and has a tuning factor.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Feed Forward (page G-115)

Set PID Feed Forward Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the feed forward gain of the PID output.

Typical Use: To set the value of the feed forward gain of the PID loop for applications requiring feed forward control.

Details:

- The initial value is normally set when the PID is configured and tuned.
- For all four PID algorithms, the Feed Forward and the Feed Forward Gain values are multiplied and then added to the output; therefore, a value of 0 for either results in no change to the output.

Arguments:

Argument 1 **PID Loop**	Argument 2 **Feed Fwd Gain**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Feed Forward Gain

PID Loop	HEATER_3	*PID Loop*
Feed Fwd Gain	PID_FEED_FD_GAIN	*Float Variable*

OptoScript Example:

SetPidFeedForwardGain(*PID Loop, Feed Fwd Gain***)**

```
SetPidFeedForwardGain(HEATER_3, PID_FEED_FD_GAIN);
```

This is a procedure command; it does not return a value.

Notes: See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Feed Forward Gain (page G-116)

Set PID Forced Output When Input Over Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the forced value that will be sent to the PID output if the input is over the established range.

Typical Use: To set the PID output to a known value if the input goes higher than its normal range.

Details: The PID must be in auto mode for this command to take effect.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Forced Output**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Forced Output When Input Over Range

PID Loop	HEATER_3	*PID Loop*
Forced Output	PID_OUTPUT_OVER_RANGE	*Float Variable*

OptoScript Example:

`SetPidForcedOutputWhenInputOverRange(`*PID Loop, Forced Output***`)`**

```
SetPidForcedOutputWhenInputOverRange(HEATER_3, PID_OUTPUT_OVER_RANGE);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- A forced output when the input is out of range (either over or under) can also be set when you configure the PID loop.

See Also: Get PID Forced Output When Input Over Range (page G-117), Set PID Forced Output When Input Under Range (page S-63)

Set PID Forced Output When Input Under Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the forced value that will be sent to the PID output if the input is under the established range.

Typical Use: To set the PID output to a known value if the input goes lower than its normal range.

Details: The PID must be in auto mode for this command to take effect.

Arguments:

Argument 1	Argument 2
PID Loop	**Forced Output**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Forced Output When Input Under Range

PID Loop	HEATER_3	*PID Loop*
Forced Output	PID_OUTPUT_UNDER_RANGE	*Float Variable*

OptoScript Example:

SetPidForcedOutputWhenInputUnderRange(*PID Loop, Forced Output***)**

```
SetPidForcedOutputWhenInputUnderRange(HEATER_3, PID_OUTPUT_UNDER_RANGE);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- A forced output when the input is out of range (either over or under) can also be set when you configure the PID loop.

See Also: Get PID Forced Output When Input Under Range (page G-118), Set PID Forced Output When Input Over Range (page S-62)

Set PID Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or 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 setpoint.
- Always use a non-zero gain value.
- Use a negative gain to reverse the direction of the PID output (typical for heating applications).
- Gain has a direct multiplying effect on the integral and derivative values. 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 **PID Loop**	Argument 2 **Gain**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Gain

PID Loop	Extruder_Zone08	*PID Loop*
Gain	Zone08_Gain	*Float Variable*

OptoScript Example:

SetPidGain(*PID Loop, Gain***)**

```
SetPidGain(Extruder_Zone08, Zone08_Gain);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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 from -1 to -40.
- Use more gain to improved 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.

See Also: Enable Communication to PID Loop (page E-6), Get PID Gain (page G-119)

Set PID Input

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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 point 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 0.8 seconds to 1.0 seconds should be adequate).

Arguments:

Argument 1	**Argument 2**
PID Loop	**Input**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Input

PID Loop	HEATER_3	*PID Loop*
Input	PID_INPUT_VALUE	*Float Variable*

OptoScript Example:

`SetPidInput(` *PID Loop, Input* **`)`**

```
SetPidInput(HEATER_3, PID_INPUT_VALUE);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Do not send the input value to the PID less frequently than the PID scan rate, as it will adversely affect the PID performance.

Dependencies:

- You 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.

See Also: Enable Communication to PID Loop (page E-6), Get PID Input (page G-120)

Set PID Input High Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the highest expected value from the PID's input.

Typical Use: To set the highest valid input from the PID.

Details: Input high range is normally set when the PID is configured, but it can be changed from a running strategy using this command.

Arguments:

Argument 1	Argument 2
PID Loop	**High Range**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Input High Range

PID Loop	HEATER_3	*PID Loop*
High Range	PID_High_Range	*Float Variable*

OptoScript Example:

`SetPidInputHighRange(`*PID Loop, High Range***`)`**

```
SetPidInputHighRange(HEATER_3, PID_HIGH_RANGE);
```

This is a procedure command; it does not return a value.

Notes:

- Input range affects the span used in the PID algorithm. It is also used in output options when the input is out of range. See Set PID Configuration Flags.
- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Get PID Input High Range (page G-121), Set PID Input Low Range (page S-67)

Set PID Input Low Range

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the lowest expected value from the PID's input.

Typical Use: To set the lowest valid input for the PID.

Details: Input low range is normally set when the PID is configured, but it can be changed from a running strategy using this command.

Arguments:

Argument 1	Argument 2
PID Loop	**Low Range**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Input Low Range

PID Loop	HEATER_3	*PID Loop*
Low Range	PID_LOW_RANGE	*Float Variable*

OptoScript Example:

`SetPidInputLowRange(`*PID Loop, Low Range***`)`**

```
SetPidInputLowRange(HEATER_3, PID_LOW_RANGE);
```

This is a procedure command; it does not return a value.

Notes:

- Input range affects the span used in the PID algorithm. It is also used in output options when the input is out of range. See Set PID Configuration Flags.
- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Get PID Input Low Range (page G-122), Set PID Input High Range (page S-66)

Set PID Max Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the maximum output change limit of the PID.

Typical Use: To define the maximum amount that the PID output is allowed to change per scan period, to make sure the output ramps up (or down) rather than increasing or decreasing too quickly.

Details:

- Maximum output change is normally set when the PID is configured, but it can be changed from a running strategy using this command.
- Units are the same as those defined for the PID output point.
- The default value is the range of the output point. This setting allows the PID output to move as much as 100 percent per scan period. For example, if the PID output point is 4–20 mA, 16.00 would be the default, representing 100 percent of the range.
- Note that the max output change limits the PID algorithm and may slow it down.

Arguments:

Argument 1 **PID Loop**	Argument 2 **Max Change**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Max Output Change

PID Loop	HEATER_3	*PID Loop*
Max Change	PID_MAX_LIMIT	*Float Variable*

OptoScript Example:

`SetPidMaxOutputChange(`*PID Loop, Max Change***`)`**

```
SetPidMaxOutputChange(HEATER_3, PID_MAX_LIMIT);
```

This is a procedure command; it does not return a value.

Notes: See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Max Output Change (page G-123), Set PID Scan Time (page S-74)

Set PID Min Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set the minimum amount of change that must occur before the PID output will change.

Typical Use: To define how much change must occur before the PID output changes, in order to avoid constant changes that might wear out parts (such as valve linkage).

Details:
- Minimum output change is normally set when the PID is configured, but it can be changed from a running strategy using this command.
- Units are the same as those defined for the PID output channel.
- The default value is zero (no minimum). The value must be a positive number.
- The change is applied when it exceeds the minimum in either direction (up or down).

Arguments:

Argument 1	**Argument 2**
PID Loop	**Min Change**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Min Output Change

PID Loop	HEATER_3	*PID Loop*
Min Change	PID_MIN_LIMIT	*Float Variable*

OptoScript Example:

`SetPidMinOutputChange(`*PID Loop, Min Change***`)`**

```
SetPidMinOutputChange(HEATER_3, PID_MIN_LIMIT);
```

This is a procedure command; it does not return a value.

Notes: See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Max Output Change (page G-123), Set PID Scan Time (page S-74)

Set PID Mode

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: Sets the auto/manual mode of the PID.

Typical Use: To change the PID from automatic to manual mode or return it to auto.

Details:

- In auto mode, the PID output functions normally. In manual mode, the PID output is not updated by the PID calculation, but retains its most recent value.
- Use these values to set auto and manual modes: auto = 0, manual = 1.
- To change the PID output value while in manual mode, use Set PID Output, Debug mode, ioManager, or ioDisplay to write directly to the PID output analog point.

Arguments:

Argument 1 **PID Loop**	Argument 2 **Mode**
PID Loop	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Mode

PID Loop	Extruder_Zone08	*PID Loop*
Mode	ZONE08_MODE	*Integer 32 Variable*

OptoScript Example:

`SetPidMode(`*PID Loop, Mode***`)`**

```
SetPidMode(Extruder_Zone08, ZONE08_MODE);
```

This is a procedure command; it does not return a value.

Notes: See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.

See Also: Get PID Mode (page G-125), Set PID Output (page S-71)

Set PID Output

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the output value of the PID.

Typical Use: To adjust the PID output when the PID is in manual mode.

Details: The value sent must have the same engineering units as the specified PID output channel.

Arguments:

Argument 1	Argument 2
PID Loop	**Output**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Output

PID Loop	HEATER_3	*PID Loop*
Output	TPO_OUTPUT	*Analog Output*

OptoScript Example:

SetPidOutput(*PID Loop, Output***)**

```
SetPidOutput(HEATER_3, TPO_OUTPUT);
```

This is a procedure command; it does not return a value.

Notes: See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Output (page G-126), Set PID Mode (page S-70), Get PID Mode (page G-125)

Set PID Output High Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the high clamp value for the PID output.

Typical Use: To change the high clamp value while the strategy is running.

Details: The output low clamp and high clamp values define the range of output for this PID loop. They are normally set when the PID is configured but can be changed from within a running strategy using this command.

Arguments:

Argument 1	**Argument 2**
PID Loop	**High Clamp**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Output High Clamp

PID Loop	HEATER_3	*PID Loop*
High Clamp	PID_HIGH_CLAMP	*Float Variable*

OptoScript Example:

`SetPidOutputHighClamp(`*PID Loop, High Clamp***`)`**

```
SetPidOutputHighClamp(HEATER_3, PID_HIGH_CLAMP);
```

This is a procedure command; it does not return a value.

Notes: See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Output High Clamp (page G-127), Set PID Output Low Clamp (page S-73)

Set PID Output Low Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the low clamp value for the PID output.

Typical Use: To change the PID output's low clamp value while a strategy is running.

Details: The output low clamp and high clamp values define the range of output for this PID loop. They are normally set when the PID is configured but can be changed from within a running strategy using this command.

Arguments:

Argument 1 **PID Loop**	Argument 2 **Low Clamp**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Output Low Clamp

PID Loop	HEATER_3	*PID Loop*
Low Clamp	PID_LOW_CLAMP	*Float Variable*

OptoScript Example:

`SetPidOutputLowClamp(`*PID Loop, Low Clamp***`)`**

```
SetPidOutputLowClamp(HEATER_3, PID_LOW_CLAMP);
```

This is a procedure command; it does not return a value.

Notes: See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Output Low Clamp (page G-128), Set PID Output High Clamp (page S-72)

Set PID Scan Time

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To set or change the PID calculation interval (the update period or scan rate).

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. In order to tune the PID, scan time should be greater than system lag (the time it takes for the controller output to have a measurable effect on the system). Also consider other PIDs and tasks on the brain competing for processing power.
- The value to send is in seconds. The default is 0.1 seconds.
- This command is useful for adapting a PID to work for either heating or cooling when the heating mode has a different dead time than the cooling mode.

Arguments:

Argument 1 **PID Loop**	Argument 2 **Scan Time (sec)**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Scan Time

PID Loop	Extruder_Zone08	*PID Loop*
Scan Time (sec)	Zone08_Scan_Time	*Float Variable*

OptoScript Example:

`SetPidScanTime(`*PID Loop, Scan Time***`)`**

```
SetPidScanTime(Extruder_Zone08, Zone08_Scan_Time);
```

This is a procedure command; it does not return a value.

Notes:

- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- Frequent use of this command can adversely affect the PID performance.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Scan Time (page G-129),

Set PID Setpoint

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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:

- To use this command, the setpoint must be configured to come from Host.
- The value you send has the same engineering units as the PID input.
- The setpoint can be an analog point, even from another I/O unit.

Arguments:

Argument 1	**Argument 2**
PID Loop	**Setpoint**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Setpoint

PID Loop	Heater_3	*PID Loop*
Setpoint	Pid_Setpoint_Value	*Float Variable*

OptoScript Example:

SetPidSetpoint(*PID Loop, Setpoint***)**

```
SetPidSetpoint(Heater_3, PID_Setpoint_Value);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Send a new setpoint value to the PID only when necessary.

Dependencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Setpoint (page G-130)

Set PID Tune Derivative

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

Function: To change the derivative value of the PID.

Typical Use: To improve performance in systems with long delays between when the PID output changes and when the PID input responds to the change.

Details:

- The derivative is used to determine how much effect the change-in-slope of the PID input should have on the PID output. It 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
- Too high a derivative value results in excessive amounts of PID output change. In systems with long delays, too low a derivative value results in a PID output that is always out of phase with the PID input.

Arguments:

Argument 1	Argument 2
PID Loop	**Tune Derivative**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Tune Derivative

PID Loop	Extruder_Zone08	*PID Loop*
Tune Derivative	Zone08_Derivative	*Float Variable*

OptoScript Example:

`SetPidTuneDerivative(`*PID Loop*, *Tune Derivative***`)`**

```
SetPidTuneDerivative(Extruder_Zone08, Zone08_Derivative);
```

This is a procedure command; it does not return a value.

Notes:

- See “PID Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Leave the derivative at zero unless you are sure you need it and until the gain and integral have been determined. Use sparingly; a little derivative goes a long way.
- Since derivative is applied only to the process variable, not to the setpoint, the setpoint can be changed without causing spikes in the derivative term.

Depedencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Tune Derivative (page G-132)

Set PID Tune Integral

PID—Ethernet Action

NOTE: This command is used for PID loops in ioControl; it is not for use with the SNAP-PID-V module.

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.
- Too high an integral value results in excessive PID output change; too low an integral value results in long-lasting errors between the PID input and the PID setpoint.

Arguments:

Argument 1 **PID Loop**	Argument 2 **Tune Integral**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Standard Example:

Set PID Tune Integral

PID Loop	Extruder_Zone08	*PID Loop*
Tune Integral	Zone08_Integral	*Float Variable*

OptoScript Example:

SetPidTuneIntegral(*From PID Loop, Tune Integral***)**

```
SetPidTuneIntegral(Extruder_Zone08, Zone08_Integral);
```

This is a procedure command; it does not return a value.

Notes:
- See "PID Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use an initial value of 1.0 until a better value is determined. Typical integral values range from 0.1 to 20.
- This PID prevents integral windup by back calculating the integral without the derivative term.

Depedencies: Communication to the PID must be enabled for this command to send the value to the PID.

See Also: Enable Communication to PID Loop (page E-6), Get PID Tune Integral (page G-133)

Set Seconds

Time/Date Action

Function: To set the seconds value (0 through 59) in the control engine's real-time clock/calendar.

Typical Use: To set the seconds from an ioControl 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 control engine's real-time clock/calendar.
- The control engine'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
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Standard Example:

Set Seconds

To	SECONDS	*Integer 32 Variable*

OptoScript Example:

SetSeconds(*To*)

```
SetSeconds(SECONDS);
```

This is a procedure command; it does not return a value.

Note: Do not issue this command continuously.

See Also: Get Day (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Hours (page S-25), Set Day (page S-17), Set Minutes (page S-43), Set Month (page S-57), Set Year (page S-89)

Set Simple 64 I/O Unit from MOMO Masks

Deprecated

NOTE: This command has been deprecated. It is still functional, however if you are developing a new strategy, use Set I/O Unit from MOMO Masks (page S-29) instead.

Function: To control multiple digital output points on the same 64-point I/O unit simultaneously with a single command (applies to I/O units with a SNAP-ENET-S64 brain only).

Typical Use: To efficiently control all digital outputs on a SNAP Simple I/O 64-point rack with one command.

Details:

- This command is 64 times faster than using Turn On or Turn Off 64 times.
- Updates the IVALs and XVALs for all 64 points. Affects only selected output points. Does not affect input points.
- To turn on a point, set the respective bit in the 64-bit data field of argument 1 (the must-on bit mask) to a value of "1."To turn off a point, set the respective bit in the 64-bit data field of argument 2 (the must-off bit mask) to a value of "1." To leave a point unaffected, set its bits to a value of 0 in *both* arguments 1 and 2. (Check for conflicts; if the same bit is set to 1 in both masks, the point is turned off.)
- The least significant bit corresponds to point zero.
- If a specific point is disabled or if the entire I/O unit is disabled, only the internal values (IVALs) will be written.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Must On Mask	**Must Off Mask**	**Simple 64 I/O Unit**
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-S64
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

Set Simple 64 I/O Unit from MOMO Masks		
Must On Mask	0x060003C0000000C2	*Integer 64 Literal*
Must Off Mask	0xB0F240010308A020	*Integer 64 Literal*
Simple 64 I/O Unit	PUMP_CTRL_UNIT	*SNAP-ENET-S64*

The effect of this command is illustrated below:

	Point Number	63	62	61	60	59	58	57	56	→	7	6	5	4	3	2	1	0
Must-on Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			
Must-off Bit Mask	Binary	1	0	1	1	0	0	0	0	→	0	0	1	0	0	0	0	0
	Hex	B				0				→	2				0			

To save space, the example shows only the first eight points and the last eight points on the rack. For the points shown, points 58, 57, 7, 6, and 1 will be turned on. Points 63, 61, 60, and 5 will be turned off. Other points shown are not changed.

OptoScript Example:

`SetSimple64IoUnitFromMomo(`*Must-On Mask*, *Must-Off Mask*, *Simple 64 I/O Unit***`)`**

```
SetSimple64IoUnitFromMomo(0x060003C0000000C2i64, 0xB0F240010308A020i64,
PUMP_CTRL_UNIT);
```

This is a procedure command; it does not return a value. (Note that Integer 64 literals in OptoScript code take an `i64` suffix.)

Notes: Use Bit Set or Bit Clear to change individual bits in an integer variable.

See Also: Get I/O Unit as Binary Value (page G-65)

Set Target Address State

I/O Unit Action

Function: To control which target addresses in a redundant system should be enabled on an I/O unit.

Typical Use: To control which network is used for a specific I/O unit in a redundant system.

Details:

- A target address is the IP address of an Ethernet interface on an I/O unit.
- In a redundant network architecture, you can assign two target addresses to an I/O unit. In ioControl these are called the Primary Address and the Secondary Address. By default, the Primary Address is used, but the server will switch to the Secondary Address if the primary address is not available.
- Each target address has an *enabled* state and an *active* state. If both target addresses are enabled, they are available to be used. However, only one address can be used at a given time, so there can only be one active address.
- Use Argument 1 to enable one or both addresses.
- Use Argument 2 to disable one or both addresses.
- Use Argument 3 to make one address active.
- Use Argument 4 to designate the I/O unit type.
- Only the last bit of the 32-bit data field is used. Therefore, for arguments 1, 2, and 3 you can use the integers 0, 1, 2, and 3 to indicate the following:
  0=No change
  1=Primary Target Address
  2=Secondary Target Address
  3=Primary and Secondary Target Addresses

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Must On Mask	**Must Off Mask**	**Active Mask**	**I/O Unit**
Integer 32 Literal	Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64
			SNAP-ENET-S64
			SNAP-UP1-M64
			SNAP-B3000-ENET,
			SNAP-ENET-RTC
			SNAP-UP1-ADS
			SNAP-PAC-R1
			SNAP-PAC-R2

Standard Example: This example assumes that there are redundant networks. It enables the secondary address, disables the primary address, and makes the secondary address active.

Set Target Address States

Must On Mask	2	*Integer 32 Literal*
Must Off Mask	1	*Integer 32 Literal*
Active Mask	2	*Integer 32 Literal*
I/O Unit	UNIT	*SNAP-UP1-ADS*

OptoScript Example: **`SetTargetAddressState(`***Must-On Mask, Must-Off Mask, Active Mask, I/O Unit***`)`**

```
SetTargetAddressState(2, 1, 2, UNIT);
```

This is a procedure command; it does not return a value.

Notes:

- See "I/O Unit Commands " in Chapter 10 of the *ioControl User's Guide*.
- Arguments 1 and 2 (the Must On Mask and the Must Off Mask) together comprise the enable mask. You can use the enable mask in the following combinations:

To do this:	**Must On Mask:**	**Must Off Mask:**
Enable both addresses	3	0
Enable Primary	1	0
Enable Secondary	2	0
Enable *only* Primary	1	2
Enable *only* Secondary	2	1
Disable Primary	0	1
Disable Secondary	0	2
Disable both addresses	0	3

- Argument 3 makes one address active or both addresses inactive as follows:

To do this:	**Active Mask:**
Make both addresses inactive	0
Activate Primary	1
Activate Secondary	2

- A fully redundant system may also include ioDisplay clients and OptoOPCServers. These commands only deal with the control engine communicating with I/O units. ioDisplay and OptoOPCServer have their own mechanism for controlling their use of the network.

See Also: Set All Target Address States (page S-5), Get Target Address State (page G-141)

Set Time

Time/Date Action

Function: To set the time in the control engine's real-time clock/calendar from a string variable.

Typical Use: To set the time from an ioControl program.

Details:

- The *From* parameter (*Argument 1*) can be a constant or string variable, although a string variable 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 control engine's real-time clock/calendar.
- The control engine'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
From
String Literal
String Variable

Standard Example:

Set Time		
From	TIME_STRING	*String Variable*

OptoScript Example:

SetTime(*To*)

```
SetTime(TIME_STRING);
```

This is a procedure command; it does not return a value.

Notes:

- To change the time, use an integer variable 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 control engine'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 (DD/MM/YYYY) (page C-59), Copy Date to String (MM/DD/YYYY) (page C-60), Copy Time to String (page C-61), Set Date (page S-16)

Set TPO Percent

Digital Point Action

Function: To set the on time of an output point 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 point 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.
- On mistic brains, 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). On Ethernet brains, the current pulse train is immediately cancelled and replaced with the new one, starting with the off state.

Arguments:

Argument 1	Argument 2
To (Percent)	**On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set TPO Percent

To (Percent)	20	*Integer 32 Literal*
On Point	Heater_Output	*Time Proportional Output*

OptoScript Example:

`SetTpoPercent(`*To Percent, On Point***`)`**

```
SetTpoPercent(20, Heater_Output);
```

This is a procedure command; it does not return a value.

Notes:

- When using the output of a PID to drive a digital TPO, scale the analog output point (for the PID) to 0–100. (This analog point does not have to exist physically, but must be one of the 16 points on the I/O unit.) Use Move to copy the PID analog output value to the digital TPO point 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.
- To ensure that the TPO period will always be correct, store this and other changeable I/O unit values in flash memory (EEPROM) at the I/O unit using the Debug mode in ioControl. Some older hardware and firmware will not support this feature. For more information, see the *ioControl User's Guide*.
- Setting the value of a digital TPO overrides any prior Turn On or Turn Off command for the digital point.

Dependencies:
- A Set TPO Period command must be used at least once before this command to define the time period.
- Applies only to output points configured with the TPO feature.

See Also: Set TPO Period (page S-85)

Set TPO Period

Digital Point Action

Function: To set the time proportional output (TPO) period of an output point.

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 TPO Percent command.

Arguments:

Argument 1 **To (Seconds)**	Argument 2 **On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Set TPO Period

To (Seconds)	60.0	*Float Literal*
On Point	Heater_Output	*Time Proportional Output*

OptoScript Example:

`SetTpoPeriod(`*To Seconds, On Point***`)`**

```
SetTpoPeriod(60.0, Heater_Output);
```

This is a procedure command; it does not return a value.

Notes:
- The time proportion period specifies only the total time over which the output is varied. Set TPO 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 point 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 flash memory (EEPROM) at the I/O unit using the Debug mode in ioControl. Some older hardware and firmware will not support this feature. For more information, see the *ioControl User's Guide*.

- If the TPO period is not stored in flash memory at the I/O unit, use this command immediately before Set TPO 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 too frequently, since this can cause unnecessary interruptions in ongoing processes.

Dependencies: Applies only to output points configured with the TPO feature.

See Also: Set TPO Percent (page S-84)

Set Up Timer Target Value

Timing Action

Function: To set the target value of an up timer.

Typical Use: To initialize an up timer.

Details:

- This command sets the target value *but does not start the timer*. You must start the timer using the Start Timer command.
- Up timers do not stop timing when they reach their target value. Use the Up Timer Target Time Reached? command to determine if the target time has been reached.
- The target value must be a positive number in seconds.

Arguments:

Argument 1	Argument 2
Target Value	**Up Timer**
Float Literal	Up Timer Variable
Float Variable	

Standard Example:

Set Up Timer Target Value

Target Value	60.0	*Float Literal*
Up Timer	OVEN_TIMER	*Up Timer Variable*

OptoScript Example:

`SetUpTimerTarget(`*Target Value, Up Timer***`)`**

```
SetUpTimerTarget(60.0, Oven_Timer);
```

This is a procedure command; it does not return a value.

Notes:

- See "Timing Commands" in Chapter 10 of the *ioControl User's Guide* for more information on timers.
- To set the target value and start the timer in one step, use the Move command to move the target value to the timer. The timer will immediately start from zero. Using the Move command overwrites any target value previously set.

See Also: Start Off-Pulse (page S-96), Stop Timer (page S-102), Continue Timer (page C-39), Pause Timer (page P-1), Up Timer Target Time Reached? (page U-1)

Set Variable False

Logical Action

Function: To move a False (0) value into a variable.

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
[Value]
Float Variable
Integer 32 Variable

Standard Example:

Set Variable False
    Flag_Hopper_Full   *Integer 32 Variable*

OptoScript Example:

`SetVariableFalse(`*Variable***`)`**

```
SetVariableFalse(Flag_Hopper_Full);
```

This is a procedure command; it does not return a value.

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- *Speed Tip:* This command is faster than Move for moving a zero to a variable.

See Also: Set Variable True (page S-88)

Set Variable True

Logical Action

Function: To move a True (+1) value into a variable.

Typical Use: To set a variable to true.

Details: All numeric variables are False by default unless initialized to a non-zero value.

Arguments:

Argument 1
[Value]
Float Variable
Integer 32 Variable

Standard Example:

Set Variable True
    FLAG_JOB_DONE   *Integer 32 Variable*

OptoScript Example:

`SetVariableTrue(`*Variable***`)`**

```
SetVariableTrue(FLAG_JOB_DONE);
```

This is a procedure command; it does not return a value.

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.
- *Speed Tip:* This command is faster than Move for moving a +1 value to a variable.

See Also: Set Variable False (page S-87)

Set Year

Time/Date Action

Function: To set the year value (2000 through 2099) in the control engine's real-time clock/calendar.

Typical Use: To set the year from an ioControl program.

Details:

- The *To* parameter (*Argument 1*) can be an integer or a float, although an integer is preferred.
- Executing this command would set the year (2000 through 2099) in the control engine's real-time clock/calendar.
- The control engine's real-time clock/calendar will automatically increment the time and date after they are set.
- All erroneous year values are ignored.

Arguments:

Argument 1
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Standard Example:

Set Year		
To	YEAR	*Integer 32 Variable*

OptoScript Example:

SetYear(*To*)

```
SetYear(YEAR);
```

This is a procedure command; it does not return a value.

Notes:

- The control engine'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 (page G-49), Get Day of Week (page G-50), Get Hours (page G-63), Get Minutes (page G-86), Get Month (page G-97), Get Seconds (page G-135), Get Year (page G-145), Set Hours (page S-25), Set Day (page S-17), Set Minutes (page S-43), Set Month (page S-57), Set Seconds (page S-78)

Shift Numeric Table Elements

Miscellanous Action

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**
Shift Count	**Table**
Integer 32 Literal	Float Table
Integer 32 Variable	Integer 32 Table

Standard Example:

Shift Numeric Table Elements

Shift Count	-5	*Integer 32 Literal*
Table	MY_TABLE	*Float Table*

OptoScript Example:

`ShiftNumTableElements(`*Shift Count*, *Table***`)`**

```
ShiftNumTableElements(-5, MYTABLE);
```

This is a procedure command; it does not return a value.

Notes:
- Use Move from Numeric Table Element before this command to capture values that will be shifted out of the table, if they need to be used.
- Use Move to Numeric Table Element (for example) after this command to fill vacated entries, if desired.

See Also: Move Numeric Table Element to Numeric Table (page M-13), Move from Numeric Table Element (page M-8), Move to Numeric Table Element (page M-17)

Sine

Mathematical Action

Function: To derive the sine of an angle.

Typical Use: Trigonometric function for computing triangular height of the angle.

Details:

- Calculates the sine of *Argument 1* and places the result in *Argument 2*.
- *Argument 1* has a theoretical range of -infinity to +infinity, but is limited by the type of variable used.
- The range of *Argument 2* is -1.0 to 1.0, inclusive.
- The following are examples of sine calculations to four decimal places:

Radians	Degrees	Result
0.0	0	0.0
0.7854	45	0.7071
1.5708	90	1.0000
2.3562	135	0.7071
3.1416	180	0.0000
3.9270	225	-0.7071
4.7124	270	-1.0000
5.4978	315	-0.7071
6.2832	360	0.0000

Arguments:

Argument 1 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Sine

Of	Radians	*Float Variable*
Put Result in	SINE	*Float Variable*

OptoScript Example:

`Sine(`*`Of`*`)`

```
SINE = Sine(Radians);
```

This is a function command; it returns the sine of the angle. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- To convert units of degrees to units of radians, divide degrees by 57.29578 (or 180 / pi).
- Use Arcsine if the sine is known and the angle is desired.

See Also: Arcsine (page A-12), Cosine (page C-62), Tangent (page T-1)

Square Root

Mathematical Action

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 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Square Root

Of	Area_of_Square	*Integer 32 Variable*
Put Result in	Height_of_Square	*Integer 32 Variable*

OptoScript Example:

`SquareRoot(`*`Of`*`)`

```
Height_of_Square = SquareRoot(Area_of_Square);
```

This is a function command; it returns square root of the value. The returned value can be consumed by a variable (as in the example shown) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- Executes faster than raising a number to the 0.5 power.
- Taking the square root of zero or of a negative value will result in zero, and a queue error. Use > or Greater? to check the value before using the command.
- 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.

Queue Errors: -14 = Invalid number.

See Also: Raise to Power (page R-2), Greater? (page G-146)

Start Chart

Chart Action

Function: To request that a stopped chart begin executing at Block 0 or to request that a suspended chart continue executing from the point at which it was suspended.

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.
- If the chart is stopped and fewer than the maximum number of tasks are running, then this chart will be added to the task queue and this command will succeed. Otherwise, it has no effect. If the chart is suspended, then the chart is already part of the task queue, and this command will continue the chart from the point at which it is suspended.
- The maximum number of charts for a SNAP Ultimate brain is 8; the maximum number of charts for a SNAP-LCE controller is 16.
- Upon success, the chart will start at its next scheduled time in the task queue.

Arguments:

Argument 1	**Argument 2**
Chart	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Standard Example:

Start Chart

Chart	CHART_B	*Chart*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`StartChart(`*Chart***`)`**

```
STATUS = StartChart(CHART_B);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- This command should be used judiciously. It can take up to 100 ms for the chart to start. Use this command only when timing is not critical. Otherwise, instead of Start Chart, use a chart that runs continuously and uses subroutines for any kind of repetitive logic.
- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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.

Dependencies: If the chart is stopped, then a task must be available in the task queue.

Status Codes: 0 = success

-5 = failure

See Also: Continue Chart (page C-38), Stop Chart (page S-99)

Start Continuous Square Wave

Digital Point Action

Function: To generate a square wave on an output point.

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 point. *On Time* specifies the amount of time in seconds that the point will remain on during each pulse; *Off Time* specifies the amount of time the point 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. 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.
- 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.
- On mistic brains, 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). On Ethernet brains, the current pulse train is immediately cancelled and replaced with the new one, starting with the off state.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
On Time (Seconds)	**Off Time (Seconds)**	**On Point**
Float Literal	Float Literal	Digital Output
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	

Standard Example:

Start Continuous Square Wave

On Time (Seconds)	0.100	*Integer 32 Literal*
Off Time (Seconds)	0.500	*Integer 32 Literal*
On Point	BLINKING_LAMP	*Digital Output*

OptoScript Example:

`StartContinuousSquareWave(` *On Time (Seconds), Off Time (Seconds), On Point* **`)`**

```
StartContinuousSquareWave(0.100, 0.500, BLINKING_LAMP);
```

This is a procedure command; it does not return a value.

Notes:

- Once the pulse train has started, the digital I/O unit maintains the waveform indefinitely.
- Pulse trains on mistic brains are cancelled when a Turn Off or Turn On is sent to the output, or when the output is configured (for example, when a strategy is first run and I/O units are initialized). Ethernet brains do NOT cancel pulse trains on an output upon configuration, or when the output is turned off or on. To programmatically cancel a pulse train on an Ethernet brain, use this command with both the on times and off times set to 0. Pulse trains on both Ethernet and mistic brains will also be cancelled if the brain receives a reset command.

Dependencies:

- Applies only to outputs.

- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Generate N Pulses (page G-5)

Start Counter

Digital Point Action

Function: To reactivate a standard digital input counter or quadrature counter.

Typical Use: To restart a digital input counter or quadrature counter after it has been stopped.

Details:
- Standard digital only; high-density digital counters cannot be stopped or started.
- Must be used to activate quadrature counter inputs on serial (Mistic) I/O units. On Ethernet-based (MMP) I/O units, counters start as soon as they are configured, and Start Counter is only used after you have used the Stop Counter command.
- Does not reset the counter or quadrature counter to zero.
- Retains any previously accumulated counts.
- A quadrature counter occupies two adjacent points, so quadrature modules appear with only points 00 and 02 available.

Arguments:

Argument 1
On Point
Counter
Quadrature Counter

Standard Example:

Start Counter

On Point	BAGGAGE_COUNTER	*Counter*

OptoScript Example:

`StartCounter(`*On Point***`)`**

```
StartCounter(BAGGAGE_COUNTER);
```

This is a procedure command; it does not return a value.

Notes: Use Clear Counter to clear a counter or quadrature counter to zero.

Dependencies: Applies only to standard digital inputs configured with the counter or quadrature counter feature.

See Also: Get Counter (page G-48), Get & Clear Counter (page G-18), Clear Counter (page C-22), Stop Counter (page S-101)

Start Off-Pulse

Digital Point Action

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 point on.
- The time may be specified from 0.0005 to 429,496.7000 seconds (4.97 days), with a resolution of 100 microseconds. 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.
- During the execution of this command, if another Start Off-Pulse 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
Off Time (Seconds)	**On Point**
Float Literal	Digital Output
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Start Off-Pulse

Off Time (Seconds)	RESET_TIME	*Float Literal*
On Point	PUMP_2_STOP	*Digital Output*

OptoScript Example:

`StartOffPulse(`*Off Time (Seconds), On Point***`)`**

```
StartOffPulse(RESET_TIME, PUMP_2_STOP);
```

This is a procedure command; it does not return a value.

Notes:

- A Turn On command may be used to abort an off-pulse before the end of the off time.
- ***Caution:*** If this command is used more frequently than the specified delay, the output will remain off.

Dependencies:

- Applies only to outputs.
- Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Start On-Pulse (page S-97), Turn Off (page T-25), Turn On (page T-27)

Start On-Pulse

Digital Point Action

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 point off.
- The time may be specified from 0.0005 to 429,496.7000 seconds (4.97 days), with a resolution of 100 microseconds. 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.
- During the execution of this command, if another Start On-Pulse is performed, the current on-pulse is cancelled 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
On Time (Seconds)	**On Point**
Float Literal	Digital Output
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Standard Example:

Start On-Pulse

On Time (Seconds)	MIN_LATCH_TIME	*Float Variable*
On Point	PUMP_2_RUN	*Digital Output*

OptoScript Example:

`StartOnPulse(`*On Time (Seconds), On Point***`)`**

```
StartOnPulse(MIN_LATCH_TIME, PUMP_2_RUN);
```

This is a procedure command; it does not return a value.

Notes:

- A Turn Off command may be used to abort an on-pulse before the end of the on time.
- ***Caution:*** If this command is used more frequently than the specified delay, the output will remain on.

Dependencies: Available on mistic multifunction I/O units, SNAP PAC R-series controllers, and SNAP EIO and UIO brains with firmware version 7.0 or higher. For a list of mistic multifunction brains, see the Appendix Opto 22 Brain Families.

See Also: Start Off-Pulse (page S-96), Turn Off (page T-25), Turn On (page T-27)

Start Timer

Timing Action

Function: To start a timer variable.

Typical Use: To start an up timer or a down timer. To measure time elapsed since an event occurred.

Details:
- Use this command to start an up timer. Up timer variables start from 0 and count up.
- Also use this command to start a down timer. Down timer variables start from their preset value and count down to 0. Since the default preset value for a down timer is zero, nothing will happen if you start the timer without first using the Set Down Timer Preset Value command.

Arguments:

Argument 1
Timer
Down Timer Variable
Up Timer Variable

Standard Example:

Start Timer

Timer	Oven_Timer	*Down Timer Variable*

OptoScript Example:

StartTimer(*Timer***)**

```
StartTimer(Oven_Timer);
```

This is a procedure command; it does not return a value.

Notes:
- See “Timing Commands” in Chapter 10 of the *ioControl User’s Guide* for more information on timers.
- To set the target value (for an up timer) or the preset value (for a down timer) and start the timer at the same time, use the Move command.
- Start Timer always starts up timers from zero and down timers from their preset value. To restart a timer from the value where it was paused, use the command Continue Timer instead.

See Also: Stop Timer (page S-102), Continue Timer (page C-39), Pause Timer (page P-1), Set Down Timer Preset Value (page S-21), Set Up Timer Target Value (page S-86)

Stop Chart

Chart Action

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 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
Chart

Standard Example:

Stop Chart

Chart	CHART_B	*Chart*

OptoScript Example:

`StopChart(`*Chart***`)`**

`StopChart(CHART_B);`

This is a procedure command; it does not return a value.

Notes:
- This command should be used judiciously. It can take up to 100 ms for the chart to stop. Use this command only when timing is not critical. Otherwise, instead of Stop Chart, use a chart that runs continuously and uses subroutines for any kind of repetitive logic.
- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use Suspend Chart if you want to continue a chart from where it left off.

See Also: Start Chart (page S-93), Suspend Chart (page S-106), Chart Stopped? (page C-10)

Stop Chart on Error

Error Handling Action

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

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

Details:

- Since ioControl is a multitasking environment, an error handler chart cannot stop another chart instantaneously with this command, because the error handler chart itself is executed periodically. The actual time required depends on how many charts are running simultaneously.
- See the Errors Appendix in the *ioControl User's Guide* for a list of errors that may appear in the message queue.

Arguments: None.

Standard Example: **Stop Chart on Error**

OptoScript Example: **`StopChartOnError()`**

```
StopChartOnError();
```

This is a procedure command; it does not return a value.

Notes:

- See "Error Handling Commands" and Chart Commands" in Chapter 10 of the *ioControl User's Guide*.
- To get to each error in the message queue, the top error must be discarded, bringing the next error to the top. Use Remove Current Error and Point to Next Error to do this.

See Also: Remove Current Error and Point to Next Error (page R-22), Get Error Count (page G-53), Suspend Chart on Error (page S-107)

Stop Counter

Digital Point Action

Function: To deactivate a standard digital input counter or quadrature counter.

Typical Use: To inhibit a counter or quadrature counter until further notice.

Details:

- Standard digital only. High-density digital counters cannot be stopped or started.
- Stops the specified counter or quadrature counter.
- Stops counting incoming quadrature pulses until Start Counter is used.
- Does not reset the counter or quadrature counter to zero.
- Retains any previously accumulated counts.
- A quadrature counter occupies two adjacent points, so quadrature modules appear with only points 00 and 02 available.

Arguments:

Argument 1
On Point
Counter
Quadrature Counter

Standard Example:

Stop Counter

On Point	BEAN_COUNTER	*Counter*

OptoScript Example:

StopCounter(*On Point***)**

```
StopCounter(BEAN_COUNTER);
```

This is a procedure command; it does not return a value.

Notes: Use Clear Counter to set counts to zero.

Dependencies: Applies only to standard digital inputs configured with the counter or quadrature counter feature.

See Also: Get Counter (page G-48), Get & Clear Counter (page G-18), Clear Counter (page C-22), Start Continuous Square Wave (page S-94)

Stop Timer

Timing Action

Function:	To stop a timer variable.
Typical Use:	To stop timing an event.
Details:	• Once an up timer or a down timer has been stopped, its value is zero. If you stop a timer and move the value to a variable, you will always get 0.0. • To store the timer's value at the time it was stopped, or to be able to continue a timer, use the command Pause Timer instead.
Arguments:	**Argument 1** **Timer** Down Timer Variable Up Timer Variable
Standard Example:	Stop Timer *Timer* OVEN_TIMER *Down Timer Variable*
OptoScript Example:	**StopTimer(***Timer***)** `StopTimer(OVEN_TIMER);` This is a procedure command; it does not return a value.
Notes:	See "Timing Commands" in Chapter 10 of the *ioControl User's Guide* for more information on timers.
See Also:	Start Off-Pulse (page S-96), Continue Timer (page C-39), Pause Timer (page P-1), Set Down Timer Preset Value (page S-21), Set Up Timer Target Value (page S-86)

String Equal?

String Condition

Function: To compare two strings for equality.

Typical Use: To check passwords or barcodes for an exact match.

Details:

- Determines if strings in *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."
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- Functionally equivalent to the Test Equal Strings action.
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1	Argument 2
Is	**To**
String Literal	String Literal
String Variable	String Variable

Standard Example:

String Equal? *Is*	NEW_ENTRY	*String Variable*
To	PASSWORD	*String Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `==` operator.

```
if (NEW_ENTRY == PASSWORD) then
```

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- The example shown is only one way to use the `==` operator. For more information on using comparison operators and strings in OptoScript code, see Chapter 11 of the *ioControl User's Guide*
- Use String Equal to String Table Element? to compare with strings in a table.

See Also: Test Equal Strings (page T-3), String Equal to String Table Element? (page S-104)

String Equal to String Table Element?

String Condition

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."
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.
- A valid range for the *At Index* parameter (*Argument 2*) is zero to the table length (size).
- Functionally equivalent to the Test Equal Strings action.
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Is	**At Index**	**Of Table**
String Literal	Integer 32 Literal	String Table
String Variable	Integer 32 Variable	

Standard 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.

Is	NEW_BARCODE	*String Variable with Barcode*
String Equal to String Table Element?		
At Index	Loop_Index	*Integer 32 Variable*
Of Table	Current_Products	*String Table*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `==` operator.

```
if (NEW_BARCODE == Current_Products[Loop_Index]) then
```

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- The example shown is only one way to use the `==` operator. For more information on using comparison operators and strings in OptoScript code, see Chapter 11 of the *ioControl User's Guide*

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Test Equal Strings (page T-3), String Equal? (page S-103)

Subtract

Mathematical Action

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 [Value]	Argument 2 Minus	Argument 3 Put Result in
Analog Input	Analog Input	Analog Output
Analog Output	Analog Output	Down Timer Variable
Down Timer Variable	Down Timer Variable	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	Up Timer Variable
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Subtract

	Num_Widgets_to_Produce	*Integer 32 Variable*
Minus	Num_Widgets_Produced	*Integer 32 Variable*
Put Result in	Num_Widgets_Left_to_Make	*Integer 32 Variable*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `-` operator.

```
Num_Widgets_Left_to_Make = NuSm_Widgets_to_Produce -
Num_Widgets_Produced;
```

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In OptoScript code, the `-` operator has many uses. For more information on mathematical expressions in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.

Queue Errors: -13 = Overflow error—result too large.

See Also: Decrement Variable (page D-1), Add (page A-3)

Suspend Chart

Chart Action

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 task queue.
- A chart can suspend itself or any other chart.
- IMPORTANT: A chart that suspends itself may not do so immediately. Depending on activity in the control engine, the chart may continue for another command or two. To start another chart and immediately suspend the first chart, use the command Call Chart instead.
- 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 from the point at which they were suspended (using either Start Chart or Continue Chart), or they can be stopped (using Stop Chart).

Arguments:

Argument 1	Argument 2
Chart	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Standard Example:

Suspend Chart

Chart	CHART_B	*Chart*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

SuspendChart(*Chart***)**

```
STATUS = SuspendChart(CHART_B);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- This command should be used judiciously. It can take up to 100 ms for the chart to suspend. Use this command only when timing is not critical. Otherwise, instead of Suspend Chart, use a chart that runs continuously and uses subroutines for any kind of repetitive logic.
- See "Chart Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = success.

-5 = failure.

See Also: Start Chart (page S-93), Continue Chart (page C-38), Chart Suspended? (page C-11)

Suspend Chart on Error

Error Handling Action

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

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

Details:

- Since ioControl is a multitasking environment, an error handler chart cannot suspend another chart instantaneously with this command, because the error handler chart itself is executed periodically. The actual time required depends on how many charts are running simultaneously as well as on the priority of each.
- See the Errors Appendix in the *ioControl User's Guide* for a list of errors that may appear in the message queue.

Arguments:

Argument 1
Put Status in
Float Variable
Integer 32 Variable

Standard Example:

Suspend Chart on Error

Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`SuspendChartOnError()`

```
STATUS = SuspendChartOnError();
```

This is a function command; it returns one of the status codes listed below.

Notes:

- See “Error Handling Commands” and “Chart Commands” in Chapter 10 of the *ioControl User's Guide*.
- To get to each error in the message queue, the top error must be discarded, which brings the next error to the top. Use Remove Current Error and Point to Next Error to do this.

Status Codes:

0 = success

-5 = failure

See Also: Remove Current Error and Point to Next Error (page R-22), Get Error Count (page G-53), Stop Chart on Error (page S-100)

Tangent

Mathematical Action

Function: To derive the tangent of an angle.

Typical Use: Trigonometric function for computing angular rise.

Details:

- Computes the tangent (in radians) of *Argument 1* and places the result in *Argument 2*.
- Tangent produces a result that theoretically ranges from -infinity to +infinity, but is limited by the type of the argument.
- Computing a tangent at (pi / 2) ± (n * pi) yields unpredictable results, since ± infinity cannot be represented. Use Within Limits? to check for a valid *Argument 1* value before calling the Tangent command.
- Tangent is sin (angle) / cos (angle).

Arguments:

Argument 1 **Of**	**Argument 2** **Put Result in**
Analog Input	Analog Output
Analog Output	Down Timer Variable
Down Timer Variable	Float Variable
Float Literal	Integer 32 Variable
Float Variable	Up Timer Variable
Integer 32 Literal	
Integer 32 Variable	
Up Timer Variable	

Standard Example:

Tangent

Of	RADIANS	*Float Variable*
Put Result in	TANGENT	*Float Variable*

OptoScript Example:

Tangent(*Of*)

```
TANGENT = Tangent(RADIANS);
```

This is a function command; it returns the tangent of the angle. The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Mathematical Commands" in Chapter 10 of the *ioControl User's Guide*.
- To convert units of degrees to units of radians, divide degrees by 57.29578 (or 180 / pi).
- Use Arctangent if the tangent is known and the angle is desired.

See Also: Arctangent (page A-13), Cosine (page C-62), Sine (page S-91)

Test Equal

Logical Action

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*. The result is non-zero (True) if both values are the same, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument 3
0	0	True
-1	0	False
255	65280	False
22.22	22.22	True

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 [Value]	Argument 2 With	Argument 3 Put Result in
Analog Input	Analog Input	Digital Output
Analog Output	Analog Output	Float Variable
Digital Input	Digital Input	Integer 32 Variable
Digital Output	Digital Output	Up Timer Variable
Down Timer Variable	Down Timer Variable	
Float Literal	Float Literal	
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Test Equal

	TOP_LEVEL	*Integer 32 Variable*
With	1000	*Integer 32 Literal*
Put Result in	FLAG_AT_THE_TOP	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the Equal? command.

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- In many cases 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. Be careful when testing equality of floating point values, since the values must be *exactly* identical for a true result to occur. Consider using the following test:

```
AbsolutedValue(test_float – compare_float) < zero_tolerance
```

See Also: Test Greater (page T-4), Test Less (page T-6), Test Greater or Equal (page T-5), Test Less or Equal (page T-7), Test Not Equal (page T-8)

Test Equal Strings

String Action

Function: To compare two strings for equality.

Typical Use: To check passwords or barcodes for an exact match.

Details:

- Determines if *Argument 1* and *Argument 2* are equal and puts result in *Argument 3*. The result is non-zero (True) if both strings are exactly the same, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument 3
"OPTO"	"OPTO"	True
"OPTO"	"Opto"	False
"22"	"22"	True
"2 2"	"22"	False

- 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 action is functionally equivalent to the String Equal? condition.
- Quotes ("") are used in OptoScript code, but not in standard ioControl code.

Arguments:

Argument 1 **Compare**	Argument 2 **With**	Argument 3 **Put Result in**
String Literal	String Literal	Digital Output
String Variable	String Variable	Float Variable
		Integer 32 Variable

Standard 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. Quotes are shown for clarity only; do not use them in standard commands.

Test Equal Strings

Compare	Password	*String Variable*
With	"LISA"	*String Literal*
Put Result in	IS_AUTHORIZED	*Integer 32 Variable*

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

Compare	BARCODE	*String Variable*
With	BARCODE_FROM_LIST	*String Variable*
Put Result In	IS_IN_LIST	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the String Equal? command.

Notes:

- See "String Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use String Equal to String Table Element? to compare with strings in a table.

See Also: Compare Strings (page C-35), String Equal? (page S-103) String Equal to String Table Element? (page S-104)

Test Greater

Logical Action

Function: To determine if one value is greater than another.

Typical Use: To determine if an analog value is too high.

Details:

- Determines if *Argument 1* is greater than *Argument 2* and puts result in *Argument 3*. The result is non-zero (True) if *Argument 1* is greater than *Argument 2*, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument 3
0	0	False
-1	0	False
-1	-3	True
22.221	22.220	True

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 Is	Argument 2 Greater than	Argument 3 Put Result in
Analog Input	Analog Input	Digital Output
Analog Output	Analog Output	Float Variable
Digital Input	Digital Input	Integer 32 Variable
Digital Output	Digital Output	Up Timer Variable
Down Timer Variable	Down Timer Variable	
Float Literal	Float Literal	
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Test Greater

Is	TEMP	*Analog Input*
Greater than	1000	*Integer 32 Literal*
Put Result in	TEMP_COMPARISON	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the Greater? command.

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Consider using Test Greater or Equal instead.

See Also: Test Equal (page T-2), Test Less (page T-6), Test Greater or Equal (page T-5), Test Less or Equal (page T-7), Test Not Equal (page T-8)

Test Greater or Equal

Logical Action

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*. The result is non-zero (True) if *Argument 1* is greater than or equal to *Argument 2*, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument 3
0	0	False
-1	0	False
-1	-3	True
22.221	22.220	True

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 Is	Argument 2 > or =	Argument 3 Put Result in
Analog Input	Analog Input	Digital Output
Analog Output	Analog Output	Float Variable
Digital Input	Digital Input	Integer 32 Variable
Digital Output	Digital Output	Up Timer Variable
Down Timer Variable	Down Timer Variable	
Float Literal	Float Literal	
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Test Greater or Equal

Is	ROOM_TEMP	*Analog Input*
> or =	78.5000	*Float Literal*
Put Result in	FLAG_ROOM_TEMP_OK	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the Greater Than or Equal? command.

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- 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: Test Equal (page T-2), Test Less (page T-6), Test Greater (page T-4), Test Less or Equal (page T-7), Test Not Equal (page T-8)

Test Less

Logical Action

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*. The result is non-zero (True) if *Argument 1* is less than *Argument 2*, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument 3
0	0	False
-1	0	True
-1	-3	False
22.221	22.220	False

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 **Is**	**Argument 2** **Less than**	**Argument 3** **Put Result in**
Analog Input	Analog Input	Digital Output
Analog Output	Analog Output	Float Variable
Digital Input	Digital Input	Integer 32 Variable
Digital Output	Digital Output	Up Timer Variable
Down Timer Variable	Down Timer Variable	
Float Literal	Float Literal	
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Test Less

Is	TANK_LEVEL	*Analog Input*
Less than	FULL_TANK_LEVEL	*Integer 32 Variable*
Put Result in	FLAG_TANK_FILL_VALVE	*Digital Output*

OptoScript Example: For an OptoScript equivalent, see the Less? command.

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Consider using Test Less or Equal instead, since exact matches of non-integer types are rare.

See Also: Test Greater (page T-4), Test Equal (page T-2), Test Greater or Equal (page T-5), Test Less or Equal (page T-7), Test Not Equal (page T-8)

Test Less or Equal

Logical Action

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*. The result is non-zero (True) if *Argument 1* is less than or equal to *Argument 2*, 0 (False) otherwise. Examples:

Argument 1	Argument 2	Argument 3
0	0	True
-1	0	True
-1	-3	False
22.221	22.220	False

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 **Is**	**Argument 2** **< or =**	**Argument 3** **Put Result in**
Analog Input	Analog Input	Digital Output
Analog Output	Analog Output	Float Variable
Digital Input	Digital Input	Integer 32 Variable
Digital Output	Digital Output	Up Timer Variable
Down Timer Variable	Down Timer Variable	
Float Literal	Float Literal	
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Test Less or Equal

Is	TEMPERATURE	*Float Variable*
< or =	98.6	*Float Literal*
Put Result in	FLAG_TEMP_OK	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the Less Than or Equal? command.

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.
- 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: Test Greater (page T-4), Test Less (page T-6), Test Greater or Equal (page T-5), Test Equal (page T-2), Test Not Equal (page T-8)

Test Not Equal

Logical Action

Function: To determine if two values are different.

Typical Use: To check a variable against a standard.

Details:

- Determines if *Argument 1* is different from *Argument 2* and puts result in *Argument 3*. The result is non-zero (True) if *Argument 1* is not the same as *Argument 2*, 0 (False) if they are equal. Examples:

Argument 1	Argument 2	Argument 3
0	0	False
-1	0	True
255	65280	True
22.22	22.22	False

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 **Is**	**Argument 2** **Not Equal to**	**Argument 3** **Put Result in**
Analog Input	Analog Input	Digital Output
Analog Output	Analog Output	Float Variable
Digital Input	Digital Input	Integer 32 Variable
Digital Output	Digital Output	Up Timer Variable
Down Timer Variable	Down Timer Variable	
Float Literal	Float Literal	
Float Variable	Float Variable	
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	
Up Timer Variable	Up Timer Variable	

Standard Example:

Test Not Equal

Is	COUNTER_VALUE	*Integer 32 Variable*
Not Equal to	100	*Integer 32 Literal*
Put Result in	FLAG_NOT_DONE	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the Not Equal? command.

Notes:

- See “Logical Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Be careful when testing equality of floating point values, since the values must be *exactly* identical for a false result to occur. Consider using the following test:

```
AbsolutedValue(test_float – compare_float) > float_tolerance
```

See Also: Test Greater (page T-4), Test Less (page T-6), Test Greater or Equal (page T-5), Test Less or Equal (page T-7), Test Equal (page T-2)

Test Within Limits

Logical Action

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: A logical True (non-zero) is returned if within limits, otherwise a logical False (0) is returned.

Arguments:

Argument 1 **Is**	**Argument 2** **>=**	**Argument 3** **And <=**	**Argument 4** **Put Result in**
Analog Input	Float Literal	Float Literal	Float Variable
Analog Output	Float Variable	Float Variable	Integer 32 Variable
Down Timer Variable	Integer 32 Literal	Integer 32 Literal	
Float Literal	Integer 32 Variable	Integer 32 Variable	
Float Variable	Integer 64 Literal	Integer 64 Literal	
Integer 32 Literal	Integer 64 Variable	Integer 64 Variable	
Integer 32 Variable			
Integer 64 Literal			
Integer 64 Variable			
Up Timer Variable			

Standard Example:

Test Within Limits

Is	CURRENT_TEMP	*Float Variable*
>=	COLDEST_TEMP	*Float Variable*
And <=	HOTTEST_TEMP	*Float Variable*
Put Result in	RESULT	*Integer 32 Variable*

OptoScript Example: For an OptoScript equivalent, see the Within Limits? command.

See Also: Test Greater (page T-4), Test Less (page T-6), Test Greater or Equal (page T-5), Test Less or Equal (page T-7), Test Equal (page T-2), Test Not Equal (page T-8)

Timer Expired?

Timing Condition

Function: To determine if the specified timer has reached its target value. For down timers, the target value is zero. For up timers, it is the value set by the command Set Up Timer Target Value.

Typical Use: To determine if it is time to take an appropriate action.

Details: Evaluates True if the specified timer has reached its target value, False otherwise.

Arguments:

Argument 1
Is
Down Timer Variable
Up Timer Variable

Standard Example:

Timer Expired?		
Is	EGG_TIMER	*Down Timer Variable*

OptoScript Example:

`HasTimerExpired(`*Timer***`)`**

```
if (HasTimerExpired(EGG_TIMER)) then
```

This is a function command; it returns a non-zero (True) if the timer has expired, 0 (False) if not. The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Timing Commands" in Chapter 10 of the *ioControl User's Guide* for more information on using timers.
- This command can be used the same as Down Timer Expired? and Up Timer Target Time Reached?

See Also: Set Up Timer Target Value (page S-86), Set Down Timer Preset Value (page S-21), Start Off-Pulse (page S-96), Up Timer Target Time Reached? (page U-1), Down Timer Expired? (page D-21)

Transfer N Characters

Communication Action

Function: To send data from one communication handle to another.

Typical Uses: To store data from a serial module to a log file, or to take data from a log file and send it via FTP to another device on the network.

Details:

- This command essentially receives data on the source communication handle (*Argument 2*) and transmits it on the destination handle (*Argument 1*), without any processing. When you use this command, the data sent is not limited to the size of a string. This command is also faster than receiving data, storing it in a variable, and then transmitting it.
- If you need to process the data from the source handle before sending it to the destination handle, do not use this command. Instead, create a variable to receive the data from the source handle, process the data using any of the string commands, and then transmit it to the destination handle.
- To use this command, first use Open Outgoing Communication to both communication handles.
- Either use Get Number of Characters Waiting to determine how many bytes of data to transfer and enter that number in *Argument 3, Num Chars*, or enter -1 in *Argument 3* to transfer as many characters as are available.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
Destination Handle	**Source Handle**	**Num Chars**	**Put Status In**
Communication Handle	Communication Handle	Integer 32 Literal	Float Variable
		Integer 32 Variable	Integer 32 Variable

Standard Example:

Transfer N Characters

Destination Handle	UIO_3	*Communication Handle*
Source Handle	UIO_4	*Communication Handle*
Num Chars	3000	*Integer 32 Variable*
Put Status in	ERROR_CODE	*Integer 32 Variable*

OptoScript Example:

`TransferNChars(`*Destination Handle, Source Handle, Num Chars***`)`**

```
ERROR_CODE = TransferNChars(UIO_3, UIO_4, 3000);
```

This is a function command; it returns a zero (indicating success) or an error (indicating failure). The returned value can be consumed by a variable (as shown in the example) or by a control structure, mathematical expression, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- The two communication handles must be unique.
- See "CommunicationCommands" in Chapter 10 of the *ioControl User's Guide*.
- For receiving information using FTP communication handles, this command will only work following the Send Communication Handle Command (dir option) to retrieve directory information about the local or a remote FTP server. To retrieve a file from a remote FTP

server, use Send Communication Handle Command (get option) to bring the file into the local file system, then use a File communication handle to access the file locally.

Status Codes: 0 = Success

-3 = Buffer overrun or invalid length. The only negative number valid for *Argument 3* is -1.

-25 = Port not locked. Communication handles in *Argument 1* and *Argument 2* must be different. If trying to transfer characters to a file, may be insufficient file space.

-36 = Invalid command or feature not implemented for this type of communication handle in this version of firmware. To retrieve a file from a remote FTP server, use Send Communication Handle Command (*get* option) to bring the file into the local file system, then use a File communication handle to access the file locally.

-37 = Lock port timeout.

-38 = Send timeout.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-69 = Invalid parameter (null pointer) passed to command.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

See Also: Open Outgoing Communication (page O-4), Get Number of Characters Waiting (page G-101), Close Communication (page C-29)

Transmit Character

Communication Action

Function: To send a single character to the entity specified by the communication handle.

Typical Uses:
- To send a message to another device or file one character at a time.

Details:
- Character values sent are 0–255. Only the last eight bits are sent when the value is >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.
- With a File communication handle, the character is transmitted immediately.
- With any other communication handle, this command does not transmit the character. The character stays in the buffer until you use Transmit Newline or Transmit String to send it.

Arguments:

Argument 1	Argument 2	Argument 3
From	**Communication Handle**	**Put Status in**
Float Literal	Communication Handle	Float Variable
Float Variable		Integer 32 Variable
Integer 32 Literal		
Integer 32 Variable		

Standard Example:

Transmit Character

From	10	*Integer 32 Literal*
Communication Handle	UIO_4	*Communication Handle*
Put Status in	ERROR_CODE	*Integer 32 Variable*

OptoScript Example:

`TransmitChar(`*Character, Communication Handle***`)`**

```
ERROR_CODE = TransmitChar(10, UIO_4);
```

This is a function command; it returns one of the status codes listed below.

In OptoScript code, you can also use a character literal for Argument 1. For example, you could use `TransmitChar('a', UIO_4);` rather than having to use `TransmitChar(97, UIO_4);` making the code more readable. Unprintable character codes would still require a number, however.

Notes:
- See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide*.
- Use Transmit String instead when there are a lot of characters to send.

Status Codes:

0 = Success

-36 = Invalid command. Does not apply to the type of communication handle you are using.

-38 = Timeout. If you are using a File communication handle, you may have used a read-only parameter.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-69 = Invalid parameter (null pointer) passed to command.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

See Also: Transmit String (page T-22)

Transmit NewLine

Communication Action

Function: To send the message in the transmit buffer. No carriage return is appended.

Typical Use: For TCP/IP communication, to send characters that have been placed in the buffer using the Transmit Character command.

Details:

- **CAUTION:** The message could be sent and acknowledged but discarded by the destination with no error if the receiving end's buffer is full.
- If the communication handle does not use a buffer (for example, a File communication handle), this command has no effect.

Arguments:

Argument 1	Argument 2
Communication Handle	**Put Status in**
Communication Handle	Float Variable
	Integer 32 Variable

Standard Example:

Transmit NewLine

Communication Handle	UIO_4	*Communication Handle*
Put Status in	ERROR_CODE	*Integer 32 Variable*

OptoScript Example:

`TransmitNewLine(`*Communication Handle***`)`**

```
ERROR_CODE = TransmitNewLine(UIO_4);
```

This is a function command; it returns one of the status codes listed below.

Notes: See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.

Status Codes:

0 = Success

-36 = Invalid command. Does not apply to the type of communication handle you are using.

-37 = Lock port timeout.

-38 = Send timeout.

-42 = Invalid limit.

-69 = Invalid parameter (null pointer) passed to command.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

See Also: Transmit String (page T-22)

Transmit Numeric Table

Communication Action

Function: Sends a specific number of numeric table values to another entity, such as another control engine or a binary file.

Typical Use: Efficient method of writing binary data to a file.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
Length	**Start at Index**	**Of Table**	**Communication Handle**	**Put Status in**
Integer 32 Literal	Integer 32 Literal	Float Table	Communication Handle	Float Variable
Integer 32 Variable	Integer 32 Variable	Integer 32 Table		Integer 32 Variable
		Integer 64 Table		

Standard Example:

Transmit Numeric Table

Length	Table_length	*Integer 32 Variable*
Start at Index	0	*Integer 32 Literal*
Of Table	Peer_data_table	*Float Table*
Communication Handle	UIO_5	*Communication Handle*
Put Status in	Xmit_status	*Integer 32 Variable*

OptoScript Example:

`TransmitNumTable(`*Length, Start at Index, Of Table, Communication Handle***`)`**

```
Xmit_status = TransmitNumTable(Table_length, 0, Peer_data_table, UIO_5);
```

This is a function command; it returns one of the status codes listed below.

Notes: Use Transmit Character first to send a destination index, table ID, etc. if desired. These values could be sent as fixed length or carriage return delimited.

Dependencies: Must first use Open Outgoing Communication to establish a session, or (for TCP communication handles) Listen for Incoming Communication and Accept Incoming Communication to accept a session initiated by a TCP/IP peer. See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide* for more information.

Status Codes:

0 = Success

-36 = Invalid command. Does not apply to the type of communication handle you are using.

-37 = Lock port timeout.

-38 = Send timeout. If you are using a File communication handle, you may have used a read-only parameter.

-42 = Invalid limit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-69 = Invalid parameter (null pointer) passed to command.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

See Also: Receive Numeric Table (page R-16), Receive String Table (page R-21), Receive Pointer Table (page R-17), Transmit String (page T-22), Transmit Character (page T-13), Transmit String Table (page T-23), Transmit Pointer Table (page T-16), Transfer N Characters (page T-11)

Transmit Pointer Table

Communication Action

Function: Sends a specific number of pointer table values to another entity, such as another control engine or a file. (The values pointed to are transmitted, not the pointers themselves.)

Typical Use: Efficient method of data transfer to a file.

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**	**Argument 5**
Length	**Start at Index**	**Of Table**	**Communication Handle**	**Put Status in**
Integer 32 Literal	Integer 32 Literal	Pointer Table	Communication Handle	Float Variable
Integer 32 Variable	Integer 32 Variable			Integer 32 Variable

Standard Example:

Transmit Pointer Table

Length	Table_length	*Integer 32 Variable*
Start at Index	0	*Integer 32 Literal*
Of Table	Peer_data_table	*Pointer Table*
Communication Handle	UIO_5	*Communication Handle*
Put Status in	Xmit_status	*Integer 32 Variable*

OptoScript Example:

`TransmitPtrTable(`*Length, Start at Index, Of Table, Communication Handle***`)`**

```
Xmit_status = TransmitPtrTable(Table_length, 0, Peer_data_table, UIO_5);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- Use Transmit Character first to send a destination index, table ID, etc. if desired. These values could be sent as fixed length or carriage return delimited.
- Pointers in the table must not point to another table.
- Make sure that the tables used on both ends of the communication point to the same types and sizes of data. For example, if you transmit a table with pointers to a float, an integer, and a string with width 10, make sure the table on the receiving end is exactly the same.

Dependencies: Must first use Open Outgoing Communication to establish a session, or (for TCP communication handles) Listen for Incoming Communication and Accept Incoming Communication to accept a session initiated by a TCP/IP peer. See "Communication Commands" in Chapter 10 of the *ioControl User's Guide* for more information.

Status Codes: 0 = Success

-36 = Invalid command. Does not apply to the type of communication handle you are using.

-37 = Lock port timeout.

-38 = Send timeout.

-42 = Invalid limit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-69 = Invalid parameter (null pointer) passed to command.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

-29 = Wrong object type. Pointers in the table must point to strings, integers, or floats. Tables are not allowed.

See Also: Receive Numeric Table (page R-16), Receive String Table (page R-21), Receive Pointer Table (page R-17), Transmit String (page T-22), Transmit Character (page T-13), Transmit String Table (page T-23), Transmit Numeric Table (page T-15), Transfer N Characters (page T-11)

Transmit/Receive Mistic I/O Hex String

Communication Action

Function: Assists in sending custom commands using hex to a mistic I/O unit.

Typical Uses: Reading a group of 16 event latches from a multifunction I/O unit.

Details:

- Sends the command string, gets the response, and verifies the DVF (data verification field, such as a CRC). A zero result indicates the response was received and verified.
- A hex string representation (of a mistic command in "binary command format") is used to make the command string and the response string more readable to the user. The control engine will convert this hex string and append the DVF (data verification field, such as a CRC) appropriate for the I/O unit passed.
- The control engine can optionally pre-pend the required (by mistic's "binary command format") address and length bytes.

  Otherwise, if the option flag is 0, the command string passed should include those four characters at the beginning of the string.

  Set the option flag to 0 for backwards-compatibility with the old OptoControl commands: Transmit/Receive Mistic I/O Hex String with CRC and Transmit/Receive Mistic I/O Hex String with Checksum. The "with Checksum" or "with CRC" are no longer relevant since the control engine will use whatever I/O unit it is configured for.
- An option flag of 0 pre-pends no address info; it behaves like the OptoControl commands. A non-zero flag will cause the control engine to pre-pend the address of the board passed and the appropriate length of the mistic command passed.

Arguments:

Argument 1 **mistic Command**	Argument 2 **I/O Unit**	Argument 3 **Option Flag**	Argument 4 **mistic Response**	Argument 5 **Put status in**
String Literal	B100	Integer 32 Literal	String Variable	Float Variable
String Variable	B200	Integer 32 Variable		Integer 32 Variable
	B3000 (Analog)			
	B3000 (Digital)			
	G4A8R, G4RAX			
	G4D16R			
	G4D32RS			

Standard Example:

Transmit/Receive Mistic I/O Hex String

mistic Command	B3000_1	*String Variable*
I/O Unit	IO_COMMAND	*B3000 (Digital)*
Option Flag	0	*Integer 32 Literal*
mistic Response	RESPONSE	*String Variable*
Put Status in	RECV_STATUS	*Integer 32 Variable*

OptoScript Example:

`TransReceMisticIoHexString(`*Hex String, I/O Unit, Option Flag, Hex Response***`)`**

```
RECV_STATUS = TransReceMisticIoHexString(IO_Command, B3000_1, 0,
                                          Response);
```

This is a function command; it returns one of the status codes listed below.

Notes: Use Convert Hex String to Number when the response represents a count or bit pattern. The commands Convert Mistic I/O Hex String to Float and Convert Number to Mistic I/O Hex String may also come in handy.

Status Codes: 0 = Success.

-44 = String too short. The string passed was empty, or an odd number of characters. (To represent one byte as a hex string requires two characters, for example an 'f' is "66" so sending an odd number of characters is incorrect.)

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

Errors returned by the brain: See form #270, Mistic Protocol user's guide for more information on these errors. The mistic error code is shown in [square] brackets.

-1 = [1] Undefined command

-2 = [2] CRC or checksum mismatch

-531 = [3] Buffer overrun – too many characters received

-4 = [4] Powerup clear expected.

-70 = [5] Not enough chars received

-7 = [6] Communication watchdog timed out (activated).

-6 = [7] Specified data invalid error.

-35 = [9] Invalid point type for the command sent.

-94 = [10] Invalid event entry or command.

-42 = [11] Invalid time delay limit reached – digital only.

Queue Errors: -538 = The address string passed does not match the passed I/O unit's address. This will appear as a warning when using the option flag set to 0 if the address passed (represented by the first two characters in the mistic command string) does not match the address of the I/O unit.

The "errors returned by the brain" listed above (with the exception of the PUC error) will also appear in the message queue for each retry.

See Also: Convert Hex String to Number (page C-41), Convert Mistic I/O Hex String to Float (page C-45), Convert Number to Mistic I/O Hex String (page C-49)

Transmit/Receive String

Communication Action

Function: Sends a message, and then waits for an end-of-message delimited response when communicating via TCP.

Typical Use: Sending and receiving messages and data to/from other devices. via TCP/IP.

Details:
- See the Details section for Transmit String and Receive String. This command is the equivalent of using Transmit String followed by Receive String.
- If the response has multiple embedded end-of-message (EOM) characters, use Receive String to get each additional EOM-delimited section.
- Do not use this command with FTP or File communication handles.
- If the EOM-delimited string is longer than the destination string length, a -23 error is returned and as many characters as fit in the destination string are placed there. To see how many characters were received, use a Get Length command for the destination string. The characters remaining, minus the data just received, may be retrieved by a subsequent call to Receive String.

Arguments:

Argument 1	Argument 2	Argument 4	Argument 5
From	**Communication Handle**	**Put Result in**	**Put Status in**
String Literal	Communication Handle	String Variable	Float Variable
String Variable			Integer 32 Variable

Standard Example:

Transmit/Receive String

From	XMIT_MSG	*String Variable*
Communication Handle	UIO_4	*Communication Handle*
Put Result in	RECV_MSG	*String Variable*
Put Status in	TR_STATUS	*Integer 32 Variable*

OptoScript Example:

`TransmitReceiveString(`*String, Communication Handle, Put Result in***`)`**

```
TR_STATUS = TransmitReceiveString(XMIT_MSG, UIO_4, RECV_MSG);
```

This is a function command; it returns one of the status codes listed below.

Notes:
- Use Move String, Append String to String or Append Character to String to build the string to send.
- Use Receive String or Receive N Characters in the destination device followed by Transmit String for the reply.
- See more details in Transmit String and Receive String.
- See "Communication Commands" in Chapter 10 of the *ioControl User's Guide*.

Dependencies:
- Must first use Open Outgoing Communication to establish a session, or (for TCP communication handles) Accept Incoming Communication to accept a session initiated by a TCP/IP peer.
- After using Open Outgoing Communication, use the Set End-Of-Message Terminator command to change the default of 13 (carriage return) if needed.

Status Codes: 0 = Success

-23 = Destination string too short.

-37 = Lock port timeout.

-38 = Send timeout.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-58 = No data received. May have timed out if no response in 10 seconds. Check I/O unit power.

-76 = At end of file.

-69 = Invalid parameter (null pointer) passed to command.

-408 = Error during file access. For example, attempted to write to a file opened for reading.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Transmit String (page T-22), Receive String (page R-19), Open Outgoing Communication (page O-4), Get End-Of-Message Terminator (page G-51), Set End-Of-Message Terminator (page S-22), Transfer N Characters (page T-11)

Transmit String

Communication Action

Function: To send a message to another entity.

Typical Use: To write a string to a text file.

Details:

- For communication handles that use buffers (for example, TCP), if the transmit buffer of the specified handle has any characters in it (previously placed there by Transmit Character), they will be sent first, followed by any characters that may be in the string. If the string is empty, the transmit buffer contents will be sent. If both the string and the transmit buffer are empty, the packet will not be sent.
- When using a file, the string is immediately written to the file.

Arguments:

Argument 1	Argument 2	Argument 3
From	**Communication Handle**	**Put Status in**
String Literal	Communication Handle	Float Variable
String Variable		Integer 32 Variable

Standard Example:

Transmit String

From	XMIT_MSG	*String Variable*
Communication Handle	UIO_5	*Communication Handle*
Put Status in	COMM_STATUS	*Integer 32 Variable*

OptoScript Example:

`TransmitString(`*String, Communication Handle***`)`**

```
COMM_STATUS = TransmitString(XMIT_MSG, UIO_5);
```

This is a function command; it returns one of the status codes listed below.

Dependencies:

- Must first use Open Outgoing Communication to establish a session, or (for TCP communication handles) Accept Incoming Communication to accept communication initiated by a TCP/IP peer.
- See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide*.

Status Codes:

0 = Success

-37 = Lock port timeout.

-38 = Send timeout. For example, attempted to write to a file opened for reading.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-69 = Invalid parameter (null pointer) passed to command.

-408 = Error during file access. For example, attempted to write to a file open for reading.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

Note: This command does not automatically append the current end-of-message (EOM) delimiter for the communication handle to the end of the string. Only the string passed will be transmitted. If the EOM is needed (for example, to be received on the other end using Receive String), use Append Character to String to append the EOM to the string.

See Also: Receive String (page R-19), Transmit/Receive String (page T-20), Open Outgoing Communication (page O-4), Append Character to String (page A-9), Get End-Of-Message Terminator (page G-51), Set End-Of-Message Terminator (page S-22)

Transmit String Table

Communication Action

Function: Sends a specific number of string table values to another entity, such as another control engine or a file.

Typical Use: Efficient method of writing delimited data to a file.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
Length	**Start at Index**	**Of Table**	**Communication Handle**	**Put Status in**
Integer 32 Literal	Integer 32 Literal	String Table	Communication Handle	Float Variable
Integer 32 Variable	Integer 32 Variable			Integer 32 Variable

Standard Example:

Transmit String Table

Length	Table_length	*Integer 32 Variable*
Start at Index	0	*Integer 32 Literal*
Of Table	Peer_data_table	*String Table*
Communication Handle	UIO_5	*Communication Handle*
Put Status in	Xmit_status	*Integer 32 Variable*

OptoScript Example:

TransmitStrTable(*Length, Start at Index, Of Table, Communication Handle***)**

```
Xmit_status = TransmitStrTable(Table_length, 0, Peer_data_table, UIO_5);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- Each string that is transmitted will be followed by the current end-of-message character for this communication handle.
- Use Set End-of-Message Terminator to specify the end-of-message character to use. The default is 13 (carriage return).
- Use Transmit Character first to send a destination index, table ID, etc. if desired. These values could be sent as fixed length or carriage return delimited.

Dependencies: Must first use Open Outgoing Communication to establish a session, or (for TCP communication handles) Listen for Incoming Communication and Accept Incoming Communication to accept a session initiated by a TCP/IP peer. See “Communication Commands” in Chapter 10 of the *ioControl User’s Guide* for more information.

Status Codes: 0 = Success

-3 = Invalid length. Length (Argument 1) is greater than number of elements in the source table.

-12 = Invalid table index. Index was negative or greater than or equal to the table size.

-37 = Lock port timeout.

-38 = Send timeout. For example, attempted to write to a file opened for reading.

-42 = Invalid limit.

-52 = Invalid connection—not opened. The communication handle may have been closed by a previous command that failed. Check status codes returned on other communication handle commands.

-69 = Invalid parameter (null pointer) passed to command.

-531 = Buffer full. You may be attempting to send data to the serial port faster than the port can send and buffer data. Try a faster baud rate or a delay between Transfer/Transmit commands.

Queue Errors: -12 = Invalid table index value—index was negative or greater than or equal to the table size.

See Also: Receive String Table (page R-21), Receive Numeric Table (page R-16), Receive Pointer Table (page R-17), Transmit String (page T-22), Transmit Character (page T-13), Transmit Pointer Table (page T-16), Transmit Numeric Table (page T-15), Set End-Of-Message Terminator (page S-22)Transfer N Characters (page T-11)

Truncate

Mathematical Action

Function: Discards the fractional part of a number without changing the whole part.

Typical Use: To separate the whole part of a number from the fractional part.

Arguments:

Argument 1 **[Value]**	**Argument 2** **Put Result in**
Down Timer Variable	Down Timer Variable
Float Literal	Float Variable
Float Variable	Integer 32 Variable
Up Timer Variable	Integer 64 Variable
	Up Timer Variable

Standard Example:

Truncate		
	Flow_Total_Raw	*Float Variable*
Put Result in	Flow_Total_Integer	*Integer 32 Variable*

OptoScript Example: **`Truncate(`*Value*`)`**

```
Flow_Total_Integer = Truncate(Flow_Total_Raw);
```

This is a function command; it returns the whole part of the truncated number.

Notes: Subtracting the resulting integer from the float will remove the whole part from the fractional part.

See Also: Round (page R-24)

Turn Off

Digital Point Action

Function: To turn off a standard digital output point.

Typical Use: To deactivate devices connected to digital outputs, such as motors, pumps, lights, etc.

Details:

- Standard digital only. For high-density digital, see Turn Off HDD Module Point.
- Turns off the specified output.
- The output will remain off until directed otherwise.

Arguments:

Argument 1
[Value]
Digital Output

Standard Example:

Turn Off
The_Lights *Digital Output*

OptoScript Example:

TurnOff(*Output***)**

```
TurnOff(The_Lights);
```

This is a procedure command; it does not return a value.

In OptoScript code, you could also assign the output a zero value to turn it off:

```
The_Lights = 0;
```

Notes:

- To cause an output on one I/O unit to assume the state of an input on another I/O unit, use Move in standard commands or an assignment in OptoScript code.
- Use NOT to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Speed Tip: Use Set Digital-64 I/O Unit from MOMO Masks or Set Mixed I/O Unit from MOMO Masks to turn off all outputs at once.

Dependencies: If the output point or the I/O unit is disabled, no action will occur at the output point (XVAL). The IVAL, however, will be updated.

See Also: Set Digital-64 I/O Unit from MOMO Masks (page S-19), Set Mixed I/O Unit from MOMO Masks (page S-55), Turn On (page T-27)

Turn Off HDD Module Point

High Density Digital Module Action

Function: To turn off a specific point on a high-density digital output module.

Typical Use: To turn off one point only.

Details: Works only on high-density digital output modules, not on standard digital output modules.

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Module Number**	Argument 3 **Point Number**	Argument 4 **Put Status In**
SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64 SNAP-PAC-R1 SNAP-PAC-R2	Integer 32 Literal] Integer 32 Variable	Integer 32 Literal Integer 32 Variable	Integer 32 Variable

Standard Example:

Turn Off HDD Module Point

I/O Unit	Installation_42	*SNAP-ENET-S64*
Module Number	8	*Integer 32 Literal*
Point Number	Meter	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

OptoScript Example:

`TurnOffHddModulePoint(`*I/O Unit, Module Number, Point Number***`)`**

```
Status_Code = TurnOffHddModulePoint(Installation_42, 8, Meter);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To turn on or off several points at once, use Set HDD Module from MOMO Masks.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Turn On HDD Module Point (page T-28), Set HDD Module from MOMO Masks (page S-23)

Turn On

Digital Point Action

Function: To turn on a standard digital output point.

Typical Use: To activate devices connected to digital outputs, such as motors, pumps, lights, etc.

Details:
- Standard digital only. For high-density digital, see Turn On HDD Module Point.
- Turns on the specified output.
- The output will remain on until directed otherwise.

Arguments:

Argument 1
[Value]
Digital Output

Standard Example:

Turn On
INLET_VALVE *Digital Output*

OptoScript Example:

`TurnOn(`*Output***`)`**

```
TurnOn(INLET_VALVE);
```

This is a procedure command; it does not return a value.

In OptoScript code, you could also assign the output any non-zero value to turn it on:

```
INLET_VALVE = -1;
```

Notes:
- To cause an output on one I/O unit to assume the state of an input on another I/O unit, use Move in standard commands or an assignment in OptoScript code.
- Use NOT to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Speed Tip: Use Set Digital-64 I/O Unit from MOMO Masks or Set Mixed I/O Unit from MOMO Masks to turn on all outputs at once.

Dependencies: If the output point or the I/O unit is disabled, no action will occur at the output point (XVAL). The IVAL, however, will be updated.

See Also: Set Digital-64 I/O Unit from MOMO Masks (page S-19), Set Mixed I/O Unit from MOMO Masks (page S-55), Turn Off (page T-25)

Turn On HDD Module Point

High Density Digital Module Action

Function: To turn on a specific point on a high-density digital output module.

Typical Use: To turn on one point only.

Details: Works only on high-density digital output modules, not on standard digital output modules.

Arguments:

Argument 1	Argument 2	Argument 3	Argument 4
I/O Unit	**Module Number**	**Point Number**	**Put Status In**
SNAP-B3000-ENET,	Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
SNAP-ENET-RTC	Integer 32 Variable	Integer 32 Variable	
SNAP-UP1-ADS			
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Turn On HDD Module Point

I/O Unit	Installation_42	*SNAP-ENET-S64*
Module Number	8	*Integer 32 Literal*
Point Number	Meter	*Integer 32 Variable*
Put Status in	Status_Code	*Integer 32 Variable*

OptoScript Example:

`TurnOnHddModulePoint(`*I/O Unit, Module Number, Point Number***`)`**

```
Status_Code = TurnOnHddModulePoint(Installation_42, 8, Meter);
```

This is a function command; it returns one of the status codes shown below.

Notes:

- To turn on or off several points at once, use Set HDD Module from MOMO Masks.
- See "High Density Digital Module Commands" in Chapter 10 of the *ioControl User's Guide*, and see form #1547, the *SNAP High-Density Digital Module User's Guide*.

Status Codes:

0 = Success

-43 = Received a NACK from the I/O unit.

-58 = No data received. Make sure I/O unit has power.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Turn Off HDD Module Point (page T-26), Set HDD Module from MOMO Masks (page S-23)

U

Up Timer Target Time Reached?

Timing Condition

Function: To check if an up timer has reached its target time.

Typical Use: Used to go to the next step in a sequential process.

Details:
- Up timers do not stop timing when they reach their target value.
- Use the Set Up Timer Target Value command to set the target time.

Arguments:

Argument 1
Up Timer
Up Timer Variable

Standard Example:

Up Timer Target Time Reached?
Up Timer OVEN_TIMER *Up Timer Variable*

OptoScript Example:

`HasUpTimerReachedTargetTime(`*Up Timer***`)`**

```
if (HasUpTimerReachedTargetTime(OVEN_TIMER)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes: See "Timing Commands" in Chapter 10 of the *ioControl User's Guide* for more information on using timers.

See Also: Start Off-Pulse (page S-96), Stop Timer (page S-102), Continue Timer (page C-39), Pause Timer (page P-1), Set Up Timer Target Value (page S-86)

V

Variable False?

Logical Condition

Function: To determine if the specified variable is zero.

Typical Use: To determine if further processing should take place.

Details: Evaluates True if the value of the integer variable is zero, False otherwise. False is defined as zero.

Arguments:

Argument 1
Is
Float Variable
Integer 32 Variable
Integer 64 Variable

Standard Example:

Variable False?	*Is*	Pressure_Difference	*Integer 32 Variable*

OptoScript Example:

`IsVariableFalse(`*Variable***`)`**

```
if (IsVariableFalse(Pressure_Difference)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

A shorter way to achieve the same result in OptoScript code is to use the following:

```
if (not Pressure_Difference) then
```

See Also: Variable True? (page V-2)

Variable True?

Logical Condition

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 value of the integer is not zero, False otherwise. True is defined as any non-zero value.

Arguments:

Argument 1
Is
Float Variable
Integer 32 Variable
Integer 64 Variable

Standard Example:

Variable True?		
Is	Pressure_Difference	*Integer 32 Variable*

OptoScript Example:

`VariableTrue(`*Variable***`)`**

```
if (IsVariableTrue(Pressure_Difference)) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

A shorter way to achieve the same result in OptoScript code is to use the following:

```
if (Pressure_Difference) then
```

See Also: Variable False? (page V-1)

Verify Checksum on String

String Action

Function: To check the validity of a received message.

Typical Use: Ensuring the integrity of the data in a message prior to using it.

Details:

- Checksum type is eight-bit.
- The *Start Value* is also known as the "seed." It is usually zero.
- All characters except the last byte are included in the verification.
- The last byte must be the checksum.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Verify Checksum on String

Start Value	0	*Integer 32 Literal*
On String	RESPONSE_MSG	*String Variable*
Put Status In	CKSUM_STATUS	*Integer 32 Variable*

OptoScript Example:

`VerifyChecksumOnString(`*Start Value, On String***`)`**

```
CKSUM_STATUS = VerifyChecksumOnString(0, RESPONSE_MSG);
```

This is a function command; it returns one of the status codes listed below.

Status Codes:

0 = No error; valid checksum.

-2 = Invalid checksum; checksum verification failed.

-44 = String too short or string was empty.

Notes: The checksum used by this command is an 8-bit (one byte) value. The method used to calculate the checksum is:

1 Take the numerical sum of the ASCII numerical representation of each character in the string.

2 Divide the result by 256.

3 The integer remainder is the 8-bit checksum.

See Also: Generate Checksum on String (page G-1)

Verify Forward CCITT on String

String Action

Function: To check the validity of a received message.

Typical Use: Ensuring the integrity of the data in a message prior to using it.

Details:

- CRC type is 16-bit forward CCITT.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- All characters except the last two are included in the verification.
- The last two characters must be the CRC.

Arguments:

Argument 1	**Argument 2**	**Argument 3**
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Verify Forward CCITT on String

Start Value	-1	*Integer 32 Literal*
On String	RESPONSE_MSG	*String Variable*
Put Status In	CRC_STATUS	*Integer 32 Variable*

OptoScript Example:

`VerifyForwardCcittOnString(`*Start Value, On String***`)`**

```
CRC_STATUS = VerifyForwardCcittOnString(-1, RESPONSE_MSG);
```

This is a function command; it returns one of the status codes listed below.

Status Codes:

0 = No error; valid checksum.

-2 = Invalid checksum; checksum verification failed.

-44 = String too short or string was empty.

See Also: Verify Reverse CCITT on String (page V-6), Generate Forward CCITT on String (page G-3)

Verify Forward CRC-16 on String

String Action

Function: To check the validity of a received message.

Typical Use: Ensuring the integrity of the data in a message prior to using it.

Details:

- CRC type is 16-bit forward.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- All characters except the last two are included in the verification.
- The last two characters must be the CRC.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Verify Forward CRC-16 on String

Start Value	-1	*Integer 32 Literal*
On String	RESPONSE_VSS	*String Variable*
Put Status in	CRC_STATUS	*Integer 32 Variable*

OptoScript Example:

VerifyForwardCrc16OnString(*Start Value, On String***)**

```
CRC_STATUS = VerifyForwardCrc16OnString(-1, RESPONSE_VSS);
```

This is a function command; it returns one of the status codes listed below.

Status Codes:

0 = No error; valid checksum.

-2 = Invalid checksum; checksum verification failed.

-44 = String too short or string was empty.

See Also: Verify Reverse CRC-16 on String (page V-7), Generate Forward CRC-16 on String (page G-4)

Verify Reverse CCITT on String

String Action

Function: To check the validity of a received message.

Typical Use: Ensuring the integrity of the data in a message prior to using it.

Details:

- CRC type is 16-bit reverse CCITT.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- All characters except the last two are included in the verification.
- The last two characters must be the CRC.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Verify Reverse CCITT on String

Start Value	-1	*Integer 32 Literal*
On String	RESPONSE_MSG	*String Variable*
Put Status in	CRC_STATUS	*Integer 32 Variable*

OptoScript Example:

`VerifyReverseCcittOnString(`*Start Value, On String***`)`**

```
CRC_STATUS = VerifyReverseCcittOnString(-1, RESPONSE_MSG);
```

This is a function command; it returns one of the status codes listed below.

Status Codes:

0 = No error; valid checksum.

-2 = Invalid checksum; checksum verification failed.

-44 = String too short or string was empty.

See Also: Verify Forward CCITT on String (page V-4), Generate Reverse CCITT on String (page G-7)

Verify Reverse CRC-16 on String

String Action

Function: To check the validity of a received message.

Typical Use: Ensuring the integrity of the data in a message prior to using it.

Details:
- CRC type is 16-bit reverse.
- The *Start Value* is also known as the "seed." It is usually zero or -1.
- All characters except the last two are included in the verification.
- The last two characters must be the CRC.

Arguments:

Argument 1	Argument 2	Argument 3
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Standard Example:

Verify Reverse CRC-16 on String

Start Value	-1	*Integer 32 Literal*
On String	RESPONSE_MSG	*String Variable*
Put Status in	CRC_STATUS	*Integer 32 Variable*

OptoScript Example:

`VerifyReverseCrc16OnString(`*Start Value, On String***`)`**

```
CRC_STATUS = VerifyReverseCrc16OnString(-1, RESPONSE_MSG);
```

This is a function command; it returns one of the status codes listed below.

Status Codes:

0 = No error; valid checksum.

-2 = Invalid checksum; checksum verification failed.

-44 = String too short or string was empty.

See Also: Verify Forward CRC-16 on String (page V-5), Generate Reverse CRC-16 on String (page G-8)

W

Within Limits?

Logical Condition

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*. Evaluates True if *Argument 1* falls between *Argument 2* and *Argument 3* or equals either value. Evaluates False if *Argument 1* is less than *Argument 2* or 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

Arguments:

Argument 1	Argument 2	Argument 3
Is	**> =**	**And < =**
Analog Input	Float Literal	Float Literal
Analog Output	Float Variable	Float Variable
Down Timer Variable	Integer 32 Literal	Integer 32 Literal
Float Literal	Integer 32 Variable	Integer 32 Variable
Float Variable	Integer 64 Literal	Integer 64 Literal
Integer 32 Literal	Integer 64 Variable	Integer 64 Variable
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		
Up Timer Variable		

Standard 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.

Is	Current_Temp	*Float Variable*
Within Limits?		
>=	Coldest_Temp	*Float Variable*
And <=	Hottest_Temp	*Float Variable*

OptoScript Example: **`IsWithinLimits(`***Value, Low Limit, High Limit***`)`**

```
if IsWithinLimits(Current_Temp, Coldest_Temp, Hottest_Temp) then
```

This is a function command; it returns a value of true (non-zero) or false (0). The returned value can be consumed by a control structure (as in the example shown) or by a variable, I/O point, etc. See Chapter 11 of the *ioControl User's Guide* for more information.

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*.
- Use to replace two conditions: Less Than or Equal? and Greater Than or Equal?

See Also: Less Than or Equal? (page L-3) Greater Than or Equal? (page G-148)

Write I/O Unit Configuration to EEPROM

I/O Unit Action

Function: Stores all point features, watchdog settings, and other configurations to flash memory (EEPROM) at the I/O unit.

Typical Use: Allows the I/O unit to be fully functional at powerup. No further configuration by a control engine is needed.

Details:

- Instead of using this command in the strategy, it is better to store configurations to flash using ioManager (see the *ioManager User's Guide* for instructions) or using ioControl in Debug mode (see the *ioControl User's Guide*).
- This command takes about two seconds to complete and causes the connection to the I/O unit to be closed. If this command is used in the strategy, it should be placed where it will execute just once each time the program runs—typically in the Powerup chart *after* all special configuration commands are sent to the I/O unit. After a delay, use Enable Communication to I/O Unit to open the connection again.
- **CAUTION:** If you use this command in a strategy, make certain it is not in a loop. You can literally wear out the hardware if you write to flash too many times.

Arguments:

Argument 1
On I/O Unit
B100
B200
B3000 (Analog)
B3000 (Digital)
G4A8R, G4RAX
G4D16R
G4D32RS
SNAP-ENET-D64
SNAP-UP1-D64
SNAP-UP1-M64
SNAP-ENET-S64
SNAP-B3000-ENET, SNAP-ENET-RTC
SNAP-UP1-ADS
SNAP-PAC-R1
SNAP-PAC-R2

Standard Example:

Write I/O Unit Configuration to EEPROM

On I/O Unit	FURNACE_CONTROL	*SNAP-UP1-ADS*

OptoScript Example:

WriteIoUnitConfigToEeprom(*On I/O Unit***)**

```
WriteIoUnitConfigToEeprom(FURNACE_CONTROL);
```

This is a procedure command; it does not return a value.

Queue Errors:

-52 = Invalid connection—not opened

-534 = Attempts to communicate with I/O unit failed. Make sure I/O unit is turned on.

Write Number to I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Write a value from an integer 32 or float variable into an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map address.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioManager or ioControl. The control engine must be on the I/O unit or connected to the I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use the Scratch Pad commands instead (Set I/O Unit Scratch Pad Integer 32 Element and related commands).

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
I/O Unit	**Mem Address**	**From**	**Put Status in**
SNAP-ENET-D64	Integer 32 Literal	Float Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	Float Variable	
SNAP-UP1-M64		Integer 32 Literal	
SNAP-ENET-S64		Integer 32 Variable	
SNAP-B3000-ENET,		Integer 64 Literal	
SNAP-ENET-RTC		Integer 64 Variable	
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Write Number to I/O Unit Memory Map

I/O Unit	MYIOUNIT	*SNAP-UP1-ADS*
Mem Address	0xFFFFFFFF	*Integer 32 Literal*
From	MYINTVAR	*Integer 32 Variable*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`WriteNumToIoUnitMemMap(`*I/O Unit, Mem Address, Variable***`)`**

```
STATUS = WriteNumToIoUnitMemMap(MYIOUNIT, 0xFFFFFFFF, MYINTVAR);
```

This is a function command; it returns one of the status codes listed below.

Notes:

- Use hex integer display in ioControl for easy entering of memory map addresses. Be sure there are no spaces within the memory map address.
- The control engine does not convert the variable type to match the area of memory map being written to. The control engine has no knowledge of which memory map areas are integers and which are floats. You must write the correct type of data to the specified memory map address.

  For example, if you are using the SNAP PID module (SNAP-PID-V), use an integer to write the setpoint, which is in counts, and use a float to write the analog output. As another example, unpredictable results would occur if you try to write an integer 32 variable to the analog point area of the memory map. Use a float variable instead. See the *SNAP Ethernet-Based*

I/O Units Protocols and Programming Guide (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

Status Codes: 0 = Success

-36 = Tried to write a float value to a memory map address that takes only integer values.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-56 = Invalid memory map address or read-only address.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Write Numeric Table to I/O Unit Memory Map (page W-4), Read Numeric Table from I/O Unit Memory Map (page R-7), Read Number from I/O Unit Memory Map (page R-5), Set I/O Unit Scratch Pad Float Element (page S-32), Set I/O Unit Scratch Pad Integer 32 Element (page S-36)

Write Numeric Table to I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Write a range of values from an integer 32 or float table into an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map address.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioManager or ioControl. The control engine must be on the I/O unit or connected to the I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use the Scratch Pad commands instead (Set I/O Unit Scratch Pad Integer 32 Table and related commands).
- *Argument 1*, Length, is the number of table elements and also the length of data in the memory map in quads (groups of four bytes).
- *Argument 4*, Mem address, includes only the last eight hex digits (four bytes) of the memory map address (the lower 32 bits).

Arguments:

Argument 1	**Argument 2**	**Argument 3**	**Argument 4**
Length	**Start Index**	**I/O Unit**	**Mem Address**
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64	Integer 32 Variable
		SNAP-UP1-M64	
		SNAP-ENET-S64	
		SNAP-B3000-ENET,	
		SNAP-ENET-RTC	
		SNAP-UP1-ADS	
		SNAP-PAC-R1	
		SNAP-PAC-R2	

Argument 5	**Argument 6**
From	**Put Status in**
Float Table	Integer 32 Variable
Integer 32 Table	

Standard Example:

Write Numeric Table to I/O Unit Memory Map

Length	0x10	*Integer 32 Literal*
Start Index	0x5	*Integer 32 Literal*
I/O Unit	MYIOUNIT	*SNAP-UP1-ADS*
Mem Address	0xFFFFFFFF	*Integer 32 Literal*
From	MYINTTABLE	*Integer 32 Table*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

WriteNumTableToIoUnitMemMap(*Length, Start Index, I/O Unit, Mem Address, Table***)**

```
STATUS = WriteNumTableToIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYINTTABLE);
```

This is a function command; it returns one of the status codes listed below.

In OptoScript, you can use hex in some arguments and decimal in others, for example:

```
STATUS = WriteNumTableToIoUnitMemMap(16, 5, MYIOUNIT, 0xFFFFFFFF,
MYINTTABLE);
```

Notes:

- Use hex integer display for easy entering of memory map addresses. When you display integers in hex, note that the length of data and start index arguments are also in hex.
- The control engine does not convert the table type to match the area of the memory map being written to. The control engine has no knowledge of which memory map areas are integers and which are floats. You must write the correct type of data to the specified memory map address.

  For example, unpredictable results would occur if you try to write an integer 32 table to the analog bank area of the memory map. A float table should be used instead. See the *SNAP Ethernet-Based I/O Units Protocols and Programming Guide* (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = Success

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-56 = Invalid memory map address or read-only address.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Read Number from I/O Unit Memory Map (page R-5), Read Numeric Table from I/O Unit Memory Map (page R-7), Write Number to I/O Unit Memory Map (page W-3), Set I/O Unit Scratch Pad Float Table (page S-34), Set I/O Unit Scratch Pad Integer 32 Table (page S-38)

Write String Table to I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Write a range of values from a string table into the Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioManager or ioControl. The control engine must be on the I/O unit or connected to the I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use the Scratch Pad commands instead (Set I/O Unit Scratch Pad String Table and related commands).
- *Argument 1*, Length, is the number of table elements.
- *Argument 4*, Mem address, includes only the last eight digits of the memory map address (the lower 32 bits).
- This command treats strings like chunks of binary data. Each string must be divisible by 4, or you receive a -70 error. Strings are simply appended together and written to the memory map location specified in *Argument 4*.

Arguments:

Argument 1 **Length**	Argument 2 **Start Index**	Argument 3 **I/O Unit**	Argument 4 **Mem Address**
Integer 32 Literal	Integer 32 Literal	SNAP-ENET-D64	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable	SNAP-UP1-D64	Integer 32 Variable
		SNAP-UP1-M64	
		SNAP-ENET-S64	
		SNAP-B3000-ENET,	
		SNAP-ENET-RTC	
		SNAP-UP1-ADS	
		SNAP-PAC-R1	
		SNAP-PAC-R2	

Argument 5 **From**	Argument 6 **Put Status in**
String Table	Integer 32 Variable

Standard Example:

Write String Table to I/O UnitMemory Map

Length	0x10	*Integer 32 Literal*
Start Index	0x5	*Integer 32 Literal*
I/O Unit	MYIOUNIT	*SNAP-UP1-ADS*
Mem Address	0xFFFFFFFF	*Integer 32 Literal*
From	MYSTRINGTABLE	*String Table*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`WriteStrTableToIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem Address, Table***`)`**

```
STATUS = WriteStrTableToIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYSTRINGTABLE);
```

This is a function command; it returns one of the status codes listed below.

In OptoScript, you can use hex in some arguments and decimal in others, for example:

```
STATUS = WriteStrTableToIoUnitMemMap(16, 5, MYIOUNIT, 0xFFFFFFFF,
MYSTRINGTABLE);
```

Notes:

- Use hex integer display for easy entering of memory map addresses. When you display integers in hex, note that the length of data and start index arguments are also in hex.
- The control engine does not convert the table type to match the area of the memory map being written to. The control engine has no knowledge of which memory map areas are strings and which are other formats. You must write the correct type of data to the specified memory map address.

  For example, unpredictable results would occur if you try to write a string table to the analog bank area of the memory map. A float table should be used instead. See the *SNAP Ethernet-Based I/O Units Protocols and Programming Guide* (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = Success

-3 = Invalid length. Length must be greater than zero.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-56 = Invalid memory map address or read-only address.

-70 = Not enough data supplied. Each string must be divisible by 4.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also:

Read String from I/O Unit Memory Map (page R-9), Read String Table from I/O Unit Memory Map (page R-11), Write String to I/O Unit Memory Map (page W-9), Set I/O Unit Scratch Pad String Table (page S-41), Set I/O Unit Scratch Pad String Element (page S-40)

Write String to I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Write a value from a string variable into an Opto 22 SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O memory map address.

Typical Use: To access areas of the memory map not directly supported by ioControl.

Details:

- To use this command with a controller (such as a SNAP-LCE or SNAP-PAC-S1), create an I/O Unit of the type SNAP-UP1-M64 Unit with the controller's IP address.
- This command works with SNAP Ultimate, SNAP Ethernet, and SNAP Simple I/O units that have been configured in ioManager or ioControl. The control engine must be on the I/O unit or connected to the I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use the Scratch Pad commands instead (Set I/O Unit Scratch Pad String Element and related commands).

Arguments:

Argument 1 **I/O Unit**	Argument 2 **Mem Address**	Argument 3 **From**	Argument 4 **Put Status in**
SNAP-ENET-D64	Integer 32 Literal	String Literal	Integer 32 Variable
SNAP-UP1-D64	Integer 32 Variable	String Variable	
SNAP-UP1-M64			
SNAP-ENET-S64			
SNAP-B3000-ENET,			
SNAP-ENET-RTC			
SNAP-UP1-ADS			
SNAP-PAC-R1			
SNAP-PAC-R2			

Standard Example:

Write String to I/O Unit Memory Map

I/O Unit	MYIOUNIT	*SNAP-UP1-ADS*
Mem Address	0xFFFFFFFF	*Integer 32 Literal*
From	MYSTRINGVAR	*String Variable*
Put Status in	STATUS	*Integer 32 Variable*

OptoScript Example:

`WriteStrToIoUnitMemMap(` *I/O Unit, Mem Address, Variable* **`)`**

```
STATUS = WriteStrToIoUnitMemMap(MYIOUNIT, 0xFFFFFFFF, MYSTRINGVAR);
```

This is a function command; it returns a status code as listed below.

Notes:

- Use hex integer display for easy entering of memory map addresses.
- The control engine does not convert the variable type to match the area of memory map being written to. The control engine has no knowledge of which memory map areas are strings and which are other formats. You must write the correct type of data to the specified memory map address.

  For example, unpredictable results would occur if you try to write a string variable to the analog point area of the memory map. A float variable should be used instead. See the *SNAP Ethernet-Based I/O Units Protocols and Programming Guide* (Opto 22 form 1465) to determine the data types for specific areas of the memory map.

Status Codes: 0 = Success

-3 = Invalid length. Length must be greater than zero.

-12 = Invalid table index value—index was negative or greater than the table size.

-43 = Received a NACK from the I/O unit.

-52 = Invalid connection—not opened.

-56 = Invalid memory map address or read-only address.

-81 = Error writing to memory map. Invalid memory map address.

-93 = I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.

See Also: Write String Table to I/O Unit Memory Map (page W-7), Read String from I/O Unit Memory Map (page R-9), Read String Table from I/O Unit Memory Map (page R-11), Set I/O Unit Scratch Pad String Element (page S-40), Set I/O Unit Scratch Pad String Table (page S-41)

XOR

Logical Action

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, or to compare two logic states to see if they are different.

Details:

- Performs a logical EXCLUSIVE OR on *Argument 1* and *Argument 2* and puts the result in *Argument 3*. The result is True (non-zero) if either Argument 1 or Argument 2 value is non-zero (but not both); otherwise the result is False (0). Examples:

Argument 1	Argument 2	Argument 3
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	22	False

- The result can be sent directly to a digital output if desired.

Arguments:

Argument 1 [Value]	Argument 2 With	Argument 3 Put Result in
Digital Input	Digital Input	Digital Output
Digital Output	Digital Output	Float Variable
Float Literal	Float Literal	Integer 32 Variable
Float Variable	Float Variable	Integer 64 Variable
Integer 32 Literal	Integer 32 Literal	
Integer 32 Variable	Integer 32 Variable	
Integer 64 Literal	Integer 64 Literal	
Integer 64 Variable	Integer 64 Variable	

Standard Example:

XOR

	SUPPLY_FAN	*Digital Output*
With	1	*Integer 32 Literal*
Put Result in	SUPPLY_FAN	*Digital Output*

In this example, if SUPPLY FAN is on it will turn off, and vice versa.

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `xor` operator.

```
Supply_Fan = Supply_Fan xor 1;
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `xor` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- It is best to use only integers or digital points with this command.
- To manipulate individual bits or toggle a value between zero and another value, use Bit XOR.

See Also: Bit XOR (page B-18), Not Equal? (page N-4) Turn On (page T-27), Turn Off (page T-25), On? (page O-3) Off? (page O-1)

XOR?

Logical Condition

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. 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. 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

- Functionally equivalent to the Not Equal? condition when using allowable values.

Arguments:

Argument 1 Is	Argument 2 Is
Digital Input	Digital Input
Digital Output	Digital Output
Float Literal	Float Literal
Float Variable	Float Variable
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Literal
Integer 64 Variable	Integer 64 Variable

Standard Example:

	Is	Limit_Switch1_Prev	*Integer 32 Variable*
XOR?			
	Is	Limit_Switch1	*Digital Input*

OptoScript Example: OptoScript doesn't use a command; the function is built in. Use the `xor` operator.

```
if (Limit_Switch_Prev xor Limit_Switch) then
```

Notes:

- See "Logical Commands" in Chapter 10 of the *ioControl User's Guide*. The example shown is only one of many ways to use the `xor` operator. For more information on logical operators in OptoScript code, see Chapter 11 of the *ioControl User's Guide*.
- It is best to use only integers or digital points with this command.
- To test two values for equivalent True/False states, use the False exit.

See Also: NOT (page N-2), AND? (page A-8), OR? (page O-7)

Opto 22 Brain Families

Use this table to determine which family an Opto 22 brain belongs to:

Part number	mistic multi-function	mistic sub-family	mistic simple	Ethernet/ Optomux	SNAP Ethernet I/O (EIO)	SNAP Ultimate I/O (UIO)	Simple I/O (SIO)
B3000	●	mixed		●			
B200	●	analog					
B100	●	digital					
E1				●			
E2				●			
G4A8R	●	analog					
G4D16R	●	digital					
G4D32RS			●				
SNAP-B3000-ENET					●		
SNAP-BRS			●				
SNAP-ENET-D64					●		
SNAP-ENET-RTC							●
SNAP-ENET-S64							●
SNAP-UP1-ADS						●	
SNAP-UP1-D64						●	
SNAP-UP1-M64						●	

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

U

V

W

X