Form 1701

PAC CONTROL COMMAND REFERENCE

PAC CONTROL COMMAND REFERENCE

Form 1701-230516—May 2023

OPTO 22
Your Edge in Automation.™

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

PAC Control Command Reference
Form 1701-230516—May 2023

The information in this manual has been checked carefully and is believed to be accurate; however, Opto 22 assumes no responsibility for possible inaccuracies or omissions. Specifications are subject to change without notice.

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 newer are guaranteed for life. This lifetime warranty excludes reed relay modules, *groov* and 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. Refer to Opto 22 form 1042 for complete warranty information.

Wired+Wireless controllers and brains are licensed under one or more of the following patents: U.S. Patent No(s). 5282222, RE37802, 6963617; Canadian Patent No. 2064975; European Patent No. 1142245; French Patent No. 1142245; British Patent No. 1142245; Japanese Patent No. 2002535925A; German Patent No. 60011224.

Opto 22
Your Edge in Automation.

Table of Contents

1: Welcome

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

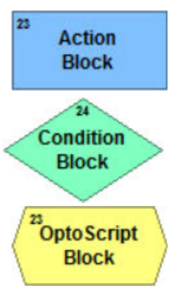
This command reference describes in detail all PAC Control programming commands and instructions. Its purpose is to help you understand what each command can do so that you can fully utilize the power of PAC Control.

This reference assumes that you have some experience using Microsoft® Windows® on personal computers. Programming experience is helpful, but not required.

ABOUT THIS REFERENCE

PAC Control comes in two forms: PAC Control Basic and PAC Control Professional. Commands that are available only in PAC Control Professional are noted with a Pro icon.

There are two types of commands: Actions and Conditions.



Actions are commands used in PAC Control Action blocks. They typically perform a control function; for example, the **Turn On** command turns on a digital output point.

Conditions are used in PAC Control Condition blocks to determine the flow of logic. Conditions end with a question mark (for example, **On?** and **Greater Than or Equal?**) and ask a question, such as "Is the digital input ON?" The answer determines whether the flow takes the True connection or the False connection to exit the block.

Each Action and Condition also has an equivalent form in **OptoScript**™ (PAC Control's built-in programming language). The equivalents can be used in OptoScript blocks.

Command Groups

Each command is assigned to a group (although many commands could naturally fit into several groups). Commands are listed with their respective group. If you can't find the command you are looking for in one group, try looking in a related group.

The groups are:

If you're using an electronic version of this reference, you can also use the Find feature (typically, the Ctrl+F key combination) to easily search for commands. If you're using a printed version of this reference, use the index to locate commands.

Choosing Documentation

This command reference contains the latest information you need to use PAC Control with your SNAP PAC system. However, if you are using Opto 22 "legacy" products designed to work with pre-SNAP PAC systems, see the *PAC Control Command Reference, Legacy Edition* (form 1711). The legacy version includes references to pre-SNAP PAC devices and commands, which are not included in this document.

For information on what we mean by "legacy" and instructions to migrate from an older system to a SNAP PAC system, see the *SNAP PAC System Migration* technical note (form 1688).

For information on how to enable legacy functionality in PAC Control, see "Enabling Legacy Options" in the *PAC Control User's Guide* (form 1700).

OTHER RESOURCES

For Software Developers: SNAP PAC REST API

If you're a developer who'd like to use PAC Control strategy tags in communications with other devices, the Opto 22 SNAP PAC REST API is a secure and powerful way to do just that. The API is available in SNAP PAC R-series and S-series controllers with PAC firmware R9.5a and higher. To configure HTTPS access to your PAC's RESTful server and learn how to call the API, visit developer.opto22.com.

Help, Documents, and Opto22.com Resources

To help you learn, understand, and use PAC Control systems, we offer a wide variety of options:

- Help is available in PAC Control and in all of the applications in the PAC Project™ Software Suite from Opto 22.
  - To see screen and field-level help, click the Help button on any PAC Control screen. While entering commands (instructions), click the Command Help button to see details about the command.

- In the PAC Control menu bar, click Help to view the list of help topics.
- Click Help > Manuals to open the PDF versions of the documents that are installed when you install PAC Control, including:
  - The *PAC Control User's Guide* (form 1700)—Shows how to install and use PAC Control.
  - The *PAC Control Command Reference* (form 1701)—Contains detailed information about each command available in PAC Control.
  - The *PAC Control Commands Quick Reference* (form 1703)—Lists all PAC Control commands plus their OptoScript code equivalents and arguments.

- Our website at www.opto22.com offers has a broad range of resources—from helpful videos to online blogs, forums, and news to free online self-training. We even offer free hands-on training at our headquarters in Temecula, California.

Product Support

If you have questions about PAC Control and can't find the help you need in this command reference or in the user's guide, contact Opto 22 Product Support.

Phone:	800-TEK-OPTO (800-835-6786 toll-free in the U.S. and Canada) 951-695-3080 Monday through Friday, 7 a.m. to 5 p.m. Pacific Time
Email:	support@opto22.com
Opto 22 website:	www.opto22.com

When calling for technical support, you can help us help you *faster* if you provide the following information to the Product Support engineer:

- A screen capture of the Help > About dialog box showing software product and version (available by clicking Help > About in the application's menu bar).
- Opto 22 hardware part numbers or models that you're working with.
- Firmware version:
  - For *groov* EPIC processors and *groov* RIO modules: available in *groov* Manage by clicking Info and Help > About.
  - For SNAP controllers and brains: available in PAC Manager by clicking Tools > Inspect.
- Specific error messages you received.
- Version of your computer's operating system.
- For PAC Control, PAC Display, OptoOPCServer, or PAC Manager, you may be requested to provide additional information, such as log or dump files. You can find these files in a support files sub-folder:
  a. On your Windows Desktop, double-click the PAC Project 10.5 folder.
  b. Double-click Support Files.
  c. Double-click on the appropriate shortcut to open the sub-folder containing the requested files.

  Note: *PAC Control, PAC Display, OptoOPCServer, and PAC Manager create appropriate sub-folders when they create diagnostic log or dump files. If they have not created these files, the sub-folder may not exist; in this case, the shortcut will not work.*

PAC CONTROL COMMANDS QUICK REFERENCE

OptoScript Equivalents

The following tables list both Actions and Conditions—and their OptoScript equivalents—within their respective command groups.

Commands with an asterisk (*) are available only in PAC Control Professional.

The Type column in these tables indicates whether the OptoScript command is a function command (F) or a procedure command (P). Function commands return a value from their action; procedure commands do not. For more information on OptoScript and command types, see Chapter 11 in the *PAC Control User's Guide* (form 1700).

For a list of symbols and where to find examples, see "Chapter Symbols" on page 20.

For OptoScript examples, see the individual listings for the commands (beginning on page 23).

Key

[1] Available only in PAC Control™ Professional

[2] Ethernet I/O™, Ultimate I/O™, and Simple I/O™ units

[3] Original High-Density Digital (HDD) modules

[4] *mistic*™ I/O units (Available only in PAC Control Professional)

To enable commands for Ethernet I/O, Ultimate I/O, Simple I/O, High-Density Digital modules, and *mistic* I/O units:

1. On PAC Control's menu bar, click File > Strategy Options.
2. On the Legacy tab, click the option you want to enable, and then click Yes.
3. To enable more than one option, repeat step 2.
4. When finished, click OK.

The Type column shows whether the OptoScript™ command is a function command (f) or a procedure command (p). Function commands return a value from their action; procedure commands do not.

Analog Point

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Calculate & Set Analog Gain	`CalcSetAnalogGain(On Point)`	f
Calculate & Set Analog Offset	`CalcSetAnalogOffset(On Point)`	f
Get & Clear Analog Filtered Value[1, 4]	`GetClearAnalogFilteredValue(From)`	f
Get & Clear Analog Maximum Value	`GetClearAnalogMaxValue(From)`	f
Get & Clear Analog Minimum Value	`GetClearAnalogMinValue(From)`	f
Get & Clear Analog Totalizer Value	`GetClearAnalogTotalizerValue(From)`	f
Get Analog Filtered Value[1, 4]	`GetAnalogFilteredValue(From)`	f
Get Analog Maximum Value	`GetAnalogMaxValue(From)`	f
Get Analog Minimum Value	`GetAnalogMinValue(From)`	f
Get Analog Square Root Filtered Value[1, 4]	`GetAnalogSquareRootFilteredValue(From)`	f
Get Analog Square Root Value[1, 4]	`GetAnalogSquareRootValue(From)`	f
Get Analog Totalizer Value	`GetAnalogTotalizerValue(From)`	f
Get HART Unique Address	`GetHARTUniqueAddress (Point, Polling Address, Unique Address, Timeout)`	f
Ramp Analog Output	`RampAnalogOutput(Ramp Endpoint, Units/Sec, Point to Ramp)`	p
Receive HART Response	`ReceiveHARTResponse(Point, Unique Address, Response, Timeout)`	f
Receive HART Burst Response	`ReceiveHARTBurstResponse(Point, Unique Address, Response, Timeout)`	f
Send/Receive HART Command	`SendReceiveHARTCommand (Point, Unique Address, Command, Parameters, Response, Timeout)`	f
Set Analog Filter Weight	`SetAnalogFilterWeight(To, On Point)`	p
Set Analog Gain	`SetAnalogGain(To, On Point)`	p
Set Analog Load Cell Fast Settle Level	`SetAnalogLoadCellFastSettleLevel(To, On Point)`	p
Set Analog Load Cell Filter Weight	`SetAnalogLoadCellFilterWeight(To, On Point)`	p
Set Analog Offset	`SetAnalogOffset(To, On Point)`	p
Set Analog Totalizer Rate	`SetAnalogTotalizerRate(To Seconds, On Point)`	p
Set Analog TPO Period	`SetAnalogTpoPeriod(To, On Point)`	p

Chart

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Call Chart	`CallChart(Chart)`	f
Calling Chart Running?	`IsCallingChartRunning()`	f
Calling Chart Stopped?	`IsCallingChartStopped()`	f
Calling Chart Suspended?	`IsCallingChartSuspended()`	f
Chart Running?	`IsChartRunning(Chart)`	f
Chart Stopped?	`IsChartStopped(Chart)`	f
Chart Suspended?	`IsChartSuspended(Chart)`	f
Continue Calling Chart	`ContinueCallingChart()`	f
Continue Chart	`ContinueChart(Chart)`	f
Get Chart Status	`GetChartStatus(Chart)`	f
Start Chart	`StartChart(Chart)`	f
Stop Chart	`StopChart(Chart)`	f
Suspend Chart	`SuspendChart(Chart)`	f

Communication

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Accept Incoming Communication	`AcceptIncomingCommunication(Communication Handle)`	f
Clear Communication Receive Buffer	`ClearCommunicationReceiveBuffer(Communication Handle)`	p
Close Communication	`CloseCommunication(Communication Handle)`	f
Communication Open?	`IsCommunicationOpen(Communication Handle)`	f
Get Active Interrupt Mask[1, 4]	`GetActiveInterruptMask()`	f
Get Communication Handle Value	`GetCommunicationHandleValue(From, To)`	f
Get End-Of-Message Terminator	`GetEndOfMessageTerminator (Communication Handle)`	f
Get Number of Characters Waiting	`GetNumCharsWaiting(On Communication Handle)`	f
HTTP Get	`HttpGet(Response Content, Response Header, Get Header, Security Mode, URL Path, Put HTTP Status In, Port, Hostname)`	f
HTTP Post from String Table	`HttpPostFromStringTable(Response Content, Response Header, Post Content, Post Header, Security Mode, URL Path, Put HTTP Status In, Port, Hostname)`	f
HTTP Post Calculate Content Length	`HttpPostCalcContentLength(Post Content, Post Header, Length Index)`	f
Listen for Incoming Communication	`ListenForIncomingCommunication(Communication Handle)`	f
Open Outgoing Communication	`OpenOutgoingCommunication(Communication Handle)`	f
Receive Character	`ReceiveChar(Communication Handle)`	f
Receive N Characters	`ReceiveNChars(Put In, Number of Characters, Communication Handle)`	f
Receive Numeric Table	`ReceiveNumTable(Length, Start at Index, Of Table, Communication Handle)`	f
Receive Numeric Table Ex	`ReceiveNumTableEx(Length, Start at Index, Endian Mode, Bytes per Value, Of Table, Communication Handle)`	f
Receive Numeric Variable	`ReceiveNumVariable(Endian mode, Number of Bytes, Put in, Communication Handle)`	f
Receive Pointer Table	`ReceivePtrTable(Length, Start at Index, Of Table, Communication Handle)`	f
Receive String	`ReceiveString(Put In, Communication Handle)`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Receive String Table	ReceiveStrTable(*Length, Start at Index, Of Table, Communication Handle*)	f
Send Communication Handle Command	SendCommunicationHandleCommand(*Communication Handle, Command*)	f
Send Email	SendEmail(*Server Information, Recipients, Message Body*)	f
Send Email with Attachments	SendEmailWithAttachments (*Server Information, Recipients, Message Body, Attachment File Names*)	f
Set Communication Handle Value	SetCommunicationHandleValue(*Value, Communication Handle*)	p
Set End-Of-Message Terminator	SetEndOfMessageTerminator(*Communication Handle, To Character*)	p
Transfer N Characters	TransferNChars(*Destination Handle, Source Handle, Num Chars*)	f
Transmit Character	TransmitChar(*Character, Communication Handle*)	f
Transmit NewLine	TransmitNewLine(*Communication Handle*)	f
Transmit Numeric Table	TransmitNumTable(*Length, Start at Index, Of Table, Communication Handle*)	f
Transmit Pointer Table	TransmitPtrTable(*Length, Start at Index, Of Table, Communication Handle*)	f
Transmit/Receive Mistic I/O Hex String[1, 4]	TransReceMisticIoHexStringWithCrc(*Hex String, On Port, Put Result in*)	f
Transmit/Receive String	TransmitReceiveString(*String, Communication Handle, Put Result in*)	f
Transmit String	TransmitString(*String, Communication Handle*)	f
Transmit String Table	TransmitStrTable(*Length, Start at Index, Of Table, Communication Handle*)	f

Control Engine

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Calculate Strategy CRC	CalcStrategyCrc()	f
Erase Files in Permanent Storage	EraseFilesInPermanentStorage()	f
Get Available File Space	GetAvailableFileSpace(*File System Type*)	f
Get Available File Space Ex	GetAvailableFileSpaceEx(*File System Type*)	f
Get Control Engine Address	GetControlEngineAddress()	f
Get Control Engine Type	GetEngineType()	f
Get Firmware Version	GetFirmwareVersion(*Put in*)	p
Get Number of Charts Running	GetNumChartsRunning(Put in)	f
Get Redundant Controller State[1]	GetRedundantControllerState(State)	f
Get Redundant Controller Status[1]	GetRedundantControllerState(Status)	f
Get Strategy Name	GetStrategyName(Put in)	p
Load Files From Permanent Storage	LoadFilesFromPermanentStorage()	f
Retrieve Strategy CRC	RetrieveStrategyCrc()	f
Save Files To Permanent Storage	SaveFilesToPermanentStorage()	f
Start Alternate Host Task	StartAlternateHostTask()	f

Digital Point

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clear All Latches	ClearAllLatches(*On I/O Unit*)	p

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clear Counter	ClearCounter(*On Point*)	p
Clear Off-Latch	ClearOffLatch(*On Point*)	p
Clear On-Latch	ClearOnLatch(*On Point*)	p
Generate N Pulses	GenerateNPulses(*On Time (Seconds), Off Time (Seconds), Number of Pulses, On Point*)	p
Get & Clear Counter	GetClearCounter(*From Point*)	f
Get & Clear Off-Latch	GetClearOffLatch(*From Point*)	f
Get & Clear On-Latch	GetClearOnLatch(*From Point*)	f
Get & Restart Off-Pulse Measurement	GetRestartOffPulseMeasurement(*From Point*)	f
Get & Restart Off-Time Totalizer	GetRestartOffTimeTotalizer(*From Point*)	f
Get & Restart On-Pulse Measurement	GetRestartOnPulseMeasurement(*From Point*)	f
Get & Restart On-Time Totalizer	GetRestartOnTimeTotalizer(*From Point*)	f
Get & Restart Period	GetRestartPeriod(*From Point*)	f
Get Counter	GetCounter(*From Point*)	f
Get Frequency	GetFrequency(*From Point*)	f
Get Off-Latch	GetOffLatch(*From Point*)	f
Get Off-Pulse Measurement	GetOffPulseMeasurement(*From Point*)	f
Get Off-Pulse Measurement Complete Status	GetOffPulseMeasurementCompleteStatus(*From Point*)	f
Get Off-Time Totalizer	GetOffTimeTotalizer(*From Point*)	f
Get On-Latch	GetOnLatch(*From Point*)	f
Get On-Pulse Measurement	GetOnPulseMeasurement(*From Point*)	f
Get On-Pulse Measurement Complete Status	GetOnPulseMeasurementCompleteStatus(*From Point*)	f
Get On-Time Totalizer	GetOnTimeTotalizer(*From Point*)	f
Get Period	GetPeriod(*From Point*)	f
Get Period Measurement Complete Status	GetPeriodMeasurementCompleteStatus(*From Point*)	f
Get TPO Percent	GetTpoPercent(From Point, Put in)	f
Get TPO Period	GetTpoPeriod(From Point, Put in)	f
Off?	IsOff(Point)	f
Off-Latch Set?	IsOffLatchSet(*On Point*)	f
On?	IsOn(*Point*)	f
On-Latch Set?	IsOnLatchSet(*On Point*)	f
Set TPO Percent	SetTpoPercent(*To Percent, On Point*)	p
Set TPO Period	SetTpoPeriod(*To Seconds, On Point*)	p
Start Continuous Square Wave	StartContinuousSquareWave(*On Time (Seconds), Off Time (Seconds), On Point*)	p
Start Counter	StartCounter(*On Point*)	p
Start Off-Pulse	StartOffPulse(*Off Time (Seconds), On Point*)	p
Start On-Pulse	StartOnPulse(*On Time (Seconds), On Point*)	p
Stop Counter	StopCounter(*On Point*)	p
Turn Off	TurnOff(*Output*)	p
Turn On	TurnOn(*Output*)	p

Error Handling

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Add Message to Queue	`AddMessageToQueue(`*`Severity, Message`*`)`	p
Add User Error to Queue	`AddUserErrorToQueue(`*`Error Number`*`)`	p
Add User I/O Unit Error to Queue	`AddUserIoUnitErrorToQueue(`*`Error Number, I/O Unit`*`)`	p
Caused a Chart Error?	`HasChartCausedError(`*`Chart`*`)`	f
Caused an I/O Unit Error?	`HasIoUnitCausedError(`*`I/O Unit`*`)`	f
Clear All Errors	`ClearAllErrors()`	p
Copy Current Error to String	`CurrentErrorToString(`*`Delimiter, String`*`)`	p
Disable I/O Unit Causing Current Error	`DisableIoUnitCausingCurrentError()`	p
Enable I/O Unit Causing Current Error	`EnableIoUnitCausingCurrentError()`	p
Error?	`IsErrorPresent()`	f
Error on I/O Unit?	`IsErrorOnIoUnit()`	f
Get Error Code of Current Error	`GetErrorCodeOfCurrentError()`	f
Get Error Count	`GetErrorCount()`	f
Get ID of Block Causing Current Error	`GetIdOfBlockCausingCurrentError()`	f
Get Line Causing Current Error	`GetLineCausingCurrentError()`	f
Get Name of Chart Causing Current Error	`GetNameOfChartCausingCurrentError(`*`Put in`*`)`	p
Get Name of I/O Unit Causing Current Error	`GetNameOfIoUnitCausingCurrentError(`*`Put in`*`)`	p
Get Severity of Current Error	`GetSeverityOfCurrentError()`	f
Remove Current Error and Point to Next Error	`RemoveCurrentError()`	p
Stop Chart on Error	`StopChartOnError()`	p
Suspend Chart on Error	`SuspendChartOnError()`	f

Event/Reaction

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clear All Event Latches[1,4]	`ClearAllEventLatches(`*`On I/O Unit`*`)`	p
Clear Event Latch[1,4]	`ClearEventLatch(`*`On Event/Reaction`*`)`	p
Clear I/O Unit Interrupt[1,4]	`ClearIoUnitInterrupt(`*`On I/O Unit`*`)`	p
Disable Interrupt on Event[1,4]	`DisableInterruptOnEvent(`*`Event/Reaction`*`)`	p
Disable Scanning for All Events[1,4]	`DisableScanningForAllEvents(`*`On I/O Unit`*`)`	p
Disable Scanning for Event[1,4]	`DisableScanningForEvent(`*`Event/Reaction`*`)`	p
Disable Scanning of Event/Reaction Group[1,4]	`DisableScanningOfEventReactionGroup(`*`E/R Group`*`)`	p
Enable Interrupt on Event[1,4]	`EnableInterruptOnEvent(`*`Event/Reaction`*`)`	p
Enable Scanning for All Events[1,4]	`EnableScanningForAllEvents(`*`On I/O Unit`*`)`	p
Enable Scanning for Event[1,4]	`EnableScanningForEvent(`*`Event/Reaction`*`)`	p
Enable Scanning of Event/Reaction Group[1,4]	`EnableScanningOfEventReactionGroup()`	p
Event Occurred?[1,4]	`HasEventOccurred(`*`Event/Reaction`*`)`	f
Event Occurring?[1,4]	`IsEventOccurring(`*`Event/Reaction`*`)`	f
Event Scanning Disabled?[1,4]	`IsEventScanningDisabled(`*`Event/Reaction`*`)`	f
Event Scanning Enabled?[1,4]	`IsEventScanningEnabled(`*`Event/Reaction`*`)`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get & Clear Event Latches[1,4]	`GetClearEventLatches(`*E/R Group*`)`	f
Get Event Latches[1,4]	`GetEventLatches(`*E/R Group*`)`	f
Generating Interrupt?[1,4]	`IsGeneratingInterrupt(`*I/O Unit*`)`	f
Interrupt Disabled for Event?[1,4]	`IsInterruptDisabledForEvent(`*Event/Reaction*`)`	f
Interrupt Enabled for Event?[1,4]	`IsInterruptEnabledForEvent(`*Event/Reaction*`)`	f
Read Event/Reaction Hold Buffer[1,4]	`ReadEventReactionHoldBuffer(`*Event/Reaction*`)`	f

Digital Module

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get & Clear Module Counters	`GetClearModuleCounters(`*I/O Unit, Module Number, Of Table, Starting Index*`)`	p
Get & Clear Module Off-Latches	`GetClearModuleOffLatches(`*I/O Unit, Module Number*`)`	p
Get & Clear Module On-Latches	`GetClearModuleOnLatches(I/O Unit, Module Number, Put Result In)`	f
Get All Module States	`GetAllModuleStates(`*From I/O Unit, Starting Index, Of Table*`)`	p
Get Module Counters	`GetModuleCounters(`*I/O Unit, Module Number, Of Table, Starting Index*`)`	f
Get Module Off-Latches	`GetModuleOffLatches(`*I/O Unit, Module Number, Put Result In*`)`	f
Get Module On-Latches	`GetModuleOnLatches(`*I/O Unit, Module Number, Put Result In*`)`	f
Get Module States	`GetModuleStates(`*I/O Unit, Module Number, Put Result In*`)`	f
Set Module States from MOMO Masks	`SetModuleStatesfromMOMOMasks(`*Must-Off Mask, Must-On Mask, I/O Unit, Module Number*`)`	p
IVAL Get All Module States	`IvalGetAllModuleStates(`*From I/O Unit, Starting Index, Of Table*`)`	p

High Density Digital Module

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clear HDD Module Off-Latches[3]	`ClearHddModuleOffLatches(`*I/O Unit, Module Number, Clear Mask*`)`	f
Clear HDD Module On-Latches[3]	`ClearHddModuleOnLatches(`*I/O Unit, Module Number, Clear Mask*`)`	f
Get & Clear All HDD Module Off-Latches[3]	`GetClearAllHddModuleOffLatches(`*I/O Unit, Start Index, Put Result In*`)`	f
Get & Clear All HDD Module On-Latches[3]	`GetClearAllHddModuleOnLatches(`*I/O Unit, Start Index, Put Result In*`)`	f
Get & Clear HDD Module Counter[3]	`GetClearHddModuleCounter(`*I/O Unit, Module Number, Point Number, Put Result In*`)`	f
Get & Clear HDD Module Counters[3]	`GetClearHddModuleCounters(`*I/O Unit, Module Number, Start Table Index, Put Result In*`)`	f
Get & Clear HDD Module Off-Latches[3]	`GetClearHddModuleOffLatches(`*I/O Unit, Module Number, Put Result In*`)`	f
Get & Clear HDD Module On-Latches[3]	`GetClearHddModuleOnLatches(`*I/O Unit, Module Number, Put Result In*`)`	f
Get All HDD Module Off-Latches[3]	`GetAllHddModuleOffLatches(`*I/O Unit, Start Index, Put Result In*`)`	f
Get All HDD Module On-Latches[3]	`GetAllHddModuleOnLatches(`*I/O Unit, Start Index, Put Result In*`)`	f
Get All HDD Module States[3]	`GetAllHddModuleStates(`*I/O Unit, Start Index, Put Result In*`)`	f
Get HDD Module Counters[3]	`GetHddModuleCounters(`*I/O Unit, Module Number, Start Table Index, Put Result In*`)`	f
Get HDD Module Off-Latches[3]	`GetHddModuleOffLatches(`*I/O Unit, Module Number, Put Result In*`)`	f
Get HDD Module On-Latches[3]	`GetHddModuleOnLatches(`*I/O Unit, Module Number, Put Result In*`)`	f
Get HDD Module States[3]	`GetHddModuleStates(`*I/O Unit, Module Number, Put Result In*`)`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Set HDD Module from MOMO Masks[3]	`SetHddModulefromMOMOMasks(`*`I/O Unit, Module Number, Must-On Mask, Must-Off Mask`*`)`	f
Turn Off HDD Module Point[3]	`TurnOffHDDModulePoint(`*`I/O Unit, Module Number, Point Number`*`)`	f
Turn On HDD Module Point[3]	`TurnOnHddModulePoint(`*`I/O Unit, Module Number, Point Number`*`)`	f

I/O Quality

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get I/O Channel Quality	`GetChannelQuality(`*`From Channel, Put in`*`)`	f
Get I/O Unit Quality	`GetIoUnitQuality(`*`From I/O Unit, Put in`*`)`	f

I/O Unit

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clear I/O Unit Configured Flag	`ClearIoUnitConfiguredFlag(`*`I/O Unit`*`)`	p
Get I/O Unit as Binary Value	`GetIoUnitAsBinaryValue(`*`I/O Unit`*`)`	f
Get I/O Unit as Binary Value 64	`GetIoUnitAsBinaryValue64(`*`I/O Unit`*`)`	f
Get I/O Unit Module States	`GetIoUnitModuleStates (`*`From I/O Unit, To Table, With Starting Index`*`)`	p
Get Target Address State[1]	`GetTargetAddressState(`*`Enable Mask, Active Mask, I/O Unit`*`)`	p
I/O Unit Ready?	`IsIoUnitReady(`*`I/O Unit`*`)`	f
IVAL Move Numeric Table to I/O Unit	`IvalMoveNumTableToIoUnit(`*`Start at Index, Of Table, Move to`*`)`	p
IVAL Move Numeric Table to I/O Unit Ex	`IvalMoveNumTableToIoUnitEx(`*`From Table, With Starting Index, To I/O Unit`*`)`	p
Move I/O Unit to Numeric Table	`MoveIoUnitToNumTable(`*`I/O Unit, Starting Index, Of Table`*`)`	p
Move I/O Unit to Numeric Table Ex	`MoveIoUnitToNumTableEx(`*`From I/O Unit, To Table, With Starting Index`*`)`	p
Move Numeric Table to I/O Unit	`MoveNumTableToIoUnit(`*`Start at Index, Of Table, Move to`*`)`	p
Move Numeric Table to I/O Unit Ex	`MoveNumTableToIoUnitEx(`*`From Table, With Starting Index, To I/O Unit`*`)`	p
Set All Target Address States[1]	`SetAllTargetAddressStates(`*`Must-On Mask, Must-Off Mask, Active Mask`*`)`	p
Set I/O Unit Configured Flag	`SetIoUnitConfiguredFlag(`*`For I/O Unit`*`)`	p
Set I/O Unit from MOMO Masks	`SetIoUnitFromMomo(`*`Must-On Mask, Must-Off Mask, Digital I/O Unit`*`)`	p
Set Target Address State[1]	`SetTargetAddressState(`*`Must-On Mask, Must-Off Mask, Active Mask, I/O Unit`*`)`	p
Write I/O Unit Configuration to EEPROM	`WriteIoUnitConfigToEeprom(`*`On I/O Unit`*`)`	p

I/O Unit—Event Message

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get I/O Unit Event Message State	`GetIoUnitEventMsgState(`*`I/O Unit, Event Message #, Put Result in`*`)`	f
Get I/O Unit Event Message Text	`GetIoUnitEventMsgText(`*`I/O Unit, Event Message #, Put Result in`*`)`	f
Set I/O Unit Event Message State	`SetIoUnitEventMsgState(`*`I/O Unit, Event Message #, State`*`)`	f
Set I/O Unit Event Message Text	`SetIoUnitEventMsgText(`*`I/O Unit, Event Message #, Message Text`*`)`	f

I/O Unit—Memory Map

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Read Number from I/O Unit Memory Map	`ReadNumFromIoUnitMemMap(`*I/O Unit, Mem address, To*`)`	f
Read Numeric Table from I/O Unit Memory Map	`ReadNumTableFromIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, To*`)`	f
Read String from I/O Unit Memory Map	`ReadStrFromIoUnitMemMap(`*Length, I/O Unit, Mem address, To*`)`	f
Read String Table from I/O Unit Memory Map	`ReadStrTableFromIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, To*`)`	f
Write Number to I/O Unit Memory Map	`WriteNumToIoUnitMemMap(`*I/O Unit, Mem address, Variable*`)`	f
Write Numeric Table to I/O Unit Memory Map	`WriteNumTableToIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, Table*`)`	f
Write String Table to I/O Unit Memory Map	`WriteStrTableToIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem address, Table*`)`	f
Write String to I/O Unit Memory Map	`WriteStrToIoUnitMemMap(`*I/O Unit, Mem address, Variable*`)`	f

I/O Unit—Scratch Pad

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get I/O Unit Scratch Pad Bits	`GetIoUnitScratchPadBits(`*I/O Unit, Put Result in*`)`	f
Get I/O Unit Scratch Pad Float Element	`GetIoUnitScratchPadFloatElement(`*I/O Unit, Index, Put Result in*`)`	f
Get I/O Unit Scratch Pad Float Table	`GetIoUnitScratchPadFloatTable(`*I/O Unit, Length, From Index, To Index, To Table*`)`	f
Get I/O Unit Scratch Pad Integer 32 Element	`GetIoUnitScratchPadInt32Element(`*I/O Unit, Index, Put Result in*`)`	f
Get I/O Unit Scratch Pad Integer 32 Table	`GetIoUnitScratchPadInt32Table(`*I/O Unit, Length, From Index, To Index, To Table*`)`	f
Get I/O Unit Scratch Pad Integer 64 Element	`GetIoUnitScratchPadInt64Element(`*I/O Unit, Index, Put Result in*`)`	f
Get I/O Unit Scratch Pad Integer 64 Table	`GetIoUnitScratchPadInt64Table(`*I/O Unit, Length, From Index, To Index, To Table*`)`	f
Get I/O Unit Scratch Pad String Element	`GetIoUnitScratchPadStringElement(`*I/O Unit, Index, Put Result in*`)`	f
Get I/O Unit Scratch Pad String Table	`GetIoUnitScratchPadStringTable(`*I/O Unit, Length, From Index, To Index, To Table*`)`	f
Set I/O Unit Scratch Pad Bits from MOMO Mask	`SetIoUnitScratchPadBitsFromMomo(`*I/O Unit, Must-On Mask, Must-Off Mask*`)`	f
Set I/O Unit Scratch Pad Float Element	`SetIoUnitScratchPadFloatElement(`*I/O Unit, Index, From*`)`	f
Set I/O Unit Scratch Pad Float Table	`SetIoUnitScratchPadFloatTable(`*I/O Unit, Length, To Index, From Index, From Table*`)`	f
Set I/O Unit Scratch Pad Integer 32 Element	`SetIoUnitScratchPadInt32Element(`*I/O Unit, Index, From*`)`	f
Set I/O Unit Scratch Pad Integer 32 Table	`SetIoUnitScratchPadInt32Table(`*I/O Unit, Length, To Index, From Index, From Table*`)`	f
Set I/O Unit Scratch Pad Integer 64 Element	`SetIoUnitScratchPadInt64Element(`*I/O Unit, Index, From*`)`	f
Set I/O Unit Scratch Pad Integer 64 Table	`SetIoUnitScratchPadInt64Table(`*I/O Unit, Length, To Index, From Index, From Table*`)`	f
Set I/O Unit Scratch Pad String Element	`SetIoUnitScratchPadStringElement(`*I/O Unit, Index, From*`)`	f
Set I/O Unit Scratch Pad String Table	`SetIoUnitScratchPadStringTable(`*I/O Unit, Length, To Index, From Index, From Table*`)`	f

Logical

PAC Control Command	OptoScript Equivalent (Arguments)	Type
AND	`x and y`	f
AND?	`See AND`	f
Bit AND	`x bitand y`	f
Bit AND?	`See Bit AND`	f
Bit Change	`BitChange(`*Set flag, Bit to Change, Output*`)`	p
Bit Clear	`BitClear(`*Item, Bit to Clear*`)`	f
Bit Copy	`BitCopy(`*Server Bit to Set, Destination, Destination Index, Bit to Read, Source, Source Index*`)`	f
Bit NOT	`bitnot x`	f
Bit NOT?	`See Bit NOT`	f
Bit Off in Numeric Table Element?	`IsBitOffInNumTableElement(`*At Index, Of Table, Bit*`)`	f
Bit Off?	`IsBitOff(`*In, Bit*`)`	f
Bit On in Numeric Table Element?	`IsBitOnInNumTableElement(`*At Index, Of Table, Bit*`)`	f
Bit On?	`IsBitOn(`*In, Bit*`)`	f
Bit OR	`x bitor y`	f
Bit OR?	`See Bit OR`	f
Bit Rotate	`BitRotate(`*Item, Count*`)`	f
Bit Set	`BitSet(`*Item, Bit to Set*`)`	f
Bit Shift	`x << nBitsToShift`	f
Bit Test	`BitTest(`*Item, Bit to Test*`)`	f
Bit XOR	`x bitxor y`	f
Bit XOR?	`See Bit XOR`	f
Equal to Numeric Table Element?	`n == nt[0]`	f
Equal?	`x == y`	f
Flip Flop JK	`FlipFlopJK(`*Set [J], Reset [K], Output [Q]*`)`	p
Float to Int32 Bits	`FloatToInt32Bits(`*Server URL*`)`	f
Get High Bits of Integer 64	`GetHighBitsOfInt64(`*High Bits From*`)`	f
Get Low Bits of Integer 64	`GetLowBitsOfInt64(`*Integer 64*`)`	f
Greater Than Numeric Table Element?	`x > nt[0]`	f
Greater Than or Equal to Numeric Table Element?	`x >= t[0]`	f
Greater Than or Equal?	`x >= y`	f
Greater?	`x > y`	f
Int32 to Float Bits	`Int32ToFloatBits(`*nInt32*`)`	f
Less Than Numeric Table Element?	`x < nt[0]`	f
Less Than or Equal to Numeric Table Element?	`x <= nt[0]`	f
Less Than or Equal?	`x <= y`	f
Less?	`x < y`	f
Make Integer 64	`MakeInt64(`*High Integer, Low Integer*`)`	f
Move 32 Bits	`Move32Bits(`*From, To*`)`	p
NOT	`not x`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Not Equal to Numeric Table Element?	`n <> nt[0]`	f
Not Equal?	`x <> y`	f
NOT?	`not x`	f
Numeric Table Element Bit Clear	`NumTableElementBitClear(`*Element Index, Of Integer Table, Bit to Clear*`)`	p
Numeric Table Element Bit Set	`NumTableElementBitSet(`*Element Index, Of Integer Table, Bit to Set*`)`	p
Numeric Table Element Bit Test	`NumTableElementBitTest(`*Element Index, Of Integer Table, Bit to Test*`)`	f
OR	`x or y`	f
OR?	`See OR`	f
Set Variable False	`SetVariableFalse(`*Variable*`)`	p
Set Variable True	`SetVariableTrue(`*Variable*`)`	p
Test Equal	`See Equal?`	f
Test Greater	`See Greater?`	f
Test Greater or Equal	`See Greater Than or Equal?`	f
Test Less	`See Less?`	f
Test Less or Equal	`See Less Than or Equal?`	f
Test Not Equal	`See Not Equal?`	f
Test Within Limits	`See Within Limits?`	f
Variable False?	`IsVariableFalse(`*Variable*`)`	f
Variable True?	`IsVariableTrue(`*Variable*`)`	f
Within Limits?	`IsWithinLimits(`*Value, Low Limit, High Limit*`)`	f
XOR	`x xor y`	f
XOR?	`See XOR`	f

Mathematical

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Absolute Value	`AbsoluteValue(`*Of*`)`	f
Add	`x + y`	f
Arccosine	`Arccosine(`*Of*`)`	f
Arcsine	`Arcsine(`*Of*`)`	f
Arctangent	`Arctangent(`*Of*`)`	f
Clamp Float Table Element	`ClampFloatTableElement(`*High Limit, Low Limit, Element Index, Of Float Table*`)`	p
Clamp Float Variable	`ClampFloatVariable(`*High Limit, Low Limit, Float Variable*`)`	p
Clamp Integer 32 Table Element	`ClampInt32TableElement(`*High Limit, Low Limit, Element Index, Of Integer 32 Table*`)`	p
Clamp Integer 32 Variable	`ClampInt32Variable(`*High Limit, Low Limit, Integer 32 Variable*`)`	p
Complement	`-x`	p
Cosine	`Cosine(`*Of*`)`	f
Decrement Variable	`DecrementVariable(`*Variable*`)`	p
Divide	`x / y`	f
Generate Random Number	`GenerateRandomNumber()`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Hyperbolic Cosine	`HyperbolicCosine(`*Of*`)`	f
Hyperbolic Sine	`HyperbolicSine(`*Of*`)`	f
Hyperbolic Tangent	`HyperbolicTangent(`*Of*`)`	f
Increment Variable	`IncrementVariable(`*Variable*`)`	p
Maximum	`Max(`*Compare, With*`)`	f
Minimum	`Min(`*Compare, With*`)`	f
Modulo	`x % y`	f
Multiply	`x * y`	f
Natural Log	`NaturalLog(`*Of*`)`	f
Raise e to Power	`RaiseEToPower(`*Exponent*`)`	f
Raise to Power	`Power(`*Raise, To the*`)`	f
Round	`Round(`*Value*`)`	f
Seed Random Number	`SeedRandomNumber()`	p
Sine	`Sine(`*Of*`)`	f
Square Root	`SquareRoot(`*Of*`)`	f
Subtract	`x - y`	f
Tangent	`Tangent(`*Of*`)`	f
Truncate	`Truncate(`*Value*`)`	f

Miscellaneous

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Comment (Block)	`/* block comment */`	p
Comment (Single Line)	`// single line comment`	f
Flag Lock	`FlagLock(`*Flag, Timeout*`)`	f
Flag Unlock	`FlagUnlock(`*Flag, Force unlock*`)`	f
Float Valid?	`IsFloatValid(`*Float*`)`	f
Generate Reverse CRC-16 on Table (32 bit)	`GenerateReverseCrc16OnTable32(`*Start Value, Table, Starting Element, Number of Elements*`)`	f
Get Length of Table	`GetLengthOfTable(`*Table*`)`	f
Get Type From Name	`GetTypeFromName(`*Name*`)`	f
Get Value From Name	`GetValueFromName(`*Name, Put Result In*`)`	f
Move	`x = y;`	p
Move from Numeric Table Element	`x = nt[0];`	f
Move Numeric Table Element to Numeric Table	`nt1[0] = nt2[5];`	p
Move Numeric Table to Numeric Table	`MoveNumTableToNumTable(`*From Table, From Index, To Table, To Index, Length*`)`	p
Move to Numeric Table Element	`nt[0] = x;`	p
Move to Numeric Table Elements	`MoveToNumTableElements(`*From, Start Index, End Index, Of Table*`)`	p
Shift Numeric Table Elements	`ShiftNumTableElements(`*Shift Count, Table*`)`	p
Shift String Table Elements	`ShiftStringTableElements(`*Shift Count, Table*`)`	p

PID—Ethernet

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get PID Configuration Flags	GetPidConfigFlags(*PID Loop*)	f
Get PID Current Input	GetPidCurrentInput(*PID Loop*)	f
Get PID Current Setpoint	GetPidCurrentSetpoint(*PID Loop*)	f
Get PID Feed Forward	GetPidFeedForward(*PID Loop*)	f
Get PID Feed Forward Gain	GetPidFeedForwardGain(*PID Loop*)	f
Get PID Forced Output When Input Over Range	GetPidForcedOutputWhenInputOverRange(*PID Loop*)	f
Get PID Forced Output When Input Under Range	GetPidForcedOutputWhenInputUnderRange(*PID Loop*)	f
Get PID Gain	GetPidGain(*PID Loop*)	f
Get PID Input	GetPidInput(*PID Loop*)	f
Get PID Input High Range	GetPidInputHighRange(*PID Loop*)	f
Get PID Input Low Range	GetPidInputLowRange(*PID Loop*)	f
Get PID Max Output Change	GetPidMaxOutputChange(*PID Loop*)	f
Get PID Min Output Change	GetPidMinOutputChange(*PID Loop*)	f
Get PID Mode	GetPidMode(*PID Loop*)	f
Get PID Output	GetPidOutput(*PID Loop*)	f
Get PID Output High Clamp	GetPidOutputHighClamp(*PID Loop*)	f
Get PID Output Low Clamp	GetPidOutputLowClamp(*PID Loop*)	f
Get PID Scan Time	GetPidScanTime(*PID Loop*)	f
Get PID Setpoint	GetPidSetpoint(*PID Loop*)	f
Get PID Status Flags	GetPidStatusFlags(*PID Loop*)	f
Get PID Tune Derivative	GetPidTuneDerivative(*PID Loop*)	f
Get PID Tune Integral	GetPidTuneIntegral(*PID Loop*)	f
Set PID Configuration Flags	SetPidConfigFlags(*PID Loop, Configuration Flags*)	p
Set PID Feed Forward	SetPidFeedForward(*PID Loop, Feed Forward*)	p
Set PID Feed Forward Gain	SetPidFeedForwardGain(*PID Loop, Feed Fwd Gain*)	p
Set PID Forced Output When Input Over Range	SetPidForcedOutputWhenInputOverRange(*PID Loop, Forced Output*)	p
Set PID Forced Output When Input Under Range	SetPidForcedOutputWhenInputUnderRange(*PID Loop, Forced Output*)	p
Set PID Gain	SetPidGain(*PID Loop, Gain*)	p
Set PID Input	SetPidInput(*PID Loop, Input*)	p
Set PID Input High Range	SetPidInputHighRange(*PID Loop, High Range*)	p
Set PID Input Low Range	SetPidInputLowRange(*PID Loop, Low Range*)	p
Set PID Max Output Change	SetPidMaxOutputChange(*PID Loop, Max Change*)	p
Set PID Min Output Change	SetPidMinOutputChange(*PID Loop, Min Change*)	p
Set PID Mode	SetPidMode(*PID Loop, Mode*)	p
Set PID Output	SetPidOutput(*PID Loop, Output*)	p
Set PID Output High Clamp	SetPidOutputHighClamp(*PID Loop, High Clamp*)	p
Set PID Output Low Clamp	SetPidOutputLowClamp(*PID Loop, Low Clamp*)	p
Set PID Scan Time	SetPidScanTime(*PID Loop, Scan Time*)	p
Set PID Setpoint	SetPidSetpoint(*PID Loop, Setpoint*)	p
Set PID Tune Derivative	SetPidTuneDerivative(*PID Loop, Derivative*)	p

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Set PID Tune Integral	`SetPidTuneIntegral(`*`PID Loop, Integral`*`)`	p

PID—Mistic

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clamp Mistic PID Output[1, 4]	`ClampMisticPidOutput(`*`High Clamp, Low Clamp, On PID Loop`*`)`	p
Clamp Mistic PID Setpoint[1, 4]	`ClampMisticPidSetpoint(`*`High Clamp, Low Clamp, On PID Loop`*`)`	p
Disable Mistic PID Output[1, 4]	`DisableMisticPidOutput(`*`Of PID Loop`*`)`	p
Disable Mistic PID Output Tracking in Manual Mode[1, 4]	`DisableMisticPidOutputTrackingInManualMode(`*`On PID Loop`*`)`	p
Disable Mistic PID Setpoint Tracking in Manual Mode[1, 4]	`DisableMisticPidSetpointTrackingInManualMode(`*`On PID Loop`*`)`	p
Enable Mistic PID Output[1, 4]	`EnableMisticPidOutput(`*`On PID Loop`*`)`	p
Enable Mistic PID Output Tracking in Manual Mode[1, 4]	`EnableMisticPidOutputTrackingInManualMode(`*`On PID Loop`*`)`	p
Enable Mistic PID Setpoint Tracking in Manual Mode[1, 4]	`EnableMisticPidSetpointTrackingInManualMode(`*`On PID Loop`*`)`	p
Get Mistic PID Control Word[1, 4]	`GetMisticPidControlWord(`*`From PID Loop`*`)`	f
Get Mistic PID D Term[1, 4]	`GetMisticPidDTerm(`*`From PID Loop`*`)`	f
Get Mistic PID I Term[1, 4]	`GetMisticPidITerm(`*`From PID Loop`*`)`	f
Get Mistic PID Input[1, 4]	`GetMisticPidInput(`*`PID Loop`*`)`	f
Get Mistic PID Mode[1, 4]	`GetMisticPidMode(`*`PID Loop`*`)`	f
Get Mistic PID Output[1, 4]	`GetMisticPidOutput(`*`PID Loop`*`)`	f
Get Mistic PID Output Rate of Change[1, 4]	`GetMisticPidOutputRateOfChange(`*`From PID Loop`*`)`	f
Get Mistic PID P Term[1, 4]	`GetMisticPidPTerm(`*`From PID Loop`*`)`	f
Get Mistic PID Scan Rate[1, 4]	`GetMisticPidScanRate(`*`From PID Loop`*`)`	f
Get Mistic PID Setpoint[1, 4]	`GetMisticPidSetpoint(`*`PID Loop`*`)`	f
Set Mistic PID Control Word[1, 4]	`SetMisticPidControlWord(`*`On-Mask, Off-Mask, For PID Loop`*`)`	p
Set Mistic PID D Term[1, 4]	`SetMisticPidDTerm(`*`To, On PID Loop`*`)`	p
Set Mistic PID I Term[1, 4]	`SetMisticPidITerm(`*`To, On PID Loop`*`)`	p
Set Mistic PID Input[1, 4]	`SetMisticPidInput(`*`Input, PID Loop`*`)`	p
Set Mistic PID Mode to Auto[1, 4]	`SetMisticPidModeToAuto(`*`On PID Loop`*`)`	p
Set Mistic PID Mode to Manual[1, 4]	`SetMisticPidModeToManual(`*`On PID Loop`*`)`	p
Set Mistic PID Output Rate of Change[1, 4]	`SetMisticPidOutputRateOfChange(`*`To, On PID Loop`*`)`	p
Set Mistic PID P Term[1, 4]	`SetMisticPidPTerm(`*`To, On PID Loop`*`)`	p
Set Mistic PID Scan Rate[1, 4]	`SetMisticPidScanRate(`*`To, On PID Loop`*`)`	p
Set Mistic PID Setpoint[1, 4]	`SetMisticPidSetpoint(`*`PID Loop, Setpoint`*`)`	p

Pointers

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Clear Pointer	`pn1 = null;`	f
Clear Pointer Table Element	`pt[0] = null;`	p
Get Pointer From Name	`GetPointerFromName(`*`Name, Pointer`*`)`	p
Move from Pointer Table Element	`pn = pt[0];`	f
Move to Pointer	`pn = &n;`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Move to Pointer Table Element	`pt[0] = &n;`	f
Pointer Equal to Null?	`pn == null`	f
Pointer Table Element Equal to Null?	`pt[0] == null`	f

Simulation

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Communication to All I/O Points Enabled?	`IsCommToAllIoPointsEnabled()`	f
Communication To All I/O Units Enabled?	`IsCommToAllIoUnitsEnabled()`	f
Disable Communication to All I/O Points	`DisableCommuncationToAllIoPoints()`	p
Disable Communication to All I/O Units	`DisableCommunicationToAllIoUnits()`	p
Disable Communication to Event/Reaction[1, 4]	`DisableCommunicationToEventReaction (`*Event/Reaction*`)`	p
Disable Communication to I/O Unit	`DisableCommunicationToIoUnit(`*I/O Unit*`)`	p
Disable Communication to Mistic PID Loop[1, 4]	`DisableCommunicationtoMisticPidLoop(`*PID Loop*`)`	p
Disable Communication to PID Loop	`DisableCommunicationtoPidLoop(`*PID Loop*`)`	p
Disable Communication to Point	`DisableCommunicationToPoint(`*Point*`)`	p
Disable Event/Reaction Group[1]	`DisableEventReactionGroup(`*E/R Group*`)`	p
Enable Communication to All I/O Points	`EnableCommunicationToAllIoPoints()`	p
Enable Communication to All I/O Units	`EnableCommunicationToAllIoUnits()`	p
Enable Communication to Event/Reaction[1, 4]	`EnableCommunicationToEventReaction(`*Event/Reaction*`)`	p
Enable Communication to I/O Unit	`EnableCommunicationToIoUnit(`*I/O Unit*`)`	p
Enable Communication to Mistic PID Loop[1, 4]	`EnableCommunicationToMisticPidLoop(`*PID Loop*`)`	p
Enable Communication to PID Loop	`EnableCommunicationtoPidLoop(`*PID Loop*`)`	p
Enable Communication to Point	`EnableCommunicationToPoint(`*Point*`)`	p
Enable Event/Reaction Group[1, 4]	`EnableEventReactionGroup(`*E/R Group*`)`	p
Event/Reaction Communication Enabled?[1, 4]	`IsEventReactionCommEnabled (`*Event/Reaction*`)`	f
Event/Reaction Group Communication Enabled?[1, 4]	`IsEventReactionGroupEnabled(`*E/R Group*`)`	f
I/O Point Communication Enabled?	`IsIoPointCommEnabled(`*I/O Point*`)`	f
I/O Unit Communication Enabled?	`IsIoUnitCommEnabled(`*I/O Unit*`)`	f
IVAL Set Analog Filtered Value	`IvalSetAnalogFilteredValue(`*To, On Point*`)`	p
IVAL Set Analog Maximum Value	`IvalSetAnalogMaxValue(`*To, On Point*`)`	p
IVAL Set Analog Minimum Value	`IvalSetAnalogMinValue(`*To, On Point*`)`	p
IVAL Set Analog Point	`IvalSetAnalogPoint(`*To, On Point*`)`	p
IVAL Set Counter	`IvalSetCounter(`*To, On Point*`)`	p
IVAL Set Frequency	`IvalSetFrequency(`*To, On Point*`)`	p
IVAL Set I/O Unit from MOMO Masks	`IvalSetIoUnitfromMOMO(On Mask, Off Mask, On I/O Unit)`	p
IVAL Set Mistic PID Control Word[1, 4]	`IvalSetPidControlWord(`*On Mask, Off Mask, For PID Loop*`)`	p
IVAL Set Mistic PID Process Term[1, 4]	`IvalSetMisticPidProcessTerm(`*To, On PID Loop*`)`	p
IVAL Set Off-Latch	`IvalSetOffLatch(`*To, On Point*`)`	p
IVAL Set Off-Pulse	`IvalSetOffPulse(`*To, On Point*`)`	p
IVAL Set Off-Totalizer	`IvalSetOffTotalizer(`*To, On Point*`)`	p

PAC Control Command	OptoScript Equivalent (Arguments)	Type
IVAL Set On-Latch	`IvalSetOnLatch(`*`To, On Point`*`)`	p
IVAL Set On-Pulse	`IvalSetOnPulse(`*`To, On Point`*`)`	p
IVAL Set On-Totalizer	`IvalSetOnTotalizer(`*`To, On Point`*`)`	p
IVAL Set Period	`IvalSetPeriod(`*`To, On Point`*`)`	p
IVAL Set TPO Percent	`IvalSetTpoPercent(`*`To, On Point`*`)`	p
IVAL Set TPO Period	`IvalSetTpoPeriod(`*`Value, On Point`*`)`	p
IVAL Turn Off	`IvalTurnOff(`*`Point`*`)`	p
IVAL Turn On	`IvalTurnOn(`*`Point`*`)`	p
Mistic PID Loop Communication Enabled?[1, 4]	`IsMisticPidLoopCommEnabled(`*`PID Loop`*`)`	p
PID Loop Communication Enabled?	`IsPidLoopCommEnabled(`*`PID Loop`*`)`	f

String

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Append Character to String	`s1 = s1 + "a";`	p
Append String to String	`s1 = s1 + s2;`	p
Compare Strings	`CompareStrings(`*`String 1, String 2`*`)`	f
Convert Float to String	`FloatToString(`*`Convert, Length, Decimals, Put Result in`*`)`	p
Convert Hex String to Number	`HexStringToNumber(`*`Convert`*`)`	f
Convert IEEE Hex String to Number	`IEEEHexStringToNumber(`*`Convert`*`)`	f
Convert Integer 32 to IP Address String	`Int32ToIpAddressString(`*`Convert, Put Result In`*`)`	f
Convert IP Address String to Integer 32	`IpAddressStringToInt32(`*`Convert`*`)`	f
Convert Mistic I/O Hex String to Float[1, 4]	`MisticIoHexToFloat(`*`Convert`*`)`	f
Convert Number to Formatted Hex String	`NumberToFormattedHexString(`*`Convert, Length, Put Result in`*`)`	p
Convert Number to Hex String	`NumberToHexString(`*`Convert, Put Result in`*`)`	p
Convert Number to Mistic I/O Hex String[1, 4]	`NumberToMisticIoHex(`*`Convert, Put Result in`*`)`	p
Convert Number to String	`NumberToString(`*`Convert, Put Result in`*`)`	p
Convert Number to String Field	`NumberToStringField(`*`Convert, Length, Put Result in`*`)`	p
Convert String to Float	`StringToFloat(`*`Convert`*`)`	f
Convert String to Integer 32	`StringToInt32(`*`Convert`*`)`	f
Convert String to Integer 64	`StringToInt64(`*`Convert`*`)`	f
Convert String to Lower Case	`StringToLowerCase(`*`Convert`*`)`	p
Convert String to Upper Case	`StringToUpperCase(`*`Convert`*`)`	p
Find Character in String	`FindCharacterInString(`*`Find, Start at Index, Of String`*`)`	f
Find Substring in String	`FindSubstringInString(`*`Find, Start at Index, Of String`*`)`	f
Generate Checksum on String	`GenerateChecksumOnString(`*`Start Value, On String`*`)`	f
Generate Forward CCITT on String	`GenerateForwardCcittOnString(`*`Start Value, On String`*`)`	f
Generate Forward CRC-16 on String	`GenerateForwardCrc16OnString(`*`Start Value, On String`*`)`	f
Generate Reverse CCITT on String	`GenerateReverseCcittOnString(`*`Start Value, On String`*`)`	f
Generate Reverse CRC-16 on String	`GenerateReverseCrc16OnString(`*`Start Value, On String`*`)`	f
Get Nth Character	`GetNthCharacter(`*`From String, Index`*`)`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get String Length	`GetStringLength(Of String)`	f
Get Substring	`GetSubstring(From String, Start at Index, Num. Characters, Put Result in)`	p
Move from String Table Element	`s = st[0];`	p
Move String	`s1 = s2;`	p
Move to String Table Element	`st[0] = s;`	p
Move to String Table Elements	`MoveToStrTableElements(From, Start Index, End Index, Of Table)`	p
Pack Float into String	`PackFloatIntoString(From Value, To String, Start Index, Width, Endian Type, Put Result In)`	f
Pack Integer 32 into String	`PackInt32IntoString(From Value, To String, Start Index, Width, Endian Type, Put Result In)`	f
Pack Integer 64 into String	`PackInt64IntoString(From Value, To String, Start Index, Width, Endian Type, Put Result In)`	f
Pack String into String	`PackStringIntoString(From Value, To String, Start Index, Width, Data Type, Put Result In)`	f
Set Nth Character	`SetNthCharacter(To, In String, At Index)`	f
String Equal?	`s1 == s2`	f
String Equal to String Table Element?	`s == st[0]`	f
Test Equal Strings	`s1 == s2`	f
Trim String	`TrimString(String, Option)`	f
Unpack String	`UnpackString(From String, Start Index, Width, To Value, Data Type, Endian Type, Put Result In)`	f
Verify Checksum on String	`VerifyChecksumOnString(Start Value, On String)`	f
Verify Forward CCITT on String	`VerifyForwardCcittOnString(Start Value, On String)`	f
Verify Forward CRC-16 on String	`VerifyForwardCrc16OnString(Start Value, On String)`	f
Verify Reverse CCITT on String	`VerifyReverseCcittOnString(Start Value, On String)`	f
Verify Reverse CRC-16 on String	`VerifyReverseCrc16OnString(Start Value, On String)`	f

Time/Date

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Copy Date to String (DD/MM/YYYY)	`DateToStringDDMMYYYY(String)`	p
Copy Date to String (MM/DD/YYYY)	`DateToStringMMDDYYYY(String)`	p
Copy Time to String	`TimeToString(String)`	p
Convert Date & Time to NTP Timestamp	`DateTimeToNtpTimestamp(Date&Time, NTP Timestamp, Put Result in)`	f
Convert NTP Timestamp to Date & Time	`NtpTimestampToDateTime(Date&Time, NTP Timestamp, Put Result in)`	f
Get Date & Time	`GetDateTime(Table)`	f
Get Day	`GetDay()`	f
Get Day of Week	`GetDayOfWeek()`	f
Get Hours	`GetHours()`	f
Get Julian Day	`GetJulianDay()`	f
Get Minutes	`GetMinutes()`	f
Get Month	`GetMonth()`	f
Get Seconds	`GetSeconds()`	f
Get Seconds Since Midnight	`GetSecondsSinceMidnight()`	f

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Get System Time	`GetSystemTime()`	f
Get Time Zone Description	`GetTimeZoneDescription(`*`Configuration, Description`*`)`	f
Get Time Zone Offset	`GetTimeZoneOffset(`*`Configuration`*`)`	f
Get Year	`GetYear()`	f
Set Date	`SetDate(`*`To`*`)`	p
Set Day	`SetDay(`*`To`*`)`	p
Set Hours	`SetHours(`*`To`*`)`	p
Set Minutes	`SetMinutes(`*`To`*`)`	p
Set Month	`SetMonth(`*`To`*`)`	p
Set Seconds	`SetSeconds(`*`To`*`)`	p
Set Time	`SetTime(`*`To`*`)`	p
Set Time Zone Configuration	`SetTimeZoneConfiguration(`*`Configuration`*`)`	f
Set Year	`SetYear(`*`To`*`)`	p
Synchronize Clock SNTP	`SynchronizeClockSNTP(`*`Timeout, Server URL, Put Result in`*`)`	f

Timing

PAC Control Command	OptoScript Equivalent (Arguments)	Type
Continue Timer	`ContinueTimer(`*`Timer`*`)`	p
Delay (mSec)	`DelayMsec(`*`Milliseconds`*`)`	p
Delay (Sec)	`DelaySec(`*`Seconds`*`)`	p
Down Timer Expired?	`HasDownTimerExpired(`*`Down Timer`*`)`	f
Get & Restart Timer	`GetRestartTimer(`*`Timer`*`)`	f
Pause Timer	`PauseTimer(`*`Timer`*`)`	p
Set Down Timer Preset Value	`SetDownTimerPreset(`*`Target Value, Down Timer`*`)`	p
Set Up Timer Target Value	`SetUpTimerTarget(`*`Target Value, Up Timer`*`)`	p
Start Timer	`StartTimer(`*`Timer`*`)`	p
Stop Timer	`StopTimer(`*`Timer`*`)`	p
Timer Expired?	`HasTimerExpired(`*`Timer`*`)`	f
Up Timer Target Time Reached?	`HasUpTimerReachedTargetTime(`*`Up Timer`*`)`	f

SYMBOLS

Symbol	Usage	See
`-`	`-x`	“Chapter Complement” on page 500
	`x - y`	“Chapter Subtract” on page 523
`%`	`x % y`	“Chapter Modulo” on page 512
`+`	`x + y`	“Chapter Add” on page 492
`*`	`x * y`	“Chapter Multiply” on page 513
`/`	`x / y`	“Chapter Divide” on page 504

Symbol	Usage	See
+=	s1 += “a”;	“Chapter Append Character to String” on page 643
	s1 += s2;	“Chapter Append String to String” on page 645
<	x < nt[0]	“Chapter Less Than Numeric Table Element?” on page 445
	x < y	“Chapter Less?” on page 451
<<	x << nBitsToShift	“Chapter Bit Shift” on page 421
<=	x <= nt[0]	“Chapter Less Than or Equal to Numeric Table Element?” on page 447
	x <= y	“Chapter Less Than or Equal?” on page 449
<>	n <> nt[0]	“Chapter Not Equal to Numeric Table Element?” on page 457
	x <> y	“Chapter Not Equal?” on page 459
=	nt[0] = x;	“Chapter Move to Numeric Table Element” on page 547
	nt1[0] = nt2[5];	“Chapter Move Numeric Table Element to Numeric Table” on page 545
	pn = &n;	“Chapter Move to Pointer” on page 599
	pn = pt[0];	“Chapter Move from Pointer Table Element” on page 598
	pn1 = null;	“Chapter Clear Pointer” on page 595
	pt[0] = &n;	“Chapter Move to Pointer Table Element” on page 602
	pt[0] = null;	“Chapter Clear Pointer Table Element” on page 596
	s = st[0];	“Chapter Move from String Table Element” on page 680
	s1 = s2;	“Chapter Move String” on page 682
	st[0] = s;	“Chapter Move to String Table Element” on page 683
	x = nt[0];	“Chapter Move from Numeric Table Element” on page 544
	x = y;	“Chapter Move” on page 542
==	n == nt[0]	“Chapter Equal to Numeric Table Element?” on page 428
	pn == null	“Chapter Pointer Equal to Null?” on page 605
	pt[0] == null	“Chapter Pointer Table Element Equal to Null?” on page 606
	s == st[0]	“Chapter String Equal to String Table Element?” on page 694
	s1 == s2	“Chapter String Equal?” on page 696
	x == y	“Chapter Equal?” on page 430
>	x > nt[0]	“Chapter Greater Than Numeric Table Element?” on page 436
	x > y	“Chapter Greater?” on page 442
>=	x >= t[0]	“Chapter Greater Than or Equal To Numeric Table Element?” on page 438
	x >= y	“Chapter Greater Than or Equal?” on page 440

2: Analog Point Commands

Calculate & Set Analog Gain

Analog Point Action

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

Typical Uses: To improve calibration on an input.

Details:

- For all SNAP PAC and Ethernet brains, reads the current value of a specified analog input and interprets it as the point's maximum nominal value.
- Calculates a gain based on the current value that will cause this value to read the point's maximum nominal value.
- Stores the calculated gain in *Put Result in* (Argument 1) for subsequent use by Set Analog Gain, if desired.
- Use a calibrator to input the signal corresponding to the point's maximum nominal value.
- 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 0	Argument 1
On Point	**Put Result in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Calculate & Set Analog Gain		
Argument Name	Type	Name
On Point	*Analog Input*	*Boiler_Temperature*
Put Result in	*Float Variable*	*Gain_Coefficient*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Ethernet brains only: Instead of using this command, we recommend that you calibrate inputs using PAC Manager. For instructions, see the *PAC Manager User's Guide* (form 1704).

- For more information on offset and gain, see the Using Offset and Gain technical note (form 1359).

Dependencies:
- Always use Calculate & Set Analog Offset before using this command.
- Always set the analog input to the point's maximum nominal value before using this command.

See Also:
"Calculate & Set Analog Offset" on page 25
"Set Analog Gain" on page 48
"Set Analog Offset" on page 52

Calculate & Set Analog Offset

Analog Point Action

Function: To improve accuracy of an analog input signal.

Typical Uses: To improve calibration on an input.

Details:

- For all SNAP PAC and Ethernet brains, use a calibrator to input the signal corresponding to zero engineering units on the analog input point.
- Stores the calculated offset in *Put Result in* (Argument 1) 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 0	**Argument 1**
On Point	**Put Result in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Calculate & Set Analog Offset		
Argument Name	Type	Name
On Point	*Analog Input*	*Boiler_Temperature*
Put Result in	*Integer 32 Variable*	*OFFSET*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Instead of using this command, we recommend that you calibrate inputs on Ethernet brains using PAC Manager. For instructions, see the *PAC Manager User's Guide* (form 1704).
- This command is intended to be used in conjunction with Calculate & Set Analog Gain.
When using this command, first calculate and set the offset, and then calculate and set the gain. The offset must be calculated at the signal corresponding to zero engineering units (EU), and the gain must be calculated at the signal corresponding to the point's maximum input range value-or, for inverted scaling (also called "negative scaling"), the point's minimum input range value. (In inverted scaling, the lower scaled value is greater than the upper scaled value).
- For more information on offset and gain, see the Using Offset and Gain technical note (form 1359).

See Also:

"Calculate & Set Analog Gain" on page 23
"Set Analog Gain" on page 48
"Set Analog Offset" on page 52

Clear Counter 64

Analog Point Action

Function: To reset the accumulated energy or summation channels on a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module.

Typical Use: To reset any of the following 64-bit analog input channels configured on a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module:

- 13–17 (Phase A accumulated energy)
- 31–35 (Phase B accumulated energy)
- 49–53 (Phase C accumulated energy)
- 57–62 (Summation)

Details: Resets the specified analog input to zero as soon as it is used.

Arguments:

Argument 0	Argument 1
On Channel	**Put Status In**
Analog Input	Float Variable Integer 32 Variable

Action Block Example:

Clear Counter 64		
Argument Name	Type	Name
On Channel	*Analog Input*	*Panel_A_PM_Mod_CH13_PhsA_NetEn*
Put Status In	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

ClearCounter64(*On Channel*)

```
Status_Code = ClearCounter64(Panel_A_PM_Mod_CH13_PhsA_NetEn);
```

This is a function command; it returns one of the status codes below. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Dependencies: Applies only to analog inputs on GRV-IVAPM-3 or GRV-R7-I1VAPM-3 modules.

Status Codes: 0 = Success.

-38 = Timeout on send.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-58 = No data received. I/O unit may be turned off or unreachable.

-444 = Error on socket send.

-8612 = Old response to new command.

See Also:
"Get Counter 64" on page 35
"Get & Clear Counter 64" on page 30
"Set Counter 64" on page 56

or used in an item such as a Condition block, an OptoScript block, or a mathematical expression

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.
- 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.

Arguments:

Argument 0	Argument 1
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Clear Analog Maximum Value		
Argument Name	Type	Name
From	*Analog Input*	*Pres_Sensor*
Put in	*Float Variable*	*MAX_KPA*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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

See Also: "Get & Clear Analog Minimum Value" on page 28
"Get Analog Minimum Value" on page 33

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 0	Argument 1
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Clear Analog Minimum Value		
Argument Name	Type	Name
From	*Analog Input*	*PRES_SENSOR*
Put in	*Float Variable*	*MIN_KPA*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 27
"Get Analog Maximum Value" on page 32

Get & Clear Analog Totalizer Value

Analog Point Action

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, and 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 either limit.
- Totalizing will resume after using Get & Clear Analog Totalizer Value.
- Totalizing will stop if the input goes under range.

Arguments:

Argument 0	**Argument 1**
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Clear Analog Totalizer Value

Argument Name	Type	Name
From	*Analog Input*	*Flow_Rate*
Put in	*Float Variable*	*Total_Barrels*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the PAC Control User's Guide (form 1700).

Notes:

- Before using this command, use Set Analog Totalizer Rate once to:
  - (for *groov* EPIC processors) set the sampling rate.
  - (for all other supported controllers and brains) set the sampling rate and start the totalizer.

  Use this command to clear the total before actual readings start.
- See notes for Set Analog Totalizer Rate before using this command.
- Do not use this command frequently when the total is a small value. Doing so may degrade the cumulative accuracy.

Dependencies: Available on *groov* EPIC processors, SNAP PAC R-series controllers SNAP PAC EB-series and SB-series brains, and .

See Also:

"Get Analog Totalizer Value" on page 34
"Set Analog Totalizer Rate" on page 53

Get & Clear Counter 64

Analog Point Action

Function: To read and clear select 64-bit analog input channels.

Typical Use: To measure the accumulated amount of energy—per phase or for all phases combined—consumed by equipment.

Details:

- Reads the current value of any of the following analog input channels configured on a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module and places it in *Put In* (Argument 1):
  - 13–17 (Phase A accumulated energy)
  - 31–35 (Phase B accumulated energy)
  - 49–53 (Phase C accumulated energy)
  - 57–62 (Summation channels)
- Sets the specified analog input channel to zero. Does not stop the channel from continuing to accumulate or sum energy values.

Arguments:

Argument 0	Argument 1	Argument 2
Put in	**From Channel**	**Put Status in**
Integer 64 Variable	Analog Input	Float Variable Integer 32 Variable

Action Block Example:

Get & Clear Counter 64		
Argument Name	Type	Name
Put in	*Integer 64 Variable*	*Stn_2_Phase_A_Net_Energy*
From Channel	*Analog Input*	*Panel_A_PM_Mod_Ch13_PhsA_NetEn*
Put Status in	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

GetClearCounter64 (*Put In, From Channel*)

```
Status_Code = GetClearCounter64(Stn_2_Phase_A_Net_Energy,Panel_A_PM_Mod_Ch13_PhsA_NetEn);
```

This is a function command; it returns one of the status codes below. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: This value rolls over from 9,223,372,036,854,775,807 to -9,223,372,036,854,775,808.

Dependencies: Applies to accumulation and summation values on a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module.

Status Codes:

0 = Success

-38 = Timeout on send.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-58 = No data received. I/O unit may be turned off or unreachable.

-444 = Error on socket send.

-8612 = Old response to new command.

See Also:
"Get Counter 64" on page 35
"Clear Counter 64" on page 26

or used in an item such as a Condition block, an OptoScript block, or a mathematical expression

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 0	Argument 1
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Get Analog Maximum Value		
Argument Name	Type	Name
From	*Analog Input*	*PRES_SENSOR*
Put in	*Float Variable*	*MAX_KPA*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 27
"Get & Clear Analog Minimum Value" on page 28
"Get Analog Minimum Value" on page 33

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 0	Argument 1
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Get Analog Minimum Value		
Argument Name	Type	Name
From	*Analog Input*	*PRES_SENSOR*
Put in	*Float Variable*	*MIN_KPA*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 Maximum Value" on page 27
"Get & Clear Analog Minimum Value" on page 28
"Get Analog Maximum Value" on page 32

Get Analog Totalizer Value

Analog Point Action

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 if the input goes under range.

Arguments:

Argument 0	Argument 1
From	**Put in**
Analog Input	Float Variable
	Integer 32 Variable

Action Block Example:

Get Analog Totalizer Value		
Argument Name	Type	Name
From	*Analog Input*	*Flow_Rate*
Put in	*Float Variable*	*Total_Barrels*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 begin.

Dependencies:

- Available on *groov* EPIC processors, SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.
- Before using this command, Set Analog Totalizer Rate must be executed.

See Also:

"Get & Clear Analog Totalizer Value" on page 29
"Set Analog Totalizer Rate" on page 53

Get Counter 64

Analog Point Action

Function: To read the 64-bit analog input values on the following channels of a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module:

- 13–17 (Phase A accumulated energy)
- 31–35 (Phase B accumulated energy)
- 49–53 (Phase C accumulated energy)
- 57–62 (Summation)

Typical Use: To read accumulation or summation energy values from electrical equipment.

Details:

- Reads the current value of an energy accumulation or summation point and places it in *Put in* (argument 1).
- Does *not* reset the value at the channel to zero.
- Does *not* stop the counter from continuing to count.
- The counter rolls over from 9,223,372,036,854,775,807 to -9,223,372,036,854,775,808.

Arguments:

Argument 0 **Put in**	Argument 1 **From Channel**	Argument 2 **Put Status in**
Integer 64 Variable	Analog Input	Float Variable Integer 32 Variable

Action Block Example:

Get Counter 64		
Argument Name	Type	Name
Put in	*Integer 64 Variable*	*Stn_2_Phase_A_Net_Energy*
From Channel	*Analog Input*	*Panel_A_PM_Mod_Ch13_PhsA_NetEn*
Put Status in	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

GetCounter64(*Put in, From Channel***)**

```
Status_Code = GetCounter64(Stn_2_Phase_A_Net_Energy, Panel_A_PM_Mod_Ch13_PhsA_NetEn);
```

This is a function command; it returns one of the status codes below. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Dependencies: Applies to GRV-IVAPM-3 and GRV-R7-I1VAPM-3 modules.

Status Codes:

0 = Success

-38 = Timeout on send.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-58 = No data received. I/O unit may be turned off or unreachable.

-444 = Error on socket send.

-8612 = Old response to new command.

See Also:
"Clear Counter 64" on page 26
"Set Counter 64" on page 56
"Get Counter" on page 177

Get HART Unique Address

Analog Point Action

Function: Retrieves the unique address of a HART device.

Typical Use: To get a unique address of a HART device in order to use other HART commands to talk to the device.

Details: This command allows you to supply the polling address, and in return to get back the unique address you need for other HART commands.

The arguments are:

- *Point*: Indicate an input or output point on a HART module.
- *Polling Address*: HART polling address, a value of 0 to 63.
- *Unique Address*: Destination for the unique address retrieved by this command based on the polling address. For example: 11c87c8bcf
- *Timeout (Seconds)*: Timeout in seconds.
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3	Argument 4
Point	**Polling Address**	**Unique Address**	**Timeout (Seconds)**	**Put Result in**
Analog Input	Integer 32 Literal	String Variable	Float Literal	Integer 32 Variable
Analog Output	Integer 32 Variable		Float Variable	
			Integer 32 Literal	
			Integer 32 Variable	

Action Block Example:

Get HART Unique Address		
Argument Name	Type	Name
Point	*Analog Input*	*HART_0*
Polling Address	*Integer 32 Variable*	*PollAddress*
Unique Address	*String Variable*	*UniqueAdd*
Timeout (Seconds)	*Float Variable*	*Timeout*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

GetHARTUniqueAddress (*Point, Polling Address, Unique Address, Timeout (Seconds)*)

```
Result = GetHARTUniqueAddress(HART_0, PollAddress, UniqueAdd, Timeout);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Note: This command and the other HART-specific PAC Control commands work with HART SNAP I/O modules in a SNAP PAC system with a SNAP PAC programmable automation controller and PAC Control automation software. These HART-specific PAC Control commands make it possible for these modules to communicate any HART command that adheres to the HART request-response model or the burst message model.

In your PAC Control strategy you will need to assemble all request data sent to the module and to parse all response data returned by the module. In this way the modules can communicate with any wired HART device using any HART command.

Status Codes: 0 = Success.

-2 = Data verification field mismatch.

-3 = Invalid length.

-8 = Invalid data.

-20 = Resource busy. This error is returned if no response is available within the supplied timeout period.

-43 = NACK received.

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

-107 = I/O unit not enabled. No attempt to read or write was made. Reenable the I/O unit and try again.

-443 = Could not receive on socket.

-444 = Error on socket send.

-8612 = Old response to new command.

-13013 = Operation failed.

See Also:
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691
"Unpack String" on page 700

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 PAC brains, *mistic* I/O units, and some legacy Ethernet brains. For a detailed comparison of features supported by Ethernet-based SNAP I/O brains and on-the-rack controllers, see Legacy and Current SNAP Product Comparison and Compatibility Charts (form 1693).
- For SNAP I/O units, this command works only on the first four channels (channels 0-3) of a module.
- 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 0	Argument 1	Argument 2
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	

Action Block Example:

Ramp Analog Output		
Argument Name	Type	Name
Ramp Endpoint	*Float Variable*	*SOAK_TEMP*
Units/Sec	*Float Variable*	*RAMP_RATE*
Point to Ramp	*Analog Output*	*TEMP_CONTROL*

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.

Dependencies

Available on:

- *groov* EPIC processors
- *groov* RIO modules
- SNAP-PAC-R2, -R1, and -R1-B controllers
- SNAP-PAC-SB2, -SB1, -EB2, and -EB1 brains with firmware 8.1 or later

Queue Errors: -34 = Invalid I/O command. (The selected channel is out of range.)

-42 = Invalid limit. (The ramp rate was less than or equal to zero.)

Receive HART Burst Response

Analog Point Action

Function: Receives a response from a HART command burst— a special mode where periodic data is sent asynchronously from a sensor.

Typical Use: To receive responses from a HART command burst in which in which a sensor sends asynchronous data in response to a trigger, such as a set time period or a particular value.

Details: The arguments are:

- *Point*: A point on a HART module.
- *Unique Address*: HART device address (ex: 11c87c8bcf).
- *Response*: Destination for HART response.
- *Timeout (Seconds)*: Timeout, in seconds.
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0 **Point**	Argument 1 **Unique Address**	Argument 2 **Response**	Argument 3 **Timeout (Seconds)**	Argument 4 **Put Result in**
Analog Input	String Literal	String Variable	Float Literal	Integer 32 Variable
Analog Output	String Variable		Float Variable	
			Integer 32 Literal	
			Integer 32 Variable	

Action Block Example:

Receive HART Burst Response		
Argument Name	Type	Name
Point	*Analog Input*	*HART_0*
Unique Address	*String Variable*	*sUniqueAddr*
Response	*String Variable*	*sRcv*
Timeout (Seconds)	*Float Variable*	*fTimeout*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

ReceiveHARTBurstResponse(*Point, Unique Address, Response, Timeout (Seconds)***)**

```
Result = ReceiveHARTBurstResponse(HART_0, sUniqueAddr, sRcv, fTimeout);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- The Receive HART Burst Response command receives *only* burst responses and keeps them separate from standard command responses. The last burst response is buffered until requested by the user; if a new burst response arrives before the buffered response is retrieved, the old response is overwritten by the new one.
- This command and the other HART-specific PAC Control commands work with HART SNAP I/O modules in a SNAP PAC system with a SNAP PAC programmable automation controller and PAC Control automation software. These HART-specific PAC Control commands make it possible for these modules to communicate any HART command that adheres to the HART request-response model or the burst message model.

In your PAC Control strategy you will need to assemble all request data sent to the module and to parse all response data returned by the module. In this way the modules can communicate with any wired HART device using any HART command.

Status Codes: 0 = Success.

-2 = Data verification field mismatch.

-20 = Resource busy. This error is returned if no response is available within the supplied timeout period.

-23 = String too short.

-34 = Invalid I/O command.

-40 = Not enough data returned.

-43 = NACK received.

-443 = Could not receive on socket.

-13013 = Operation failed.

See Also: "Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691
"Unpack String" on page 700

Receive HART Response

Analog Point Action

Function: Receives the response to a command that was sent previously.

Typical Use: If you use Send/Receive HART Command and get a -443 (could not receive on socket), you can try again with this instruction. If a response from a previous command is available, this instruction will receive it.

This command might be used in the case where you send a series of commands to multiple HART devices without waiting for a response, then go back to pick up those responses. Because HART is relatively slow, using this method would be much faster than sending a command to one device, waiting for its response, sending a command to another device and waiting for its response, and so on.

Details: The arguments are:

- *Point*: A point on a HART module.
- *Unique Address*: HART device address (ex: 11c87c8bcf).
- *Response*: Destination for HART response.
- *Timeout (Seconds)*: Timeout, in seconds.
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0 **Point**	Argument 1 **Unique Address**	Argument 2 **Response**	Argument 3 **Timeout (Seconds)**	Argument 4 **Put Result in**
Analog Input	String Literal	String Variable	Float Literal	Integer 32 Variable
Analog Output	String Variable		Float Variable	
			Integer 32 Literal	
			Integer 32 Variable	

Action Block Example:

Receive HART Response

Argument Name	Type	Name
Point	*Analog Input*	*HART_0*
Unique Address	*String Variable*	*sUniqueAddr*
Response	*String Variable*	*sRcv*
Timeout (Seconds)	*Float Variable*	*fTimeout*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

ReceiveHARTResponse(*Point, Unique Address, Response, Timeout (Seconds)***)**

```
Result = ReceiveHARTResponse(HART_0, sUniqueAddr, sRcv, fTimeout);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Note: This command and the other HART-specific PAC Control commands work with HART SNAP I/O modules in a SNAP PAC system with a SNAP PAC programmable automation controller and PAC Control automation software. These HART-specific PAC Control commands make it possible for these modules to communicate any HART command that adheres to the HART request-response model or the burst message model.

In your PAC Control strategy you will need to assemble all request data sent to the module and to parse all response data returned by the module. In this way the modules can communicate with any wired HART device using any HART command.

Status Codes:

0 = success

-3 = Invalid length.

-8 = Invalid data.

-20 = Resource busy. This error is returned if no response is available within the supplied timeout period.

-23 = String too short.

-34 = Invalid I/O command.

-40 = Not enough data returned.

-43 = NACK received.

-107 = I/O unit not enabled. No attempt to read or write was made. Reenable the I/O unit and try again.

-443 = Could not receive on socket.

-444 = Error on socket send.

-13013 = Fail.

See Also:

"Get HART Unique Address" on page 37
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691
"Unpack String" on page 700

Send/Receive HART Command

Analog Point Action

Function: Sends a HART command and receives its response in a single operation.

Typical Use: To send a HART command and receive a response in a single operation.

Details: The arguments are:

- *Point*: A point on a HART module.
- *Unique Address*: Enter the unique HART device address, for example, 11c87c8bcf. You can use the Get HART Unique Address command to obtain the address.
- *Command*: The HART command number.
- *Parameters*: Parameters needed by the command (created using the string packing commands). Note that HART uses Big-Endian order.
- *Response*: Destination for HART response.
- *Timeout (Seconds)*: Timeout, in seconds.
  - Setting the timeout to zero results in non-blocking behavior. If the response isn't immediately ready, the command returns with a "resource busy" error (-20). You might want to do this if you're sending commands to a series of devices and don't want to wait for each response to complete. A much faster method is to do the sends, and then pick up the responses when you've finished.
  - Depending on the value you use, you may or may not get a response within your timeout period. If a response isn't available before you time out, you'll get a socket receive error (-443).
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
Point	**Unique Address**	**Command**	**Parameters**
Analog Input	String Literal	Integer 32 Literal	String Literal
Analog Output	String Variable	Integer 32 Variable	String Variable

Argument 4	Argument 5	Argument 6
Response	**Timeout (Seconds)**	**Put Result in**
String Variable	Float Literal	Integer 32 Variable
	Float Variable	
	Integer 32 Literal	
	Integer 32 Variable	

Action Block Example:

Send/Receive HART Command

Argument Name	Type	Name
Point	*Analog Input*	*HART_0*
Unique Address	*String Variable*	*UniqueAdd*
Command	*Integer 32 Variable*	*0*
Parameters	*String Variable*	*sParam*
Response	*String Variable*	*sRcv*
Timeout (Seconds)	*Float Variable*	*fTimeout*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

SendReceiveHARTCommand(*Point, Unique Address, Command, Parameters, Response, Timeout (Seconds)***)**

```
Result = SendReceiveHARTCommand(HART_0, UniqueAdd, 0, sParam, sRcv,
fTimeout);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Note: HART® SNAP I/O modules for analog current input and output provide communication with other Highway Addressable Remote Transducer (HART) current loop devices. These modules allow you to use any HART command that adheres to the HART request-response model or the burst message model. All request data sent to the module must be put together by the user, and all response data returned by the module must be parsed by the user. This allows the HART SNAP I/O modules to communicate to any wired HART device with any command, provided that the user can construct and parse the data of that command.

Status Codes: 0 = Success.

-3 = Invalid length.

-8 = Invalid data.

-20 = Resource busy. This error is returned if no response is available within the supplied timeout period.

-23 = String too short.

-34 = Invalid I/O command.

-40 = Not enough data returned.

-43 = NACK received.

-107 = I/O unit not enabled. No attempt to read or write was made. Reenable the I/O unit and try again.

-443 = Could not receive on socket.

-444 = Error on socket send.

-8612 = Old response to new command.

-13013 = Operation failed.

See Also:
"Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691
"Unpack String" on page 700

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. See the "Analog & High-Density Digital Scanner" on the Status Read window in PAC Manager.
- A zero disables filtering. A larger value increases filtering.
- Filtering is applied to the engineering units value as well as the min and max values.
- The digital filtering algorithm is an implementation of a first-order lag filter: New Filtered Value = ((Current Reading - Old Filter Value) / Filter Weight) + Old Filter Value.
- To calculate the filter weight value that will result in a particular time constant value, use: Filter Weight = (Time Constant + Scan Interval)/Scan Interval. (Time values are in seconds.)
- To calculate the time constant that a particular filter weight will result in, use:
  Time Constant = (Scan Interval * Filter Weight) – Scan Interval. (Time values are in seconds.)

  A Time Constant is the amount of time it will take to reach 63.21% of the final value. After an amount of time equal to five time constants, the value will be very close to its final value (99.33% of the final value).
- The filter weight will be used until power is removed from the I/O unit, or it will always be used if it is stored in permanent memory at the I/O unit.

Arguments:

Argument 0	Argument 1
To	**On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Set Analog Filter Weight		
Argument Name	Type	Name
To	*Integer 32 Variable*	*FILTER_WEIGHT*
On Point	*Analog Input*	*TEMP_IN1*

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 an input.

Details:

- Always use Set Analog Offset before using this command.
- The default value is 1.0.
- For all SNAP PAC and Ethernet brains, a setting of 0.0 will cause the brain to use the default value of 1.0.
- 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 0	Argument 1
To	**On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Set Analog Gain		
Argument Name	Type	Name
To	*Float Variable*	*GAIN_COEFFICIENT*
On Point	*Analog Input*	*PRESS_IN*

OptoScript Example:

SetAnalogGain(*To, On Point***)**

```
SetAnalogGain(GAIN_COEFFICIENT, PRESS_IN);
```

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

Notes:

- 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 PAC Manager.)
- For more information on offset and gain, see the Using Offset and Gain technical note (form 1359).

Dependencies: Must use Set Analog Offset first.

See Also:

"Set Analog Offset" on page 52
"Calculate & Set Analog Gain" on page 23

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 PAC Display 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 0	Argument 1
To	**On Point**
Integer 32 Literal	Analog Input
Integer 32 Variable	

Action Block Example: This example sets the fast settle level of the analog point to 5.

Set Analog Load Cell Fast Settle Level		
Argument Name	Type	Name
To	*Integer 32 Literal*	*5*
On Point	*Analog Input*	*Load_Cell_A*

OptoScript Example:

SetAnalogLoadCellFastSettleLevel(*To, On Point***)**

```
SetAnalogLoadCellFastSettleLevel(5, Load_Cell_A);
```

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

Note: 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 PAC Manager.)

Dependencies: This command is valid only when used on a properly configured SNAP-AILC module.

See Also: "Set Analog Load Cell Filter Weight" on page 51

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 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 each time the module scans the 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 Fast Settle Level) to get the filtered reading faster.
- The filtered reading is on channel 2 of the SNAP-AILC module.

Arguments:

Argument 0	**Argument 1**
To	**On Point**
Integer 32 Literal	Analog Input
Integer 32 Variable	

Action Block Example: This example sets the filter weight to 25.

Set Analog Load Cell Filter Weight		
Argument Name	Type	Name
To	*Integer 32 Literal*	*25*
On Point	*Analog Input*	*Load_Cell_A*

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 PAC Manager.)
- The filtered weight is reduced when the difference between the unfiltered value and the filtered value 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" on page 49

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 an input.

Details:

- Always use Set Analog Gain after using this command.
- The default offset value is 0.
- For all SNAP PAC and Ethernet brains, the offset value is in engineering units.
- 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 0 **To**	Argument 1 **On Point**
Float Literal Float Variable Integer 32 Literal Integer 32 Variable	Analog Input

Action Block Example:

Set Analog Offset

Argument Name	Type	Name
To	*Integer 32 Variable*	*OFFSET*
On Point	*Analog Input*	*PRESS_IN*

OptoScript Example:

SetAnalogOffset(*To, On Point***)**

```
SetAnalogOffset(OFFSET, PRESS_IN);
```

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

Notes:

- 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 PAC Manager.)
- For more information on offset and gain, see the Using Offset and Gain technical note (form 1359).

See Also:

"Set Analog Gain" on page 48
"Calculate & Set Analog Offset" on page 25

Set Analog Totalizer Rate

Analog Point Action

Function: For *groov* EPIC processors, to set the sampling rate. For all other supported controllers and brains, to start the totalizer and to set the sampling rate.

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

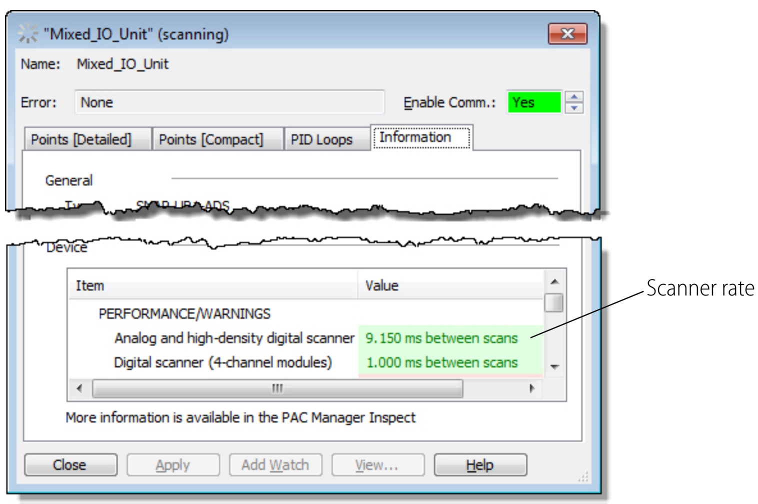
Details:

- For *groov* EPIC processor, this command only sets the sampling rate. To start sampling, you need to run the Get and Clear Analog Totalizer Value command.
- SNAP PAC brains continuously add fractional samples to the totalized value, rather than one full sample at the end of the period.
- The sampled value is added to the previous accumulated total.
- Valid range for the sampling rate is 0.0 to 3276.7 seconds for *mistic* I/O units.
- SNAP PAC brains stop totalizing when the totalized input goes out of range or offline, and continue totalizing when the input becomes valid again.
- SNAP PAC brains allow totalize rates as slow as every 4294967.295 seconds (49.7 days), where the sampling rate resolution is typically on the order of milliseconds; it depends on the rate of the Analog & High-Density Digital Scanner. For more information, see the *OptoMMP Protocol Guide* (form 1465).

  To see the scanner rate in PAC Control:

  a. Open the strategy in PAC Control in debug mode.

  b. Double-click the I/O unit, and then click the Information tab.



- Setting the sampling rate to 0.0 seconds will discontinue totalizing, but not clear or change the total.
- Setting the sampling rate to a non-zero value will clear any existing total.
- Totalizing will be bi-directional if the input range is bi-directional, such as -10 to +10.

Arguments:

Argument 0	**Argument 1**
To (Seconds)	**On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Set Analog Totalizer Rate		
Argument Name	Type	Name
To (Seconds)	*Float Variable*	*TOTALIZE_RATE*
On Point	*Analog Input*	*FUEL_FLOW*

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, but watch that the total doesn't overflow the destination variable.
- The following series of commands reads the accumulated total from the I/O unit, scales it, and 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*

Dependencies: Available on *groov* EPIC processors, SNAP PAC R-series controllers and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also: "Get Analog Totalizer Value" on page 34
"Get & Clear Analog Totalizer Value" on page 29

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 resolution, it is rounded to the nearest period value.
- For a SNAP-AOD-29-HFi module, TPO periods are multiples of 20.8 nanoseconds, ranging from 0.00001 sec to 64.25 sec. If the value entered is not an exact multiple of the period resolution, it is rounded to the nearest 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 0	Argument 1
To (Seconds)	**On Point**
Float Literal	Analog Output
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block 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

Argument Name	Type	Name
To (Seconds)	*Float Literal*	*5.0*
On Point	*Analog Input*	*TPO_OUTPUT*

OptoScript Example: **SetAnalogTpoPeriod(***To (Seconds), 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 PAC Control. For more information, see the *PAC Control User's Guide* (form 1700).
- 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 Counter 64

Analog Point Action

Function: To set a specific 64-bit analog input value to any of the following channels of a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module:

- 13–17 (Phase A accumulated energy)
- 31–35 (Phase B accumulated energy)
- 49–53 (Phase C accumulated energy)
- 57–62 (Summation)

Typical Use: To set a specific 64-bit analog input value to any of the accumulation or summation channels on a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module.

Details:

- Counters start as soon as they are configured.
- Does not reset the counter to zero.
- The new value is immediately saved to permanent storage.
- The range of valid values is 9,223,372,036,854,775,807 to -9,223,372,036,854,775,808

Arguments:

Argument 0	Argument 1	Argument 2
To	**On Channel**	**Put Status In**
Integer 32 Literal	Analog Input	Float Variable
Integer 32 Variable		Integer 32 Variable
Integer 64 Literal		
Integer 64 Variable		

Action Block Example:

Set Counter 64

Argument Name	Type	Name
To	*Integer 64 Variable*	*Stn_2_Net_Energy*
On Channel	*Analog Input*	*Panel_A_PM_Mod_CH13_PhsA_NetEn*
Put Status In	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

SetCounter64(*To, On Channel***)**

```
Status_Code = SetCounter64(Stn_2_Net_Energy,Panel_A_PM_Mod_CH13_PhsA_NetEn);
```

This is a function command; it returns one of the status codes below. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = Success

-38 = Timeout on send.

-39 = Timeout on receive.

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-58 = No data received. I/O unit may be turned off or unreachable.

-444 = Error on socket send.

-8612 = Old response to new command.

Notes: Use Clear Counter 64 or Get & Clear Counter 64 to clear a counter to zero.

Dependencies:

- Applies to digital inputs configured with the counter feature.
- Available on *groov* EPIC processors and *groov* RIO modules.

See Also:

"Clear Counter 64" on page 26
"Get & Clear Counter 64" on page 30
"Get Counter 64" on page 35

3: Chart Commands

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. Since each called chart ends before the next chart is called, this reduces 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. Keep in mind that the suspended chart still takes up a task in the queue. Call Chart attempts to start the specified chart and if successful, suspends the chart that issued the command. When the called chart finishes, the calling chart automatically continues.
- The status variable indicates success (0) or failure (error code -5 if the chart is already running).

Arguments:

Argument 0	Argument 1
Chart	**Put Status in**
Chart	Float Variable Integer 32 Variable

Action Block Example:

Call Chart		
Argument Name	Type	Name
Chart	*Chart*	*Tank_Monitor*
Put Status in	*Integer 32 Variable*	*Call_Status*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- Make sure to check the STATUS variable to ensure the chart has actually started.

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

Status Codes: 0 = success

-5 = Operation Failed. Possible causes on a non-redundant system:

  – The maximum number of charts is already running.
  – The chart is already running.

-36 = Invalid command or feature not implemented. This code will be returned on a redundant system.

See Also: "Continue Calling Chart" on page 67
"Start Chart" on page 71
"Suspend Chart" on page 73

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: If the calling chart is running, the logic will take the True path.
If the calling chart is *not* running, the logic will take the False path.

Arguments: None.

Condition Block Example:

Calling Chart Running?
No arguments

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "Chart Commands" in the *PAC Control User's Guide* (form 1700).

See Also:

"Continue Calling Chart" on page 67
"Calling Chart Suspended?" on page 63
"Calling Chart Stopped?" on page 62

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: If the calling chart is stopped, the logic will take the True path.
If the calling chart is *not* stopped, the logic will take the False path.

Arguments: None.

Condition Block Example:

Calling Chart Stopped?
No arguments

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "Chart Commands" in the *PAC Control User's Guide* (form 1700).

See Also: "Continue Calling Chart" on page 67
"Calling Chart Suspended?" on page 63
"Calling Chart Running?" on page 61

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: If the calling chart is suspended, the logic will take the True path.
If the calling chart is *not* suspended, the logic will take the False path.

Arguments: None.

Condition Block Example:

Calling Chart Suspended?
No arguments

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Chart Commands" in the *PAC Control User's Guide* (form 1700).
- Always use before Continue Calling Chart to ensure its success. See the Continue Calling Chart action for details.

See Also:

"Continue Calling Chart" on page 67
"Calling Chart Stopped?" on page 62
"Calling Chart Running?" on page 61

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: If the specified chart is running, the logic will take the True path.
If the specified chart is *not* running, the logic will take the False path.

Arguments:

Argument 0
Is
Chart

Condition Block Example:

Chart Running?		
Argument Name	Type	Name
Is	*Chart*	*CHART_B*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Chart Commands" in the *PAC Control User's Guide* (form 1700).
- When a chart uses 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?" on page 66
"Chart Stopped?" on page 65
"Call Chart" on page 59,
"Start Chart" on page 71
"Stop Chart" on page 72

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 succeeds.

Details: If the specified chart is stopped, the logic will take the True path.
If the specified chart is *not* stopped, the logic will take the False path.

Arguments:

Argument 0
Is
Chart

Condition Block Example:

Chart Stopped?		
Argument Name	Type	Name
Is	*Chart*	*CHART_B*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "Chart Commands" in the *PAC Control User's Guide* (form 1700).

See Also:

"Chart Suspended?" on page 66
"Chart Running?" on page 64
"Call Chart" on page 59
"Start Chart" on page 71
"Stop Chart" on page 72

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: If the specified chart is suspended, the logic will take the True path.
If the specified chart is *not* suspended, the logic will take the False path.

Arguments:

Argument 0
Is
Chart

Condition Block Example:

Chart Suspended?		
Argument Name	Type	Name
Is	*Chart*	*CHART_B*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Chart Commands" in the *PAC Control User's Guide* (form 1700).
- Use before Continue Chart to ensure success.

See Also:

"Chart Running?" on page 64
"Chart Stopped?" on page 65
"Continue Chart" on page 68
"Suspend Chart" on page 73

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 has 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, AND 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 0
Put Status in
Float Variable
Integer 32 Variable

Action Block Example:

Continue Calling Chart		
Argument Name	Type	Name
Put Status in	*Integer 32 Variable*	*STATUS*

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 the *PAC Control User's Guide* (form 1700).

See Also:

"Call Chart" on page 59
"Calling Chart Suspended?" on page 63

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 has 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 0	Argument 1
Chart	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Action Block Example:

Continue Chart

Argument Name	Type	Name
Chart	*Chart*	*CHART_A*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

ContinueChart(*Chart*)

```
STATUS = 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 the *PAC Control User's Guide* (form 1700).
- Loop on Chart Suspended? before this command if success is critical.

See Also:

"Suspend Chart" on page 73
"Chart Suspended?" on page 66

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. Applicable bits are 0–3 and 17:
  - 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)
  - Bit 17: Starting (0 = chart is not in the process of starting; 1 = chart is in the process of starting or is currently running)
- Unused bits from 4-16, 18-31 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.
- Starting Mode is on whenever the chart is in the process of starting or is currently running.
- A chart that is paused is considered to be running and in Step Mode.
- A chart that has never been started is considered stopped. A chart that is not suspended is either running or stopped.

Arguments:

Argument 0	Argument 1
Chart	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Action Block Example:

Get Chart Status		
Argument Name	Type	Name
Chart	*Chart*	*CHART_A*
Put Status in	*Integer 32 Variable*	*STATUS*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 4-16, 18-31 can have any value
  – Starting Mode, Bit 17 = 1

- In order to check the stopped/running state of a chart, it is necessary to check bits 0 and 17. If the chart is stopped and is not in the process of starting, bits 0 and 17 will both be 0. If the chart is in the process of starting, but is not running yet, bit 0 will be 0 and bit 17 will be 1. If the chart is running, bit 0 will be 1 and bit 17 will be 1.
- Avoid putting the returned status into a float variable, since the bits cannot be tested.
- If the strategy is stopped, the status variable is not updated before it is stopped, so the bits are left in their last known state before stopping. For example, if you stop the strategy, Bit 0 is not updated and may incorrectly indicate that a chart is still running when it is not.

See Also:
"Chart Stopped?" on page 65
"Chart Running?" on page 64
"Bit Test" on page 423

Start Chart

Chart Action

Function: To request that a stopped chart begin executing at Block 0.

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

Details:

- This command is a request only. If the chart is stopped—and fewer than the maximum number of charts are running—then this chart will be added to the task queue and the command will succeed. Otherwise, the command has no effect.
- The maximum number of charts that can be running at any one time is based on the control engine you are using:
  - 32 charts (and Alternate Host Tasks*) on a SNAP PAC S-series controller
  - 16 charts (and Alternate Host Tasks*) on a SNAP PAC R-series
  - 64 charts (and Alternate Host Tasks*) on a SoftPAC controller

  * Each successful call to the command Start Alternate Host Task will reduce the number of charts than can be run simultaneously by one chart.
- Upon success, the chart will start at its next scheduled time in the task queue.

Arguments:

Argument 0	Argument 1
Chart	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Action Block Example:

Start Chart

Argument Name	Type	Name
Chart	*Chart*	*CHART_B*
Put Status in	*Integer 32 Variable*	*STATUS*

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 the *PAC Control User's Guide* (form 1700).
- Make sure to check the STATUS variable to ensure the chart has actually started.
- Use Continue Chart if you want to continue a suspended chart.

Dependencies: If the chart is stopped, then a chart must be available in the task queue.

Status Codes: 0 = success

-5 = failure

See Also: "Continue Chart" on page 68
"Stop Chart" on page 72

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 chart 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 0
Chart
Chart

Action Block Example:

Stop Chart		
Argument Name	Type	Name
Chart	*Chart*	*CHART_B*

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 the *PAC Control User's Guide* (form 1700).
- Use Suspend Chart if you want to continue a chart from where it left off.

See Also:

"Start Chart" on page 71
"Suspend Chart" on page 73
"Chart Stopped?" on page 65

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 Continue Chart), or they can be stopped (using Stop Chart).

Arguments:

Argument 0	Argument 1
Chart	**Put Status in**
Chart	Float Variable
	Integer 32 Variable

Action Block Example:

Suspend Chart		
Argument Name	Type	Name
Chart	*Chart*	*CHART_B*
Put Status in	*Integer 32 Variable*	*STATUS*

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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success.

-5 = failure.

See Also:

"Start Chart" on page 71
"Continue Chart" on page 68
"Chart Suspended?" on page 66

4: Communication Commands

Accept Incoming Communication

Communication Action

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

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 0	Argument 1
Communication Handle	**Put Result in**
Communication Handle	Float Variable
	Integer 32 Variable

Action Block Example:

Accept Incoming Communication

Argument Name	Type	Name
Communication Handle	*Communication Handle*	*SNAP_PAC_A*
Put Result in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

AcceptIncomingCommunication(*Communication Handle***)**

```
STATUS = AcceptIncomingCommunication(SNAP_PAC_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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Ports 22004 and 22005 are assigned specifically to Opto 22 and are the ports we recommend for communication via TCP. You *can* use other ports, but be aware that other processes may also try to use those ports.
- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- It is necessary to use the Listen for Incoming Communication command only once per port, even if you use the Accept Incoming Communication command several times.

- For those familiar with sockets programming, PAC Control 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.)
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

Status Codes: 0 = Success

-10 = Invalid port number. Check format 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" on page 91
"Get Number of Characters Waiting" on page 82
"Receive String" on page 107
"Open Outgoing Communication" on page 93
"Communication Open?" on page 79

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 0
Communication Handle
Communication Handle

Action Block Example:

Clear Communication Receive Buffer		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_B*

OptoScript Example:

ClearCommunicationReceiveBuffer(*Communication Handle***)**

```
ClearCommunicationReceiveBuffer(PAC_B);
```

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

Notes:

- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 78
"Get Number of Characters Waiting" on page 82
"Receive N Characters" on page 97

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

Action Block Example:

Close Communication		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_A*
Put Status in	*Integer 32 Variable*	*Ethernet_Status*

OptoScript Example:

CloseCommunication(*Communication Handle***)**

```
Ethernet_Status = CloseCommunication(PAC_A);
```

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

Notes:

- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- 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.
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.
- (FTP handles only) After the communication handle is open, you can set the FTP mode by issuing one of the following commands:
  - For Active FTP:

    ```
    i32ResultCode = SendCommunicationHandleCommand(cmh, "active");
    ```
  - For Passive FTP:

    ```
    i32ResultCode = SendCommunicationHandleCommand(cmh, "passive");
    ```

Status Codes:

0 = Success

-37 = Lock port timeout.

-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.

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

-437 = No acceptable socket interface found. An Ethernet "accept" or "open" was attempted, but no more sessions are available.

See Also:

"Open Outgoing Communication" on page 93
"Send Communication Handle Command" on page 113

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 0
Communication Handle
Communication Handle

Condition Block Example:

Communication Open?		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_A*

OptoScript Example:

IsCommunicationOpen(*Communication Handle***)**

```
if (IsCommunicationOpen(PAC_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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- 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 another command's 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.
- When using TCP/IP, use this command to evaluate if the command Open Outgoing Communication has been called on this handle. To evaluate the status of the handle later, use the response of the command Get Number of Characters Waiting. This command will indicate if a remote host has issued a message to close the session.

See Also:

"Accept Incoming Communication" on page 75
"Open Outgoing Communication" on page 93
"Close Communication" on page 78

Get Communication Handle Value

Communication Action

Function: Returns a string that is the current value (the arguments) of the communication handle.

Typical Use: To find out the current communication arguments for a communication handle.

Arguments:

Argument 0	Argument 1
From	**To**
Communication Handle	String Variable

Action Block Example:

Get Communication Handle Value		
Argument Name	Type	Name
From	*Communication Handle*	*COMM_B*
To	*String Variable*	*COMM_VALUE*

OptoScript Example:

GetCommunicationHandleValue(*From, To***)**

```
GetCommunicationHandleValue(COMM_B, COMM_VALUE);
```

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

Note: If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

See Also: "Set Communication Handle Value" on page 126

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 the *PAC Control User's Guide* form 1700.) 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 0	Argument 1
Communication Handle	**Put Status In**
Communication Handle	Integer 32 Variable

Action Block Example:

Get End-Of-Message Terminator		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_A*
Put Status In	*Integer 32 Variable*	*EOM_Term*

OptoScript Example:

`GetEndOfMessageTerminator``(`*Communication Handle*`)`

```
EOM_Term = GetEndOfMessageTerminator(PAC_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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Note: If this command returns an error code, the communication handle will close and need to be reopened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

Status Codes: -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.

See Also:

"Set End-Of-Message Terminator" on page 127
"Open Outgoing Communication" on page 93

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 0	Argument 1
Communication Handle	**Put In**
Communication Handle	Float Variable
	Integer 32 Variable

Action Block Example:

Get Number of Characters Waiting		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_A*
Put in	*Integer 32 Variable*	*CHAR_COUNT*

OptoScript Example:

`GetNumCharsWaiting(`*Communication Handle***`)`**

```
CHAR_COUNT = GetNumCharsWaiting(PAC_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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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).
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

NOTE: 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.

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 = Receive timeout.

-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.

-53 = Connection number not valid.

-443 = Socket receive error. Usually occurs when the connection was closed by the other side.

See Also: "Send Communication Handle Command" on page 113, especially the getpos and setpos commands

HTTP Get

Communication Action

Function: Sends a request to retrieve a web page.

Typical Use: To get HTTP content from a specific file and location.

Details:

- In order for this command to work, the controller must be configured with DNS and Gateway addresses. Make sure that the controller is configured with an IP address and the appropriate Subnet Mask, DNS, and Gateway for the network. For details, see the *PAC Manager User's Guide* (form 1704).
- For *Response Content* (Argument 0) and *Response Header* (Argument 1), make sure the table is big enough for the response to fit. Each element is filled with as much of the HTTP content as it will hold, then the next element is used.
- To change the timeouts for transmitting, receiving, and connecting, use element [0] of the table in *Get Header* (Argument 2). The default is 10,000 milliseconds (10 seconds).

  To change one or more timeouts from the default, specify the value as follows:

To do this:	Use this command where "nnnn" is the timeout in milliseconds
Set transmit timeout	"to.tx:nnnn"
Set receive timeout	"to.rx:nnnn"
Set connect timeout	"to.cx:nnnn"

  Each timeout command is independent of the others so you can use one, two, or all three of the commands in any order. If you use more than one command, put a space between each one.

  The following example sets the transmit timeout to 20 seconds and the receive timeout to 15 seconds. The connect timeout is not changed.

```
HTTP_SrcStrTblBody[0]="to.tx:20000 to.rx:15000"
```

Table name Element 0 Transmit timeout Receive timeout

HTTP_SrcStrTblBody[0]="to.tx:20000 to.rx:15000"

Table name Element 0 Transmit timeout Receive timeout

- *Port* (Argument 6) represents the port number. For example, 80 is the standard HTTP port; 443 is the standard for SSL.
- For *Security Mode* (Argument 3), use 0 (zero) for a non-secure connection. Use a number other than zero to use SSL (Secure Socket Layer).
  - Using 0 for *Security Mode* and 80 for *Port* (Argument 6) is the equivalent of typing HTTP into your browser.
  - Using a non-zero number for *Security Mode* and 443 for *Port* is the equivalent of typing HTTPS into your browser.
- *Hostname* (Argument 7) is the web address **without** the `http://` or `https://`.
  Example: `www.hostaddress.com`

- *URL Path* (Argument 4) is the specific sub-page for *Hostname* (Argument 7).
  For example, `/products` specifies the *products* sub-page. Otherwise, just use a forward slash (/) for the root.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3	Argument 4
Response Content	**Response Header**	**Get Header**	**Security Mode**	**URL Path**
String Table	String Table	String Table	Integer 32 Literal	String Literal
			Integer 32 Variable	String Variable

Argument 5	Argument 6	Argument 7	Argument 8
Put HTTP Status in	**Port**	**Hostname**	**Put Result in**
Integer 32 Variable	Integer 32 Literal	String Literal	Integer 32 Variable
	Integer 32 Variable	String Variable	

Action Block Example:

HTTP Get

Argument Name	Type	Name
Response Content	*String Table*	*DstStrTblBody*
Response Header	*String Table*	*DstStrTblHdr*
Get Header	*String Table*	*GET_SrcStrTblBody*
Security Mode	*Integer 32 Literal*	*0*
URL Path	*String Literal*	*StrCmd*
Put HTTP Status in	*Integer 32 Variable*	*nHttpStatus*
Port	*Integer 32 Literal*	*nPort*
Hostname	*String Literal*	*StrHostname*
Put Result in	*Integer 32 Variable*	*nSendStatus*

OptoScript Example:

`HttpGet(`*Response Content, Response Header, Get Header, Security Mode, URL Path, Put HTTP Status in, Port, Hostname***`)`**

```
nSendStatus = HttpGet(DstStrTblBody, DstStrTblHdr, GET_SrcStrTblBody, 0,
StrCmd, nHttpStatus, nPort, StrHostname);
```

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

Notes:

- If result = 0, the HTTP GET request was successfully sent to the server.
- If result < 0, there was an error executing this command.

HTTP Status Codes:

The following standard types of HTTP status codes may be returned.

100 series = Informational

200 series = Success

300 series = Redirection. The client must take additional action to complete the request.

400 series = Client error

Status Codes:

0 = Success.

-12 = Invalid table index. Try increasing the length of the string tables passed to this command.

-20 = Device busy. May be in use by another user or another application.

-26 = Unknown response. This error is returned if you attempt to do a secure HTTP GET but specify zero rather than one for the Security Mode. The host is expecting an encrypted response, but instead receives plain text, which it attempts to decrypt.

-34 = Invalid I/O command or invalid memory location.

-38 = Timeout on send.

-39 = Timeout on Receive.

-59 = Could not receive data. PAC Sim returns this error on a secure HTTP Get if 8192 bytes or more are returned.

-443 = Could not receive on socket.

-454 = Unable to connect to DNS server. Check DNS and gateway configuration.

-2000 = SSL: Bad certificate

-2001 = SSL: Certificate revoked

-2002 = SSL: Certificate expired

-2103 = SSL: Handshake failed.

-2104 = SSL: Handshake failed due to invalid or unverifiable certificate.

-2105 = SSL: Unspecified asynchronous platform error.

-2106 = SSL: SSL session is not open.

-2110 = SSL: Server failed to start.

-13019 = An argument in a string is bad.

See Also: "HTTP Post from String Table" on page 88
"HTTP Post Calculate Content Length" on page 87

HTTP Post Calculate Content Length

Communication Action

Function: Calculates the length (the number of characters) of the HTTP content stored in a string table.

Typical Use: When sending an HTTP Post from String Table, you have to tell the server the total length of the header and body data you're going to send. This command calculates the size of the data stream for that purpose.

Details:

- *Post Content* (Argument 0) and *Post Header* (Argument 1) are the tables you plan on sending with the HTTP Post from String Table command.
- Fill in the header and body tables with their data, and then call this command. The total length of the header and body data you're going to send is automatically placed in the header table element you specify in *Length Index* (Argument 2).
- In order for this command to work, the controller must be configured with DNS and Gateway addresses. Make sure that the controller is configured with an IP address and the appropriate Subnet Mask, DNS, and Gateway for the network. For details, see the *PAC Manager User's Guide* (form 1704).

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
Post Content	**Post Header**	**Length Index**	**Put Result in**
String Table	String Table	Integer 32 Literal Integer 32 Variable	Integer 32 Variable

Action Block Example:

HTTP Post Calculate Content Length		
Argument Name	Type	Name
Post Content	*String Table*	*SrcStrTblBody*
Post Header	*String Table*	*SrcStrTblHdr*
Length Index	*Integer 32 Literal*	*2*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

HttpPostCalcContentLength(*Post Content, Post Header, Length Index***)**

```
nResult = HttpPostCalcContentLength(SrcStrTblBody, SrcStrTblHdr, 2);
```

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

Notes:

- If result = 0, the HTTP content length was calculated successfully.
- If result < 0, there was an error executing this command.

Status Codes:

0 = Success.

-12 = Invalid table index value. Index was negative or greater than or equal to the table size.

-20 = Resource busy. May be in use by another user or another application.

-59 = Could not receive data. PAC Sim returns this error on a secure HTTP Post Calculate Content Length if 8192 bytes or more are returned.

See Also:

"Get Number of Characters Waiting" on page 82
"HTTP Post from String Table" on page 88

HTTP Post from String Table

Communication Action

Function: Posts simple form-type HTTP content stored in the strategy to an HTTP server. The response from the server is returned as a string table.

Typical Use: To send values for fields on a webpage and return the webpage with the fields filled in.

Details:

- In order for this command to work, the controller must be configured with DNS and Gateway addresses. Make sure that the controller is configured with an IP address and the appropriate Subnet Mask, DNS, and Gateway for the network. For details, see the *PAC Manager User's Guide* (form 1704).
- For *Response Content* (Argument 0) and *Response Header* (Argument 1), make sure the table is big enough for the response to fit. Each element is filled with as much of the HTTP content as it will hold, then the next element is used.
- To change the timeouts for transmitting, receiving, and connecting, use element [0] of the table in *Get Header* (Argument 2). The default is 10,000 milliseconds (10 seconds). To change one or more timeouts from the default, specify the value as follows:

To do this:	Use this command where "nnnn" is the timeout in milliseconds
Set transmit timeout	"to.tx:nnnn"
Set receive timeout	"to.rx:nnnn"
Set connect timeout	"to.cx:nnnn"

Each timeout command is independent of the others so you can use one, two, or all three of the commands in any order. If you use more than one command, put a space between each one.

The following example sets the transmit timeout to 20 seconds and the receive timeout to 15 seconds. The connect timeout is not changed.

```
HTTP_SrcStrTblBody[0]="to.tx:20000 to.rx:15000"
```

Table name Element 0 Transmit timeout Receive timeout

- *Port* (Argument 6) represents the port number. For example, 80 is the standard HTTP port; 443 is the standard for SSL.
- For *Security Mode* (Argument 3), use 0 (zero) for a non-secure connection. Use a number other than zero to use SSL (Secure Socket Layer).
  - Using 0 for *Security Mode* and 80 for *Port* (Argument 6) is the equivalent of typing HTTP into your browser.
  - Using a non-zero number for *Security Mode* and 443 for *Port* is the equivalent of typing HTTPS into your browser.
- *Hostname* (Argument 7) is the web address **without** the `http://` or `https://`.
  Example: `www.hostaddress.com`
- *URL Path* (Argument 4) is the specific sub-page for *Hostname* (Argument 7).
  For example, `/products` specifies the *products* sub-page. Otherwise, just use a forward slash (/) for the root.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3	Argument 4
Response Content	**Response Header**	**Post Content**	**Post Header**	**Security Mode**
String Table	String Table	String Table	String Table	Integer 32 Literal Integer 32 Variable

Argument 5	Argument 6	Argument 7	Argument 8	Argument 9
URL Path	**Put HTTP Status in**	**Port**	**Hostname**	**Put Result in**
String Table String Variable	Integer 32 Variable	Integer 32 Literal Integer 32 Variable	String Literal String Variable	Integer 32 Variable

Action Block Example:

HTTP Post from String Table

Argument Name	Type	Name
Response Content	*String Table*	*DstStrTblBody*
Response Header	*String Table*	*DstStrTblHdr*
Post Content	*String Table*	*SrcStrTblBody*
Post Header	*String Table*	*SrcStrTblHdr*
Security Mode	*Integer 32 Literal*	*0*
URL Path	*String Literal*	*StrUrlPath*
Put HTTP Status in	*Integer 32 Variable*	*nHttpStatus*
Port	*Integer 32 Literal*	*nPort*
Hostname	*String Literal*	*StrHostname*
Put Result in	*Integer 32 Variable*	*nSendStatus*

OptoScript Example:

HttpPostFromStringTable(*Response Content, Response Header, Post Content, Post Header, Security Mode, URL Path, Put HTTP Status in, Port, Hostname***)**

```
nSendStatus = HttpPostFromStringTable(DstStrTblBody, DstStrTblHdr
SrcStrTblBody, SrcStrTblHdr, 0, StrUrlPath, nHttpStatus, nPort,
StrHostname);
```

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

Notes:

- If result = 0, the HTTP content was posted successfully.
- If result < 0, there was an error executing this command.

HTTP Status Codes:

The following standard types of HTTP status codes may be returned.

100 series = Informational

200 series = Success

300 series = Redirection. The client must take additional action to complete the request.

400 series = Client error

500 = Internal server error

Status Codes:

0 = Success.

-20 = Device busy. May be in use by another user or another application.

-26 = An unknown response was received from the server.

-34 = Invalid I/O command or invalid memory location.

-38 = Timeout on send.

-39 = Timeout on Receive.

-50 = Open connection timeout. Could not establish connection within the timeout period. Make sure that the controller is configured with a DNS and Gateway for the network.

-59 = Could not receive data. PAC Sim returns this error on a secure HTTP Post from String Table if 8192 bytes or more are returned.

-438 = Could not create socket.

-443 = Could not receive on socket.

-454 = Unable to connect to DNS server. Check DNS and gateway configuration.

-450 = DNS could not resolve hostname.

-2000 = SSL: Bad certificate

-2001 = SSL: Certificate revoked

-2002 = SSL: Certificate expired

-2103 = SSL: Handshake failed.

-2104 = SSL: Handshake failed due to invalid or unverifiable certificate.

-2105 = SSL: Unspecified asynchronous platform error.

-2106 = SSL: SSL session is not open.

-2110 = SSL: Server failed to start.

-13019 = An argument in a string is bad.

Queue Error: -12 = Invalid table index value. The destination table is too small to hold the HTTP response. Increase the table's width or length.

See Also: "Get Number of Characters Waiting" on page 82
"HTTP Post Calculate Content Length" on page 87

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 server, and the session is opened by the client.)

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 0	Argument 1
Communication Handle	**Put Status In**
Communication Handle	Integer 32 Variable

Action Block Example:

Listen for Incoming Communication		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*SNAP_PAC_A*
Put Status In	*Integer 32 Variable*	*STATUS*

OptoScript Example:

`ListenForIncomingCommunication(`*Communication Handle***`)`**

```
STATUS = ListenForIncomingCommunication(SNAP_PAC_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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- You need to perform Listen for Incoming Communication only once per listening port. Until then, the controller will reject any incoming session requests.
- After performing Listen for Incoming Communication, use Accept Incoming Communication to accept any request for a new incoming connection. Note that Accept Incoming Connection will fail until a request has been received. Therefore, you should periodically perform Accept Incoming Connection so that it can accept any new request.
- To determine whether the connection is still open, use Get Number of Characters Waiting or Communication Open?
- The number of Ethernet sessions available to listen for incoming communications depends in part on the control engine and the number of Ethernet sessions in use. (Ethernet sessions are used for a variety of purposes, such as making connections for FTP, email, Modbus/TCP, Ethernet/IP, and so forth.)

  If you send this command when there are no more available sessions, a -438 code is returned.

  Here's a list of devices and the approximate number of total Ethernet sessions that can manage:

  – GRV-EPIC-PR1 ≈ 500 sessions
  – SNAP-PAC-S1 ≈ 100
  – SNAP-PAC-S2 ≈ 100
  – SNAP-PAC-R1 ≈ 75
  – SNAP-PAC-R1-B ≈ 75
  – SNAP-PAC-R2 ≈ 75
  – SoftPAC ≈ 400
- See also "Communication Commands" in the *PAC Control User's Guide* (form 1700).

Status Codes: For more information on the following status codes, see "List of Common Messages" in the *PAC Control User's Guide* (form 1700).

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.

-440 = Could not bind socket.

See Also:
"Accept Incoming Communication" on page 75
"Get Number of Characters Waiting" on page 82
"Communication Open?" on page 79
"Open Outgoing Communication" on page 93

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 via TCP/IP, UDP/IP, or a serial connection; send FTP data from the brain to a file on another device; or work with files in the brain's file structure.

Arguments:

Argument 0	Argument 1
Communication Handle	**Put Result in**
Communication Handle	Integer 32 Variable

Action Block Example:

Open Outgoing Communication		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*TANK_CONTROL*
Put Result in	*Integer 32 Variable*	*COMM_STATUS*

OptoScript Example:

`OpenOutgoingCommunication(`*Communication Handle***`)`**

```
COMM_STATUS = OpenOutgoingCommunication(TANK_CONTROL);
```

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

Notes:

- When using a SNAP PAC controller, Open Outgoing Communication can be used to open either Ethernet (TCP or UDP) or serial communication (using a serial port on the controller).
  Note: SNAP serial communication modules (SNAP-SCM) use TCP comm handles.
- When using SoftPAC, Open Outgoing Communication cannot be used with a computer's serial ports or USB serial ports. (SoftPAC doesn't support PC serial or USB ports.)
  However, the command can still be used with Ethernet (TCP and UDP) and SNAP-SCM comm handles.
- 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 *PAC Control User's Guide* (form 1700).
- If this command returns an error code, the communication handle closes and you'll need to re-open it. Be sure to check for status codes returned by this command, and handle any errors appropriately.

  An error typically happens when you're using the on-board 232 port on a SNAP-PAC-R and you haven't changed the default settings for that port. (It defaults to PPP.) You can resolve the issue either in your strategy (by using PAC Control) or by using PAC Manager.

  Using PAC Manager:

  a. Select Tools > Inspect.
  b. Click the Communications button and select Communications Port Control.
  c. In the Values column, click the first drop-down list, and then select None. (This is the value for Control Function for Communication Port 0, memory map address 0xFFFF F031 0400).
  d. Click Apply.
  e. Click the Status Write button.
  f. In the Operation area, click "Store configuration to flash," and then click the Send Command button. This stores the configuration to flash memory so that it retains the settings when the controller boots up.

Using PAC Control:

– Before using Open Outgoing Communication in your strategy, use the Write Number to I/O Unit Memory Map command to write the value of 0 to the address 0xFFFF F031 0400.

Status Codes: 0 = Success.

-10 = Invalid port number.

-20 = Resource busy. May be in use by another user or application. Use PAC Manager 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, and so forth) is in lowercase letters and correctly spelled.

-412 = TCP/IP: Cannot connect error. Make sure the device is on. If multiple calls to this command are made one right after another, delay at least one second between the back-to-back calls.

-417 = Cannot open file. Check filename; verify that the file exists. If you're trying to create a new file, make sure there's enough room on your file system. You'll get this error if it's full.

-437 = No acceptable socket interface found. An Ethernet "accept" or "open" was attempted, but no more sessions are available.

-438 = Could not create socket. For more information, see "List of Common Messages" in the *PAC Control User's Guide* (form 1700).

-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.

-454 = Unable to connect to DNS server. Check DNS and gateway configuration.

See Also: "Close Communication" on page 78
"Communication Open?" on page 79

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 0	Argument 1
Communication Handle	**Put in**
Communication Handle	Float Variable
	Integer 32 Variable

Action Block Example:

Receive Character		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*UNIT_2*
Put in	*Integer 32 Variable*	*CHAR*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Communication Commands" in the *PAC Control User's Guide* (form 1700). 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 work only after 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, and then use a File communication handle to access the file locally.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

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.

-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.

-76 = At end of file.

See Also: "Append Character to String" on page 643
"Get Number of Characters Waiting" on page 82
"Receive N Characters" on page 97
"Send Communication Handle Command" on page 113

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 0	Argument 1	Argument 2	Argument 3
Put in	**Num. Characters**	**Communication Handle**	**Put Status in**
String Variable	Integer 32 Literal	Communication Handle	Float Variable
	Integer 32 Variable		Integer 32 Variable

Action Block Example:

Receive N Character		
Argument Name	Type	Name
Put in	*String Variable*	*RECV_MSG*
Num. Characters	*Integer 32 Variable*	*QTY_CHARS*
Communication Handle	*Communication Handle*	*UNIT_2*
Put Status in	*Integer 32 Variable*	*RECV_STATUS*

OptoScript Example:

ReceiveNChars(*Put in, Num. 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 work only after 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, and then use a File communication handle to access the file locally.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

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, and to determine an appropriate maximum value for n.

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 (if negative value is passed)

-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" on page 95
"Get Number of Characters Waiting" on page 82
"Set End-Of-Message Terminator" on page 127
"Get End-Of-Message Terminator" on page 81
"Transfer N Characters" on page 128

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 0	Argument 1	Argument 2
Length	**Start at Index**	**Of Table**
Integer 32 Literal	Integer 32 Literal	Float Table
Integer 32 Variable	Integer 32 Variable	Integer 32 Table
		Integer 64 Table

Argument 3	Argument 4
Communication Handle	**Put Status in**
Communication Handle	Integer 32 Variable

Action Block Example:

Receive Numeric Table

Argument Name	Type	Name
Length	*Integer 32 Literal*	*64*
Start at Index	*Integer 32 Literal*	*0*
Of Table	*Float Table*	*PEER_DATA_TABLE*
Communication Handle	*Communication Handle*	*UNIT_2*
Put Status in	*Integer 32 Variable*	*RECV_STATUS*

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.

Notes:

- For receiving information using FTP communication handles, this command will work only after 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, and then use a File communication handle to access the file locally.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 107
"Receive String Table" on page 110
"Receive Pointer Table" on page 105
"Transmit Numeric Table" on page 133
"Transmit String Table" on page 139
"Transmit Pointer Table" on page 135

Receive Numeric Table Ex

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.

Details: The arguments are:

- *Length*: The number of elements to read.
- *Start at Index*: The start index in the destination table.
- *Endian Mode*: A Boolean value indicating whether the value is to be read as little-endian (true) or big-endian (false).
- *Bytes per Value*: The number of bytes that should be read for each element, (1, 2, 4, or 8).
- *Of Table*: A numeric table descriptor.
- *Communication Handle*: A communications handle descriptor.
- *Put Status in*: The result of the command. See possible Status Codes listed below.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
Length	**Start at Index**	**Endian mode**	**Bytes per value**
Integer 32 Literal	Integer 32 Literal	Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable	Integer 32 Variable

Argument 4	Argument 5	Argument 6
Of Table	**Communication Handle**	**Put Status in**
Float Table	Communication Handle	Integer 32 Variable
Integer 32 Table		
Integer 64 Table		

Action Block Example:

Receive Numeric Table Ex		
Argument Name	Type	Name
Length	*Integer 32 Literal*	*1*
Start at Index	*Integer 32 Literal*	*8*
Endian mode	*Integer 32 Variable*	*endianMode*
Bytes per value	*Integer 32 Variable*	*bytesPerElement*
Of Table	*Integer 64 Table*	*i64Table*
Communication Handle	*Communication Handle*	*chRcv*
Put Status in	*Integer 32 Variable*	*nStatus*

OptoScript Example:

ReceiveNumTableEx(*Length, Start at Index, Endian mode, Bytes per value, Of Table, Communication Handle***)**

```
nStatus= ReceiveNumTableEx(1, 8, endianMode, bytesPerElement, i64Table,
chRcv);
```

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

Notes:

- For receiving information using FTP communication handles, this command will work only after 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, and then use a File communication handle to access the file locally.

- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Status Codes: 0 = Success.

-12 = Invalid index.

-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 Numeric Variable" on page 103
"Receive Numeric Table Ex" on page 101
"Receive String" on page 107
"Receive String Table" on page 110
"Receive Pointer Table" on page 105
"Transmit Numeric Table" on page 133
"Transmit String Table" on page 139
"Transmit Pointer Table" on page 135

Receive Numeric Variable

Communication Action

Function: Moves a specific value from the device or file specified in the communication handle to an integer 32 variable.

Typical Use: Receive numeric (binary string) data directly into a numeric variable.

Details: The arguments are:

- *Endian mode*: Either true (non-zero), or false (zero); if true, this command receives in little-endian mode, if false, it receives in big-endian mode.
- *Number of Bytes*: Use 4 for 32-bit integer and floats, 8 for 64-bit integers.
- *Put in*: The variable where the bytes should be placed; it *must* be at least the size you specify in Number of Bytes. That is, if you specify 8 bytes, it has to be an Int64.
- *Communication Handle*: A communications handle descriptor.
- *Put Status in*: The result of the command. See possible Status Codes listed below.

Arguments:

Argument 0	Argument 1	Argument 2
Endian mode	**Number of Bytes**	**Put in**
Integer 32 Literal	Integer 32 Literal	Float Variable
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
		Integer 64 Variable

Argument 3	Argument 4
Communication Handle	**Put Status in**
Communication Handle	Integer 32 Variable

Action Block Example:

Receive Numeric Variable

Argument Name	Type	Name
Endian mode	*Integer 32 Literal*	*1*
Number of Bytes	*Integer 32 Literal*	*8*
Put in	*Integer 64 Variable*	*i64Temp*
Communication Handle	*Communication Handle*	*chRcv*
Put Status in	*Integer 32 Variable*	*nStatus*

OptoScript Example:

ReceiveNumVariable (*Endian mode, Number of Bytes, Put in, Communication Handle*)

```
nStatus = ReceiveNumVariable(1, 8, i64Temp, chRcv);
```

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

Notes:

- This command makes it easy to directly receive that data into a numerical variable. It is not restricted to serial data; it can be data coming in from any source using a comm handle.
- For receiving information using FTP communication handles, this command will work only after 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, and then use a File communication handle to access the file locally.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

- Set *Endian mode* (Argument 0) to reflect the order that bytes are sent to the controller from the sending device. This information should be in the sending device's documentation. When *Endian mode* is set to 0 (Big Endian), this command stores the first byte received as the most significant byte. When set to Little Endian (non-zero), it stores the first byte received as the least significant byte. For more information about Big/Little Endian, see the Using Modbus Devices with Opto 22 Products technical note (form 2011).
- You may have to experiment with *Endian mode* (Argument 0) to figure out which byte order is right for a given application. If your input values are way off from what you expected, try the other Endian order.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- Before using this command, use Get Number of Characters Waiting to see if there is a message.

Status Codes:

0 = Success.

-6 = Invalid data field. Returned if Number of Bytes is anything other than 1, 2, 4, or 8 bytes.

-29 = Wrong object type.

-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 Numeric Table" on page 99
"Receive Numeric Table Ex" on page 101
"Receive String" on page 107
"Receive String Table" on page 110
"Receive Pointer Table" on page 105
"Transmit Numeric Table" on page 133
"Transmit String Table" on page 139
"Transmit Pointer Table" on page 135

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 PAC I/O systems), especially when transferring both strings and numbers.

Arguments:

Argument 0	Argument 1	Argument 2
Length	**Start at Index**	**Of Table**
Integer 32 Literal	Integer 32 Literal	Pointer Table
Integer 32 Variable	Integer 32 Variable	

Argument 3	Argument 4
Communication Handle	**Put Status in**
Communication Handle	Integer 32 Variable

Action Block Example:

Receive Pointer Table

Argument Name	Type	Name
Length	*Integer 32 Literal*	*64*
Start at Index	*Integer 32 Literal*	*0*
Of Table	*Pointer Table*	*PEER_DATA_TABLE*
Communication Handle	*Communication Handle*	*UNIT_2*
Put Status in	*Integer 32 Variable*	*RECV_STATUS*

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- 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 Communication Receive Buffer command to make sure the buffer does not fill up.
- For receiving information using FTP communication handles, this command will work only after 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, and then use a File communication handle to access the file locally.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part

of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

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: "Clear Communication Receive Buffer" on page 77
"Receive String" on page 107
"Receive String Table" on page 110
"Receive Numeric Table" on page 99
"Transmit Numeric Table" on page 133
"Transmit String Table" on page 139
"Transmit Pointer Table" on page 135

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 that 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 handle's timeout period for an EOM to arrive. The timeout period for a file communication handle is one second. Most other types of communication handles can have their timeout value adjusted using the "set.TO" option with the Send Communication Handle Command. If no EOM is received within the timeout period, error code -39 is put in the status variable, and as many characters as will fit are copied into the string.
- 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 0	Argument 1	Argument 2
Put in	**Communication Handle**	**Put Status in**
String Variable	Communication Handle	Float Variable
		Integer 32 Variable

Action Block Example:

Receive String

Argument Name	Type	Name
Put in	*String Variable*	*RECV_MSG*
Communication Handle	*Communication Handle*	*UNIT_2*
Put Status in	*Integer 32 Variable*	*RECV_STATUS*

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.
- For more information, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- When using FTP communication handles, you must first use "Send Communication Handle Command" on page 113 to retrieve the information from the FTP server; then, you can use Receive String to access a local copy of the file.

Example

1. Open a communication handle to an FTP server.
2. Use Send Communication Handle Command (with the get option) to retrieve a remote file and store it locally.
3. Close the FTP communications handle.
4. Set the arguments for a new communication handle.
5. Open the new communication handle.

6. Use Receive String to access the local file.

```
nResult = OpenOutgoingCommunication(chFtp);
nResult = SendCommunicationHandleCommand(chFtp,"get:/test.txt,/LocalCopy.txt");
nResult = CloseCommunication(chFtp);
SetCommunicationHandleValue("file:r,LocalCopy.txt", chFile);
nResult = OpenOutgoingCommunication(chFile);
nResult = ReceiveString(sRcv, chFile);
```

Other (non-programmatic) ways to access files are discussed in "Tools for Managing Files," in the *PAC Manager User's Guide* (form 1704).

- FTP communication handle users of this command may find Receive String Table more helpful, especially when more than one file is stored on the FTP server. For an example, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).

  Because data should already be ready to receive following the Send Communication Handle Command (dir option), FTP handles do not wait for the timeout period.
- See notes for "Set End-Of-Message Terminator" on page 127 for issues related to handling special (often invisible) characters like carriage returns.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.
- If the response arrives slowly due to low baud rates or high latency, insert a reasonable Delay (mSec) before invoking Receive String. This will prevent excess consumption of chart timeslices.

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 Get 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.

-443 = Could not receive on socket. Often an indication that the connection was reset by the other device.

See Also:

"Get End-Of-Message Terminator" on page 81

"Open Outgoing Communication" on page 93

"Receive Numeric Table" on page 99

"Receive String Table" on page 110

"Send Communication Handle Command" on page 113

"Set End-Of-Message Terminator" on page 127

"Transmit Numeric Table" on page 133

"Transmit String" on page 137

Receive String Table

Communication Action

Function: Moves a string or specific number of EOM-delimited 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.

Details: All characters up to the first EOM are read or moved to the string table element. The EOM is discarded. If there is no EOM to be received, the control engine waits for the communication handle's timeout period for an EOM to arrive. The timeout period for a file communication handle is one second. Most other types of communication handles can have their timeout value adjusted using the "set.TO" option with the Send Communication Handle Command. If no EOM is received within the timeout period, error code -39 is put in the status variable, and as many characters as will fit are copied into the string table element.

Arguments:

Argument 0	Argument 1	Argument 2
Length	**Start at Index**	**Of Table**
Integer 32 Literal	Integer 32 Literal	String Table
Integer 32 Variable	Integer 32 Variable	

Argument 3	Argument 4
Communication Handle	**Put Status in**
Communication Handle	Integer 32 Variable

Action Block Example:

Receive String

Argument Name	Type	Name
Length	*Integer 32 Literal*	*6*
Start at Index	*Integer 32 Literal*	*0*
Of Table	*String Table*	*PEER_DATA_TABLE*
Communication Handle	*Communication Handle*	*UNIT_2*
Put Status in	*Integer 32 Variable*	*RECV_STATUS*

OptoScript Example:

ReceiveStrTable(*Length, Start at Index, Of Table, Communication Handle***)**

```
RECV_STATUS = ReceiveStrTable(6, 0, PEER_DATA_TABLE, UNIT_2);
```

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

Notes:

- In the example above, assuming the width of String Table PEER_DATA_TABLE is 5, the EOM character has been set to 44 (a comma), and this is the data coming in via Communication Handle UNIT_2: Figs,Watermelon,Plums,Bananas, the resulting PEER_DATA_TABLE will contain:

Figs
Water
melon
Plums
Banan

- See notes for "Set End-Of-Message Terminator" on page 127 for issues related to handling special (often invisible) characters like carriage returns.

- The Length specifies the number of string table elements to be filled. Use -1 to receive all data available.
- If the incoming data does not include any EOM (End-Of-Message) characters, each string element will be filled and the data will wrap to the next element for the number of elements specified by the Length. (Each string table has a certain user-specified width, up to 1024 characters.)
- For data which does include EOM characters, this command's behavior is similar to the Receive String command, where all characters up to the first EOM are read or moved to each string element. The EOMs are discarded.
- The default EOM is ASCII 13, the carriage return character. After the communication handle has been opened, use Set End-Of-Message Terminator to change the EOM character.
- For receiving information using FTP communication handles, this command will work only after 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, and then use a File communication handle to access the file locally. For an example, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).

  Because data should already be ready to receive following the Send Communication Handle Command (dir option), FTP handles do not wait for the timeout period.
- If this command returns an error code (other than -37 or -39), the communication handle will close and need to be re-opened. (-37 and -39 indicate a timeout during the Receive, which could be a normal part of communications.) Be sure to check for status codes returned by this command, and handle any errors appropriately.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- 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 0) is greater than the number of elements in the destination table.

-25 = Port or object is not locked. The device you're receiving data from has closed the connection, so the port cannot be accessed (locked).

-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" on page 107
"Receive Numeric Table" on page 99
"Receive Pointer Table" on page 105
"Send Communication Handle Command" on page 113
"Set End-Of-Message Terminator" on page 127
"Transmit Numeric Table" on page 133
"Transmit String Table" on page 139
"Transmit Pointer Table" on page 135
"Transfer N Characters" on page 128

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 a SNAP PAC controller or brain to change or specify a remote filename when using an FTP communication handle.

You can also use this command to change timeout values for communication handles. (See "Notes:" for default timeout values.)

Details: The following commands are available for the communication handles shown. Before using any of these commands, make sure first to use Open Outgoing Communication to open the handle.

Comm Handle Type	Commands Available	Description
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.
ser	get.to	Retrieves the communication timeout value for transmitting and receiving data, rounded to the nearest integer. A timeout value of 0.5 is rounded up to 1. A value of 0.4 is rounded down to 0 (zero).
	set.to:*seconds*	Sets the communication timeout value for transmitting and receiving data, which you must specify in seconds. For example, use 0.5 to set the timeout to 1/2 second.
udp (See "tcp" below for more)	set.dst	Set the destination address and port for data sent from a stream. For example: `set.dst:192.168.1.2:5300`
	set.ttl	Set "Time To Live" (the maximum number of hops a packet will make between routers) before it is discarded. For example: `set.ttl:15`
	get.ttl	Get the current "Time To Live" value for a stream. For example: `get.ttl`

Comm Handle Type	Commands Available	Description
ftp (See "tcp" below for more)	appe:*local filename, remote filename*	Similar to the send command except that it adds, or appends, the data to the data that already exists in the remote filename.
	cd:*directory to go to*	Changes the current working directory to the specified directory.
	delete	Removes the file named in the *dest:* command (below) 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.
	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 (above).
	dir:*optional directory name, optional table name*	When used with a directory name, retrieves a directory listing for the specified directory name (or the root, if the directory name is 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. When used with a table name, retrieves all the directory listings and inserts them into the provided table. Use this option when the directory has a large number of files. If successful, returns 0 (zero). Note that you do not use the Receive String or Receive String Table command when using this option because the data is stored directly in the table. If the directory name is omitted, a comma (,) must precede the table name; for example: `dir:,mytable`
	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.
	mkdir:*directory name*	Creates (makes) the specified directory. To create multiple layers of directories, depending on the FTP server you're connected to, you may need to create one layer at a time. For example: `mkdir:aaa` followed by a second call: `mkdir:aaa\bbb` and so forth. Depending on the server, you might use a forward slash instead of a backslash.
	rmdir:*directory name*	Deletes (removes) the specified directory. The directory must be empty for this command to work. (Use the *delete* command to delete each file in the directory before using *rmdir* to remove the directory.)
	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 remote filename already exists, the file is overwritten.

Comm Handle Type	Commands Available	Description
ftp (See "tcp" below for more)	appe:*local filename, remote filename*	Similar to the send command except that it adds, or appends, the data to the data that already exists in the remote filename.
	cd:*directory to go to*	Changes the current working directory to the specified directory.
	delete	Removes the file named in the *dest:* command (below) 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.
	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 (above).
	dir:*optional directory name, optional table name*	When used with a directory name, retrieves a directory listing for the specified directory name (or the root, if the directory name is 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. When used with a table name, retrieves all the directory listings and inserts them into the provided table. Use this option when the directory has a large number of files. If successful, returns 0 (zero). Note that you do not use the Receive String or Receive String Table command when using this option because the data is stored directly in the table. If the directory name is omitted, a comma (,) must precede the table name; for example: `dir:,mytable`
	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.
	mkdir:*directory name*	Creates (<u>mak</u>es) the specified directory. To create multiple layers of directories, depending on the FTP server you're connected to, you may need to create one layer at a time. For example: `mkdir:aaa` followed by a second call: `mkdir:aaa\bbb` and so forth. Depending on the server, you might use a forward slash instead of a backslash.
	rmdir:*directory name*	Deletes (<u>rem</u>oves) the specified directory. The directory must be empty for this command to work. (Use the *delete* command to delete each file in the directory before using *rmdir* to remove the directory.)
	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 remote filename already exists, the file is overwritten.

Comm Handle Type	Commands Available	Description
ftp (See "tcp" below for more)	appe:*local filename, remote filename*	Similar to the send command except that it adds, or appends, the data to the data that already exists in the remote filename.
	cd:*directory to go to*	Changes the current working directory to the specified directory.
	delete	Removes the file named in the *dest:* command (below) 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.
	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 (above).
	dir:*optional directory name, optional table name*	When used with a directory name, retrieves a directory listing for the specified directory name (or the root, if the directory name is 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. When used with a table name, retrieves all the directory listings and inserts them into the provided table. Use this option when the directory has a large number of files. If successful, returns 0 (zero). Note that you do not use the Receive String or Receive String Table command when using this option because the data is stored directly in the table. If the directory name is omitted, a comma (,) must precede the table name; for example: `dir:,mytable`
	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.
	mkdir:*directory name*	Creates (makes) the specified directory. To create multiple layers of directories, depending on the FTP server you're connected to, you may need to create one layer at a time. For example: `mkdir:aaa` followed by a second call: `mkdir:aaa\bbb` and so forth. Depending on the server, you might use a forward slash instead of a backslash.
	rmdir:*directory name*	Deletes (removes) the specified directory. The directory must be empty for this command to work. (Use the *delete* command to delete each file in the directory before using *rmdir* to remove the directory.)
	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 remote filename already exists, the file is overwritten.

Comm Handle Type	Commands Available	Description

tcp (Also applies to udp and ftp)	get.src	When accepting incoming connections, it is sometimes necessary to determine the IP address from which the connection originated. To accomplish this you would do the following in OptoScript:
		`Result = SendCommunicationHandleCommand( chComHandle, "get.src" );`
		Result would be set to the address of the peer. For instance, if the connection originated from 192.168.1.12, the value of Result, in hex, would be:
		`0xC0A8010C`
		`(C0 = 192, A8 = 168, 01 = 1, 0C = 12)`

	get.srcport	When accepting incoming connections, it is sometimes necessary to determine the IP port from which the connection originated. To accomplish this you would do the following in OptoScript: `Result = SendCommunicationHandleCommand( chComHandle, "get.srcport" );` Result would be set to the incoming port of the peer.
	get.to	Retrieves the communication timeout value for receiving data, rounded to the nearest integer. A timeout value of 0.5 is rounded up to 1. A value of 0.4 is rounded down to 0 (zero).
	set.to:*seconds*	Sets the communication timeout value for receiving data, which you must specify in seconds. For example, use 0.5 to set the timeout to 1/2 second.
	set.to.open.i:n.n	Sets the time to wait to for an *incoming* connection to complete. To do this, use a Listen for Incoming Communication command, and then an Accept Incoming Communication command to open the connection. Unlike most communication handle commands, send the set.to.open command *before* establishing the connection, since it determines how long the Accept Incoming Communication command should wait before giving up. n.n means the timeout is specified with a decimal number, rather than specifying milliseconds with an integer. If you want 500 mSec, `use: set.to.open.i:0.5`
	set.to.open.o:n.n	Sets the time to wait for an *outgoing* connection to be established. To do this, use an Open Outgoing Communication to open the connection. Unlike most communication handle commands, send the set.to.open command *before* establishing the connection, since it determines how long the Open Outgoing Communication command should wait before giving up. n.n means the timeout is specified with a decimal number, rather than specifying milliseconds with an integer. If you want 500 mSec, use: `set.to.open.o:0.5`

Arguments:

Argument 0	Argument 1	Argument 2
Communication Handle	**Command**	**Put Status in**
Communication Handle	String Literal	Integer 32 Variable
	String Variable	

Action Block Example:

Send Communication Handle Command		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*Log_File*
Command	*String Literal*	*delete*
Put Status in	*Integer 32 Variable*	*Status_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 the *PAC Control User's Guide* (form 1700).
- If this command returns an error code, the communication handle will close and you'll need to re-open it. Be sure to check for status codes returned by this command and handle errors appropriately.
- Default timeout values for communication handles are:
  - TCP: 10 seconds
  - FTP: 30 seconds
  - Serial: 1 second
  - File: 1 second

Status Codes:

0 = Success.

-11 = Could not send data.

-28 = Object not found. When used with ftp dir:*optional table name*, indicates the table name was not found. Verify that the table name exists and that its name is spelled correctly.

-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, and so forth).

-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.

-58 = No data received. If using a file communication handle *find*, make sure file was opened in r (read) mode.

-59 = Could not receive data. Make sure you don't have any extra spaces or tabs in the command.

-74 = Invalid filename. Check length and special characters.

-76 = End of file error. (File communication handle) Didn't find the string you were looking for.

-408 = Error during file access. (File or FTP communication handle) Possible cause: file is in use or read-only, or error accessing directory.

-446 = FTP: Login failed. Check user name, password, and maximum number of logins on server.

-449 = FTP: Error while setting local port number that incoming data connections should use.

-497 = The remote filename used for an ftp *get* doesn't exist.

See Also:

"Open Outgoing Communication" on page 93
"Get Communication Handle Value" on page 80
"Close Communication" on page 78

Send Email

Communication Action

Function: Sends a text-only email via SMTP.

Typical Use: To create a text-only email and send it to recipients via SMTP.

Details:

- This command has many uses including the following:
  - Send an email in case of an alarm condition.
  - Send a daily status report.
  - Have a critical system send hourly status emails; if no email arrives after an hour, it means something is wrong.
- In order for this command to work, you must configure the SNAP PAC controller or *groov* EPIC processor with the correct DNS and Gateway addresses and proper root certificates. For instructions, see "Sending an Email" in the *PAC Control User's Guide* (form 1700).
- You provide information (like email addresses and the message body) through string tables. For each of the string tables, make sure there are enough elements of adequate length for the data you want to enter. See the OptoScript example below.
- It is recommended that you use OptoScript to construct the email. This allows you to see the entire email at the same time. See the OptoScript Example below.
- You can change the transmit, receive, or connect timeout values, as well as indicate whether the control engine should ignore certain SSL errors. For instructions, see "Sending an Email with Attachments Example" in the *PAC Control User's Guide* (form 1700).

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
Server Information	**Recipients**	**Message Body**	**Put Result in**
String Table	String Table	String Table	Integer 32 Variable

Action Block Example:

Send Email

Argument Name	Type	Name
Server Information	*String Table*	*arrstrServer*
Recipients	*String Table*	*arrstrRecipients*
Message Body	*String Table*	*arrstrBody*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

SendEmail(*Server Information, Recipients, Message Body***)**

```
nResult = SendEmail(arrstrServer, arrstrRecipients, arrstrBody);
```

This is a function command; it returns a zero (0) on success or an error code when it fails. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Server Information (Argument 0) must have the entries in the order shown in the following example. (If the table contains more elements, they will be ignored.)

```
arrstrServer[0] = "myaccount@speedmail.com"; // User Account
arrstrServer[1] = "mypassword";              // Password
arrstrServer[2] = "smtp.speedmail.com";      // Server
arrstrServer[3] = "587";                     // Port#
arrstrServer[4] = "tls";                     // "ssl", "tls", "none"
```

Recipients: *Argument 1* List the email address for each recipient.

```
arrstrRecipients[0] = "xyz@mymail.com";      // Recipient list
arrstrRecipients[1] = "abc@yourmail.com";
arrstrRecipients[2] = "";                    //This blank ends the 
recipient list
arrstrRecipients[3] = "johndoe@hismail.com"; //This recipient will be 
ignored
```

Message Body (Argument 2) contains the body of the email. Element zero is always the subject line. The email body starts with element one, and continues to the first empty element or the end of the table, whichever comes first. Each line is entered as a separate string. However, they will be concatenated in the body of the email. Note that a blank line is not the same as an empty element.

```
arrstrBody[0]   = "Email From a Controller";  // Subject line
arrstrBody[1]   = "Hello:"+CRLF+CRLF;         // Body starts here
arrstrBody[2]   = CRLF;                       // Could have been combined 
above
arrstrBody[3]   = "This is an example of an email that can be sent from a 
PAC ";
arrstrBody[4]   = "controller. Even though these lines are separate 
strings, ";
arrstrBody[5]   = "they will be concatenated in the body of the 
email."+CRLF+CRLF;
arrstrBody[6]   = "The pair of CR/LFs above create a blank line for the 
start ";
arrstrBody[7]   = "of a new paragraph.";
arrstrBody[8]   = CRLF+CRLF;
arrstrBody[9]   = "Sincerely,"+CRLF;
arrstrBody[10]  = "SMTP-Controller";
arrstrBody[11]  = "";                         // 1st blank element = body 
end
arrstrBody[12]  = "This is not included in the body.";
```

NOTE: In this example, CRLF is defined at the beginning of the block as a carriage-return / linefeed combination: CRLF = chr(13)+chr(10). Each CRLF starts a new line in the email body. For a double carriage-return / linefeed combination, use +CRLF+CRLF.

Notes:

- If result = 0, the email was sent successfully.
- If result < 0, there was an error executing this command.
- If multiple emails with the similar subject and body text are sent through the same email account, the emails may be flagged as SPAM by the email service provider and not be delivered. To avoid this, you may need to add code to the subject and body text that changes the text dynamically with each new email.
- Make sure all server information table elements are filled in and that the recipients list contains at least one valid entry.

Status Codes:

0 = Success. The email was sent successfully.

-11 = Could not send data. There was an error sending to the mail server.

NOTE: If you have firmware R9.1b, -11 could also indicate that a NACK was received. See -43 below.

-20 = Resource busy. May be in use by another user or another application.

-34 = Invalid I/O command or invalid memory location.

-36 = Invalid command. This error may be returned if you are using PAC Sim. Send Email commands are not supported in PAC Sim.

-38 = Timeout on send.

-39 = Timeout on Receive.

-43 = Received NACK (Negative Acknowledgment). The mail server returned an error (other than a 200- or 300-series response).

-50 = Open connection timeout. Could not establish connection within the timeout period.

-58 = No data received. May have timed out if no response in 10 seconds. Check I/O unit power.

-70 = Not enough data supplied. Check Server Information table elements. Make sure all server information table elements are filled in and that the recipients list contains at least one valid entry.

-443 = Could not receive on socket.

-450 = DNS could not resolve host name to an IP address.

-451 = SMTP login failed.

-454 = Cannot connect to the DNS server. Check the gateway and DNS addresses and configuration.

-2000 = SSL: Bad certificate

-2001 = SSL: Certificate revoked

-2002 = SSL: Certificate expired

-2103 = SSL: Handshake failed due to an unmatched certificate.

-2104 = SSL: Handshake failed due to invalid or unverifiable certificate.

-2105 = SSL: Unspecified asynchronous platform error.

-2106 = SSL: SSL session is not open.

-13019 = An argument in a string is incorrect.

See Also: "Send Email with Attachments" on page 123

Send Email with Attachments

Communication Action

Function: To send an email with text and one or more attachments via SMTP.

Typical Use: This command is similar to Send Email, except that along with the text you can also attach one or more files such as log files, status reports, images, and so on.

Details:

- An example use might be to send a data log file along with the email. The file you want to attach must exist on the controller's file system, (RAM, flash, or SD card). For more information, see the OptoScript example below.
- In order for this command to work, you must configure the SNAP PAC controller or *groov* EPIC processor with the correct DNS and Gateway addresses and proper root certificates. For instructions, see "Sending an Email" in the *PAC Control User's Guide* (form 1700).
- You provide information (like email addresses and the message body) through string tables. For each of the string tables, make sure there are enough elements of adequate length for the data you want to enter. See the OptoScript example below.
- It is recommended that you use OptoScript to construct the email. This allows you to see the entire email at the same time. See the OptoScript Example below.
- You can change the transmit, receive, or connect timeout values, as well as indicate whether the control engine should ignore certain SSL errors. For instructions, see "Sending an Email with Attachments Example" in the *PAC Control User's Guide* (form 1700).

Arguments:

Argument 0	Argument 1	Argument 2
Server Information	**Recipients**	**Message Body**
String Table	String Table	String Table

Argument 3	Argument 4
Attachment File Names	**Put Result in**
String Table	Integer 32 Variable

Action Block Example:

Send Email with Attachments		
Argument Name	Type	Name
Server Information	*String Table*	*arrstrServer*
Recipients	*String Table*	*arrstrRecipients*
Message Body	*String Table*	*arrstrBody*
Attachment File Names	*String Table*	*arrstrAttach*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

`SendEmailWithAttachments` **(***Server Information, Recipients, Message Body, Attachment File Names***)**

```
nResult = SendEmailWithAttachments(arrstrServer, arrstrRecipients,
arrstrBody, arrstrAttach);
```

This is a function command; it returns the number of attachments that were sent successfully. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Server Information (Argument 0) must have the entries in the order shown in the following example. (If the table contains more elements, they will be ignored.)

```
arrstrServer[0] = "myaccount@speedmail.com"; // User Account
```

```
arrstrServer[1] = "mypassword";             // Password
arrstrServer[2] = "smtp.speedmail.com";     // Server
arrstrServer[3] = "587";                    // Port#
arrstrServer[4] = "tls";                    // "ssl", "tls", "none"
```

Recipients: In *Argument 1* List the email address for each recipient.

```
arrstrRecipients[0] = "xyz@mymail.com";     // Recipient list
arrstrRecipients[1] = "abc@yourmail.com";
arrstrRecipients[2] = "";                   //This blank ends the 
recipient list
arrstrRecipients[3] = "johndoe@hismail.com";  //This recipient will be 
ignored
```

Message Body (Argument 2) contains the body of the email. Element zero is always the subject line. The email body starts with element one, and continues to the first empty element or the end of the table, whichever comes first. Each line is entered as a separate string. However, they will be concatenated in the body of the email. Note that a blank line is not the same as an empty element.

```
arrstrBody[0]   = "Opto Email Example";       // Subject line
arrstrBody[1]   = "Hello:"+CRLF+CRLF;         // The email body starts 
here
arrstrBody[3]   = "This is an example of an email that can be sent from a 
PAC ";
arrstrBody[5]   = "controller.”
arrstrBody[6]   = "Sincerely,"+CRLF;
arrstrBody[7]  = "Opto Controller";
arrstrBody[8]  = "";                          // This blank denotes end of 
body
arrstrBody[9]  = "This is not included in the body.";
```

NOTE: In this example, CRLF is defined at the beginning of the block as a carriage-return / linefeed combination: CRLF = chr(13)+chr(10). Each CRLF starts a new line in the email body. For a double carriage-return / linefeed combination, use +CRLF+CRLF.

Attachment File Names (Argument 3) specified in the attachments table must exist on the controller's file system, (RAM, flash, or SD card), and must include the full directory specification. For example, if a file is in the root directory, use just the filename. However, if it is in the temp directory of the SD card, you must put /sdcard0/temp/ in front of the filename. Any kind of file may be attached, so long as there is room in the file system. However, be cautious about file size, many ISPs still have a 10 MB attachment limit.

```
arrstrAttach[0] = "OptoLogo Color.jpg";     // List of files to attach
arrstrAttach[1] = "OptoLogo BW.jpg";
arrstrAttach[2] = "SMTP.zip";
arrstrAttach[3] = "";                       // This blank ends the 
attachment list
arrstrAttach[4] = "Attachment.jpg";         // This attachment will be 
ignored
```

Notes:

- If result > 0, [result] indicates the number of attachments that were sent successfully.
- If result = 0, the text of the email was sent successfully, but no attachments were sent.
- If result < 0, there was an error executing this command.
- If multiple emails with the similar subject and body text are sent through the same email account, the emails may be flagged as SPAM by the email service provider and not be delivered. To avoid this, you may need to add code to the subject and body text that changes the text dynamically with each new email.
- Make sure all server information table elements are filled in and that the recipients list contains at least one valid entry.
- SNAP PAC Simulator should return -36 (unsupported in sim) for SendEmail commands.

Status Codes: If Status Code > 0, it indicates the number of attachments that were sent successfully.

0 = Success sending email with no attachments. This might happen if the attachment table element 0 is empty.

-11 = Could not send data. There was an error sending to the mail server.

NOTE: If you have firmware R9.1b, -11 could also indicate that a NACK was received. See -43 below.

-20 = Resource busy. May be in use by another user or another application.

-34 = Invalid I/O command or invalid memory location.

-36 = Invalid command. This error may be returned if you are using PAC Sim. Send Email commands are not supported in PAC Sim.

-38 = Timeout on send.

-39 = Receive timeout.

-43 = Received NACK (Negative Acknowledgment). The mail server returned an error (other than a 200- or 300-series response).

-50 = Open connection timeout. Could not establish connection within the timeout period.

-70 = Not enough data supplied. Check Server Information table elements. Make sure all server information table elements are filled in and that the recipients list contains at least one valid entry.

-407 = File not found. Make sure any files specified in the attachments table exist on the controller's file system, (RAM, flash, or SD card) including the full directory specification.

-443 = Could not receive on socket.

-450 = DNS could not resolve host name to an IP address.

-451 = SMTP login failed.

-454 = DNS could not connect.

-2000 = SSL: Bad certificate

-2001 = SSL: Certificate revoked

-2002 = SSL: Certificate expired

-2103 = SSL: Handshake failed due to an unmatched certificate.

-2104 = SSL: Handshake failed due to invalid or unverifiable certificate.

-2105 = SSL: Unspecified asynchronous platform error.

-2106 = SSL: SSL session is not open.

-13019 = An argument in a string is bad.

See Also: "Send Email" on page 120

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 arguments for a communication handle before using an Open Outgoing Communication command.

Arguments:

Argument 0	Argument 1
From	**To**
Communication Handle String Literal String Variable	Communication Handle

Action Block Example:

Set Communication Handle Value		
Argument Name	Type	Name
From	*String Literal*	*tcp:10.22.30.40:22005*
To	*Communication Handle*	*COMM_Y*

OptoScript Example:

SetCommunicationHandleValue(*From, To*)

```
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:

- When using a SNAP PAC controller, Set Communication Handle Value can be used for either Ethernet (TCP or UDP) or serial communication (using a serial port on the controller) communication handles. Note: SNAP serial communication modules (SNAP-SCM) use TCP comm handles.
- When using SoftPAC, Set Communication Handle Value cannot be used with a computer's serial ports or USB serial ports. (SoftPAC doesn't support PC serial or USB ports.)
  However, the command can still be used with Ethernet (TCP and UDP) and SNAP-SCM comm handles.
- The example shown above is for outgoing communication using a TCP communication handle. For details about communication handle types and values, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- If you use a string literal in *To* (Argument 0), make sure the communication handle type (for example, `tcp`, `udp`, `ftp`, `file`) is in lowercase letters.
- Do not use this command on a COM handle that is already open. If you do, an "Already Open" (-47) error will be placed in the queue. Before using this command, use Communication Open? to determine whether the handle is already open. If the handle is open, close the handle before changing the value.

Status Codes: -47 = This command must be called while the communication handle is closed; these settings will take effect during the open.

See Also:

"Get Communication Handle Value" on page 80
"Open Outgoing Communication" on page 93
"Communication Open?" on page 79

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 (on the Receive only), Transmit String Table.

Details:

- The EOM character is **not** appended to the transmitted string for the commands Transmit String or Transmit/Receive String. Therefore, before calling those commands, you may first want to call Get End-Of-Message Terminator, and then Append Character to String.
- 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 the *PAC Control User's Guide* (form 1700). 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). If using OptoScript, you may put the character you want to use in single quotes, as shown in the OptoScript example below.

Arguments:

Argument 0	Argument 1
Communication Handle	**To Character**
Communication Handle	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set End-Of-Message Terminator		
Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_A*
To Character	*Integer 32 Variable*	*EOM_Term*

OptoScript Example:

SetEndOfMessageTerminator(*Communication Handle, To Character***)**

```
SetEndOfMessageTerminator(PAC_A, ':');
```

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

Notes:

- While debugging your string-parsing strategy, consider using PAC Control's debugger menu option View > Hex String Display so you will be able to see all your characters, including special formatting characters which may be invisible and are often confusing. For example the CR/LF combination mentioned in the next note will appear in hex as: 0D 0A.
- When you create a text file and finish a line with the Enter key, be aware that most text editors put two invisible characters in the file, ASCII 13 (carriage return or CR) followed by 10 (line feed or LF). When parsing a text file with these CR/LF characters, keep in mind that the end of message terminator is only a single character.
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

Queue Errors: -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.

See Also:

"Get End-Of-Message Terminator" on page 81
"Open Outgoing Communication" on page 93
"Receive String Table" on page 110
"Transmit String Table" on page 139

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 1) and transmits it on the destination handle (Argument 0), 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.
- To transfer as many characters as are available, either enter -1 in *Num Chars* (Argument 2), or use Get Number of Characters Waiting to determine how many bytes of data to transfer, and then enter that number in *Num Chars*.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
Destination Handle	**Source Handle**	**Num. Chars**	**Put Status in**
Communication Handle	Communication Handle	Integer 32 Literal	Float Variable
		Integer 32 Variable	Integer 32 Variable

Action Block Example:

Transfer N Characters

Argument Name	Type	Name
Destination Handle	*Communication Handle*	*PAC_3*
Source Handle	*Communication Handle*	*PAC_4*
Num. Chars	*Integer 32 Variable*	*3000*
Put Status in	*Integer 32 Variable*	*ERROR_CODE*

OptoScript Example:

TransferNChars(*Destination Handle, Source Handle, Num Chars*)

```
ERROR_CODE = TransferNChars(PAC_3, PAC_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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- The two communication handles must be unique.
- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- For receiving information using FTP communication handles, this command will work only after 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, and then use a File communication handle to access the file locally.
- When using this command, it is possible to overrun the receiver, so do not send data faster than it can be received. If necessary, break the transfer into multiple transactions to allow the receiver time to process the incoming data.

- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

Status Codes: 0 = Success

-3 = Buffer overrun or invalid length. The only negative number valid for *Num Chars* (Argument 2) is -1.

-25 = Port not locked. Communication handles in *Destination Handle* (Argument 0) and *Source Handle* (Argument 1) must be different. If trying to transfer characters to a file, the file space may be insufficient.

-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, and 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" on page 93
"Get Number of Characters Waiting" on page 82
"Close Communication" on page 78

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 0	Argument 1	Argument 2
From	**Communication Handle**	**Put Status in**
Float Literal	Communication Handle	Float Variable
Float Variable		Integer 32 Variable
Integer 32 Literal		
Integer 32 Variable		

Action Block Example:

Transmit Character		
Argument Name	Type	Name
From	*Integer 32 Literal*	*10*
Communication Handle	*Communication Handle*	*PAC_4*
Put Status in	*Integer 32 Variable*	*ERROR_CODE*

OptoScript Example:

`TransmitChar` **(***From, Communication Handle***)**

```
ERROR_CODE = TransmitChar(10, PAC_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 *From* (Argument 0). For example, you could use `TransmitChar('a', PAC_4);` rather than having to use `TransmitChar(97, PAC_4);` making the code more readable. Unprintable character codes would still require a number, however.

Notes:

- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- When there are a lot of characters to send, use Transmit String instead.
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

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" on page 137
"Transmit NewLine" on page 132

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 0	**Argument 1**
Communication Handle	**Put Status in**
Communication Handle	Float Variable
	Integer 32 Variable

Action Block Example:

Transmit NewLine

Argument Name	Type	Name
Communication Handle	*Communication Handle*	*PAC_4*
Put Status in	*Integer 32 Variable*	*ERROR_CODE*

OptoScript Example:

`TransmitNewLine(`*Communication Handle***`)`**

```
ERROR_CODE = TransmitNewLine(PAC_4);
```

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

Notes:

- See "Communication Commands" in the *PAC Control User's Guide* (form 1700).
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.
- Characters are transmitted "all at once." Task switching within PAC Control does not interrupt the transmission of the data.

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" on page 137

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 0	Argument 1	Argument 2
Length	**Start at Index**	**Of Table**
Integer 32 Literal	Integer 32 Literal	Float Table
Integer 32 Variable	Integer 32 Variable	Integer 32 Table
		Integer 64 Table

Argument 3	Argument 4
Communication Handle	**Put Status in**
Communication Handle	Float Variable
	Integer 32 Variable

Action Block Example:

Transmit Numeric Table

Argument Name	Type	Name
Length	*Integer 32 Variable*	*Table_length*
Start at Index	*Integer 32 Literal*	*0*
Of Table	*Float Table*	*Peer_data_table*
Communication Handle	*Communication Handle*	*PAC_5*
Put Status in	*Integer 32 Variable*	*Xmit_status*

OptoScript Example:

TransmitNumTable(*Length, Start at Index, Of Table, Communication Handle*)

```
Xmit_status = TransmitNumTable(Table_length, 0, Peer_data_table, PAC_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, and so forth, if desired. These values could be sent as fixed length or carriage return delimited.
- This command does not generate a numeric string equivalent table; it creates a binary table of numeric data.
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 99
"Receive String Table" on page 110
"Receive Pointer Table" on page 105
"Transmit String" on page 137
"Transmit Character" on page 130
"Transmit String Table" on page 139
"Transmit Pointer Table" on page 135
"Transfer N Characters" on page 128

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 0	Argument 1	Argument 2
Length	**Start at Index**	**Of Table**
Integer 32 Literal	Integer 32 Literal	Pointer Table
Integer 32 Variable	Integer 32 Variable	

Argument 3	Argument 4
Communication Handle	**Put Status in**
Communication Handle	Float Variable
	Integer 32 Variable

Action Block Example:

Transmit Pointer Table

Argument Name	Type	Name
Length	*Integer 32 Variable*	*Table_length*
Start at Index	*Integer 32 Literal*	*0*
Of Table	*Pointer Table*	*Peer_data_table*
Communication Handle	*Communication Handle*	*PAC_5*
Put Status in	*Integer 32 Variable*	*Xmit_status*

OptoScript Example:

`TransmitPtrTable(`*Length, Start at Index, Of Table, Communication Handle***`)`**

```
Xmit_status = TransmitPtrTable(Table_length, 0, Peer_data_table, PAC_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, and so forth, 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.
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 99
"Receive String Table" on page 110
"Receive Pointer Table" on page 105
"Transmit String" on page 137
"Transmit Character" on page 130
"Transmit String Table" on page 139
"Transmit Numeric Table" on page 133
"Transfer N Characters" on page 128

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 0	Argument 1	Argument 2
From	**Communication Handle**	**Put Status in**
String Literal	Communication Handle	Float Variable
String Variable		Integer 32 Variable

Action Block Example:

Transmit String

Argument Name	Type	Name
From	*String Variable*	*XMIT_MSG*
Communication Handle	*Communication Handle*	*PAC_5*
Put Status in	*Integer 32 Variable*	*COMM_STATUS*

OptoScript Example:

`TransmitString(`*From, Communication Handle***`)`**

```
COMM_STATUS = TransmitString(XMIT_MSG, PAC_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 the *PAC Control User's Guide* (form 1700).
- Characters are transmitted "one at a time." Task switching within the controller will interrupt the transmission of data. This will cause the transmission to pause momentarily.

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.

Notes:

- 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.

- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

See Also:

"Receive String" on page 107
"Transmit/Receive String" on page 141
"Open Outgoing Communication" on page 93
"Append Character to String" on page 643
"Get End-Of-Message Terminator" on page 81
"Set End-Of-Message Terminator" on page 127

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 0
Length
Integer 32 Literal
Integer 32 Variable

Argument 1
Start at Index
Integer 32 Literal
Integer 32 Variable

Argument 2
Of Table
String Table

Argument 3
Communication Handle
Communication Handle

Argument 4
Put Status in
Float Variable
Integer 32 Variable

Action Block Example:

Transmit String Table		
Argument Name	Type	Name
Length	*Integer 32 Variable*	*Table_length*
Start at Index	*Integer 32 Literal*	*0*
Of Table	*String Table*	*Peer_data_table*
Communication Handle	*Communication Handle*	*PAC_5*
Put Status in	*Integer 32 Variable*	*Xmit_status*

OptoScript Example:

TransmitStrTable(*Length, Start at Index, Of Table, Communication Handle*)

```
Xmit_status = TransmitStrTable(Table_length, 0, Peer_data_table, PAC_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, and so forth, if desired. These values could be sent as fixed length or carriage return delimited.
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

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. For details, see "Communication Commands" in the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = Success

-3 = Invalid length. *Length* (Argument 1) is greater than number of elements in the source table.

-11 = Could not send data. For example, there may not be enough space to write the data.

-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.

-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.

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

See Also:
"Receive String Table" on page 110
"Receive Numeric Table" on page 99
"Receive Pointer Table" on page 105
"Transmit String" on page 137
"Transmit Character" on page 130
"Transmit Pointer Table" on page 135
"Transmit Numeric Table" on page 133
"Set End-Of-Message Terminator" on page 127
"Transfer N Characters" on page 128

Transmit/Receive String

Communication Action

Function: Sends a message, and then waits for an end-of-message delimited response.

Typical Use: Sending and receiving messages and data to/from other devices via TCP/IP.

Details:

- The EOM character is **not** appended to the transmitted string. Therefore, before using this command you may first want to call Get End-Of-Message Terminator, and then Append Character to String.
- 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 0	Argument 1	Argument 2	Argument 3
From	**Communication Handle**	**Put Result in**	**Put Status in**
String Literal	Communication Handle	String Variable	Float Variable
String Variable			Integer 32 Variable

Action Block Example:

Transmit/Receive String		
Argument Name	Type	Name
From	*String Variable*	*XMIT_MSG*
Communication Handle	*Communication Handle*	*PAC_4*
Put Result in	*String Variable*	*RECV_MSG*
Put Status in	*Integer 32 Variable*	*TR_STATUS*

OptoScript Example:

TransmitReceiveString(*From, Communication Handle, Put Result in*)

```
TR_STATUS = TransmitReceiveString(XMIT_MSG, PAC_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 the *PAC Control User's Guide* (form 1700).
- If this command returns an error code, the communication handle will close and need to be re-opened. Be sure to check for status codes returned by this command, and handle any errors appropriately.

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.

-25 = Port not locked. This could be an indication that communication was lost between the Transmit and Receive (for example, reset by the other device following the Transmit).

-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.

-443 = Could not receive on socket. Often an indication that the connection was reset by the other device.

-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" on page 137
"Receive String" on page 107
"Open Outgoing Communication" on page 93
"Get End-Of-Message Terminator" on page 81
"Set End-Of-Message Terminator" on page 127
"Transfer N Characters" on page 128

5: Control Engine Commands

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 0
Put Result in
Integer 32 Variable

Action Block Example:

Calculate Strategy CRC		
Argument Name	Type	Name
Put Result in	*Integer 32 Variable*	*New_CRC_Calc*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: This command could take several minutes to execute when several charts 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" on page 156

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 selected files in flash memory.
- To determine what files are in flash memory and RAM, use PAC Manager. For instructions, see the *PAC Manager User's Guide* (form 1704).

Arguments:

Argument 0
Put Status in
Integer 32 Variable

Action Block Example:

Erase Files in Permanent Storage		
Argument Name	Type	Name
Put Status in	*Integer 32 Variable*	*Flash_Mem_Status*

OptoScript Example:

EraseFilesInPermanentStorage()

```
Flash_Mem_Status = EraseFilesInPermanentStorage()
```

This is a function command; it always returns a zero.

Notes:

- See "Control Engine Commands" in the *PAC Control User's Guide* (form 1700).
- This command always returns a zero.

See Also:

"Save Files To Permanent Storage" on page 157
"Load Files From Permanent Storage" on page 155

Get Available File Space

Control Engine Action

Function: To determine how much file space is currently available in the file system for a system up to 2 GB capacity.

Typical Use: To make sure there is sufficient file space available before writing data to a file.

Details:

- In *File System Type* (Argument 0), show whether file space data for RAM, flash, or microSD should be returned in bytes or megabytes:
  - 0 = RAM filesystem bytes
  - 1 = RAM filesystem megabytes
  - 4 = flash filesystem bytes
  - 5 = flash filesystem megabytes
  - 8 = microSD filesystem bytes
  - 9 = microSD filesystem megabytes
- 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 2.5 on a SNAP-PAC-S controller (varies slightly depending on the control engine firmware version). The file storage space available on a SoftPAC controller depends on the computer the software is installed on.

Arguments:

Argument 0	Argument 1
File System Type	**Put Result in**
Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	

Action Block Example:

Get Available File Space		
Argument Name	Type	Name
File System Type	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Variable*	*File_Space*

OptoScript Example:

GetAvailableFileSpace(*File System Type*)

```
File_Space = GetAvailableFileSpace(0);
```

This is a function command; it returns the size of file space available (a positive value, in units specified by *File System Type*), 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Managing and Accessing Files in the Control Engine's File System" in the Communication Commands section of the *PAC Control User's Guide* (form 1700).
- 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:

"Get Available File Space Ex" on page 146
"Erase Files in Permanent Storage" on page 144
"Load Files From Permanent Storage" on page 155,
"Get Number of Characters Waiting" on page 82

Get Available File Space Ex

Control Engine Action

Function: To determine how much file space is currently available in the file system for systems with more than 2 GB capacity. Supports reading free space values greater than 2 GB, using a 64-bit result value.

Typical Use: To make sure there is sufficient file space available before writing data to a file.

Details:

- For SNAP PAC controllers, requires firmware R10.0h or higher. For SNAP PAC brains, requires firmware R9.5g or higher.
- Cannot be used to read file space on a USB flash drive inserted in a *groov* EPIC processor or *groov* RIO unit.
- In *File System Type* (Argument 0), show whether file space data for RAM, flash, or microSD should be returned in bytes, megabytes, or gigabytes:
  - 0 = RAM filesystem bytes
  - 1 = RAM filesystem megabytes
  - 2 = RAM filesystem gigabytes
  - 4 = flash filesystem bytes
  - 5 = flash filesystem megabytes
  - 6 = flash filesystem gigabytes
  - 8 = microSD filesystem bytes
  - 9 = microSD filesystem megabytes
  - 10 = microSDHC filesystem gigabytes
- 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 2.5 MB on a SNAP-PAC-S controller (varies slightly depending on the control engine firmware version). The file storage space available on a SoftPAC controller depends on the computer the software is installed on.

Arguments:

Argument 0	Argument 1
File System Type	**Put Result in**
Integer 32 Literal	Integer 64 Variable
Integer 32 Variable	

Action Block Example:

Get Available File Space Ex		
Argument Name	Type	Name
File System Type	*Integer 32 Literal*	*0*
Put Result in	*Integer 64 Variable*	*File_Space*

OptoScript Example:

`GetAvailableFileSpaceEx(`*File System Type***`)`**

```
File_Space = GetAvailableFileSpaceEx(0);
```

This is a function command; it returns the size of file space available (a positive value, in units specified by *File System Type*), 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Managing and Accessing Files in the Control Engine's File System" in the Communication Commands section of the *PAC Control User's Guide* (form 1700).

- 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: "Get Available File Space" on page 145
"Erase Files in Permanent Storage" on page 144
"Load Files From Permanent Storage" on page 155,
"Get Number of Characters Waiting" on page 82

Get Control Engine Address

Control Engine Action

Function: Gets the address of the controller on which the strategy is running. The address will be a string in this format: `10.20.30.40`

Typical Use: To identify the IP address of the control engine on which the strategy is running; for example, to display the address in your HMI, or to identify the origin of a message in peer-to-peer communication.

Details:

- To convert the IP address to a 4-byte value, use the String Command, "Convert IP Address String to Integer 32" on page 653.
- If the string passed to this command is too short, it will be truncated to fit. To avoid this situation, be sure to use a string configured for a width of at least 15 characters.
- For PAC-R and PAC-S controllers, the address of ETH1 is returned.
- For SoftPAC, your result may vary depending on your SoftPAC version and computer setup.
  - If your computer has more than one NIC with a valid IP address, the command returns the address of the first DHCP-enabled NIC with a valid address that it finds.
  - If your computer does not have any DHCP-enabled NICs with a valid address, the command returns the first NIC with a valid IP address that it finds.
  - If your computer has no NICs with a valid IP address, the command returns: `0.0.0.0`

Arguments:

Argument 0
Put in
String Variable

Action Block Example:

Get Control Engine Address		
Argument Name	Type	Name
Put in	*String Variable*	*Address_Code*

OptoScript Example:

GetControlEngineAddress(*Put in***)**

```
GetControlEngineAddress(Address_Code);
```

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

See Also:

"Get Control Engine Type" on page 149
"Get Firmware Version" on page 150
"Convert IP Address String to Integer 32" on page 653

Get Control Engine Type

Control Engine Action

Function: Returns a numeric code indicating 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 400 for SNAP PAC Simulator.
- Returns 401 for a SoftPAC controller.
- Returns 512 for all types of SNAP-PAC-S controllers.
- Returns 513 for all types of SNAP-PAC-R controllers.

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Control Engine Type		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*TYPE_CODE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Note: To determine the hardware type of any controller or I/O Unit, use the command "Read Number from I/O Unit Memory Map" on page 343 to read the Unit Type at address 0xF0300020. For more information, see the *OptoMMP Protocol Guide* (form 1465).

See Also:
"Get Control Engine Address" on page 148
"Get Firmware Version" on page 150

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 0
Put in
String Variable

Action Block Example:

Get Firmware Version		
Argument Name	Type	Name
Put in	*String Variable*	*REV_CODE*

OptoScript Example:

GetFirmwareVersion(*Put in*)

```
GetFirmwareVersion(REV_CODE);
```

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

See Also:

"Get Control Engine Address" on page 148
"Get Control Engine Type" on page 149

Get Number of Charts Running

Control Engine Action

Function: Returns the total number of charts that are Running, Starting, or Suspended.

Typical Use: To determine how the approximate number of charts that count toward the maximum number of charts the processor or controller can run simultaneously. You could use this command when optimizing your processes to decide whether you can run additional charts in parallel.

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Number of Charts Running		
Argument Name	Type	Name
Put in	*Float Variable* *Integer 32 Variable*	*Number_Charts_Running*

OptoScript Example:

```
GetNumChartsRunning()
Number_Charts_Running = GetNumChartsRunning();
```

This is a function command; it returns the number of charts running.

Notes: *groov* EPIC processors and SoftPAC computer-based controllers each can simultaneously run up to 64 charts; SNAP PAC S-series controllers, up to 32 charts; R-series controllers, up to 16 charts.

See Also:
"Call Chart" on page 59
"Get Chart Status" on page 69
"Start Chart" on page 71

Get Redundant Controller State

Control Engine Action

Function: Returns a code indicating whether the active controller in a redundant controller system can communicate with all I/O units.

Typical Use: To determine if any I/O units cannot communicate with the controller.

Details: When using redundant controllers (which requires the SNAP PAC Redundancy Option Kit), the active controller periodically tests whether I/O units configured for communication can respond to a "ping" request. If any I/O unit fails to respond to the ping request, then this command sends back a non-zero value (meaning one or more I/O units are "impaired").

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Get Redundant Controller State		
Argument Name	Type	Name
Put in	*Float Variable* *Integer 32 Variable* *Integer 64 Variable*	*State*

OptoScript Example:

GetRedundantControllerState()

```
State = GetRedundantControllerState();
```

This is a function command; it returns a value to indicate whether any I/O unit is unable to communicate with the controller.

Notes:

- Applicable only when using PAC Control Professional, SNAP PAC S-series controllers, and the SNAP PAC Redundancy Option Kit (part number SNAP-PAC-ROK).
- The "Enable communications from control engine" check box does *not* have to be selected for the controller to verify communicates with the I/O units. (Redundant systems check communications to all I/O units, even if communications are not enabled.)

State Codes

- Valid return codes are:
  - 0 = All I/O units are communicating with the active controller.
  - Any number other than 0 (zero) = One or more I/O units is not responding to the controller.

See Also: "Get Redundant Controller Status" on page 153

Get Redundant Controller Status

Control Engine Action

Function: Returns a code indicating whether the active controller is:

- The primary controller
- The secondary controller
- Not operating in redundant mode

Typical Use: To determine which controller is active when using redundant controllers.

Details: Applicable only when using PAC Control Professional, SNAP PAC S-series controllers, and the SNAP PAC Redundancy Option Kit (part number SNAP-PAC-ROK).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Get Redundant Controller Status		
Argument Name	Type	Name
Put in	*Float Variable* *Integer 32 Variable* *Integer 64 Variable*	*Status*

OptoScript Example:

```
GetRedundantControllerStatus()
Status = GetRedundantControllerStatus();
```

This is a function command; it returns a code indicating the controller's status.

Notes: Valid return codes are:

0 = The controller is not operating in redundant mode.
1 = The active controller is the primary controller.
2 = The active controller is the secondary controller.

See Also: "Get Redundant Controller State" on page 152

Get Strategy Name

Control Engine Action

Function: Returns a string containing the name of the strategy currently running.

Typical Use: To make the strategy name available for other uses; for example, to display the strategy name in your HMI.

Details: The returned string is the strategy's file name without the file path or the ".idb" extension.

Arguments:

Argument 0
Put in
String Variable

Action Block Example:

Get Strategy Name		
Argument Name	Type	Name
Put in	*String Variable*	*Strategy_ Name*

OptoScript Example:

GetStrategyName()

```
GetStrategyName(Strategy_Name);
```

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

Load Files From Permanent Storage

Control Engine Action

Function: To copy the files in a SNAP PAC controller's flash memory to the file system in RAM.

Typical Use: To retrieve files previously saved to flash memory.

Details:

- Copies all files in flash memory to the controller's file system in RAM, replacing any files of the same name. (Does not replace other files or folders in the root directory, or files within folders.)
- Does not affect firmware, strategy, point configuration, or other configuration parameters.

Arguments:

Argument 0
Put Status in
Integer 32 Variable

Action Block Example:

Load Files From Permanent Storage		
Argument Name	Type	Name
Put Status in	*Integer 32 Variable*	*STATUS*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- 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 RAM file system are deleted; only the files saved to flash memory are loaded back into RAM when the controller is turned on again.
- To determine which files are in flash memory, which are in RAM, and their file sizes, use PAC Manager. For details, see the *PAC Manager User's Guide* (form 1704).
- For more information, see "Control Engine Commands" in the *PAC Control User's Guide* (form 1700).

Status Codes: 0 = Success

-408 = Error during file access. No files are currently saved in flash memory.

See Also:

"Erase Files in Permanent Storage" on page 144
"Save Files To Permanent Storage" on page 157

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 0
Put in
Integer 32 Variable

Action Block Example:

Retrieve Strategy CRC		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*ORIGINAL_CRC*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also: "Calculate Strategy CRC" on page 143

Save Files To Permanent Storage

Control Engine Action

Function: To save the files that are in the root directory of a SNAP PAC controller's file system to flash memory.

Typical Use: To avoid losing files if the brain or controller is turned off or loses power.

Details:

- Copies all files in the root directory of the file system to flash memory, replacing files of the same name.
- Each file stored in flash memory requires about 72 bytes of overhead.
  - Maximum flash memory file storage in a SNAP PAC R-series controller is about 384 KB.
  - Maximum flash memory file storage in a SNAP-PAC S-series controller is about 2.5 MB.

  For more information about memory usage in SNAP PAC R- and S-series controllers, see the SNAP PAC Memory Usage technical note (form 1646).
- Does not affect firmware, strategy, point configuration, or other configuration parameters.

Arguments:

Argument 0
Put Status in
Integer 32 Variable

Action Block Example:

Save Files To Permanent Storage		
Argument Name	Type	Name
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

SaveFilesToPermanentStorage()

```
STATUS = SaveFilesToPermanentStorage();
```

This is a function command that always returns a zero.

Notes: *CAUTION: If you use this command in a strategy, make certain it is not in a loop. You can wear out the flash if you write to flash too many times.*

- To save a file to flash, put it in the root directory. Files that are inside folders cannot be saved to flash. (Likewise, folders cannot be saved to flash.)
- For more information, see "Control Engine Commands" in the *PAC Control User's Guide* (form 1700).
- 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 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 the *PAC Control User's Guide* (form 1700).
- To determine which files are in flash memory, which are in RAM, and their file sizes, use PAC Manager. For details, see the *PAC Manager User's Guide* (form 1704).

See Also:

"Erase Files in Permanent Storage" on page 144
"Load Files From Permanent Storage" on page 155
"Get Number of Characters Waiting" on page 82

Start Alternate Host Task

Control Engine Action

Function: To start an alternate host task in PAC Control.

Typical Use: To divide up the host workload.

Details: With an alternate host task, human-machine interface (HMI) communication can be directed to the alternate host port, leaving the default host free for use with the debugger.

Arguments:

Argument 0	Argument 1
Connection String	**Put Result in**
String Literal	Integer 32 Variable
String Variable	

Action Block Example:

Start Alternate Host Task		
Argument Name	Type	Name
Connection String	*String Literal*	*tcp:22004*
Put Result in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

```
StartAlternateHostTask()
STATUS = StartAlternateHostTask("tcp:22004");
```

This is a function command. If invalid arguments are passed, or if all user tasks are in use, an error code is returned. Otherwise, a positive number zero is returned.

The Connection String specifies the protocol and port information. Case is important. Recommended connection strings: `tcp:22004` or `tcp:22005`. You can start more than one host task but each host task started reduces the number of charts that can be run at once.

Notes:

- TCP is the only supported protocol.
- Ports 22000 – 22003 are reserved for use by Opto 22. Do not use them in your strategy.
- Ports 22004 – 22010 are designated as user alternate host ports. Opto 22 will not use these ports, and because they are specifically assigned to Opto 22, they should never used by any other application nor should you use them in your strategy.
- *groov* **EPIC processors only.** To use this command, you must first create a new firewall rule to open the alternate host port. (This is because ports in a *groov* EPIC processor are firewall-protected.) For details, see the "Controlling access to the *groov* EPIC processor" chapter of the *groov EPIC User's Guide* (form 2267).
- A positive return value means only that the task was started. Even though successful, this could still fail—for example, if the newly started host task is unable to open the requested port. This could happen if you try to start a host on a port that is already used for something else, such as a user chart, or if you try to use the 22500 series ports, which are used for serial modules. Note that ports 22004 and 22005 are reserved for you to use; you can use other ports, but they may or may not be available.

Status Codes: Any positive number = probable success (see notes above)

-5 = Operation Failed. Possible causes:

- Maximum number of tasks is already running.
- Start command called on a chart which is in the process of starting, but is not yet running. (It takes roughly 100 mSec for the task to be set up and to actually start running once the command has been received).
- Start command encountered while the strategy is in the process of stopping.

-606 = Redundant Mode Bad. This can be caused by trying to start an alternate host on a backup controller, which is allowed only on the active controller.

Queue Errors: -77 = Host task failed. Possible causes: specified host port is already in use, host attempted to load a non-existent strategy file, or there are no more available tasks.

6: Digital Point Commands

Clear All Latches

Digital Point Action

Function: To reset all 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:
- 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 0
On I/O Unit
B100*
B3000 (Digital)*
E1
E2
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
G4D16R*
G4D32RS*
G4EB2
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Clear All Latches		
Argument Name	Type	Name
On I/O Unit	*SNAP-PAC-EB1*	*INPUT_BOARD_1*

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—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-93 = I/O unit not enabled. Previous communication failure may have disabled communication to the unit automatically. Reenable communication to the unit 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" on page 165
"Clear Off-Latch" on page 164

Clear Counter

Digital Point Action

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

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

Details:

- 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 0
On Point
Counter
Quadrature Counter

Action Block Example:

Clear Counter

Argument Name	Type	Name
On Point	*Counter*	*Bottle_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. For HDD counters, use the Get & Clear Counter command.

Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, and SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Get Counter" on page 177
"Get & Clear Counter" on page 168
"Start Continuous Square Wave" on page 199
"Stop Counter" on page 204

Clear Off-Latch

Digital Point Action

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

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

Details:

- Resets the off-latch of a single digital input regardless of the on/off status of the input.
- The next time the input 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 0
On Point
Digital Input

Action Block Example:

Clear Off-Latch		
Argument Name	Type	Name
On Point	*Digital Input*	*BUTTON_1*

OptoScript Example:

ClearOffLatch(*On Point*)

```
ClearOffLatch(BUTTON_1);
```

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

Queue Errors: -52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

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

See Also:

"Get Off-Latch" on page 180
"Clear All Latches" on page 161

Clear On-Latch

Digital Point Action

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

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

Details:

- Resets the on-latch of a single digital input regardless of the on/off status of the input.
- The next time the input 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 0
On Point
Digital Input

Action Block Example:

Clear On-Latch		
Argument Name	Type	Name
On Point	*Digital Input*	*Button_1*

OptoScript Example:

ClearOnLatch(*On Point***)**

```
ClearOnLatch(Button_1);
```

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

Queue Errors: -52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

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

See Also:

"Get On-Latch" on page 184
"Clear All Latches" on page 161

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 (Seconds) (Argument 0) specifies the amount of time in seconds that the channel will remain on during each pulse.

  Off Time (Seconds) (Argument 1) specifies the amount of time the channel will remain off.
- Each SNAP PAC brain enforces a minimum period. The period is the On Time plus the Off Time. If you specify an On Time and Off Time that total less than the minimum period, the times are scaled proportionally to the minimum period.

  The minimum periods for SNAP PAC brains are:

SNAP-PAC-R1	0.006 seconds
SNAP-PAC-R1-B	0.006 seconds
SNAP-PAC-EB1	0.040 seconds
SNAP-PAC-SB1	0.050 seconds
SNAP-PAC-R2	0.1 seconds
SNAP-PAC-EB2	0.1 seconds
SNAP-PAC-SB2	0.1 seconds

  For example, if you specify 0.001 seconds on / 0.002 seconds off for a SNAP-PAC-R1, it will instead produce 0.002 seconds on / 0.004 seconds off, maintaining the On Time's 33% of the total period. In the same situation, a SNAP-PAC-EB1 would produce 0.013 seconds on / 0.027 seconds off.
- The maximum *On Time (Seconds)* and *Off Time (Seconds)* is 429,496.7000 seconds (4.97 days on, 4.97 days off).
- Valid range for *Number of Pulses* (Argument 2) 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 0 **On Time (Seconds)**	Argument 1 **Off Time (Seconds)**	Argument 2 **Number of Pulses**	Argument 3 **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	

Action Block Example:

Generate N Pulses		
Argument Name	Type	Name
On Time (Seconds)	*Float Literal*	*0.250*
Off Time (Seconds)	*Float Literal*	*0.500*
Number of Pulses	*Float Variable*	*Number_of_Pulses*
On Point	*Digital Output*	*DIG_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 are canceled when a Turn Off or Turn On is sent to the output.
- 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.
- Supported on standard 4-channel digital SNAP modules, single channel digital modules on a *mistic* brick, and high-density digital modules.

See Also: "Start Continuous Square Wave" on page 199

Get & Clear Counter

Digital Point Action

Function: To read and clear a digital input counter, quadrature counter, or HDD module counter value.

Typical Use: To count pulses from turbine flow meters, magnetic pickups, encoders, proximity switches, and so forth. To read incremental encoders for positional or velocity measurement.

Details:

- Reads the current value of a digital input counter, quadrature counter, or HDD module counter and places it in *Put In* (Argument 1).
- 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 4,294,967,295 counts. Valid range for a quadrature counter is -2,147,483,647 to 2,147,483,648 counts.
- For a quadrature counter, a positive value indicates forward movement (phase A leads phase B), and a negative value indicates reverse movement (phase B leads phase A).
- 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 0	Argument 1
From Point	**Put In**
Counter	Float Variable
Quadrature Counter	Integer 32 Variable
	Integer 64 Variable

Action Block Example:

Get & Clear Counter		
Argument Name	Type	Name
From Point	*Counter*	*Bottle_Counter*
Put in	*Integer 32 Variable*	*Number_of_Bottles*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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. For details, see the SNAP Quadrature Input Module Data Sheet (form 1053).

Dependencies:

- Always use Start Counter once before using this command for the first time. This does not apply to HDD module counters.
- Applies to high-density digital inputs as well as standard digital inputs configured with the counter or quadrature counter feature.
- Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, and SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Get Counter" on page 177
"Start Continuous Square Wave" on page 199
"Stop Counter" on page 204
"Clear Counter" on page 163

Get & Clear Off-Latch

Digital Point Action

Function: To read and re-arm a high-speed off-latch associated with a digital input.

Typical Use: To ensure detection of an extremely brief on-to-off transition of a digital input.

Details:

- Reads and re-arms the off-latch of a single digital input.
- The next time the input 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 *From Point* (Argument 1) 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 0	**Argument 1**
From Point	**Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Clear Off-Latch		
Argument Name	Type	Name
From Point	*Digital Input*	*BUTTON_3_LATCH*
Put in	*Digital Output*	*ALARM_HORN*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 180
"Clear Off-Latch" on page 164
"Clear All Latches" on page 161

Get & Clear On-Latch

Digital Point Action

Function: To read and re-arm a high-speed on-latch associated with a digital input.

Typical Use: To ensure detection of an extremely brief off-to-on transition of a digital input.

Details:

- Reads and re-arms the on-latch of a single digital input.
- The next time the input 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 stored in *Put In* (Argument 1). If the latch is not set, *Put In* will contain the value 0 (False). If the latch is set, *Put In* will be set to non-zero (True).

Arguments:

Argument 0	Argument 1
From Point	**Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Clear On-Latch		
Argument Name	Type	Name
From Point	*Digital Input*	*E_STOP_BUTTON*
Put in	*Integer 32 Variable*	*LATCH_VAR*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 184
"Clear On-Latch" on page 165
"Clear All Latches" on page 161

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 0	Argument 1
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Restart Off-Pulse Measurement		
Argument Name	Type	Name
From Point	*Off Pulse*	*STANDBY_SWITCH*
Put in	*Float Variable*	*OFF_TIME*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 SNAP PAC R-series controllers, and some SNAP PAC EB-series brains.
- Not supported on high-density digital modules.

See Also:

"Get Off-Pulse Measurement" on page 181
"Get Off-Pulse Measurement Complete Status" on page 182

Get & Restart Off-Time Totalizer

Digital Point Action

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 0	Argument 1
From Point	**Put in**
Off Totalizer	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Restart Off-Time Totalizer		
Argument Name	Type	Name
From Point	*Off Totalizer*	*Power_Status*
Put in	*Integer 32 Variable*	*System_Down_Time*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- An event faster than 10 milliseconds (100 Hz with a 50% duty cycle) may introduce an error into your totalized value.
- Digital totalizer commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:

- Applies only to inputs configured with the totalize-off feature.
- Available on SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also: "Get Off-Time Totalizer" on page 183

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 0	Argument 1
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Restart On-Pulse Measurement		
Argument Name	Type	Name
From Point	*On Pulse*	*Standby_Switch*
Put in	*Float Variable*	*On_Time*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 SNAP PAC R-series controllers, and some SNAP PAC EB-series brains.
- Not supported on high-density digital modules.

See Also:

"Get On-Pulse Measurement" on page 185
"Get On-Pulse Measurement Complete Status" on page 186

Get & Restart On-Time Totalizer

Digital Point Action

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 0	Argument 1
From Point	**Put in**
On Totalizer	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Restart On-Time Totalizer		
Argument Name	Type	Name
From Point	*On Totalizer*	*Circ_Motor_Pwr*
Put in	*Integer 32 Variable*	*Motor_Runtime*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- An event faster than 10 milliseconds (100 Hz with a 50% duty cycle) may introduce an error into your totalized value.
- Digital totalizer commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:
- Applies only to inputs configured with the totalize-on feature.
- Available on SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also: "Get On-Time Totalizer" on page 187

Get & Restart Period

Digital Point Action

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 *Put In* (Argument 1).
- 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 0	Argument 1
From Point	**Put in**
Period	Float Variable
	Integer 32 Variable

Action Block Example:

Get & Restart Period		
Argument Name	Type	Name
From Point	*Period*	*SHAFT_INPUT*
Put in	*Integer 32 Variable*	*SHAFT_CYCLE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Not available on older SNAP Ethernet brains. In order to use this command, upgrade to a SNAP PAC brain.
- This command should be used to start the period measurement.
- This command measures the first complete period only, and then 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.
- Period measurement commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:

- Applies only to inputs configured with the period feature.
- Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Get Period" on page 188
"Get Period Measurement Complete Status" on page 189

Get Counter

Digital & Analog Point Action

Function: To read a standard digital input counter, quadrature counter, or HDD module counter value.

With a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 (a *groov* RIO EMU), read the energy accumulation or summation channels as a 32-bit signed integer value[1].

Typical Use: To count pulses from turbine flow meters, magnetic pickups, encoders, proximity switches, and so forth.

With a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 (a *groov* RIO EMU) module, to read the 64-bit analog input values on the following channels of a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module as 32-bit signed integers:

- 13–17 (Phase A accumulated energy channels)
- 31–35 (Phase B accumulated energy channels)
- 49–53 (Phase C accumulated energy channels)
- 57–62 (Summation channels)

Details:

- Reads the current value of a digital input counter, quadrature counter, or HDD module counter and places it in *Put In* (Argument 1).
- 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 4,294,967,295 counts.
- Valid range for a quadrature counter is -2,147,483,647 to 2,147,483,648 counts.
- For GRV-IVAPM-3 and GRV-R7-I1VAPM-3 modules, the data type you use for argument 1 (Put in) affects how PAC Control interprets the analog input value when it reaches 0x8000 0000 (2,147,483,648 in decimal), which is the upper limit of a 32-bit integer:
  – If you use an integer 32, PAC Control interprets 0x8000 0000 as a signed integer, so it displays that value as -2,147,483,648. As the value continues to increase, PAC Control continues to interpret (and display) the number as negative until the sign bit rolls over from 1 to 0. When that happens, PAC Control interprets (and displays) the number as positive, again.
  – If you use an integer 64, PAC Control sign-extends the sign bit into the upper 32 bits, so it converts 0x8000 0000 to 0xFFFF FFFF 8000 0000, then interprets (and displays) the value as -2,147,483,648. As the value continues to increase, PAC Control continues to interpret (and display) the number as negative. If you want PAC Control to interpret (and display) the value as a positive number, you must use the PAC Control Bit AND command to bitwise AND 0xFFFF FFFF to the integer 64 variable, which will set the upper 32-bits to 0.

    Due to the possible confusion and increase in the margin of error that this additional calculation can introduce, do not select an integer 64 variable for argument 1; instead, use the "Get Counter 64" command.

Arguments:

Argument 0 **From Point**	Argument 1 **Put in**
Counter	Float Variable
Quadrature Counter	Integer 32 Variable
Analog Input[a]	Integer 64 Variable

[a]Only for GRV-IVAPM-3 and GRV-R7-I1VAPM-3 modules.

[1] Requires *groov* EPIC firmware version 3.5.0 or SNAP PAC firmware version 10.5a. The *groov* EPIC firmware will be available Q1 2022; the SNAP PAC firmware will be available Q2 2022.

Action Block Example:

Get Counter		
Argument Name	Type	Name
From Point	*Counter*	*Bottle_Counter*
Put in	*Integer 32 Variable*	*Number_of_Bottles*

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 or,
- the accumulation or summation value for the analog input.

The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- For a quadrature counter, the maximum encoder RPM will be related to the number of pulses per revolution that the encoder provides. For details, see the SNAP Quadrature Input Module Data Sheet (form 1053).

Dependencies:

- Always use Start Counter once before using this command for the first time. This does not apply to HDD counters, GRV-IVAPM-3 modules, or GRV-R7-I1VAPM-3 modules.
- Applies to high-density digital inputs as well as standard digital inputs configured with the counter or quadrature counter feature.
- Applies to the following channels of a GRV-IVAPM-3 or GRV-R7-I1VAPM-3 module:
  - 13–17 (Phase A accumulated energy)
  - 31–35 (Phase B accumulated energy)
  - 49–53 (Phase C accumulated energy)
  - 57–62 (Summation)

See Also:

"Get & Clear Counter" on page 168
"Get Counter 64" on page 35
"Start Continuous Square Wave" on page 199
"Stop Counter" on page 204
"Clear Counter" on page 163

Get Frequency

Digital Point Action

Function: To read digital input frequency value.

Typical Use: To read the speed of rotating machinery, velocity encoders, and so forth.

Details:

- Reads the current frequency unit of a digital input and places it in *Put In* (Argument 1).
- The resolution is 1 Hertz (see notes below).

Arguments:

Argument 0	Argument 1
From Point	**Put in**
Frequency	Float Variable
	Integer 32 Variable

Action Block Example:

Get Frequency		
Argument Name	Type	Name
From Point	*Frequency*	*SHAFT_PICKUP*
Put in	*Integer 32 Variable*	*MOTOR_SPEED*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Not available on older SNAP Ethernet brains. In order to use this command, upgrade to a SNAP PAC brain.
- Since the resolution is 1 Hertz, significant errors may be encountered at frequencies less than 100 Hertz. Use Get Period, then divide 1 by the period to get the frequency with resolution to 0.2 Hertz at 60 Hertz.
- The maximum frequency that can be read is limited by the input module's turn-on and turn-off times. Check the specifications for the module to be used.

Dependencies: Applies only to inputs configured with the frequency feature.

Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, and SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

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:

- 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 *Put In* (Argument 1). *Put In* will contain a non-zero value (True) if the latch is set, and 0 (False) if the latch is not set.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get Off-Latch		
Argument Name	Type	Name
From Point	*Digital Input*	*START_BUTTON*
Put in	*Float Variable*	*RELEASED*

OptoScript Example:

GetOffLatch(*From 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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 170
"Clear Off-Latch" on page 164
"Clear All Latches" on page 161
"Off-Latch Set?" on page 193

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 0	**Argument 1**
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Action Block Example:

Get Off-Pulse Measurement

Argument Name	Type	Name
From Point	*Off Pulse*	*Overheat_Switch*
Put in	*Float Variable*	*OFF_TIME*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 SNAP PAC R-series controllers, and some SNAP PAC EB-series brains.
- Not supported on high-density digital modules.

See Also:

"Get & Restart Off-Pulse Measurement" on page 172
"Get Off-Pulse Measurement Complete Status" on page 182

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 *Put In* (Argument 1). *Put In* will contain a non-zero value (True) if the measurement is complete, or a 0 (False) if it is incomplete.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
Off Pulse	Float Variable
	Integer 32 Variable

Action Block Example:

Get Off-Pulse Measurement Complete Status		
Argument Name	Type	Name
From Point	*Off Pulse*	*Overheat_Switch*
Put in	*Integer 32 Variable*	*Pulse_Complete*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 SNAP PAC R-series controllers, and some SNAP PAC EB-series brains.
- Not supported on high-density digital modules.

See Also:

"Get Off-Pulse Measurement" on page 181
"Get & Restart Off-Pulse Measurement" on page 172

Get Off-Time Totalizer

Digital Point Action

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 0	Argument 1
From Point	**Put in**
Off Totalizer	Float Variable
	Integer 32 Variable

Action Block Example:

Get Off-Time Totalizer

Argument Name	Type	Name
From Point	*Off Totalizer*	*Heater_Output*
Put in	*Float Variable*	*Heater_Down_Time*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- An event faster than 10 milliseconds (100 Hz with a 50% duty cycle) may introduce an error into your totalized value.
- Digital totalizer commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:
- Applies only to inputs configured with the totalize-off feature.
- Available on SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also: "Get & Restart Off-Time Totalizer" on page 173

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:

- 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 *Put In* (Argument 1). *Put In* will contain a non-zero value (True) if the latch is set, and 0 (False) if the latch is not set.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
Digital Input	Digital Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get On-Latch		
Argument Name	Type	Name
From Point	*Digital Input*	*ESTOP_BUTTON*
Put in	*Float Variable*	*EMERGENCY_STOP*

OptoScript Example:

GetOnLatch(*From 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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 171
"Clear On-Latch" on page 165
"Clear All Latches" on page 161
"On-Latch Set?" on page 195

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 0	**Argument 1**
From Point	**Put in**
On Pulse	Float Variable
	Integer 32 Variable

Action Block Example:

Get On-Pulse Measurement		
Argument Name	Type	Name
From Point	*On Pulse*	*Overspeed_Switch*
Put in	*Float Variable*	*On_Time*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 SNAP PAC R-series controllers, and some SNAP PAC EB-series brains.
- Not supported on high-density digital modules.

See Also:

"Get & Restart On-Pulse Measurement" on page 174
"Get On-Pulse Measurement Complete Status" on page 186

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 *Put In* (Argument 1). *Put In* will contain a non-zero value (True) if the measurement is complete, or a 0 (False) if it is incomplete.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
On Pulse	Float Variable
	Integer 32 Variable

Action Block Example:

Get On-Pulse Measurement Complete Status		
Argument Name	Type	Name
From Point	*On Pulse*	*Pressure_Switch*
Put in	*Integer 32 Variable*	*Pulse_Complete*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 SNAP PAC R-series controllers, and some SNAP PAC EB-series brains.
- Not supported on high-density digital modules.

See Also:

"Get & Restart On-Pulse Measurement" on page 174
"Get On-Pulse Measurement" on page 185

Get On-Time Totalizer

Digital Point Action

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 0	**Argument 1**
From Point	**Put in**
On Totalizer	Float Variable
	Integer 32 Variable

Action Block Example:

Get On-Time Totalizer		
Argument Name	Type	Name
From Point	*On Totalizer*	*Pump_Power*
Put in	*Float Variable*	*Pump_Runtime*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- An event faster than 10 milliseconds (100 Hz with a 50% duty cycle) may introduce an error into your totalized value.
- Digital totalizer commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:

- Applies only to inputs configured with the totalize-on feature.
- Available on SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also: "Get & Restart On-Time Totalizer" on page 175

Get Period

Digital Point Action

Function: To read the elapsed time during an on-off-on or an off-on-off transition of a digital input.

Typical Use: To measure the period of a slow shaft rotation.

Details:

- Measurement starts on the first transition (either off-to-on or on-to-off) and stops on the next transition of the same type (one complete cycle).
- Does not restart the period measurement.
- Returns a float representing seconds with a resolution of 100 microseconds.
- Maximum duration is 4.97 days.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
Period	Float Variable
	Integer 32 Variable

Action Block Example:

Get Period		
Argument Name	Type	Name
From Point	*Period*	*SHAFT_INPUT*
Put in	*Float Variable*	*SHAFT_CYCLE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Not available on older SNAP Ethernet brains. In order to use this command, upgrade to a SNAP PAC brain.
- 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.
- Period measurement commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:

- The Get & Restart Period command must be used to start the measurement.
- Applies only to inputs configured with the period feature.
- Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Get & Restart Period" on page 176
"Get Period Measurement Complete Status" on page 189

Get Period Measurement Complete Status

Digital Point Action

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 *Put In* (Argument 1). *Put In* will contain a non-zero value (True) if the measurement is complete, or a 0 (False) if it is incomplete.

Arguments:

Argument 0	**Argument 1**
From Point	**Put in**
Period	Float Variable
	Integer 32 Variable

Action Block Example:

Get Period Measurement Complete Status		
Argument Name	Type	Name
From Point	*Period*	*Pressure_Switch*
Put in	*Integer 32 Variable*	*Period_Complete*

OptoScript Example:

GetPeriodMeasurementCompleteStatus(*From Point***)**

```
Period_Complete = GetPeriodMeasurementCompleteStatus(Pressure_Switch);
```

This is a function command; it returns a value of true (non-zero) 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Not available on older SNAP Ethernet brains. In order to use this command, upgrade to a SNAP PAC brain.
- 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.
- Period measurement commands are supported only on standard 4-channel digital SNAP modules (as well as single channel digital modules on a *mistic* brick).

Dependencies:

- Applies only to inputs configured with the period measurement feature.
- Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Get & Restart Period" on page 176
"Get Period" on page 188

Get TPO Percent

Digital Point Action

Function: Reads the current setting of the time proportional output (TPO) percent.

Typical Use: To find out the value of the TPO percent.

Details:

- The value returned is the percentage of on time. For example, if the TPO period is 2 seconds and this command returns a value of 10, it means the channel was:
  - On for 0.2 seconds (2 seconds x 10%)
  - Off for 1.8 seconds (2 seconds - on time).
- The TPO percent is set by using the Set TPO Percent command.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
TPO	Float Variable

Action Block Example:

Get TPO Percent		
Argument Name	Type	Name
From Point	*Time Proportional Output*	*Heater_Output*
Put in	*Float Variable*	*TPO_Percent*

OptoScript Example:

GetTpoPercent(*From Point***)**

```
TPO_Percent = GetTpoPercent(Heater_Output);
```

This is a function command; it returns the value of the TPO percent.

Notes:

- Supported on:
  - SNAP PAC I/O units using standard 4-channel digital SNAP modules and SNAP high-density digital (HDD) modules.
  - Ultimate I/O units using standard 4-channel digital SNAP modules. (Ultimate I/O does not support HDD modules.)
- This command is not supported on the 2-channel SNAP-AOD-29 or SNAP-AOD-29-HFi modules.

Dependencies: Applies only to output points configured with the TPO feature.

See Also:

"Get TPO Period" on page 191
"Set TPO Percent" on page 196
"Set TPO Period" on page 197

Get TPO Period

Digital Point Action

Function: To read the current setting of the time proportional output (TPO) period.

Typical Use: To find the setting of the TPO period.

Details:

- The TPO period is set by using the Set TPO Period command.
-
- On *groov* digital output modules, the minimum TPO period is module-specific. For more information, see the specifications for your *groov* digital output module.
- This command is not supported on Mistic-protocol devices such as G4 bricks, and B3000 and B3000-B brains.

Arguments:

Argument 0	Argument 1
From Point	**Put in**
TPO	Float Variable

Action Block Example:

Get TPO Period		
Argument Name	Type	Name
From Point	*Time Proportional Output*	*Heater_Output*
Put in	*Float Variable*	*TPO_Period*

OptoScript Example:

GetTpoPeriod(*From Point***)**

```
TPO_Period = GetTpoPeriod(Heater_Output);
```

This is a function command; it returns the value of the TPO period.

Notes:

- Supported on:
  - SNAP PAC I/O units using standard 4-channel digital SNAP modules and SNAP high-density digital (HDD) modules.
  - Ultimate I/O units using standard 4-channel digital SNAP modules. (Ultimate I/O does not support HDD modules.)
- This command is not supported on the 2-channel SNAP-AOD-29 or SNAP-AOD-29-HFi modules.

Dependencies: Applies only to output points configured with the TPO feature.

See Also:

"Get TPO Percent" on page 190
"Set TPO Percent" on page 196
"Set TPO Period" on page 197

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:

- If the specified point is off, the logic will take the True path.
  If the specified point is *not* off, the logic will take the False path.
- *Speed Tip:* Use Get I/O Unit as Binary Value or Get I/O Unit as Binary Value 64 to get the state of all digital points at once. Then use Bit Test to determine the state of individual points.

Arguments:

Argument 0
Is
Digital Input
Digital Output

Condition Block Example:

Off?

Argument Name	Type	Name
Is	*Digital Input*	*Safety_Interlock*

OptoScript Example:

IsOff(*Is*)

```
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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: May be used with either input or output points.

Dependencies: Applies to all digital inputs and outputs.

See Also: "On?" on page 194

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: If the off-latch is set, the logic will take the True path.
If the off-latch is *not* set, the logic will take the False path.

Arguments:

Argument 0
On Point
Digital Input

Condition Block Example:

Off-Latch Set?		
Argument Name	Type	Name
On Point	*Digital Input*	*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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: Use Clear Off-Latch if true to reset the latch for next time.

See Also: "On-Latch Set?" on page 195

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: If the specified point is on, the logic will take the True path.
If the specified point is *not* on, the logic will take the False path.

Arguments:

Argument 0
Is
Digital Input
Digital Output

Condition Block Example:

On?		
Argument Name	Type	Name
On Point	*Digital Input*	*Motor_Power*

OptoScript Example:

IsOn(*Is*)

```
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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- May be used with either input or output points.
- *Speed Tip:* Use Get I/O Unit as Binary Value or Get I/O Unit as Binary Value 64 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?" on page 192

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: If the on-latch is set, the logic will take the True path.
If the on-latch is *not* set, the logic will take the False path.

Arguments:

Argument 0
On Point
Digital Input

Condition Block Example:

On-Latch Set?		
Argument Name	Type	Name
On Point	*Digital Input*	*Clip_Missing_Prox*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: Use Clear On-Latch if true to reset the latch for next time.

See Also: "Off-Latch Set?" on page 193

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.

Arguments:

Argument 0	Argument 1
To (Percent)	**On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Set TPO Percent		
Argument Name	Type	Name
To (Percent)	*Integer 32 Literal*	*20*
On Point	*Time Proportional Output*	*Heater_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 PAC Control. Some older hardware and firmware will not support this feature. For more information, see the *PAC Control User's Guide* (form 1700).
- Setting the value of a digital TPO overrides any prior Turn On or Turn Off command for the digital point.
- Supported on standard 4-channel digital SNAP modules, single channel digital modules on a *mistic* brick, and high-density digital modules.

Dependencies:

- Set TPO Period 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:

"Get TPO Percent" on page 190

"Get TPO Period" on page 191

"Set TPO Period" on page 197

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:

- This command must be used before the Set TPO Percent command.
- Sets the period of a TPO to the specified value. Each *groov* digital output module and SNAP PAC brain enforces a minimum period. The period is the On Time plus the Off Time. If you specify an On Time and Off Time that total less than the minimum period, the times are scaled proportionally to the minimum period.

  On *groov* digital output modules, the minimum TPO period is module-specific. For more information, see the specifications for your *groov* digital output module.

  The minimum periods for SNAP PAC brains are:

SNAP-PAC-R1	0.006 seconds
SNAP-PAC-R1-B	0.006 seconds
SNAP-PAC-EB1	0.040 seconds
SNAP-PAC-SB1	0.050 seconds
SNAP-PAC-R2	0.1 seconds
SNAP-PAC-EB2	0.1 seconds
SNAP-PAC-SB2	0.1 seconds

  For example, if you specify 0.001 seconds on / 0.002 seconds off for a SNAP-PAC-R1, it will instead produce 0.002 seconds on / 0.004 seconds off, maintaining the On Time's 33% of the total period. In the same situation, a SNAP-PAC-EB1 would produce 0.013 seconds on / 0.027 seconds off.

Arguments:

Argument 0	**Argument 1**
To (Seconds)	**On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Set TPO Period		
Argument Name	Type	Name
To (Seconds)	*Float Literal*	*60.0*
On Point	*Time Proportional Output*	*Heater_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 0.25) and off for 22.5 seconds at 30-second intervals.

- Be sure to check the digital output module specifications for the minimum turn-on and turn-off times.
- 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 PAC Control. Some older hardware and firmware will not support this feature. For more information, see the *PAC Control User's Guide* (form 1700).
- 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.
- Supported on standard 4-channel digital SNAP modules, single channel digital modules on a *mistic* brick, and high-density digital modules.

Dependencies: Applies only to output points configured with the TPO feature.

See Also: "Get TPO Percent" on page 190

"Get TPO Period" on page 191

"Set TPO Percent" on page 196

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 (Seconds) (Argument 0) specifies the amount of time in seconds that the point will remain on during each pulse.
  Off Time (Seconds) (Argument 1) specifies the amount of time the point will remain off.
- Each SNAP PAC brain enforces a minimum period. The period is the On Time plus the Off Time. If you specify an On Time and Off Time that total less than the minimum period, the times are scaled proportionally to the minimum period.
  The minimum periods for SNAP PAC brains are:

SNAP-PAC-R1	0.006 seconds
SNAP-PAC-R1-B	0.006 seconds
SNAP-PAC-EB1	0.040 seconds
SNAP-PAC-SB1	0.050 seconds
SNAP-PAC-R2	0.1 seconds
SNAP-PAC-EB2	0.1 seconds
SNAP-PAC-SB2	0.1 seconds

  For example, if you specify 0.001 seconds on / 0.002 seconds off for a SNAP-PAC-R1, it will instead produce 0.002 seconds on / 0.004 seconds off, maintaining the On Time's 33% of the total period. In the same situation, a SNAP-PAC-EB1 would produce 0.013 seconds on / 0.027 seconds off.
- On SNAP PAC brains, the total period must be less than 49.7 days.
- On SNAP PAC brains, if a square wave is already running when this command is used, the new timing becomes effective on the next off-to-on transition.

Arguments:

Argument 0	Argument 1	Argument 2
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	

Action Block Example:

Start Continuous Square Wave

Argument Name	Type	Name
On Time (Seconds)	*Integer 32 Literal*	*0.100*
Off Time (Seconds)	*Integer 32 Literal*	*0.500*
On Point	*Digital Output*	*BLINKING_LAMP*

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 are canceled when a Turn Off or Turn On is sent to the output. To programmatically cancel a pulse train, use this command with both the on times and off times set to 0. Pulse trains will also be canceled if the brain receives a reset command.
- The digital output module's minimum turn-on and turn-off times may be greater. Check the specifications for the module to be used.
- Supported on standard 4-channel digital SNAP modules, single channel digital modules on a *mistic* brick, and high-density digital modules.

Dependencies: Applies only to outputs.

See Also: "Generate N Pulses" on page 166

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. Therefore, this command is not required with HDD counters.
- 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 SNAP-IDC5Q quadrature modules appear with only points 00 and 02 available.

Arguments:

Argument 0
On Point
Counter
Quadrature Counter

Action Block Example:

Start Counter		
Argument Name	Type	Name
On Point	*Counter*	*BAGGAGE_COUNTER*

OptoScript Example:

StartCounter(*On Point*)

```
StartCounter(BAGGAGE_COUNTER);
```

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

Queue Errors: -36 = Invalid command or feature not implemented. (This command applies only to standard digital points. Digital modules with more than four points are automatically configured as counters and do not need this command.)

-93 = I/O Unit not enabled. Previous communication failure may have disabled communication to the unit automatically. Reenable communication to the unit and try again. (The I/O Unit and point must be enabled for this command to work.)

Notes: Use Clear Counter or Get & Clear Counter to clear a counter or quadrature counter to zero.

Dependencies:
- Applies to standard digital inputs configured with the counter or quadrature counter feature. HDD counters do not need this command.
- Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Clear Counter" on page 163
"Get & Clear Counter" on page 168
"Get Counter" on page 177
"Stop Counter" on page 204

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 0	Argument 1
Off Time (Seconds)	**On Point**
Float Literal	Digital Output
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Start Off-Pulse		
Argument Name	Type	Name
Off Time (Seconds)	*Float Literal*	*RESET_TIME*
On Point	*Digital Output*	*PUMP_2_STOP*

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: *CAUTION: If this command is used more frequently than the specified delay, the output will remain off.*

- A Turn On command may be used to abort an off-pulse before the end of the off time.
- Supported on standard 4-channel digital SNAP modules, single channel digital modules on a *mistic* brick, and high-density digital modules.

Dependencies: Applies only to outputs.

See Also:

"Start On-Pulse" on page 203
"Turn Off" on page 205
"Turn On" on page 206

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:

- 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 turns 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 canceled and the new On-pulse is generated.
- The output does not have to be configured with a feature to use this command.

Arguments:

Argument 0	Argument 1
On Time (Seconds)	**On Point**
Float Literal	Digital Output
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Start On-Pulse		
Argument Name	Type	Name
On Time (Seconds)	*Float Variable*	*MIN_LATCH_TIME*
On Point	*Digital Output*	*PUMP_2_RUN*

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:

- *Caution:* If this command is used more frequently than the specified delay, the output will remain on.
- A Turn Off command may be used to abort an on-pulse before the end of the on time.
- Supported on standard 4-channel digital SNAP modules, single channel digital modules on a *mistic* brick, and SNAP high-density digital modules. For details, see Legacy and Current SNAP Product Comparison and Compatibility Charts (form 1693).

See Also:

"Start Off-Pulse" on page 202
"Turn Off" on page 205
"Turn On" on page 206

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 0
On Point
Counter
Quadrature Counter

Action Block Example:

Stop Counter		
Argument Name	Type	Name
On Point	*Counter*	*BEAN_COUNTER*

OptoScript Example:

StopCounter(*On Point*)

```
StopCounter(BEAN_COUNTER);
```

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

Queue Errors: -36 = Invalid command or feature not implemented. (This command applies only to standard digital points. Digital modules with more than four points are automatically configured as counters and do not need this command.)

-93 = I/O Unit not enabled. Previous communication failure may have disabled communication to the unit automatically. Reenable communication to the unit and try again. (The I/O Unit and point must be enabled for this command to work.)

Notes: Use Clear Counter or Get & Clear Counter to set counts to zero.

Dependencies:

- Applies to standard digital inputs configured with the counter or quadrature counter feature.
- Available on GRV-EPIC processors, SNAP-PAC-R1and -R1-B controllers, SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also:

"Get Counter" on page 177
"Get & Clear Counter" on page 168
"Clear Counter" on page 163
"Start Continuous Square Wave" on page 199

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, and so forth.

Details:
- Turns off the specified output.
- The output will remain off until directed otherwise.

Arguments:
Argument 0
[Value]
Digital Output

Action Block Example:

Turn Off		
Argument Name	Type	Name
(none)	*Digital Output*	*The_Lights*

OptoScript Example:

TurnOff **(***Argument 0***)**

```
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 I/O Unit from MOMO Masks to turn off all outputs at once.

Dependencies: If communication to the specific output point or I/O unit is disabled, no action will occur at the output point (XVAL). The IVAL, however, will be updated.

See Also:
"Set I/O Unit from MOMO Masks" on page 329
"Turn On" on page 206

Turn On

Digital Point Action

Function: To turn on a digital output point.

Typical Use: To activate devices connected to digital outputs, such as motors, pumps, lights, and so forth.

Details:

- Turns on the specified output.
- The output will remain on until directed otherwise.

Arguments:

Argument 0
[Value]
Digital Output

Action Block Example:

Turn On		
Argument Name	Type	Name
(none)	*Digital Output*	*INLET_VALVE*

OptoScript Example:

TurnOn (*Argument 0***)**

```
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 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 I/O Unit from MOMO Masks" on page 329
"Turn Off" on page 205

7: Error Handling Commands

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. Message code and error code values are limited to values ranging from -32768 to +32767.
- Quotes (" ") are used in OptoScript code, but not in standard PAC Control code.

Arguments:

Argument 0	Argument 1
Severity	**Message**
Integer 32 Literal	String Literal
Integer 32 Variable	String Variable

Action Block Example:

Add Message to Queue		
Argument Name	Type	Name
Severity	*Integer 32 Literal*	*16*
Message	*String Literal*	*Pressure Tank Exploded*

OptoScript Example:

AddMessageToQueue (*Severity, Message*)

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

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

Notes: If a message code is passed that is not in the range -32768 to +32767, a -8 (Invalid Data) error will be posted to the queue.

Queue Error: -83 = Invalid severity value

See Also: "Add User Error to Queue" on page 208

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. Message code and error code values are limited to values ranging from -32768 to +32767.
- The queue holds a total of 1000 errors and messages.

Arguments:

Argument 0
Error Number
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Add User Error to Queue		
Argument Name	Type	Name
Error Number	*Integer 32 Literal*	*-22001*

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.
- If a message code is passed that is not in the range -32768 to +32767, a -8 (Invalid Data) error will be posted to the queue.

See Also:

"Add Message to Queue" on page 207
"Add User I/O Unit Error to Queue" on page 209
"Get Error Code of Current Error" on page 220

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. Message code and error code values are limited to values ranging from -32768 to +32767.

Arguments:

Argument 0	Argument 1
Error Number	**I/O Unit**
Integer 32 Literal	B100*
Integer 32 Variable	B200*
	B3000 (Analog)*
	B3000 (Digital)*
	G4A8R, G4RAX*
	G4D16R*
	G4D32RS*
	G4EB2
	Generic OptoMMP Device
	GRV-EPIC-PR1
	GRV-EPIC-PR2
	GRV-R7-MM1001-10
	GRV-R7-MM2001-10
	GRV-R7-I1VAPM-3
	SNAP-B3000-ENET, SNAP-ENET-RTC**
	SNAP-BRS*
	SNAP-ENET-D64**
	SNAP-ENET-S64**
	SNAP-PAC-EB1
	SNAP-PAC-EB2
	SNAP-PAC-R1
	SNAP-PAC-R1-B
	SNAP-PAC-R2
	SNAP-PAC-SB1
	SNAP-PAC-SB2
	SNAP-UP1-ADS**
	SNAP-UP1-D64**
	SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands)
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units)

Action Block Example:

Add User I/O Unit Error to Queue		
Argument Name	Type	Name
Error Number	*Integer 32 Literal*	*Error Number*
I/O Unit	*SNAP-PAC-R1*	*I/O Unit*

OptoScript Example:

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

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

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

Notes:

- For a complete list, see the Error Codes appendix in the *PAC Control User's Guide* (form 1700).
- If a message code is passed that is not in the range -32768 to +32767, a -8 (Invalid Data) error will be posted to the queue.

See Also:

"Add User Error to Queue" on page 208
"Get Error Code of Current Error" on page 220

Caused a Chart Error?

Error Handling Condition

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

(The current error is the oldest one and is always at the top of the message queue.)

Typical Use: To determine which chart caused the current error.

Details: If the specified chart caused the current error, the logic will take the True path.
If the specified chart did *not* cause the current error, the logic will take the False path.

Arguments:

Argument 0
I/O Unit
Chart

Condition Block Example:

Caused a Chart Error?		
Argument Name	Type	Name
I/O Unit	*Chart*	*POWERUP*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 220
"Remove Current Error and Point to Next Error" on page 227

Caused an I/O Unit Error?

Error Handling Condition

Function: To determine if the specified I/O unit caused the **current error** in the message queue.

(The current error is the oldest one and is always at the top of the message queue.)

Typical Use: To determine which I/O unit caused an error.

Details:

- If the specified I/O unit caused the current error, the logic will take the True path.
  If the specified I/O unit did *not* cause the current error, the logic will take the False path.
- You must use Error on I/O Unit? before using this command, since this command assumes the current error is an I/O error.

Arguments:

Argument 0
I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Condition Block Example:

Caused an I/O Unit Error?		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-EB1*	*DIG_UNIT_1*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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?" on page 218
"Get Error Code of Current Error" on page 220
"Remove Current Error and Point to Next Error" on page 227

Clear All Errors

Error Handling Action

Function: To clear the message queue.

Typical Use: To clear all errors from a 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.

Action Block Example:

Clear All Errors
No arguments

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 PAC Terminal (by clicking View > Messages > Clear All Messages.

See Also:

"Get Error Code of Current Error" on page 220
"Get Error Count" on page 221
"Remove Current Error and Point to Next Error" on page 227
"Get ID of Block Causing Current Error" on page 222
"Get Name of I/O Unit Causing Current Error" on page 225

Copy Current Error to String

Error Handling Action

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

(The current error is the oldest one and is always at the top of the message queue.)

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 *Delimiter* (Argument 0). 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 PAC Control code.

Arguments:

Argument 0 **Delimiter**	Argument 1 **Put Result in**
Integer 32 Literal	String Variable
Integer 32 Variable	

Action Block Example:

Copy Current Error to String		
Argument Name	Type	Name
Delimiter	*Integer 32 Literal*	*44*
Put Result in	*String Variable*	*strError*

OptoScript Example: **`CurrentErrorToString(`***Delimiter, Put Result in***`)`**

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

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

Note that the integer 32 literal, `44`, could also be entered in OptoScript as a character constant, `','`

See Also:

"Get Error Code of Current Error" on page 220
"Clear All Errors" on page 214
"Get Error Count" on page 221
"Remove Current Error and Point to Next Error" on page 227

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 errors 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.

Action Block Example:

Disable I/O Unit Causing Current Error
No arguments

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: -69 = null object. The current error in the message queue was not caused by an I/O unit. (The current error is the oldest one and is always at the top of the message queue.)

See Also: "Enable I/O Unit Causing Current Error" on page 217
"Error on I/O Unit?" on page 218

Enable I/O Unit Causing Current Error

Error Handling Action

Function: Attempts to bring the I/O Unit back online.

Typical Use: To re-establish communication between the control engine and the I/O unit after it was automatically or manually disabled.

Details: Sends a test message (Powerup Clear command) to the brain. If the test message is successful it will enable communication and configure the I/O if necessary.

Arguments: None.

Action Block Example:

Enable I/O Unit Causing Current Error
No arguments

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: -69 = Invalid parameter (null pointer) passed to command.

See Also: "Disable I/O Unit Causing Current Error" on page 216
"Error on I/O Unit?" on page 218

Error on I/O Unit?

Error Handling Condition

Function: To determine if the current error in the message queue is an I/O-related error.

(The current error is the oldest one and is always at the top of the message queue.)

Typical Use: To determine if further error handling for I/O units should be performed, for example, in an error handling chart.

Details: If the current error in the message queue is an I/O unit error, the logic will take the True path.
If the current error in the message queue is *not* an I/O unit error, the logic will take the False path.

Arguments: None.

Condition Block Example:

Error on I/O Unit?
No arguments

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Error Handling Commands" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 212
"Remove Current Error and Point to Next Error" on page 227
"Error?" on page 219
"Get ID of Block Causing Current Error" on page 222
"Get Line Causing Current Error" on page 223
"Get Name of Chart Causing Current Error" on page 224
"Get Name of I/O Unit Causing Current Error" on page 225

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: If there is an error in the message queue, the logic will take the True path.
If there is *not* an error in the message queue, the logic will take the False path.

Arguments: None.

Condition Block Example:

Error?
No arguments

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Error Handling Commands" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 218

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 *PAC Control User's Guide* (form 1700), for a list of errors that may appear in the message queue.

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Error Code of Current Error		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*ERROR_CODE*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 214
"Get Error Count" on page 221
"Remove Current Error and Point to Next Error" on page 227

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 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Error Count		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*ERROR_COUNT*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 214
"Get Error Code of Current Error" on page 220
"Remove Current Error and Point to Next Error" on page 227

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:
- Blocks are numbered starting with zero.
- If the error queue is empty, the returned value is -1.

Arguments:

Argument 0
Put in
Integer 32 Variable

Action Block Example:

Get Id of Block Causing Current Error		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*Error_Block_ID*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also:
"Get Name of Chart Causing Current Error" on page 224
"Get Name of I/O Unit Causing Current Error" on page 225

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:

- 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.
- If there are not errors in the queue, the command returns a zero.

Arguments:

Argument 0
Put in
Integer 32 Variable

Action Block Example:

Get Line Causing Current Error		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*Error_Block_ID*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also:

"Get ID of Block Causing Current Error" on page 222
"Get Name of Chart Causing Current Error" on page 224
"Get Name of I/O Unit Causing Current Error" on page 225

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: If there are no errors in the queue, the command returns a null.

Arguments:

Argument 0
Put in
String Variable

Action Block Example:

Get Name of Chart Causing Current Error		
Argument Name	Type	Name
Put in	*String Variable*	*CHART_NAME*

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.

See Also:

"Get ID of Block Causing Current Error" on page 222
"Get Line Causing Current Error" on page 223
"Get Name of I/O Unit Causing Current Error" on page 225

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 0
Put in
String Variable

Action Block Example:

Get Name of I/O Unit Causing Current Error		
Argument Name	Type	Name
Put in	*String Variable*	*IO_UNIT_NAME*

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.
- If the top queue error is not an I/O unit error, a null is returned.

Dependencies: The top queue error must be an I/O unit error.

See Also:

"Get Name of Chart Causing Current Error" on page 224
"Get ID of Block Causing Current Error" on page 222
"Get Line Causing Current Error" on page 223

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 0
Put In
Float Variable
Integer 32 Variable

Action Block Example:

Get Severity of Current Error		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*nCurrentError*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 220
"Clear All Errors" on page 214
"Get Error Count" on page 221
"Remove Current Error and Point to Next Error" on page 227

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 PAC Control 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.

Action Block Example:

Remove Current Error and Point to Next Error
No arguments

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?" on page 219
"Get Error Count" on page 221
"Get Error Code of Current Error" on page 220
"Get Name of Chart Causing Current Error" on page 224
"Get Name of I/O Unit Causing Current Error" on page 225

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 PAC Control 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 *PAC Control User's Guide* (form 1700), for a list of errors that may appear in the message queue.

Arguments: None.

Action Block Example:

Stop Chart on Error
No arguments

OptoScript Example:

StopChartOnError()

```
StopChartOnError();
```

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

Notes:

- See "Error Handling Commands" and Chart Commands" in the *PAC Control User's Guide* (form 1700).
- 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.
- Add a 500 ms delay following this command in order to detect the correct chart status with the Get Chart Status command.

See Also:

"Remove Current Error and Point to Next Error" on page 227
"Get Error Count" on page 221
"Suspend Chart on Error" on page 229

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 PAC Control 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 *PAC Control User's Guide* (form 1700), for a list of errors that may appear in the message queue.

Arguments:

Argument 0
Put Status in
Float Variable
Integer 32 Variable

Action Block Example:

Suspend Chart on Error		
Argument Name	Type	Name
Put Status in	*Integer 32 Variable*	*STATUS*

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 the *PAC Control User's Guide* (form 1700).
- 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.
- Add a 500 ms delay following this command in order to detect the correct chart status with the Get Chart Status command.

Status Codes:

0 = success

-5 = failure

See Also:

"Remove Current Error and Point to Next Error" on page 227
"Get Error Count" on page 221
"Stop Chart on Error" on page 228

8: Digital Module

Get & Clear Module Counters

Digital Module Action

Function: To read and reset the counters for all channels on a digital input module.

Typical Use: To read and reset all counters on a module in one command.

Details:

- Places counter data for all channels in the module in a table at a designated starting index, and then clears all counters. *Starting Index* (Argument 3) sets the starting index number, and *Of Table* (Argument 2) indicates the table.
- The table that receives the data must contain at least a number of elements equal to the number of channels on the module, starting from the starting index. (If the table is not large enough, PAC Control returns a -3 error.) PAC Control places data for channel zero in table element *Starting Index*, with other channels following in order. If a channel is not configured with the counter feature, PAC Control writes a zero to the corresponding table element.

Arguments:

Argument 0
I/O Unit
B100*
B3000 (digital)*
E1
G4EB2
G4D16R*
G4D32RS*
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-BRS*
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Of Table
Integer 32 Table
Integer 64 Table
Float Table

Argument 3
Starting Index
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Get & Clear Module Counters

Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
of Table	*Integer 32 Table*	*Meter_Count*
Starting Index	*Integer 32 Variable*	*Index*

For example, if the value of the variable Index is zero, the first four elements of the Meter_Count table might be filled as follows:

Index	Counter Value		
0	61	< < <	Counter data for channel 0
1	85	< < <	Counter data for channel 1
2	102	< < <	Counter data for channel 2
3	42	< < <	Counter data for channel 3

OptoScript Example:

GetClearModuleCounters(*I/O Unit, Module #, of Table, Starting Index***)**

```
GetClearModuleCounters(Bldg_A, 9, Meter_Count, Index);
```

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

Notes:

- To read counters without clearing them, use Get Module Counters.
- Counters with values of more than 2 billion may appear as negative numbers when using an integer 32 table for the counter values.

Queue Errors:

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 communication to the unit. Re-enable communication to the unit and try again.

See Also:

"Get Module Counters" on page 242

Get & Clear Module Off-Latches

Digital Module Action

Function: To read and reset the off-latches of all channels on a digital input module.

Typical Use: To read and clear off-latches on a module in one command.

Details: Places a bitmask in an integer 32 variable showing the state of off-latches for all channels on the module, and resets the latches. The least significant bit in the mask corresponds to channel 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 0
I/O Unit
B100*
B3000 (digital)*
E1
G4EB2
G4D16R*
G4D32RS*
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-BRS*
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Action Block Example:

Get & Clear Module Off-Latches		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OffLatches*

An example of the result is illustrated below. Only the first 8 and last 8 off-latches are shown.

Bit-mask	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
	channel Number	31	30	29	28	27	26	25	24	> > >	7	6	5	4	3	2	1	0

OptoScript Example:

`GetClearModuleOffLatches` **(***I/O Unit, Module #***)**

```
Fan_OffLatches = GetClearModuleOffLatches(Bldg_A, 9);
```

This is a function command; it returns a bitmask representing the status of off latches on the module. The returned value can be consumed by a variable (as shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression.

Notes: To read off-latches without clearing them, use Get Module Off-Latches.

Queue Errors: 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 communication to the unit. Reenable communication to the unit and try again.

See Also: "Get Module Off-Latches" on page 244
"Get & Clear Module Off-Latches" on page 234
"Get Module On-Latches" on page 247

Get & Clear Module On-Latches

Digital Module Action

Function: To read and reset the on-latches of all channels on a digital input module.

Typical Use: To read and reset all on-latches on a module in one command.

Details: Places a bitmask in an integer 32 variable that indicates the state of on-latches for all channels on the module, and resets the latches. The least significant bit corresponds to channel 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 0
I/O Unit
B100*
B3000 (digital)*
E1
G4EB2
G4D16R*
G4D32RS*
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-BRS*
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Action Block Example:

Get & Clear Module On-Latches		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OnLatches*

An example of the result is illustrated below. Only the first 8 and last 8 on-latches are shown.

Bit-mask	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
	channel Num-ber	31	30	29	28	27	26	25	24	> > >	7	6	5	4	3	2	1	0

OptoScript Example:

`GetClearModuleOnLatches(`*I/O Unit, Module #***`)`**

```
Fan_OnLatches = GetClearModuleOnLatches(Bldg_A, 9);
```

This is a function command; it returns a bitmask representing the status of on latches on the module. The returned value can be consumed by a variable (as shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression.

Notes: To read on-latches without clearing them, use Get Module On-Latches.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get Module On-Latches" on page 247
"Get & Clear Module On-Latches" on page 237
"Get Module Off-Latches" on page 244

Get All Module States

Digital Module Action

Function: To read the states of all channels on all digital input or output modules on one I/O unit.

Typical Use: To get the states for all digital channels on the I/O unit with a single command.

Details:

- Places all status data as bitmasks in an integer 32 table at a designated starting index. *Start Index* (Argument 1) sets the index number, and *Of Table* (Argument 2) indicates the table.
- The table that receives the data must contain at least 16 elements including the starting index. (If the table is not large enough, an error -3 is returned.) Data for the module in slot zero is placed in the first specified table element, with other modules following in order. If a slot does not contain a digital module, its corresponding table element is zero-filled.

Arguments:

Argument 0
From I/O Unit
B100*
B3000 (digital)*
E1
G4EB2
G4D16R*
G4D32RS*
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-BRS*
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Starting Index
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Of Table
Integer 32 Table
Integer 64 Table
Float Table

Action Block Example:

Get All Module States		
Argument Name	Type	Name
From I/O Unit	*GRV-EPIC-PR1*	*PR1_A*

Starting Index	*Integer 32 Variable*	*0*
Of Table	*Integer 32 Table*	*Bldg_A_Status*

For example, if the I/O unit PR1_A consists of an 8-module chassis with *groov* EPIC digital modules in slots 1–7, table Bldg_A_Status might be filled as follows:

Index	Value (Bitmask)	
0	00000000000000000000000000000000	<<< (This module is not a digital module.)
1	01100001010001110000001010110010	Each index contains the status data for the *groov* EPIC digital module in the corresponding position on the chassis. A value of 1 indicates that the channel is on; a value of 0 indicates that it is off. The least significant bit in the mask corresponds to channel zero on the module. In this example, index 2, which contains the status of all channels on the module in slot 2, shows that channels 0, 1, 2, 10, 14, and 19 are on. All other channels on the module are off.
2	00000000000010000100010000000111	
3	00100000011000000010010001000100	
4	01100001010001110000001010110010	
5	00001110000100001100100000001001	
6	10000000110000001110000000100100	
7	00110000011100001111100000000001	
8	00000000000000000000000000000000	
↓	↓	The remainder of the table is zero-filled, since there are no more modules.
15	00000000000000000000000000000000	

OptoScript Example:

`GetAllModuleStates(`*From I/O Unit, Starting Index, Of Table***`)`**

```
GetAllModuleStates(PR1_A, 0, Bldg_A_Status);
```

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

Notes:

- To read the channels on only one digital module, use Get Module States.
- You can test or 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.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get Module States" on page 250
"Set Module States from MOMO Masks" on page 253
"Move I/O Unit to Numeric Table Ex" on page 317
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

Get Module Counters

Digital Module Action

Function: To read the counters for all channels on a digital input module.

Typical Use: To get counts without clearing them.

Details:

- Places counter data for all channels in the module in a table at a designated starting index. *Starting Index* (Argument 3) sets the index number, and *Of Table* (Argument 2) indicates the table.
- The table that receives the data must contain at least a number of elements equal to the number of channels on the module, starting from the starting index. (If the table is not large enough, PAC Control returns a -3 error.) PAC Control places data for channel zero in table element *Starting Index*, with other channels following in order. If a channel is not configured with the counter feature, PAC Control writes a zero to the corresponding table element.

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Module #**
B100*	Integer 32 Literal
B3000 (digital)*	Integer 32 Variable
E1	
G4EB2	
G4D16R*	
G4D32RS*	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
SNAP-BRS*	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2 **Of Table**	Argument 3 **Starting Index**
Integer 32 Table	Integer 32 Literal
Integer 64 Table	Integer 32 Variable
Float Table	

Action Block Example:

Get Module Counters		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*Installation_42*
Module #	*Integer 32 Literal*	*10*
Of Table	*Integer 32 Variable*	*Rotations*
Starting Index	*Integer 32 Variable*	*0*

For example, the first four elements of the Rotations table might be filled as follows:

0	25678	< < <	Counter data for channel 0
1	25678	< < <	Counter data for channel 1
2	30946747	< < <	Counter data for channel 2
3	42	< < <	Counter data for channel 3

OptoScript Example:

`GetModuleCounters` **(***I/O Unit, Module #, Of Table, Starting Index***)**

```
GetModuleCounters(Installation_42, 10, Rotations, 0);
```

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

Notes:

- To read and clear all counters on a module, use Get & Clear Module Counters command.
- Counters with values of more than 2 billion may appear as negative numbers.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get & Clear Module Counters" on page 231

Get Module Off-Latches

Digital Module Action

Function: To read the off-latches of all channels on a digital input module.

Typical Use: To read off-latches without clearing latches.

Details: Uses a bitmask to indicate the state of off-latches for all channels on the module. The least significant bit corresponds to channel 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 0
I/O Unit
B100*
B3000 (digital)*
E1
G4EB2
G4D16R*
G4D32RS*
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-BRS*
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Action Block Example:

Get Module Off-Latches		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OffLatches*

An example of the result is illustrated below. Only the first 8 and last 8 off-latches are shown.

Bit-mask	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
	channel Number	31	30	29	28	27	26	25	24	> > >	7	6	5	4	3	2	1	0

OptoScript Example:

`GetModuleOffLatches(` *I/O Unit, Module #* **`)`**

```
Fan_OffLatches = GetModuleOffLatches(Bldg_A, 9);
```

This is a function command; it returns a bitmask representing the status of off latches in the module. The returned value can be consumed by a variable (as shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get Module On-Latches" on page 247
"Get & Clear Module Off-Latches" on page 234

Get Module On-Latches

Digital Module Action

Function: To read the on-latches of all channels on a digital input module.

Typical Use: To read on-latches without clearing latches.

Details: Uses a bitmask to indicate the state of on-latches for all channels on the module. The least significant bit corresponds to channel 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 0
I/O Unit
B100*
B3000 (digital)*
E1
G4EB2
G4D16R*
G4D32RS*
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-BRS*
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Action Block Example:

Get Module On-Latches		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OnLatches*

An example of the result is illustrated below. Only the first 8 and last 8 on-latches are shown.

Bit-mask	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
	channel Num-ber	31	30	29	28	27	26	25	24	> > >	7	6	5	4	3	2	1	0

OptoScript Example:

`GetModuleOnLatches(`*I/O Unit, Module #***`)`**

```
Fan_OnLatches = GetModuleOnLatches(Bldg_A, 9);
```

This is a function command; it returns a bitmask representing the status of on latches in the module. The returned value can be consumed by a variable (as shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get Module Off-Latches" on page 244
"Get & Clear Module On-Latches" on page 237

Get Module States

Digital Module Action

Function: To read the states of all channels on a digital input or output module.

Typical Use: To get information about all channels on one module in one command.

Details: Uses a bitmask to indicate the state of each channel on the module. The least significant bit corresponds to channel 0. A value of 1 in a bit means the channel is on; a value of 0 in the bit means the channel is off.

Arguments:

Argument 0	Argument 1
I/O Unit	**Module #**
B100*	Integer 32 Literal
B3000 (digital)*	Integer 32 Variable
E1	
G4EB2	
G4D16R*	
G4D32RS*	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
SNAP-BRS*	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Action Block Example:

Get Module States		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PR1_A*
Module #	*Integer 32 Literal*	*12*
Put Result in	*Integer 32 Variable*	*Fan_Status*

An example of the result is illustrated below. Only the first 8 and last 8 channels are shown.

Bit-mask	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
	channel Num-ber	31	30	29	28	27	26	25	24	> > >	7	6	5	4	3	2	1	0

OptoScript Example:

`GetModuleStates(`*I/O Unit, Module #***`)`**

```
Fan_Status = GetModuleStates(PR1_A, 12);
```

This is a function command; it returns a bitmask representing the status of each channel on the module. The returned value can be consumed by a variable (as shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression.

Notes: To read the channels on all modules on one I/O unit, use Get All Module States.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get All Module States" on page 240
"Set Module States from MOMO Masks" on page 253

Set Module States from MOMO Masks

Digital Module Action

Function: To control multiple channels on the same digital output module simultaneously with a single command.

Typical Use: To efficiently control multiple digital outputs on one module with one command.

Details:

- To turn on a channel, set the respective bit in the 32-bit data field of *Must On Mask* (Argument 1) to a value of 1.
- To turn off a channel, set the respective bit in the 32-bit data field of *Must Off Mask* (Argument 0) to a value of 1.
- To leave a channel unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*.
- Check for conflicts; if the same bit is set to 1 in both masks, the channel is turned off.
- The least significant bit corresponds to channel zero.

Arguments:

Argument 0 **Must Off Mask**	Argument 1 **Must On Mask**
Integer 32 Literal Integer 32 Variable	Integer 32 Literal Integer 32 Variable

Argument 2 **I/O Unit**	Argument 3 **Module #**
B100* B3000 (digital)* E1 G4EB2 G4D16R* G4D32RS* GRV-EPIC-PR1 GRV-EPIC-PR2 GRV-R7-MM1001-10 GRV-R7-MM2001-10 SNAP-BRS* SNAP-B3000-ENET, SNAP-ENET-RTC** SNAP-ENET-D64** SNAP-ENET-S64** SNAP-PAC-EB1 SNAP-PAC-EB2 SNAP-PAC-R1 SNAP-PAC-R1-B SNAP-PAC-R2 SNAP-PAC-SB1 SNAP-PAC-SB2 SNAP-UP1-ADS** SNAP-UP1-D64** SNAP-UP1-M64**	Integer 32 Literal Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Set Module States from MOMO Masks		
Argument Name	Type	Name
Must Off Mask	*Integer 32 Literal*	*0xB0000020*
Must On Mask	*Integer 32 Literal*	*0x060000C2*
I/O Unit	*GRV-EPIC-PR1*	*Bldg_A*
Module #	*Integer 32 Literal*	*3*

The effect of this command is illustrated below:

	channel Number	31	30	29	28	27	26	25	24	> > >	7	6	5	4	3	2	1	0
Must-on Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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 channels on the chassis. For the channels shown, channels 26, 25, 7, 6, and 1 will be turned on. Channels 31, 29, 28, and 5 will be turned off. Other channels shown are not changed.

OptoScript Example:

`SetModuleStatesFromMomoMasks` **(***Must Off Mask, Must On Mask, I/O Unit, Module #***)**

```
SetModuleStatesFromMomoMasks(0xB0000020, 0x060000C2, Bldg_A, 3);
```

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

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

IVAL Get All Module States

Digital Module Action

Function: To Read the internal values (IVALs) of all channels on all digital input or output modules on one I/O unit.

Typical Use: Simulation, testing, and certification where communication to the I/O unit is disabled.

Details:

- Places all status data (read from the IVAL) as bitmasks in an integer 32 table at a designated starting index.
  Starting Index (Argument 1) sets the index number, and *Of Table* (Argument 2) 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 digital module, its corresponding table element is zero-filled.
- For more information about IVAL, see the PAC Control User's Guide (form 1700).

Arguments:

Argument 0 **From I/O Unit**	Argument 1 **Starting Index**
B100*	Integer 32 Literal
B3000 (digital)*	Integer 32 Variable
E1	
G4EB2	
G4D16R*	
G4D32RS*	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
SNAP-BRS*	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2 **Of Table**
Integer 32 Table
Float Table

Action Block Example:

IVAL Get All Module States

Argument Name	Type	Name
From I/O Unit	*GRV-EPIC-PR1*	*PR1_A*
Starting Index	*Integer 32 Variable*	*0*
Of Table	*Integer 32 Table*	*Bldg_A_Status*

For example, if the I/O unit PR1_A consists of an 8-module chassis with *groov* EPIC digital modules in slots 1–7, table Bldg_A_Status might be filled as follows:

Index	Value (Bitmask)		
0	00000000000000000000000000000000	< < <	(This module is not a *groov* EPIC digital module.)
1	01100001010001110000001010110010		Each index contains the status data for the *groov* EPIC digital module in the corresponding position on the chassis. A value of 1 indicates that the channel is on; a value of 0 indicates that it is off. The least significant bit in the mask corresponds to channel zero on the module. In this example, index 2, which contains the status of all channels on the module in slot 2, shows that channels 0, 1, 2, 10, 14, and 19 are on. All other channels on the module are off.
2	00000000000010000100010000000111		
3	00100000011000000010010001000100		
4	01100001010001110000001010110010		
5	00001110000100001100100000001001		
6	10000000110000011100000000100100		
7	00110000011100001111100000000001		
8	00000000000000000000000000000000		
↓	↓		The remainder of the table is zero-filled, since there are no more modules.
15	00000000000000000000000000000000		

OptoScript Example:

`IvalGetAllModuleStates(`*From I/O Unit, Starting Index, Of Table***`)`**

```
IvalGetAllModuleStates(PR1_A, 0, Bldg_A_Status);
```

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

Notes:

- To read the channels on only one module, use Get Module States.
- You can test or 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.

Queue Errors:

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 communication to the unit. Reenable communication to the unit and try again.

See Also:

"Get Module States" on page 250
"Move I/O Unit to Numeric Table Ex" on page 317
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

9: High-Density Digital Module

Clear HDD Module Off-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Clear Mask
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Clear HDD Module Off-Latches		
Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Module #	*Integer 32 Variable*	*6*
Clear Mask	*Integer 32 Literal*	*0x060000C2*
Put Status in	*Integer 32 Variable*	*OffLatch_Status*

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
Bitmask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Clear HDD Module On-Latches" on page 261
"Get & Clear HDD Module Off-Latches" on page 271
"Get & Clear All HDD Module Off-Latches" on page 263

Clear HDD Module On-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Clear Mask
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Clear HDD Module On-Latches

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

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
Bitmask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Clear HDD Module Off-Latches" on page 259
"Get & Clear HDD Module On-Latches" on page 273
"Get & Clear All HDD Module On-Latches" on page 265

Get & Clear All HDD Module Off-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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. *Start Index* (Argument 1) sets the index number, and *Put Result in* (Argument 2) 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Start Index
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Table

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get & Clear All HDD Module Off-Latches		
Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Start Index	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Table*	*Bldg_A_OffL*
Put Status in	*Integer 32 Variable*	*Status_Code*

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.)
1	01100001010001110000001010110010		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.
2	00000000000010000100010000000111		
3	00100000011000000010010001000100		
4	01100001010001110000001010110010		
5	00001110000100001100100000001001		
6	10000000110000001110000000100100		
7	00110000011100001111100000000001		
8	00000000000000000000000000000000		
↓	↓		The remainder of the table is zero-filled, since there are no more modules.
15	00000000000000000000000000000000		

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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get & Clear HDD Module Off-Latches" on page 271
"Get All HDD Module Off-Latches" on page 275
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

Get & Clear All HDD Module On-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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. *Start Index* (Argument 1) sets the index number, and *Put Result in* (Argument 2) 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Start Index
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Table

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get & Clear All HDD Module On-Latches

Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Start Index	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Table*	*Bldg_A_OnLatches*
Put Status in	*Integer 32 Variable*	*Status_Code*

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	< < <	(This module is not a HDD module.)
1	01100001010001110000001010110010		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.
2	00000000000010000100010000000111		
3	00100000011000000010010001000100		
4	01100001010001110000001010110010		
5	00001110000100001100100000001001		
6	10000000110000011100000000100100		
7	00110000011100001111100000000001		
8	00000000000000000000000000000000		
↓	↓		The remainder of the table is zero-filled, since there are no more modules.
15	00000000000000000000000000000000		

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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get & Clear HDD Module On-Latches" on page 273
"Get All HDD Module On-Latches" on page 277
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

Get & Clear HDD Module Counter

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0 **I/O Unit**	Argument 1 **Module #**
GRV-R7-MM1001-10	Integer 32 Literal
GRV-R7-MM2001-10	Integer 32 Variable
SNAP-B3000-ENET, SNAP-ENET-RTC*	
SNAP-ENET-S64*	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS*	
SNAP-UP1-M64*	

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2 **Point #**	Argument 3 **Put Result in**	Argument 4 **Put Status in**
Integer 32 Literal	Integer 32 Variable	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Get & Clear HDD Module Counter

Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Ins_42*
Module #	*Integer 32 Literal*	*8*
Point #	*Integer 32 Variable*	*Meter*
Put Result in	*Integer 32 Variable*	*Meter_8_Counts*
Put Status in	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

GetClearHddModuleCounter(*I/O Unit, Module #, 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.
- For more information, see “Legacy High-Density Digital Module Commands” in the *PAC Control User’s Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User’s Guide (form 1547).

- 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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also: "Get & Clear HDD Module Counters" on page 269
"Get HDD Module Counters" on page 281

Get & Clear HDD Module Counters

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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. *Start Table Index* (Argument 2) sets the index number, and *Put Result in* (Argument 3) indicates the table.
- The table that receives the data must contain at least 32 elements including 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Start Table Index
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Result in
Integer 32 Table

Argument 4
Put Status in
Integer 32 Variable

Action Block Example:

Get & Clear HDD Module Counters		
Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*In_42*
Module #	*Integer 32 Variable*	*Section*
Start Table Index	*Integer 32 Variable*	*Index*
Put Result in	*Integer 32 Table*	*Meter_Ct*
Put Status in	*Integer 32 Variable*	*Status_Code*

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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).
- 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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get & Clear HDD Module Counter" on page 267
"Get HDD Module Counters" on page 281

Get & Clear HDD Module Off-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get & Clear HDD Module Off-Latches

Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*9*
Module #	*Integer 32 Literal*	*Fan_OffLatches*
Put Result in	*Integer 32 Variable*	*Status_Code*
Put Status in	*Integer 32 Variable*	*9*

An example of the result is illustrated below. Only the first 8 and last 8 off-latches are shown.

Bit-mask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module Off-Latches" on page 283
"Get & Clear All HDD Module Off-Latches" on page 263
"Get All HDD Module On-Latches" on page 277

Get & Clear HDD Module On-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get & Clear HDD Module On-Latches

Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OnLatches*
Put Status in	*Integer 32 Variable*	*Status_Code*

An example of the result is illustrated below. Only the first 8 and last 8 on-latches are shown.

Bit-mask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module On-Latches" on page 285
"Get & Clear All HDD Module On-Latches" on page 265
"Get All HDD Module Off-Latches" on page 275

Get All HDD Module Off-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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. *Start Index* (Argument 1) sets the index number, and *Put Result in* (Argument 2) 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Start Index
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Table

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get All HDD Module Off-Latches		
Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Start Index	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Table*	*Bldg_A_OffLatches*
Put Status in	*Integer 32 Variable*	*Status_Code*

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	< < < (This module is not a HDD module.)
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:

`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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module Off-Latches" on page 283
"Get & Clear All HDD Module Off-Latches" on page 263
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

Get All HDD Module On-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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. *Start Index* (Argument 1) sets the index number, and *Put Result in* (Argument 2) 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 0 **I/O Unit**	Argument 1 **Start Index**
GRV-R7-MM1001-10	Integer 32 Literal
GRV-R7-MM2001-10	Integer 32 Variable
SNAP-B3000-ENET, SNAP-ENET-RTC*	
SNAP-ENET-S64*	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS*	
SNAP-UP1-M64*	

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2 **Put Result in**	Argument 3 **Put Status in**
Integer 32 Table	Integer 32 Variable

Action Block Example:

Get All HDD Module On-Latches

Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Start Index	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Table*	*Bldg_A_OnLatches*
Put Status in	*Integer 32 Variable*	*Status_Code*

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	<<< (This module is not a HDD module.)
1	01100001010001110000001010110010	
2	00000000000010000100010000000111	
3	00100000011000000010010001000100	
4	01100001010001110000001010110010	
5	00001110000100001100100000001001	
6	10000000110000011100000000100100	
7	00110000011100001111100000000001	
8	00000000000000000000000000000000	
↓	↓	The remainder of the table is zero-filled, since there are no more modules.
15	00000000000000000000000000000000	

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.

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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module On-Latches" on page 285
"Get & Clear All HDD Module On-Latches" on page 265
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

Get All HDD Module States

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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.
  Start Index (Argument 1) sets the index number, and *Put Result in* (Argument 2) 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Start Index
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Table

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get All HDD Module States		
Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Start Index	*Integer 32 Variable*	*0*
Put Result in	*Integer 32 Table*	*Bldg_A_Status*
Put Status in	*Integer 32 Variable*	*Status_Code*

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	< < < (This module is not a HDD module.)
1	01100001010001110000001010110010	
2	00000000000010000100010000000111	
3	00100000011000000010010001000100	
4	01100001010001110000001010110010	
5	00001110000100001100100000001001	
6	10000000110000011100000000100100	
7	00110000011100001111100000000001	
8	00000000000000000000000000000000	
↓	↓	
15	00000000000000000000000000000000	

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.

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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module States" on page 287
"Set HDD Module from MOMO Masks" on page 289
"Move I/O Unit to Numeric Table Ex" on page 317
"Numeric Table Element Bit Test" on page 464
"Bit Test" on page 423 and other Bit commands

Get HDD Module Counters

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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. *Start Index* (Argument 1) sets the index number, and *Put Result in* (Argument 2) indicates the table.
- The table that receives the data must contain at least 32 elements including 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Start Table Index
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Result in
Integer 32 Table

Argument 4
Put Status in
Integer 32 Variable

Action Block Example:

Get HDD Module Counters		
Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Installation_42*
Module #	*Integer 32 Literal*	*10*
Start Table Index	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Variable*	*Rotations*
Put Status in	*Integer 32 Variable*	*Status_Code*

For example, the first four elements of the Rotations table might be filled as follows:

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 #, 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).
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).
- 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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get & Clear HDD Module Counter" on page 267
"Get & Clear HDD Module Counters" on page 269

Get HDD Module Off-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get HDD Module Off-Latches		
Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OffLatches*
Put Status in	*Integer 32 Variable*	*Status_Code*

An example of the result is illustrated below. Only the first 8 and last 8 off-latches are shown.

Bit-mask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module On-Latches" on page 285
"Get All HDD Module Off-Latches" on page 275
"Get & Clear HDD Module Off-Latches" on page 271
"Get & Clear All HDD Module Off-Latches" on page 263

Get HDD Module On-Latches

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get HDD Module On-Latches

Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Bldg_A*
Module #	*Integer 32 Literal*	*9*
Put Result in	*Integer 32 Variable*	*Fan_OnLatches*
Put Status in	*Integer 32 Variable*	*Status_Code*

An example of the result is illustrated below. Only the first 8 and last 8 on-latches are shown.

Bit-mask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get HDD Module Off-Latches" on page 283
"Get All HDD Module On-Latches" on page 277
"Get & Clear HDD Module On-Latches" on page 273
"Get & Clear All HDD Module On-Latches" on page 265

Get HDD Module States

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get HDD Module States		
Argument Name	Type	Name
I/O Unit	*SNAP-UP1-ADS*	*UIO_A*
Module #	*Integer 32 Literal*	*12*
Put Result in	*Integer 32 Variable*	*Fan_Status*
Put Status in	*Integer 32 Variable*	*Status_Code*

An example of the result is illustrated below. Only the first 8 and last 8 points are shown.

Bit-mask	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 #, 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.
- For more information, see "Legacy High-Density Digital Module Commands" in the *PAC Control User's Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User's Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get All HDD Module States" on page 279
"Set HDD Module from MOMO Masks" on page 289
"Turn On HDD Module Point" on page 293
"Turn Off HDD Module Point" on page 291

Set HDD Module from MOMO Masks

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 *Must On Mask* (Argument 2) to a value of 1.
- To turn off a point, set the respective bit in the 32-bit data field of *Must Off Mask* (Argument 3) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Must On Mask
Integer 32 Literal
Integer 32 Variable

Argument 3
Must Off Mask
Integer 32 Literal
Integer 32 Variable

Argument 4
Put Status in
Integer 32 Variable

Action Block Example:

Set HDD Module from MOMO Masks

Argument Name	Type	Name
I/O Unit	*SNAP-UP1-M64*	*Bldg_A*
Module #	*Integer 32 Literal*	*3*
Must On Mask	*Integer 32 Variable*	*0x060000C2*
Must Off Mask	*Integer 32 Literal*	*0xB0000020*
Put Status in	*SNAP-UP1-ADS*	*Status_Code*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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 #, 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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Turn On HDD Module Point" on page 293
"Turn Off HDD Module Point" on page 291

Turn Off HDD Module Point

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Point #
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Turn Off HDD Module Point

Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Installation_42*
Module #	*Integer 32 Literal*	*8*
Point #	*Integer 32 Variable*	*Meter*
Put Status in	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

`TurnOffHddModulePoint` (*I/O Unit, Module #, Point #*)

```
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.
- For more information, see “Legacy High-Density Digital Module Commands” in the *PAC Control User’s Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User’s Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also: "Turn On HDD Module Point" on page 293
"Set HDD Module from MOMO Masks" on page 289

Turn On HDD Module Point

High-Density Digital Module Action

NOTE: This is a high-density digital (HDD) command. To enable HDD commands, from the PAC Control menu bar, click File > Strategy Options > Legacy tab > Original High Density Digital commands.

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 0
I/O Unit
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-M64*

Argument 1
Module #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Point #
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Turn On HDD Module Point

Argument Name	Type	Name
I/O Unit	*SNAP-ENET-S64*	*Installation_42*
Module #	*Integer 32 Literal*	*8*
Point #	*Integer 32 Variable*	*Meter*
Put Status in	*Integer 32 Variable*	*Status_Code*

OptoScript Example:

TurnOnHddModulePoint(*I/O Unit, Module #, Point #***)**

```
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.
- For more information, see “Legacy High-Density Digital Module Commands” in the *PAC Control User’s Guide, Legacy Edition* (form 1710), and the SNAP High-Density Digital Module User’s Guide (form 1547).

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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also: "Turn Off HDD Module Point" on page 291
"Set HDD Module from MOMO Masks" on page 289

10: I/O Quality Commands

Get I/O Channel Quality

I/O Quality

Function: Returns a code indicating the data quality on a specified channel of a *groov* I/O module.

Typical Use: To determine if an exception condition occurred on the I/O channel.

Details: *NOTE: Applicable only to groov I/O in PAC Control R10.0a and higher only.*

A quality code of 0 (zero) indicates the channel isn't currently reporting any data quality issues. For details, see Quality Codes.

groov I/O modules that report quality continuously poll their channels for current data values and identify deviations between reported and expected values. For example, when a channel on a GRV-IV-24 reports a value greater than 10% above range, the module sets a quality indicator code 15 (Input is more than 10% above the maximum range) on the channel. When the issue is resolved, the quality indicator code returns to 0 (zero).

groov I/O Modules:groov RIO Modules:

Arguments:

Argument 0 **From Channel**	Argument 1 **Put in**
Analog Input	Float Variable
Analog Output	Integer 32 Variable
Digital Input	Integer 64 Variable
Digital Output	

Action Block Example:

Get I/O Channel Quality		
Argument Name	Type	Name
From Channel	*Analog Input*	*Alarm*
Put in	*Integer 32 Variable*	*Data_Quality*

OptoScript Example:

GetChannelQuality(*From Channel***)**

```
Data_Quality = GetChannelQuality(Alarm);
```

This is a function command; it returns one of the quality codes listed below.

Notes:

- This command works only with *groov* I/O modules; the "From Channel" must be on a *groov* I/O module.
- The command is available in PAC Control R10.0a and higher.

- Because the channels are constantly being polled, you may see a different code depending on the state of the channel when the command is sent. For example, if a channel reports a current reading that's above its positive range limit, the I/O module sets a quality code 21 on the channel. Later, if a wire on the same channel breaks, the current would be extremely small, and the module would change the quality code to 18 (Output fault). But when the field signal comes back into range, the quality will return to Good (0).

Quality Codes: A brief description of a channel's quality indicator is visible when you are scanning the channel in PAC Control's Debug mode. When you use a PAC Control I/O Quality command, a code is returned instead of the description.

Queue Errors:

-39 = Timeout on receive.

-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 automatically disabled communication to the unit. Reenable communication to the unit and try again. (The I/O unit and channel must be enabled for this command to work.)

See Also: "Get I/O Unit Quality" on page 297

Get I/O Unit Quality

I/O Quality

Function: Returns a numerical value representing a bitmask that indicates which *groov* I/O modules on the specified I/O unit have reported a non-zero quality code. For more about quality codes, see "Get I/O Channel Quality" on page 295.

Typical Use: To determine if any *groov* I/O module is reporting a data quality issue.

Details: *Note: This command is available in PAC Control R10.0a and higher.*

When the returned value is viewed as a binary number, each bit indicates a position in the chassis. Module positions on the chassis are numbered from left to right (0 - 15) and correspond to bits 0 - 15 in the returned value. A 0 (zero) bit means the position has no quality issues; a 1 (one) bit indicates a module in that position has at least one channel reporting a non-zero quality code.

Arguments:

Argument 0	Argument 1
From I/O Unit	**Put in**
GRV-EPIC-PR1	Float Variable
GRV-R7-I1VAPM-3	Integer 32 Variable
	Integer 64 Variable

Action Block Example:

Get I/O Unit Quality		
Argument Name	Type	Name
From I/O Unit	*GRV-EPIC-PR1*	*Local_IO*
Put in	*Integer 32 Variable*	*Quality_Mask*

OptoScript Example:

GetIoUnitQuality(*From I/O Unit*)

```
Quality_Mask = GetIOUnitQuality(Local_IO);
```

This is a function command; it returns a decimal value representing a bitmask.

Notes:

- This command works only with *groov* I/O modules; the "From I/O Unit" must be a *groov* I/O module.
- The command is available in PAC Control R10.0a and higher.
- If the returned value is 0, it means no modules are reporting a quality code.
- Any returned value greater than zero should be assessed to determine which modules are reporting quality codes. Then use Get I/O Channel Quality to find out which channel is reporting data outside of the expected values.

Queue Errors:

-39 = Timeout on receive.

-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 automatically disabled communication to the unit. Reenable communication to the unit and try again. (The I/O unit must be enabled for this command to work.)

See Also: "Get I/O Channel Quality" on page 295

11: I/O Unit Commands

Clear I/O Unit Configured Flag

I/O Unit Action

Function: Clears the flag that the controller uses to indicate that the I/O unit has been initialized by the controller.

Typical Use: Provides a workaround for the issue of Ethernet brains only supporting a single Power Up Clear (PUC).

Details:

- Issue this command immediately before enabling communication to an I/O unit so that the controller will configure the I/O unit and points.
- Using this command followed by Enable Communication to I/O Unit forces all configuration commands to be sent to the I/O unit, regardless of whether they have been previously sent.
- Use this command to force the controller to send all configuration commands to an I/O unit when you enable it. This can be useful if you are concerned that an I/O unit's PUC might have been appropriated by an HMI or other application.

Arguments:

Argument 0
On I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3

(Argument 0 continued)
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Clear I/O Unit Configured Flag		
Argument Name	Type	Name
On I/O Unit	*SNAP-PAC-EB2*	*FURNACE_PID*

OptoScript Example:

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

```
ClearIOUnitConfiguredFlag(FURNACE_PID);
```

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

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:

- For standard digital points only. For modules with more than four points, see:
  - "Move I/O Unit to Numeric Table Ex" on page 317
  - "Get All HDD Module States" on page 279
  - "Get I/O Unit Module States" on page 305.
- 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.
- SNAP-PAC-R1s and -R1-Bs return 32 bits, even though they can move up to 64 digital points. For an I/O unit with greater than 32 digital points, use "Get I/O Unit as Binary Value 64" on page 303 instead. (Note that *groov* EPIC processors return 64 bits.)
- Do not move a unit with 64 standard digital points into an integer 32 variable. Use an integer 64 variable instead.
- 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 0 **From**	Argument 1 **Put in**
B100*	Integer 32 Variable
B3000 (Digital)*	Integer 64 Variable
E1	
G4D16R*	
G4D32RS*	
G4EB2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-BRS*	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Get I/O Unit as Binary Value		
Argument Name	Type	Name
From	*SNAP-PAC-EB1*	*INPUT_BOARD_2*
Put in	*Integer 64 Variable*	*IN_BD2_STATUS*

The effect of this command is illustrated below. (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.)

	Point Number	63	62	61	60	59	58	57	56	> > >	7	6	5	4	3	2	1	0
Bitmask	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(`*From*`)`

```
IN_BD2_STATUS = GetIoUnitAsBinaryValue(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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Use Bit Test to examine individual bits.
- To understand this command, think of it as a Move command for I/O units. Get I/O Unit as Binary Value moves an Int 32 value into an Int 32 or Int 64 variable, and all the associated rules still apply.

  Here's a simple way to get the lower 32-bits using OptoScript:

  ```
  nn1 = GetIoUnitAsBinaryValue(paceb1); // nn1 is Int64
  n1  = GetLowBitsOfInt64(nn1);         //  n1 is Int32
  ```

See Also:

"Set I/O Unit from MOMO Masks" on page 329
"Get I/O Unit as Binary Value 64" on page 303
"Get Low Bits of Integer 64" on page 435
"Get High Bits of Integer 64" on page 434

Get I/O Unit as Binary Value 64

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:

- For standard digital points only. For more than four points, see:
  - "Move I/O Unit to Numeric Table Ex" on page 317
  - "Get All HDD Module States" on page 279
  - "Get I/O Unit Module States" on page 305.
- 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 (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.
- If you only need the upper or lower 32 bits, use "Get Low Bits of Integer 64" on page 435 or "Get High Bits of Integer 64" on page 434.

Arguments:

Argument 0 **From**	Argument 1 **Put in**
B100*	Integer 64 Variable
B3000 (Digital)*	
E1	
G4D16R*	
G4D32RS*	
G4EB2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-BRS*	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Get I/O Unit as Binary Value 64		
Argument Name	Type	Name
From	*SNAP-PAC-EB1*	*INPUT_BOARD_2*
Put in	*Integer 64 Variable*	*IN_BD2_STATUS*

The effect of this command is illustrated below. (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.)

	Point Number	63	62	61	60	59	58	57	56	> > >	7	6	5	4	3	2	1	0
Bitmask	Binary	0	1	1	0	1	1	0	0	> > >	0	1	0	0	0	0	1	0
	Hex	6				C				> > >	4				2			

OptoScript Example:

GetIoUnitAsBinaryValue64(*From*)

```
IN_BD2_STATUS = GetIoUnitAsBinaryValue64(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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: Use Bit Test to examine individual bits.

See Also:

"Set I/O Unit from MOMO Masks" on page 329
"Get I/O Unit as Binary Value" on page 301
"Get Low Bits of Integer 64" on page 435
"Get High Bits of Integer 64" on page 434

Get I/O Unit Module States

I/O Unit Action

Function: To read current on/off status of all digital points on an I/O unit and place them in the elements of a numeric table.

Typical Use: To efficiently read all digital points 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.
-
- Module zero corresponds to the first specified table element. The command sends point status to the table beginning at the index specified in *Starting Index* (Argument 1). If there are more modules than table elements from the specified index to the end of the table, no data is written to the table, and a -12 error is placed in the controller's message queue.
- least significant bit corresponding to point zero.GRV-IAC-24 and a 12-channel GRV-ODCI-12, the first two elements in the table might show Points that are on are indicated by a 1. Points that are off are indicated by a zero. In this example, channel 0 on module 0 is on; channel 0 on module 1 is off.
- Analog modules, serial modules, and modules that are not configured 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) are read.
- This command reads from the I/O unit. It updates the IVALs in the controller.

Arguments:

Argument 0 **From I/O Unit**	Argument 1 **Starting Index**	Argument 2 **Of Table**
B100*	Integer 32 Literal	Integer 32 Table
B3000 (digital)*	Integer 32 Variable	Integer 64 Table
E1		Float Table
G4EB2		
G4D16R*		
G4D32RS*		
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
SNAP-B3000-ENET, SNAP-ENET-RTC**		
SNAP-BRS*		
SNAP-ENET-D64**		
SNAP-ENET-S64**		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Get I/O Unit Module States		
Argument Name	Type	Name
From I/O Unit	*GRV-EPIC-PR1*	*My_IO_Unit*
Starting Index	*Integer 32 Literal*	*0*
Of Table	*Integer 32 Table*	*DATA_TABLE*

OptoScript Example:

GetIoUnitModuleStates **(***From I/O Unit, Starting Index, Of Table***)**

```
GetIoUnitModuleStates(My_IO_Unit, 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.

-69 = Invalid parameter (null pointer) passed to command.

See Also:

"Get All HDD Module States" on page 279
"Move I/O Unit to Numeric Table" on page 315
"Move Numeric Table to I/O Unit Ex" on page 322

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 PAC Control 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 a target address is enabled, then it is available to be used. However, only one address can be used at a given time, so there can only be one active address. The active address is the address the controller is currently using. One address is always active. If communication to the active address fails and the control engine is not able to switch to the other address, then communication to the I/O unit will become disabled.
- 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 enabled 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:
  1=Primary Address is active
  2=Secondary Address is active

Arguments:

Argument 0	**Argument 1**	**Argument 1**
Enable Mask	**Active Mask**	**I/O Unit**
Integer 32 Variable	Integer 32 Variable	Generic OptoMMP Device GRV-EPIC-PR1 GRV-EPIC-PR2 GRV-R7-MM1001-10 GRV-R7-MM2001-10 GRV-R7-I1VAPM-3 SNAP-B3000-ENET, SNAP-ENET-RTC* SNAP-ENET-D64* SNAP-ENET-S64* SNAP-PAC-EB1 SNAP-PAC-EB2 SNAP-PAC-R1 SNAP-PAC-R1-B SNAP-PAC-R2 SNAP-PAC-SB1 SNAP-PAC-SB2 SNAP-UP1-ADS* SNAP-UP1-D64* SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Get Target Address State		
Argument Name	Type	Name
Enable Mask	*Integer 32 Variable*	*ENABLE_MASK*
Active Mask	*Integer 32 Variable*	*ACTIVE_MASK*
I/O Unit	*GRV-EPIC-PR1*	*UNIT*

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 PAC Display clients and OptoOPCServers. These commands only deal with the control engine communicating with I/O units. PAC Display and OptoOPCServer have their own mechanism for controlling their use of the network.
- This command does not test communication.

See Also:

"Set All Target Address States" on page 325
"Set Target Address State" on page 331

I/O Unit Ready?

I/O Unit Condition

Function: Tests communication with the specified I/O unit.

Typical Use: To determine if the controller can physically communicate with the I/O unit.

Details:

- I/O Unit Ready? will test communication to the I/O unit regardless of whether it is enabled or not.
- When constructing a chart to verify I/O unit communications, it is good practice to use I/O Unit Ready? first. Then after getting a True response, use *"Enable Communication to I/O Unit" on page 617* to make sure communication to the I/O unit is enabled. Also see the appendix on troubleshooting in the *PAC Control User's Guide* (form 1700).
- In order for I/O Unit Ready? to be True, the I/O Unit must respond to the test message (identify type) with the correct I/O Unit type, or with a "PUC Expected" error.

Arguments:

Argument 0
Is
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Condition Block Example:

I/O Unit Ready?		
Argument Name	Type	Name

Is	*SNAP-PAC-EB1*	*PUMP_HOUSE*

OptoScript Example:

IsIoUnitReady(*Is*)

```
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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: Ideal for determining "System Ready" status.

See Also:

"I/O Point Communication Enabled?" on page 620
"I/O Unit Communication Enabled?" on page 621

IVAL Move Numeric Table to I/O Unit

I/O Unit Action

NOTE: Use this command for I/O units with modules that have only four points. For modules with more than four points, use "IVAL Move Numeric Table to I/O Unit Ex" on page 313 instead.

Function: Writes to the internal value (IVAL) of all 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 0	Argument 1	Argument 2
Start at Index	**Of Table**	**Move to**
Integer 32 Literal	Float Table	B100*
Integer 32 Variable	Integer 32 Table	B200*
	Pointer Table	B3000 (Analog)*
		B3000 (Digital)*
		E1
		E2
		G4A8R, G4RAX*
		G4D16R*
		G4D32RS*
		G4EB2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC**
		SNAP-BRS*
		SNAP-ENET-D64**
		SNAP-ENET-S64**
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS**
		SNAP-UP1-D64**
		SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

IVAL Move Numeric Table to I/O Unit		
Argument Name	Type	Name
Start at Index	*Integer 32 Variable*	*4*
Of Table	*Integer 32 Table*	*IO_STATUS_TABLE*
Move to	*SNAP-PAC-R1*	*VALVE_CONTROL*

OptoScript Example:

IvalMoveNumTabletoIoUnit(*Start at Index, Of Table, Move to***)**

```
IvalMoveNumTabletoIoUnit(4, IO_STATUS_TABLE, VALVE_CONTROL);
```

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

Notes: Primarily used to write to inputs.

Queue Errors: -69 = Invalid parameter (null pointer) passed to command. Received if a null table object pointer is passed.

See Also: "IVAL Set Analog Point" on page 626
"Disable Communication to All I/O Units" on page 610,
"Disable Communication to I/O Unit" on page 611

IVAL Move Numeric Table to I/O Unit Ex

I/O Unit Action

Function: Writes to the internal value (IVAL) of all 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 output IVALs to be modified as if they were being changed by real I/O.
- Please see "Table Index Offsets" on page 753 for the length of the table required for values of points-per-module.
- The range of *Points per Module* (Argument 3) is 1 to 32.

Arguments:

Argument 0
From Table
Float Table
Integer 32 Table
Pointer Table

Argument 1
With Starting Index
Integer 32 Literal
Integer 32 Variable

Argument 2
To I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 3
Points per Module
Integer 32 Literal
Integer 32 Variable

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

IVAL Move Numeric Table to I/O Unit Ex		
Argument Name	Type	Name
From Table	*Integer 32 Table*	*IO_STATUS_TABLE*
With Starting Index	*Integer 32 Variable*	*4*
To I/O Unit	*GRV-EPIC-PR1*	*VALVE_CONTROL*
Points per Module	*Integer 32 Literal*	*4*

OptoScript Example:

IvalMoveNumTableToIoUnitEx(*From Table, With Starting Index, To I/O Unit, Points per Module***)**

```
IvalMoveNumTableToIoUnitEx(IO_STATUS_TABLE, 4, VALVE_CONTROL, 4);
```

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

Notes: Primarily used to write to inputs.

Queue Errors: -3 = Invalid Length. Received if a negative value or a value greater than 32 is passed for *Points per Module* (Argument 3). The MemMap supports a maximum of 32 points per module.

-12 = Invalid table index value. Index was negative or greater than or equal to the table size.

-69 = Invalid parameter (null pointer) passed to command. Received if a null table object pointer is passed.

See Also:

"IVAL Set Analog Point" on page 626
"Disable Communication to All I/O Units" on page 610,
"Disable Communication to I/O Unit" on page 611

Move I/O Unit to Numeric Table

I/O Unit Action

NOTE: Use this command only for I/O units with modules that have four points or fewer. For modules with more than four points, use "Move I/O Unit to Numeric Table Ex" on page 317 or (for digital states only, by module) "Get I/O Unit Module States" on page 305 instead.

Function: To read the current on/off status or the current values of all points on each standard digital and analog module on an I/O unit and move the returned values to a numeric table.

Typical Use: To efficiently read all the data from all standard module points 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 standard module points.
- This command will populate 4 table elements for each module position on the rack, regardless of whether the module has 1, 2, or 4 points. Because of this and because Opto 22 racks support a maximum of 16 modules, the table needs to be at least 64 elements long (4 elements x 16 modules = 64 elements).

  If a rack is full of 2-channel analog modules, that would be 32 points total (2 points x 16 modules = 32 points). Since each rack position must be populated, and each position occupies four elements in the table, every four elements will contain the values for a 2-channel module.

  Table elements are populated as follows:

Module Position	Point	Table Element	Table Values for 2-Channel Modules	Table Values for 1-Channel Modules	Table Values for 4-Channel Modules
0	0	0	data	data	data
0	1	1	data	0.0	data
0	2	2	0.0	0.0	data
0	3	3	0.0	0.0	data
1	0	4	data	data	data
1	1	5	data	0.0	data
1	2	6	0.0	0.0	data
1	3	7	0.0	0.0	data
...	...	...	...	...	...

- Point zero corresponds to the first specified table element. The command returns status to the table beginning at the index specified in *Starting Index* (Argument 1). 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 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 0	**Argument 1**	**Argument 2**
From	**Starting Index**	**Of Table**
B100*	Integer 32 Literal	Float Table
B200*	Integer 32 Variable	Integer 32 Table
B3000 (Analog)*		Pointer Table
B3000 (Digital)*		
E1		
E2		
G4A8R, G4RAX*		
G4D16R*		
G4D32RS*		
G4EB2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
SNAP-B3000-ENET, SNAP-ENET-RTC**		
SNAP-BRS*		
SNAP-ENET-D64**		
SNAP-ENET-S64**		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Move I/O Unit to Numeric Table

Argument Name	Type	Name
From	*SNAP-PAC-R1*	*UNIT_255*
Starting Index	*Integer 32 Variable*	*0*
Of Table	*Float Table*	*DATA_TABLE*

OptoScript Example:

MoveIoUnitToNumTable(*From, Starting Index, Of Table***)**

```
MoveIoUnitToNumTable(UNIT_255, 0, DATA_TABLE);
```

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

Notes: For information on how a standard module differs from a high-density module, see the *PAC Control User's Guide* (form 1700).

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

-69 = Invalid parameter (null pointer) passed to command. Received if a null table object pointer is passed.

See Also:

"Move Numeric Table to I/O Unit" on page 320
"Get I/O Unit Module States" on page 305

Move I/O Unit to Numeric Table Ex

I/O Unit Action

NOTE: For use with high-density modules. For modules with four points or less, see "Move I/O Unit to Numeric Table" on page 315. To move digital data into elements by module rather than by point, see "Get I/O Unit Module States" on page 305.

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:

- See "Table Index Offsets" on page 753 for the length of the table required for values of points-per-module.
- Analog inputs that do not exist (for example, channels 2 and 3 of a 2-channel input module) appear as NaN (not a number).
- Analog outputs that do not exist (for example, channel 2 and 3 of a 2-channel output module) appear as 0.0.
- 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 *With Starting Index* (Argument 2). If there are more points than table elements from the specified index to the end of the table, no data is written to the table, and a -12 error is placed in the controller's message queue. For table index offsets for SNAP PAC, see "Table Index Offsets" on page 753.
- For digital points, if the point is on, there will be a 1 in the respective table element. (In a float table, on points are displayed as 1.0.) 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.
- If the points per module is less than or equal to 0, or greater than 32, a -3 error is placed in the controller's message queue.
- The range of *Points per Module* (Argument 3) is 1 to 32.
- The I/O unit may have modules with different numbers of points. For example, an I/O unit might have 4-channel, 8-channel, and 16-channel modules. In this configuration, *Points per Module* (Argument 3) determines the maximum number of points on any module. For modules that don't have the maximum number of points, zeros are stored for both analog and digital points.
- This command reads from the I/O unit. It updates the IVALs in the controller.

Arguments:

Argument 0	**Argument 1**
From I/O Unit	**To Table**
B100*	Float Table
B200*	Integer 32 Table
B3000 (Analog)*	Pointer Table
B3000 (Digital)*	
E1	
E2	
G4A8R, G4RAX*	
G4D16R*	
G4D32RS*	
G4EB2	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-BRS*	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2	**Argument 3**
With Starting Index	**Points per Module**
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable

Action Block Example:

Move I/O Unit to Numeric Table Ex

Argument Name	Type	Name
From I/O Unit	*GRV-EPIC-PR1*	*UNIT_255*
To Table	*Float Table*	*DATA_TABLE*
With Starting Index	*Integer 32 Literal*	*0*
Points per Module	*Integer 32 Literal*	*4*

OptoScript Example:

MoveIoUnitToNumTableEx (*From I/O Unit, To Table, With Starting Index, Points per Module*)

```
MoveIoUnitToNumTableEx(UNIT_255, DATA_TABLE, 0, 4);
```

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

Queue Errors:

-3 = Invalid Length. Received if a negative value, a zero, or a value greater than 32 is passed for *Points per Module* (Argument 3). The MemMap supports values of 1 through 32.

-12 = Invalid table index value. Index was negative or greater than or equal to the table size.

-69 = Invalid parameter (null pointer) passed to command.

See Also:
"Get All HDD Module States" on page 279
"Get I/O Unit Module States" on page 305
"Move I/O Unit to Numeric Table" on page 315
"Move Numeric Table to I/O Unit Ex" on page 322

Move Numeric Table to I/O Unit

I/O Unit Action

NOTE: Use this command for I/O units with modules that have only four points. For modules with more than four points, use *"Move Numeric Table to I/O Unit Ex" on page 322* *instead.*

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 On, 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.
- (PAC firmware R9.4d and higher.) This command does not write to discrete or analog points when either:
  - The float table value is set to a floating point NaN (not a number), or
  - An integer table element is set to 0x7fc00000 (the floating point representation of +NaN).

  You can use a NaN to prevent modifications to discrete and analog points. For details, see **Notes**.

Arguments:

Argument 0	Argument 1	Argument 2
Start at Index	**Of Table**	**Move to**
Integer 32 Literal	Float Table	B100*
Integer 32 Variable	Integer 32 Table	B200*
	Pointer Table	B3000 (Analog)*
		B3000 (Digital)*
		E1
		E2
		G4A8R, G4RAX*
		G4D16R*
		G4D32RS*
		G4EB2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		SNAP-B3000-ENET, SNAP-ENET-RTC**
		SNAP-BRS*
		SNAP-ENET-D64**
		SNAP-ENET-S64**
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS**
		SNAP-UP1-D64**
		SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

In this 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.

Move Numeric Table to I/O Unit		
Argument Name	Type	Name
Start at Index	*Integer 32 Variable*	*4*
Of Table	*Integer 32 Table*	*IO_STATUS_TABLE*
Move to	*GRV-EPIC-PR1*	*VALVE_CONTROL*

OptoScript Example:

`MoveNumTableToIoUnit(`*Start at Index, Of Table, Move to***`)`**

```
MoveNumTableToIoUnit(4, IO_STATUS_TABLE, VALVE_CONTROL);
```

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

Notes:

To prevent modifications to discrete and analog points, you can create a floating point variable that contains a NaN, and then assign it as a constant to table locations. For example, this OptoScript sample creates `i32Temp`, a 32-bit floating point variable that contains a NaN:

```
i32Temp = 0x7fc00000;
Move32Bits(i32Temp, f32FloatNan);
```

Queue Errors:

-12 = Invalid table index value. Index was negative or greater than or equal to the table size.

-69 = Invalid parameter (null pointer) passed to command. Received if a null table object pointer is passed.

See Also:

"Move I/O Unit to Numeric Table" on page 315

Move Numeric Table to I/O Unit Ex

I/O Unit Action

NOTE: Use this command for I/O units with modules that have more than four points. For modules with four points, use "Move Numeric Table to I/O Unit" on page 320 instead.

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:

- Please see "Table Index Offsets" on page 753 for the length of the table required for values of points-per-module.
- This command is much faster than using Turn On, Turn Off, or Move for each point.
- Updates the IVALs and XVALs for specified 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 specified points will be written to.
- The range of *Points per Module* (Argument 3) is 1 to 32.

Arguments:

Argument 0	Argument 1
From Table	**With Starting Index**
Float Table	Integer 32 Literal
Integer 32 Table	Integer 32 Variable
Pointer Table	

Argument 2	**Argument 3**
To I/O Unit	**Points per Module**
B100*	Integer 32 Literal
B200*	Integer 32 Variable
B3000 (Analog)*	
B3000 (Digital)*	
E1	
E2	
G4A8R, G4RAX*	
G4D16R*	
G4D32RS*	
G4EB2	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-BRS*	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

In this 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. For each module on the rack, four points per module are read.

Move Numeric Table to I/O Unit Ex

Argument Name	Type	Name
From Table	*Integer 32 Table*	*IO_STATUS_TABLE*
With Starting Index	*Integer 32 Variable*	*4*
To I/O Unit	*SNAP-PAC-R1*	*VALVE_CONTROL*
Points per Module	*Integer 32 Literal*	*4*

OptoScript Example:

`MoveNumTableToIoUnitEx` **(***From Table, With Starting Index, To I/O Unit, Points per Module***)**

```
MoveNumTableToIoUnitEx(IO_STATUS_TABLE, 4, VALVE_CONTROL, 4);
```

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

Queue Errors:

-3 = Invalid Length. Received if a negative value or a value greater than 32 is passed for *Points per Module* (Argument 3). The MemMap supports a maximum of 32 points per module.

-12 = Invalid table index value. Index was negative or greater than or equal to the table size.

-69 = Invalid parameter (null pointer) passed to command. Received if a null table object pointer is passed.

See Also: "Move I/O Unit to Numeric Table Ex" on page 317

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 PAC Control 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 a target address is enabled, it is available to be used. However, only one address can be used at a given time, so there can only be one active address. The active address is the address the controller is currently using. One address is always active. If communication to the active address fails and the control engine is not able to switch to the other address, then communication to the I/O unit will become disabled.
- Use *Must On Mask* (Argument 0) to enable one or both addresses.
- Use *Must Off Mask* (Argument 1) to disable one or both addresses.
- Use *Active Mask* (Argument 2) to make one address active.
- Only the last 2 bits of the 32-bit data field are used. Therefore, for Arguments 0, 1, and 2, you can use the integers 0, 1, 2, and 3 to indicate:
  0=No change.
  1=Primary Target Address.
  2=Secondary Target Address.
  3=Primary and Secondary Target Addresses. Not valid for *Active Mask* (Argument 2).

Arguments:

Argument 0	**Argument 1**	**Argument 2**
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

Action Block 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

Argument Name	Type	Name
Must On Mask	*Integer 32 Literal*	*2*
Must Off Mask	*Integer 32 Literal*	*1*
Active Mask	*Integer 32 Literal*	*2*

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 the *PAC Control User's Guide* (form 1700).
- Together, *Must On Mask* (Argument 0) and *Must Off Mask* (Argument 1) comprise the enable mask. You can use the enable mask in the following combinations:

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 2 makes one address active or both addresses inactive as follows:

Activate Primary	1
Activate Secondary	2

- A fully redundant system may also include PAC Display clients and OptoOPCServers. These commands only deal with the control engine communicating with I/O units. PAC Display and OptoOPCServer have their own mechanism for controlling their use of the network.

See Also: "Set Target Address State" on page 331
"Get Target Address State" on page 307

Set I/O Unit Configured Flag

I/O Unit Action

Function: Sets an internal flag to indicate that the I/O unit has been initialized by the controller.

Typical Use: Where there is a standby controller configured to take over communication to the I/O units in the event of a primary controller failure.

Details:

- This command should be issued for each I/O unit, preferably in the Powerup chart. Use it in both the primary and standby controller programs to keep them the same.
- By default, the controller assumes it is the only controller attached to the I/O and therefore must configure each I/O unit. This command makes the standby controller think it has already configured all the I/O units, which allows it to begin communicating with the I/O units immediately and without disrupting any control being performed by the I/O units (assuming it has just taken over as the primary). This command has no effect in a controller that has already established communication with the I/O units.

Arguments:

Argument
For I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Set I/O Unit Configured Flag

Argument Name	Type	Name
For I/O Unit	*SNAP-PAC-EB2*	*FURNACE_PID*

OptoScript Example:

`SetIoUnitConfiguredFlag(`*For I/O Unit***`)`**

```
SetIOUnitConfiguredFlag(FURNACE_PID);
```

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

Set I/O Unit from MOMO Masks

I/O Unit Action

Function: To control multiple 4-channel digital output points on the same digital I/O unit simultaneously with a single command.

Typical Use: To efficiently control a selected group of 4-channel 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 or high-density digital points in any position on the rack.
- To turn on a point, set the respective bit in the 32-bit data field of *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 32-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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 0 **Must On Mask**	Argument 1 **Must Off Mask**	Argument 2 **I/O Unit**
Integer 32 Literal	Integer 32 Literal	B100*
Integer 32 Variable	Integer 32 Variable	B3000 (Digital)*
Integer 64 Literal	Integer 64 Literal	E1
Integer 64 Variable	Integer 64 Variable	G4D16R*
		G4D32RS*
		G4EB2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC**
		SNAP-BRS*
		SNAP-ENET-D64**
		SNAP-ENET-S64**
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS**
		SNAP-UP1-D64**
		SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Set I/O Unit from MOMO Masks		
Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
I/O Unit	*SNAP-PAC-R1*	*PUMP_CTRL_UNIT*

The effect of this command is illustrated below. (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.)

	Point Number	63	62	61	60	59	58	57	56	> > >	7	6	5	4	3	2	1	0
Must-on Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	Binary	1	0	1	1	0	0	0	0	> > >	0	0	1	0	0	0	0	0
	Hex	B				0				> > >	2				0			

OptoScript Example:

`SetIoUnitFromMomo` **(***Must On Mask, Must Off Mask, I/O Unit***)**

```
SetIoUnitFromMomo(0x060003C0000000C2i64, 0xB0F240010308A020i64,
PUMP_CTRL_UNIT);
```

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

Notes:

- The I/O unit must be a digital I/O unit.
- Use Bit Set or Bit Clear to change individual bits in an integer variable.

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 PAC Control these are called the Primary Address and the Secondary Address.
  - An *enabled* address is one that has been configured to be used. An *active* address is the address the controller is currently using. Only one address can be active at a given time
  - By default, the controller uses the Primary Address. If the Primary Address is unavailable and the Secondary Address has been enabled, the controller will switch communication to the Secondary Address. At that point, the Secondary Address becomes the active address.
  - If communication to the active address fails and the control engine is not able to switch to the other address, then communication to the I/O unit will become disabled.
- Use *Must On Mask* (Argument 0) to enable one or both addresses.
- Use *Must Off Mask* (Argument 1) to disable one or both addresses.
- Use *Active Mask* (Argument 2) to make one address active.
- Use *I/O Unit* (Argument 3) to designate the I/O unit type.
- Only the last two bits of the 32-bit data field are used. Therefore, for Arguments 0, 1, and 2 you can use the integers 0, 1, 2, and 3 to indicate:
  0=No change.
  1=Primary Target Address.
  2=Secondary Target Address.
  3=Primary and Secondary Target Addresses. Not valid for *Active Mask* (Argument 2).

Arguments:

Argument 0
Must On Mask
Integer 32 Literal
Integer 32 Variable

Argument 1
Must Off Mask
Integer 32 Literal
Integer 32 Variable

Argument 2
Active Mask
Integer 32 Literal
Integer 32 Variable

Argument 3
I/O Unit
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block 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 State		
Argument Name	Type	Name
Must On Mask	*Integer 32 Literal*	*2*
Must Off Mask	*Integer 32 Literal*	*1*
Active Mask	*Integer 32 Literal*	*2*
I/O Unit	*GRV-EPIC-PR1*	*UNIT*

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 the *PAC Control User's Guide* (form 1700).
- *Must On Mask* (Argument 0) and *Must Off Mask* (Argument 1) together comprise the enable mask. You can use the enable mask in the following combinations:

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 2 makes one address active or both addresses inactive as follows:

Activate Primary	1
Activate Secondary	2

- A fully redundant system may also include PAC Display clients and OptoOPCServers. These commands only deal with the control engine communicating with I/O units. PAC Display and OptoOPCServer have their own mechanism for controlling their use of the network.
- This command does not test communication.

See Also:

"Set All Target Address States" on page 325
"Get Target Address State" on page 307

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 by using PAC Manager (see the *PAC Manager User's Guide* (form 1704) for instructions) or by using PAC Control in Debug mode (see the *PAC Control User's Guide*, (form 1700) for instructions).
- 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 0
On I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Write I/O Unit Configuration to EEPROM		
Argument Name	Type	Name
On I/O Unit	*SNAP-PAC-R1*	*FURNACE_CONTROL*

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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-534 = Attempts to communicate with I/O unit failed. Make sure I/O unit is turned on.

12: I/O Unit - Event Message Commands

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 PAC 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 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

Argument 1
Event Message #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get I/O Unit Event Message State

Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1*	*PAC_A*
Event Message #	*Integer 32 Literal*	*0*

Put Result in	*Integer 32 Variable*	*State*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

`GetIoUnitEventMsgState(`*I/O Unit, Event Message #, Put Result in***`)`**

```
Status = GetIoUnitEventMsgState(PAC_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 the *PAC Control User's Guide* (form 1700).
- Use PAC Manager 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Get I/O Unit Event Message Text" on page 337
"Set I/O Unit Event Message State" on page 339
"Set I/O Unit Event Message Text" on page 341

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 PAC 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 *Put Result in* (Argument 2). The string variable for *Put Result in* should be 128 characters long to hold the message text.

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

Argument 1
Event Message #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
String Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get I/O Unit Event Message Text

Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PAC_A*
Event Message #	*Integer 32 Literal*	*0*
Put Result in	*String Variable*	*Msg_0*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

`GetIoUnitEventMsgText``(`*I/O Unit, Event Message #, Put Result in*`)`

```
Status = GetIoUnitEventMsgText(PAC_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 the *PAC Control User's Guide* (form 1700).
- Use PAC Manager 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 *Put Result in* (Argument 2) 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 *Put Result in* (Argument 2) must be 128 characters long.

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

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also: "Get I/O Unit Event Message State" on page 335
"Set I/O Unit Event Message State" on page 339
"Set I/O Unit Event Message Text" on page 341

Set I/O Unit Event Message State

I/O Unit—Event Message Action

Function: To activate or deactivate a SNAP PAC 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 PAC Manager 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 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

Argument 1
Event Message #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
State
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Set I/O Unit Event Message State		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PAC_A*
Event Message #	*Integer 32 Literal*	*5*
State	*Integer 32 Literal*	*1*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

`SetIoUnitEventMsgState` **(***I/O Unit, Event Message #, State***)**

```
Status = SetIoUnitEventMsgState(PAC_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 the *PAC Control User's Guide* (form 1700).
- 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Event Message Text" on page 341
"Get I/O Unit Event Message State" on page 335
"Get I/O Unit Event Message Text" on page 337

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 PAC 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 PAC Manager 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 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

Argument 1
Event Message #
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Message Text
String Literal
String Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Set I/O Unit Event Message Text		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1*	*PAC_A*
Event Message #	*Integer 32 Literal*	*5*
Message Text	*String Literal*	*Machine failure*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

SetIoUnitEventMsgText(*I/O Unit, Event Message #, Message Text***)**

```
STATUS = SetIoUnitEventMsgText(PAC_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 the *PAC Control User's Guide* (form 1700).

- 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also: "Set I/O Unit Event Message State" on page 339
"Get I/O Unit Event Message State" on page 335
"Get I/O Unit Event Message Text" on page 337

13: I/O Unit - Memory Map Commands

Read Number from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a value from a SNAP PAC 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 PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with GRV-EPIC processors, SNAP PAC I/O units that have been configured in PAC Control or PAC Manager. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Get I/O Unit Scratch Pad Float Element and related commands).
- *Mem address* (Argument 1) includes only the last eight digits of the memory map address (the lower 32 bits).

Arguments:

Argument 0
I/O Unit
E1
E2
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

Argument 1
Mem address
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Float Variable
Integer 32 Variable
Integer 64 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Read Number from I/O Unit Memory Map		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1*	*MYIOUNIT*
Mem address	*Integer 32 Literal*	*0xFFFFFFFF*
Put Result in	*Integer 32 Variable*	*MYINTVAR*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

ReadNumFromIoUnitMemMap (*I/O Unit, Mem address, Put Result in*)

```
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 *OptoMMP Protocol Guide* (form 1465) to determine the data types for specific areas of the memory map.
- If *Put Result in* (Argument 2) 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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address.

-69 = Invalid parameter (null pointer) passed to command.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:
"Read Numeric Table from I/O Unit Memory Map" on page 346
"Write Numeric Table to I/O Unit Memory Map" on page 355
"Write Number to I/O Unit Memory Map" on page 353
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365

Read Numeric Table from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a range of values from a SNAP PAC 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 PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Control or PAC Manager. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Get I/O Unit Scratch Pad Integer 32 Table and related commands).
- *Length* (Argument 0) 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 300.
- *Mem address* (Argument 3) includes only the last eight digits of the memory map address (the lower 32 bits).

Arguments:

Argument 0	Argument 1	Argument 2
Length	**Start Index**	**I/O Unit**
Integer 32 Literal	Integer 32 Literal	E1
Integer 32 Variable	Integer 32 Variable	E2
		G4EB2
		Generic OptoMMP Device
		GRV-EPIC-PR1
		GRV-EPIC-PR2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC*
		SNAP-ENET-D64*
		SNAP-ENET-S64*
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS*
		SNAP-UP1-D64*
		SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3	Argument 4	Argument 5
Mem address	**Put Result in**	**Put Status in**
Integer 32 Literal	Float Table	Integer 32 Variable
Integer 32 Variable	Integer 32 Table	

Action Block Example:

Read Numeric Table from I/O Unit Memory Map		
Argument Name	Type	Name
Length	*Integer 32 Literal*	*0x10*
Start Index	*Integer 32 Literal*	*0x5*
I/O Unit	*SNAP-PAC-EB1*	*MYIOUNIT*
Mem address	*Integer 32 Literal*	*0xFFFFFFFF*
Put Result in	*Integer 32 Table*	*MYINTTABLE*

OptoScript Example:

ReadNumTableFromIoUnitMemMap(*Length, Start Index, I/O Unit, Mem address, Put Result in***)**

```
STATUS = ReadNumTableFromIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYINTTABLE);
```

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

Using this command in OptoScript code, you can use hex in some arguments and a different 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 *Length* (Argument 0) and *Start Index* (Argument 1) 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 variable from the analog point area of the memory map. A float variable should be used instead. See the *OptoMMP Protocol Guide* (form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = success

-3 = Buffer overrun or invalid length error. A value > 300 was passed for the Length.

-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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address.

-69 = Invalid parameter (null pointer) passed to command.

-81 = Error writing to memory map. Invalid memory map address.

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

See Also:

"Read Number from I/O Unit Memory Map" on page 343
"Write Numeric Table to I/O Unit Memory Map" on page 355
"Write Number to I/O Unit Memory Map" on page 353
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad Float Table" on page 365

Read String from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a string from a SNAP PAC 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 PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Control or PAC Manager. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Get I/O Unit Scratch Pad String Element and related commands).
- *Mem address* (Argument 2) includes only the last eight digits of the memory map address (the lower 32 bits).

Arguments:

Argument 0	Argument 1
Length	**I/O Unit**
Integer 32 Literal	E1
Integer 32 Variable	E2
	G4EB2
	Generic OptoMMP Device
	GRV-EPIC-PR1
	GRV-EPIC-PR2
	GRV-R7-MM1001-10
	GRV-R7-MM2001-10
	GRV-R7-I1VAPM-3
	SNAP-B3000-ENET, SNAP-ENET-RTC*
	SNAP-ENET-D64*
	SNAP-ENET-S64*
	SNAP-PAC-EB1
	SNAP-PAC-EB2
	SNAP-PAC-R1
	SNAP-PAC-R1-B
	SNAP-PAC-R2
	SNAP-PAC-SB1
	SNAP-PAC-SB2
	SNAP-UP1-ADS*
	SNAP-UP1-D64*
	SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2	Argument 3	Argument 4
Mem address	**Put Result in**	**Put Status in**
Integer 32 Literal	String Variable	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Read String from I/O Unit Memory Map		
Argument Name	Type	Name
Length	*Integer 32 Literal*	*20*
I/O Unit	*SNAP-PAC-EB1*	*MYIOUNIT*
Mem address	*Integer 32 Literal*	*0xFFFFFFFF*
Put Result in	*String Variable*	*MYSTRINGVAR*
Put Status In	*Integer 32 Variable*	*STATUS*

OptoScript Example:

ReadStrFromIoUnitMemMap(*Length, I/O Unit, Mem address, Put Result in***)**

```
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 *OptoMMP Protocol Guide* (form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = Success

-3 = Invalid length. *Length* (Argument 0) 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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-69 = Invalid parameter (null pointer) passed to command.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Read String Table from I/O Unit Memory Map" on page 350
"Write String Table to I/O Unit Memory Map" on page 357
"Write String to I/O Unit Memory Map" on page 359
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

Read String Table from I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Read a range of values from a SNAP PAC I/O memory map and store them in a string table.

Typical Use: To access areas of the memory map not directly supported by PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Control or PAC Manager. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are reading the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Get I/O Unit Scratch Pad String Table and related commands). If you read using the string command, you might get more than one string, along with whatever length bytes are there. Using the Scratch Pad commands, you get just the requested string data.
- If you're reading a string of characters from a memory map location, the length is limited by the length of your destination string, which can be up to 1024 characters long.
- *Argument 0*, Length, is the number of bytes to read in the memory map. Data is read in block sizes that are multiples of four.
- *Argument 3*, Mem address, includes only the last eight digits of the memory map address (the lower 32 bits).
- There is a limit of 64 table elements.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
Length	**Start Index**	**I/O Unit**
Integer 32 Literal	Integer 32 Literal	E1
Integer 32 Variable	Integer 32 Variable	E2
		G4EB2
		Generic OptoMMP Device
		GRV-EPIC-PR1
		GRV-EPIC-PR2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC*
		SNAP-ENET-D64*
		SNAP-ENET-S64*
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS*
		SNAP-UP1-D64*
		SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3	**Argument 4**	**Argument 5**
Mem address	**Put Result in**	**Put Status in**
Integer 32 Literal	String Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Read String Table from I/O Unit Memory Map		
Argument Name	Type	Name
Length	*Integer 32 Literal*	*0x10*
Start Index	*Integer 32 Literal*	*0x5*
I/O Unit	*SNAP-PAC-R1*	*MYIOUNIT*
Mem address	*Integer 32 Literal*	*0xFFFFFFFF*
Put Result in	*String Table*	*MYSTRINGTABLE*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

ReadStrTableFromIoUnitMemMap(*Length, Start Index, I/O Unit, Mem address, Put Result in***)**

```
STATUS = ReadStrTableFromIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYSTRINGTABLE);
```

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

Using this command in OptoScript, you can use hex in one argument but not in others.

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 *Length* (Argument 0) and *Start Index* (Argument 1) 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 *OptoMMP Protocol Guide* (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* (Argument 0) 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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address.

-69 = Invalid parameter (null pointer) passed to command.

-81 = Error writing to memory map. Invalid memory map address.

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

See Also:

"Read String from I/O Unit Memory Map" on page 348
"Write String Table to I/O Unit Memory Map" on page 357
"Write String to I/O Unit Memory Map" on page 359

"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

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 a SNAP PAC I/O memory map address.

Typical Use: To access areas of the memory map not directly supported by PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Manager or PAC Control. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Set I/O Unit Scratch Pad Integer 32 Element and related commands).

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Mem Address**
E1	Integer 32 Literal
E2	Integer 32 Variable
G4EB2	
Generic OptoMMP Device	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
SNAP-B3000-ENET, SNAP-ENET-RTC*	
SNAP-ENET-D64*	
SNAP-ENET-S64*	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS*	
SNAP-UP1-D64*	
SNAP-UP1-M64*	

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2 **From**	Argument 3 **Put Status in**
Float Literal	Integer 32 Variable
Float Variable	
Integer 32 Literal	
Integer 32 Variable	
Integer 64 Literal	
Integer 64 Variable	

Action Block Example:

Write Number to I/O Unit Memory Map		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*MYIOUNIT*
Mem Address	*Integer 32 Literal*	*0xFFFFFFFF*
From	*Integer 32 Variable*	*MYINTVAR*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

WriteNumToIoUnitMemMap(*I/O Unit, Mem Address, From***)**

```
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 PAC Control 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 *OptoMMP Protocol Guide* (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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address or read-only address.

-69 = Invalid parameter (null pointer) passed to command.

-81 = Error writing to memory map. Invalid memory map address.

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

See Also:

"Write Numeric Table to I/O Unit Memory Map" on page 355
"Read Numeric Table from I/O Unit Memory Map" on page 346
"Read Number from I/O Unit Memory Map" on page 343
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385

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 a SNAP PAC I/O memory map address.

Typical Use: To access areas of the memory map not directly supported by PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Manager or PAC Control. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Set I/O Unit Scratch Pad Integer 32 Table and related commands).
- *Argument 0*, Length, is the number of table elements and also the length of data in the memory map in quads (groups of four bytes). Maximum length is 300.
- *Argument 3*, Mem address, includes only the last eight hex digits (four bytes) of the memory map address (the lower 32 bits).

Arguments:

Argument 0	Argument 1	Argument 2
Length	**Start Index**	**I/O Unit**
Integer 32 Literal	Integer 32 Literal	E1
Integer 32 Variable	Integer 32 Variable	E2
		G4EB2
		Generic OptoMMP Device
		GRV-EPIC-PR1
		GRV-EPIC-PR2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC*
		SNAP-ENET-D64*
		SNAP-ENET-S64*
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS*
		SNAP-UP1-D64*
		SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3	Argument 4	Argument 5
Mem Address	**From**	**Put Status in**
Integer 32 Literal	Float Table	Integer 32 Variable
Integer 32 Variable	Integer 32 Table	

Action Block Example:

Write Numeric Table to I/O Unit Memory Map		
Argument Name	Type	Name
Length	*Integer 32 Literal*	*0x10*
Start Index	*Integer 32 Literal*	*0x5*
I/O Unit	*GRV-EPIC-PR1*	*MYIOUNIT*
Mem Address	*Integer 32 Literal*	*0xFFFFFFFF*
From	*Integer 32 Table*	*MYINTTABLE*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

`WriteNumTableToIoUnitMemMap(`*Length, Start Index, I/O Unit, Mem Address, From***`)`**

```
STATUS = WriteNumTableToIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYINTTABLE);
```

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

Using this command in OptoScript, you can use hex in some arguments and decimal in others.

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 *Length* (Argument 0) and *Start Index* (Argument 1) 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 *OptoMMP Protocol Guide* (form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = Success

-3 = Buffer overrun or invalid length error. A value > 300 was passed for the Length.

-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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address or read-only address.

-69 = Invalid parameter (null pointer) passed to command.

-81 = Error writing to memory map. Invalid memory map address.

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

See Also:

"Read Number from I/O Unit Memory Map" on page 343
"Read Numeric Table from I/O Unit Memory Map" on page 346
"Write Number to I/O Unit Memory Map" on page 353
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387

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 SNAP PAC I/O memory map.

Typical Use: To access areas of the memory map not directly supported by PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Manager or PAC Control. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Set I/O Unit Scratch Pad String Table and related commands).
- *Length* (Argument 0) is the number of table elements.
- *Mem address* (Argument 3) includes only the last eight digits of the memory map address (the lower 32 bits).
- There is a limit of 64 table elements.
- 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 *Mem Address* (Argument 3).

Arguments:

Argument 0 **Length**	Argument 1 **Start Index**	Argument 2 **I/O Unit**
Integer 32 Literal	Integer 32 Literal	E1
Integer 32 Variable	Integer 32 Variable	E2
		G4EB2
		Generic OptoMMP Device
		GRV-EPIC-PR1
		GRV-EPIC-PR2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC*
		SNAP-ENET-D64*
		SNAP-ENET-S64*
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS*
		SNAP-UP1-D64*
		SNAP-UP1-M64*

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3 **Mem Address**	Argument 4 **From**	Argument 5 **Put Status in**
Integer 32 Literal	String Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Write String Table to I/O Unit Memory Map		
Argument Name	Type	Name
Length	*Integer 32 Literal*	*0x10*
Start Index	*Integer 32 Literal*	*0x5*
I/O Unit	*GRV-EPIC-PR1*	*MYIOUNIT*
Mem Address	*Integer 32 Literal*	*0xFFFFFFFF*
From	*String Table*	*MYSTRINGTABLE*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

WriteStrTableToIoUnitMemMap(*Length, Start Index, I/O Unit, Mem Address, From***)**

```
STATUS = WriteStrTableToIoUnitMemMap(0x10, 0x5, MYIOUNIT, 0xFFFFFFFF,
MYSTRINGTABLE);
```

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

Using this command in OptoScript, you can use hex in some arguments and decimal in others.

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 *Length* (Argument 0) and *Start Index* (Argument 1) 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 *OptoMMP Protocol Guide* (form 1465) to determine the data types for specific areas of the memory map.

Status Codes:

0 = Success

-3 = Invalid length. *Length* (Argument 0) 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address or read-only address.

-69 = Invalid parameter (null pointer) passed to command.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Read String from I/O Unit Memory Map" on page 348
"Read String Table from I/O Unit Memory Map" on page 350
"Write String to I/O Unit Memory Map" on page 359
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395

Write String to I/O Unit Memory Map

I/O Unit—Memory Map Action

Function: Write a string variable into a SNAP PAC I/O memory map address.

Typical Use: To access areas of the memory map not directly supported by PAC Control.

Details:

- To use this command with a SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- This command works with SNAP PAC I/O units that have been configured in PAC Manager or PAC Control. The control engine must be on the I/O unit or connected to another I/O unit for this command to work.
- If you are writing to the Scratch Pad area of the memory map, use 14: I/O Unit - Scratch Pad Commands instead (Set I/O Unit Scratch Pad String Element and related commands).

Arguments:

Argument 0
I/O Unit
E1
E2
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-ENET-S64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS*
SNAP-UP1-D64*
SNAP-UP1-M64*

Argument 1
Mem Address
Integer 32 Literal
Integer 32 Variable

* Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
From
String Literal
String Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Write String to I/O Unit Memory Map		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*MYIOUNIT*
Mem Address	*Integer 32 Literal*	*0xFFFFFFFF*
From	*String Variable*	*MYSTRINGVAR*
Put Status in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

WriteStrToIoUnitMemMap(*I/O Unit, Mem Address, From***)**

```
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 *OptoMMP Protocol Guide* (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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-56 = Invalid memory map address or read-only address.

-69 = Invalid parameter (null pointer) passed to command.

-81 = Error writing to memory map. Invalid memory map address.

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

See Also:

"Write String Table to I/O Unit Memory Map" on page 357
"Read String from I/O Unit Memory Map" on page 348
"Read String Table from I/O Unit Memory Map" on page 350
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395

14: I/O Unit - Scratch Pad Commands

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 PAC controller or 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 *Put Result in* (Argument 1).

Arguments:

Argument 0 I/O Unit	Argument 1 Put Result in	Argument 2 Put Status in
G4EB2	Integer 64 Variable	Integer 32 Variable
Generic OptoMMP Device		
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Get I/O Unit Scratch Pad Bits		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PR1*
Put Result in	*Integer 64 Variable*	*MyInt64Var*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadBits(*I/O Unit, Put Result in***)**

```
Status = GetIoUnitScratchPadBits(PR1, 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 15: 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

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

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 *Index* (Argument 1). The float value is returned to the float variable named in *Put Result in* (Argument 2).

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Index
Integer 32 Literal
Integer 32 Variable

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Float Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get I/O Unit Scratch Pad Float Element

Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1*	*R1*
Index	*Integer 32 Literal*	*26*
Put Result in	*Float Variable*	*MyFloatVar*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadFloatElement(*I/O Unit, Index, Put Result in***)**

```
Status = GetIoUnitScratchPadFloatElement(R1, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad Float Element" on page 381
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 10,240 elements (index numbers 0–10,239). Enter the number of elements you want to read in *Length* (Argument 1) and the index number of the starting element in *From Index* (Argument 2).
- The float values are returned to the float table named in *To Table* (Argument 4), starting at the index shown in *To Index* (Argument 3).
- *From Index* (Argument 2) is the start index of the source table.
- *To Index* (Argument 3) is the start index of the destination table that data will be written to.

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Length**	Argument 2 **From Index**
G4EB2	Integer 32 Literal	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3 **To Index**	Argument 4 **To Table**	Argument 5 **Put Status in**
Integer 32 Literal	Float Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Get I/O Unit Scratch Pad Float Element		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PR1*
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(PR1, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length. *Length* (Argument 1) is 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad Float Table" on page 383
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 0–10,239). Enter the index number of the element you want to read in *Index* (Argument 1).
- The integer 32 value is returned to the integer 32 variable in *Put Result in* (Argument 2).

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Index
Integer 32 Literal
Integer 32 Variable

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get I/O Unit Scratch Pad Integer 32 Element		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1*	*R1*
Index	*Integer 32 Literal*	*26*
Put Result in	*Integer 32 Variable*	*MyInt32Var*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadInt32Element(*I/O Unit, Index, Put Result in***)**

```
Status = GetIoUnitScratchPadInt32Element(R1, 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 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 0–10,239). Enter the number of elements you want to read in *Length* (Argument 1), and the index number of the starting element in *From Index* (Argument 2).
- The integer values are returned to the integer 32 table in *To Table* (Argument 4), starting at the index shown in *To Index* (Argument 3).

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Length**	Argument 2 **From Index**
G4EB2	Integer 32 Literal	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3 **To Index**	Argument 4 **To Table**	Argument 5 **Put Status in**
Integer 32 Literal	Integer 32 Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Get I/O Unit Scratch Pad Integer 32 Table		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1-B*	*PAC_B*
Length	*Integer 32 Literal*	*64*
From Index	*Integer 32 Literal*	*0*
To Index	*Integer 32 Literal*	*0*
To Table	*Integer 32 Table*	*MyInt32Table*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadInt32Table(*I/O Unit, Length, From Index, To Index, To Table***)**

```
Status = GetIoUnitScratchPadInt32Table(PAC_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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length. *Length* (Argument 1) is 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

Get I/O Unit Scratch Pad Integer 64 Element

I/O Unit—Scratch Pad Action

Function: To read an integer 64 in the Scratch Pad area of a local or remote *groov* EPIC processor or *groov* RIO module.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the processor's or module's Scratch Pad area and retrieved by a peer on the network.

Details:

- You can use Set I/O Unit Scratch Pad Integer 64 Element to store the variable data in the Scratch Pad area, and then use this command to retrieve it.
- The integer 64 area of the Scratch Pad is a table containing 1,024 elements (index numbers 0–1,023). Enter the index number of the element you want to read in *Index* (Argument 1).
- The integer 64 value is returned to the integer 64 variable in *Put Result in* (Argument 2).

Arguments:

Argument 0
I/O Unit
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3

Argument 1
Index
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
Put Result in
Integer 64 Variable

Argument 3
Put Status in
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Get I/O Unit Scratch Pad Integer 64 Element		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*EPIC1*
Index	*Integer 32 Literal*	*26*
Put Result in	*Integer 64 Variable*	*MyInt64Var*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadInt64Element (*I/O Unit, Index, Put Result in*)

```
Status = GetIoUnitScratchPadInt64Element(EPIC1, 26, MyInt64Var);
```

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

Notes:

- To retrieve more than one integer 64 value in a single command, use Get I/O Unit Scratch Pad Integer 64 Table.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another *groov* EPIC 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:
"Set I/O Unit Scratch Pad Integer 64 Element" on page 389
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad Integer 64 Table" on page 373
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

Get I/O Unit Scratch Pad Integer 64 Table

I/O Unit—Scratch Pad Action

Function: To read a series of integer 64 values in the Scratch Pad area of a local or remote *groov* EPIC processor or *groov* RIO module.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the processor's or module's Scratch Pad area and retrieved by a peer on the network.

Details:

- You can use Set I/O Unit Scratch Pad Integer 64 Element more than once, or use Set I/O Unit Scratch Pad Integer 64 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 64 area of the Scratch Pad is a table containing 1,024 elements (index numbers 0–1,023). Enter the number of elements you want to read in *Length* (Argument 1), and the index number of the starting element in *From Index* (Argument 2).
- The integer values are returned to the integer 64 table in *To Table* (Argument 4), starting at the index shown in *To Index* (Argument 3).

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Length**	Argument 2 **From Index**
Generic OptoMMP Device	Integer 32 Literal	Integer 32 Literal
GRV-EPIC-PR1	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR2	Integer 64 Literal	Integer 64 Literal
GRV-R7-MM1001-10	Integer 64 Variable	Integer 64 Variable
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		

Argument 3 **To Index**	Argument 4 **To Table**	Argument 5 **Put Status in**
Integer 32 Literal	Integer 64 Table	Integer 32 Variable
Integer 32 Variable		Integer 64 Variable
Integer 64 Literal		
Integer 64 Variable		

Action Block Example:

Get I/O Unit Scratch Pad Integer 64 Table

Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*EPIC1*
Length	*Integer 32 Literal*	*64*
From Index	*Integer 32 Literal*	*0*
To Index	*Integer 32 Literal*	*0*
To Table	*Integer 64 Table*	*MyInt64Table*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadInt64Table(*I/O Unit, Length, From Index, To Index, To Table*)

```
Status = GetIoUnitScratchPadInt64Table(EPIC1,64, 0, 0, MyInt64Table);
```

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

Notes:

- To retrieve a single integer 64 value, use Get I/O Unit Scratch Pad Integer 64 Element.

- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another *groov* EPIC 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 the *PAC Control User's Guide* (form 1700).

Status Codes: 0 = success

-3 = Invalid length. *Length* (Argument 1) is less than 0 or greater than 1023.

-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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad Integer 64 Table" on page 391
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad Integer 64 Element" on page 371
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad String Table" on page 377

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 *Index* (Argument 1). The string is returned to the string variable in *Put Result in* (Argument 2).

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Index
Integer 32 Literal
Integer 32 Variable

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Put Result in
String Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Get I/O Unit Scratch Pad String Element		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PR1*
Index	*Integer 32 Literal*	*26*
Put Result in	*String Variable*	*MyStringVar*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadStringElement(*I/O Unit, Index, Put Result in***)**

```
Status = GetIoUnitScratchPadStringElement(PR1,26, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad String Element" on page 393
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad Bits" on page 361
"Get I/O Unit Scratch Pad String Table" on page 377

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 *Length* (Argument 1), and the index number of the starting element in *From Index* (Argument 2).
- The string values are returned to the string table in *To Table* (Argument 4), starting at the index shown in *To Index* (Argument 3).

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Length**	Argument 2 **From Index**
G4EB2	Integer 32 Literal	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3 **To Index**	Argument 4 **To Table**	Argument 5 **Put Status in**
Integer 32 Literal	String Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Get I/O Unit Scratch Pad String Table		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-R1-B*	*PAC_B*
Length	*Integer 32 Literal*	*8*
From Index	*Integer 32 Literal*	*0*
To Index	*Integer 32 Literal*	*0*
To Table	*String Table*	*MyStringTable*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

GetIoUnitScratchPadStringTable(*I/O Unit, Length, From Index, To Index, To Table***)**

```
Status = GetIoUnitScratchPadStringTable(PAC_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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length. *Length* (Argument 1) is less than 0 or greater than 63.

-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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

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

See Also:

"Set I/O Unit Scratch Pad String Table" on page 395
"Get I/O Unit Scratch Pad Float Element" on page 363
"Get I/O Unit Scratch Pad Float Table" on page 365
"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Get I/O Unit Scratch Pad String Element" on page 375
"Get I/O Unit Scratch Pad Bits" on page 361

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Must On Mask
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
Must Off Mask
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Set I/O Unit Scratch Pad Bits from MOMO Mask		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PR1*
Must On Mask	*Integer 64 Variable*	*MyOnMask*
Must Off Mask	*Integer 64 Variable*	*MyOffMask*
Put Status in	*Integer 32Variable*	*Status*

OptoScript Example:

SetIoUnitScratchPadBitsFromMomo(*I/O Unit, Must On Mask, Must Off Mask***)**

```
Status = SetIoUnitScratchPadBitsFromMomo(PR1, 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 *Must-on Mask* (Argument 1) and *Must-off Mask* (Argument 2). PAC Control 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

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

-52 = Invalid connection—not opened. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad Bits" on page 361
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 0–10,239). Enter the index number of the element you want to set in *Index* (Argument 1).

Arguments:

Argument 0	**Argument 1**
I/O Unit	**Index**
G4EB2	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
SNAP-B3000-ENET, SNAP-ENET-RTC*	
SNAP-ENET-D64*	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	

* *Not intended for use with these brains*, because they don't have

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2	**Argument 3**
From	**Put Status in**
Float Literal	Integer 32 Variable
Float Variable	

Action Block Example:

Set I/O Unit Scratch Pad Float Element		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-SB1*	*PAC_SB*
Index	*Integer 32 Literal*	*26*
From	*Float Literal*	*1.2*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

`SetIoUnitScratchPadFloatElement(` *I/O Unit, Index, From* **`)`**

```
Status = SetIoUnitScratchPadFloatElement(PAC_SB, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad Float Element" on page 363
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 0–10,239).
- Enter the number of elements you want to set in the Scratch Pad area in *Length* (Argument 1), and the index number of the starting element in *To Index* (Argument 2).
- In *From Index* (Argument 3), enter the starting index of the table you are writing from. In *From Table* (Argument 4), enter the name of the table.

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Length**	Argument 2 **To Index**
G4EB2	Integer 32 Literal	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3 **From Index**	Argument 4 **From Table**	Argument 5 **Put Status in**
Integer 32 Literal	Float Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Set I/O Unit Scratch Pad Float Table		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-EB1*	*PAC_EB*
Length	*Integer 32 Literal*	*64*
To Index	*Integer 32 Literal*	*0*
From Index	*Integer 32 Literal*	*0*
From Table	*Float Table*	*MyFloatTable*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

SetIoUnitScratchPadFloatTable(*I/O Unit, Length, To Index, From Index, From Table***)**

```
Status = SetIoUnitScratchPadFloatTable(PAC_EB, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length. *Length* (Argument 2) is 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad Float Table" on page 365
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 0–10,239). Enter the index number of the element you want to set in *Index* (Argument 1).

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Index
Integer 32 Literal
Integer 32 Variable

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
From
Integer 32 Literal
Integer 32 Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Set I/O Unit Scratch Pad Integer 32 Element		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-SB1*	*PAC_SB*
Index	*Integer 32 Literal*	*26*
From	*Integer 32 Literal*	*99*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

`SetIoUnitScratchPadInt32Element(`*I/O Unit, Index, From***`)`**

```
Status = SetIoUnitScratchPadInt32Element(PAC_SB, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 0–10,239). Enter the number of elements you want to set in *Length* (Argument 1), and the index number of the starting element in *To Index* (Argument 2).

Arguments:

Argument 0	Argument 1	Argument 2
I/O Unit	**Length**	**To Index**
G4EB2	Integer 32 Literal	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3	Argument 4	Argument 5
From Index	**From Table**	**Put Status in**
Integer 32 Literal	Integer 32 Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Set I/O Unit Scratch Pad Integer 32 Table

Argument Name	Type	Name
I/O Unit	*SNAP-PAC-SB1*	*PAC_SB*
Length	*Integer 32 Literal*	*64*
To Index	*Integer 32 Literal*	*0*
From Index	*Integer 32 Literal*	*0*

From Table	*Integer 32 Table*	*MyInt32Table*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example: **SetIoUnitScratchPadInt32Table(***I/O Unit, Length, To Index, From Index, From Table***)**

```
Status = SetIoUnitScratchPadInt32Table(PAC_SB, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length. *Length* (Argument 2) is 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. The connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad Integer 32 Table" on page 369
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

Set I/O Unit Scratch Pad Integer 64 Element

I/O Unit—Scratch Pad Action

Function: To write an integer 64 to the Scratch Pad area of a local or remote *groov* EPIC processor or *groov* RIO module.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the processor's or module's Scratch Pad area and retrieved by a peer on the network.

Details:

- 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 64 Element to retrieve it.
- The integer 64 area of the Scratch Pad is a table containing 1,024 elements (index numbers 0–1,023). Enter the index number of the element you want to set in *Index* (Argument 1).

Arguments:

Argument 0
I/O Unit
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3

Argument 1
Index
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
From
Integer 64 Literal
Integer 64 Variable

Argument 3
Put Status in
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Set I/O Unit Scratch Pad Integer 64 Element		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*EPIC1*
Index	*Integer 32 Literal*	*26*
From	*Integer 64 Literal*	*99*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

SetIoUnitScratchPadInt64Element(*I/O Unit, Index, From***)**

```
Status = SetIoUnitScratchPadInt64Element(EPIC1, 26, 99);
```

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

Notes:

- To write more than one integer 64 value to the Scratch Pad area in a single command, use Set I/O Unit Scratch Pad Integer 64 Table.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another *groov* EPIC 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad Integer 32 Element" on page 367
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad Integer 64 Table" on page 391
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

Set I/O Unit Scratch Pad Integer 64 Table

I/O Unit—Scratch Pad Action

Function: To write a series of integer 64 values to the Scratch Pad area of a local or remote *groov* EPIC processor or *groov* RIO module.

Typical Use: For peer-to-peer communication. Strategy variable data can be stored in the processor's or module's Scratch Pad area and retrieved by a peer on the network.

Details:

- 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 64 Element or Get I/O Unit Scratch Pad Integer 64 Table to retrieve it.
- The integer 64 area of the Scratch Pad is a table containing 1,024 elements (index numbers 0–1,023). Enter the number of elements you want to set in *Length* (Argument 1), and the index number of the starting element in *To Index* (Argument 2).

Arguments:

Argument 0	Argument 1	Argument 2
I/O Unit	**Length**	**To Index**
Generic OptoMMP Device	Integer 32 Literal	Integer 32 Literal
GRV-EPIC-PR1	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR2	Integer 64 Literal	Integer 64 Literal
GRV-R7-MM1001-10	Integer 64 Variable	Integer 64 Variable
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		

Argument 3	Argument 4	Argument 5
From Index	**From Table**	**Put Status in**
Integer 32 Literal	Integer 64 Table	Integer 32 Variable
Integer 32 Variable		Integer 64 Variable
Integer 64 Literal		
Integer 64 Variable		

Action Block Example:

Set I/O Unit Scratch Pad Integer 32 Table		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*EPIC1*
Length	*Integer 32 Literal*	*64*
To Index	*Integer 32 Literal*	*0*
From Index	*Integer 32 Literal*	*0*
From Table	*Integer 64 Table*	*MyInt64Table*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example: **SetIoUnitScratchPadInt64Table(***I/O Unit, Length, To Index, From Index, From Table***)**

```
Status = SetIoUnitScratchPadInt64Table(EPIC1, 64, 0, 0, MyInt64Table);
```

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

Notes:

- To write a single integer 64 value to the Scratch Pad area, use Set I/O Unit Scratch Pad Integer 64 Element.
- The I/O unit Scratch Pad area is for general-purpose use and is accessible to any network device (for example, another *groov* EPIC 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 the *PAC Control User's Guide* (form 1700).

Status Codes: 0 = success

-3 = Invalid length. *Length* (Argument 2) is less than 0 or greater than 1023.

-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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:
"Get I/O Unit Scratch Pad Integer 64 Table" on page 373
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad Integer 64 Element" on page 389
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad String Table" on page 395
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 *Index* (Argument 1).
- Each string element can hold 128 characters or 128 bytes of binary data.

Arguments:

Argument 0
I/O Unit
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC*
SNAP-ENET-D64*
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

Argument 1
Index
Integer 32 Literal
Integer 32 Variable

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 2
From
String Literal
String Variable

Argument 3
Put Status in
Integer 32 Variable

Action Block Example:

Set I/O Unit Scratch Pad String Element		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-EB1*	*PAC_EB*
Index	*Integer 32 Literal*	*26*
From	*String Variable*	*MyStringVar*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

`SetIoUnitScratchPadStringElement``(`*I/O Unit, Index, From*`)`

```
Status = SetIoUnitScratchPadStringElement(PAC_EB, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad String Element" on page 375
"Set I/O Unit Scratch Pad String Table" on page 395
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

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 PAC R-series controller or SNAP PAC 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 SNAP PAC S-series or SoftPAC controller, create an I/O Unit of the type Generic OptoMMP Device with the loopback address (127.0.0.1).
- 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 *Length* (Argument 1), and the index number of the starting element in *To Index* (Argument 2).
- Each string element can hold 128 characters or 128 bytes of binary data.

Arguments:

Argument 0 **I/O Unit**	Argument 1 **Length**	Argument 2 **To Index**
G4EB2	Integer 32 Literal	Integer 32 Literal
Generic OptoMMP Device	Integer 32 Variable	Integer 32 Variable
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
SNAP-B3000-ENET, SNAP-ENET-RTC*		
SNAP-ENET-D64*		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		

* *Not intended for use with these brains*, because they don't have the integer scratch pad area.

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Argument 3 **From Index**	Argument 4 **From Table**	Argument 5 **Put Status in**
Integer 32 Literal	String Table	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Set I/O Unit Scratch Pad String Table		
Argument Name	Type	Name
I/O Unit	*GRV-EPIC-PR1*	*PR1*
Length	*Integer 32 Literal*	*8*
To Index	*Integer 32 Literal*	*0*
From Index	*Integer 32 Literal*	*0*
From Table	*String Table*	*MyStringTable*
Put Status in	*Integer 32 Variable*	*Status*

OptoScript Example:

SetIoUnitScratchPadStringTable(*I/O Unit, Length, To Index, From Index, From Table***)**

```
Status = SetIoUnitScratchPadStringTable(PR1, 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 PAC R-series 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 the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length. *Length* (Argument 2) is less than 0 or greater than 63.

-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 connection may have been closed by a previous command that failed. Check status codes returned on other connection commands.

-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 communication to the unit automatically. Reenable communication to the unit and try again.

See Also:

"Get I/O Unit Scratch Pad String Table" on page 377
"Set I/O Unit Scratch Pad String Element" on page 393
"Set I/O Unit Scratch Pad Float Element" on page 381
"Set I/O Unit Scratch Pad Float Table" on page 383
"Set I/O Unit Scratch Pad Integer 32 Element" on page 385
"Set I/O Unit Scratch Pad Integer 32 Table" on page 387
"Set I/O Unit Scratch Pad Bits from MOMO Mask" on page 379

15: Logical Commands

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:

- Performs a logical AND on the value of Argument 0 and the value of *With* (Argument 1), and then stores either a non-zero value (meaning True) or a 0 (meaning False) in *Put Result in* (Argument 2).
  - If neither Argument 0 nor *With* are 0 (zero), *Put Result in* will be a non-zero value (meaning True).
  - Otherwise, the result will be 0 (zero, meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Argument 0	With	Put Result in
1	1	1
67	-32	1
0	60	0
46	0	0
0	0	0

Arguments:

Argument 0
[Value]
Digital Input
Digital Output
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 1
With
Digital Input
Digital Output
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
Put Result in
Digital Output
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

AND		
Argument Name	Type	Name
(none)	*Digital Input*	*Limit_Switch1*
With	*Digital Input*	*Limit_Switch2*
Put Result in	*Integer Variable*	*Both_Switches_Closed*

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:

- The example shown is only one of many ways to use the `and` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 PAC Control 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" on page 423
"Bit AND" on page 401
"AND?" on page 399

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 the value of *Is* (Argument 0) and the value of Argument 1.
  - If neither *Is* nor Argument 1 are 0 (zero), the logic will take the True path.
  - Otherwise, the logic will take the False path.

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Is	Argument 1	Result
1	1	True
67	-32	True
0	60	False
46	0	False
0	0	False

Arguments:

Argument 0 Is	Argument 1 [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

Condition Block Example:

AND?

Argument Name	Type	Name
Is	*Digital Input*	*Limit_Switch1*
(none)	*Digital Input*	*Limit_Switch2*

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:

- The example shown is only one of many ways to use the `and` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 PAC Control 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?" on page 403
"Variable True?" on page 484
"Variable False?" on page 483

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. Bits set to 0 (zero) will clear.

Details:

- Performs a bitwise AND on the value of Argument 0 and *With* (Argument 1), and then stores the result in *Put Result in* (Argument 2).
- Note that *With* is the mask for selecting specific bits in Argument 0.
- Acts on all bits.

Examples:

Argument 0	*With*	*Put Result in*
0	0	0
8	0	0
0	8	0
8	8	8

Arguments:

Argument 0
[Value]
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 1
With
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
Put Result in
Digital Output
Integer 32 Variable
Integer 64 Variable

Action Block Example: This example copies the four least significant bits from VALUE to RESULT and sets all remaining bits in RESULT to zero.

Bit AND		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*VALUE*
With	*Integer 32 Literal*	*15*
Put Result in	*Integer 32 Variable*	*RESULT*

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 then 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 then 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:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- To clear bits in Argument 0, set a zero for each bit to clear in the mask (all remaining bits must be 1), and then make the value of Argument 0 and *Put Result in* (Argument 2) the same.
- You may prefer to set a 1 for each bit to clear in the mask, and 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" on page 405
"Bit NOT" on page 408
"AND" on page 397
"AND?" on page 399
"Bit AND?" on page 403

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 *Is* (Argument 0) and Argument 1 (the mask).
  - If any bit set to 1 in Argument 1 is also set to 1 in *Is*, the logic will take the True path.
  - If any bit set to 1 in Argument 1 is not set to 1 in *Is*, the logic will take the False path.
- Acts on all bits.

Examples:

Is	Argument 1	Result
1	1	True
0	0	False
1	0	False
0	1	False

Arguments:

Argument 0 Is	Argument 1 [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

Condition Block Example: This example performs a bitwise AND? on the variable nMask with the constant 33,280 (1000 0010 0000 0000 binary). In this example, if either bit 15 or 9 is on, the logic takes the True path. Otherwise, the logic takes the False path.

Bit AND?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*nMask*
(none)	*Integer 32 Literal*	*33280*

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

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- Use 255 as the constant to check the lower eight points.

See Also:

"AND?" on page 399
"Bit OR?" on page 418

Bit Change

Logical Action

Function: Change a bit based on a true/false control variable.

Typical Use: Useful for selection and deselection logic.

Arguments:

Argument 0	Argument 1	Argument 2
Set flag	**Bit to Change**	**Output**
Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
Integer 32 Literal	Integer 32 Literal	Integer 64 Variable

Action Block Example:

Bit Change		
Argument Name	Type	Name
Set flag	*Integer 32 Variable*	*SET_FLAG*
Bit to Change	*Integer 32 Variable*	*Bit*
Output	*Integer 32 Variable*	*Output_Value*

OptoScript Example:

`BitChange` **(***Set flag, Bit to Change, Output***)**

```
BitChange(SET_FLAG, Bit, Output_Value);
```

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

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 the value of Argument 0, and then moves the copy to *Put Result in* (Argument 2).
- 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 *Bit to Clear* (Argument 1) are 32-bit integers, because an integer 32 provides enough range to handle either a 32-bit or a 64-bit number.

Arguments:

Argument 0	Argument 1	Argument 2
[Value]	**Bit to Clear**	**Put Result in**
Integer 32 Variable	Integer 32 Literal	Integer 32 Variable
Integer 64 Variable	Integer 32 Variable	Integer 64 Variable

Action Block Example: This example does a binary read of the variable nMask, clears bit 0, and does a binary write of the data back out to nMask.

Bit Clear

Argument Name	Type	Name
	Integer 32 Variable	*nMask*
Bit to Clear	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Variable*	*nMask*

OptoScript Example: **BitClear(***Argument 0, Bit to Clear***)**

```
nMask = BitClear(nMask, 0);
```

This is a function command; it returns the value with the specified bit cleared.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 0, make the value of Argument 0 and *Put Result in* (Argument 2) the same.
- To clear several bits at once, use Bit AND.

See Also:

"Bit AND" on page 401
"Bit Test" on page 423
"Bit Set" on page 420

Bit Copy

Logical Action

Function: Atomically copy a bit from one variable or table element into another variable or table element.

Typical Use: To copy flag or status bits from one variable or table to another (or from variable to table or table to variable).

Details:

- The *Source Index* (Argument 0) and *Destination Index* (Argument 3) are only valid for table types. They will be ignored for integer types.
- The arguments are:
  - *Source Index*: If a table, the source element number
  - *Source*: Table or variable whose bit is going to be copied FROM
  - *Bit to Read*: The source bit (bit that we're copying)
  - *Destination Index*: If a table, the destination element number
  - *Destination*: Table or variable whose bit is going to be copied TO
  - *Bit to Set*: The destination bit number (0 to 31 for Int32, 0 to 63 for Int64)
  - *Put Result in*: See possible Status Codes listed below.

Arguments:

Argument 0 **Source Index**	Argument 1 **Source**	Argument 2 **Bit to Read**	Argument 3 **Destination Index**
Integer 32 Literal	Float Variable	Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Float Table	Integer 32 Variable	Integer 32 Variable
	Integer 32 Variable		
	Integer 32 Table		
	Integer 64 Variable		
	Integer 64 Table		

Argument 4 **Destination**	Argument 5 **Bit to Set**	Argument 6 **Put Status in**
Float Variable	Integer 32 Literal	Integer 32 Variable
Float Table	Integer 32 Variable	
Integer 32 Variable		
Integer 32 Table		
Integer 64 Variable		
Integer 64 Table		

Action Block Example:

Bit Copy		
Argument Name	Type	Name
Source Index	*Integer 32 Literal*	*10*
Source	*Integer 32 Table*	*arrnSrcTable*
Bit to Read	*Integer 32 Literal*	*6*
Destination Index	*Integer 32 Literal*	*5*
Destination	*Integer 32 Table*	*arrnDstTable*
Bit to Set	*Integer 32 Literal*	*1*
Put Status in	*Integer 32 Variable*	*nStatus*

OptoScript Example: **BitCopy(** *Source Index, Source, Bit to Read, Destination Index, Destination, Bit to Set***)**

```
nStatus = BitCopy(10, arrnSrcTable, 6, 5, arrnDstTable, 1);
```

This is a function command; it atomically copies a bit from one variable or table element into another variable or table element and returns the status code for success (0) or one of the other status codes listed below. The returned value can be consumed by a variable (as shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

Status Codes: 0 = Success.

-29 = Wrong object type.

-42 = Invalid limit. Bit to Set is out of range.

Bit NOT

Logical Action

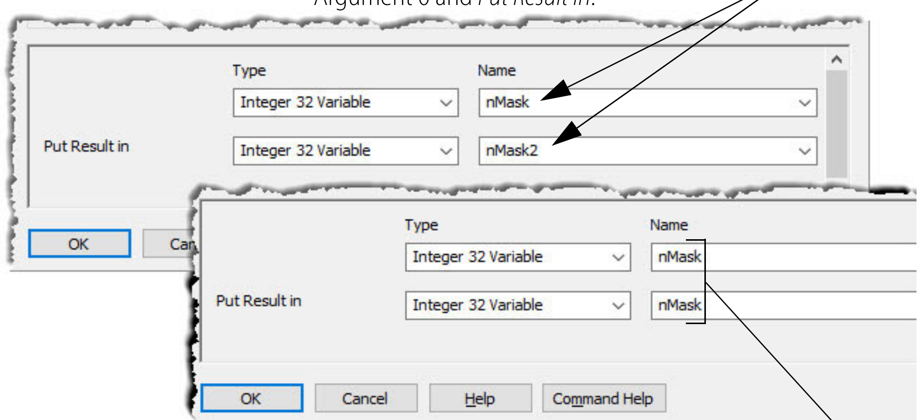
Function: To invert all bits of a 32-bit or 64-bit value.

Typical Use: To invert bits.

Details:

- Inverts the value of Argument 0. For example:
  `10100011101000111100101000001111`
  Inverted, its value would be:
  `01011100010111000011010111110000`
- Acts on all bits in the value.
- You can choose whether to leave the original value in Argument 0 and store the inverted value in *Put Result in* (Argument 1), or you can replace the original value with the inverted value.
  - To keep the original value in Argument 0, use the Name field to select *different* objects for Argument 0 and *Put Result in*.
  - To replace the original value with the inverted bits, select the same Name for Argument 0 and *Put Result in*.

To leave the original value unchanged, select different Names for Argument 0 and *Put Result in*.



To replace the original value with the inverted value, select the same Name for both arguments.

Arguments:

Argument 0 **[Value]**	**Argument 1** **Put Result in**
Integer 32 Literal	Digital Output
Integer 32 Variable	Integer 32 Variable
Integer 64 Literal	Integer 64 Variable
Integer 64 Variable	

Action Block Example:

Bit NOT		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*DATA*
Put Result in	*Integer 32 Variable*	*DATA*

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:

- 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.
- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

See Also:

"NOT" on page 455
"Bit XOR" on page 424
"XOR" on page 487
"Bit Set" on page 420
"Bit NOT?" on page 410

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:

- Evaluates the value of *Is* (Argument 0), and inverts its binary equivalent.
  - If any bit in *Is* is 0 (zero, meaning False or off), the logic will take the True path.
  - If any bit in *Is* is 1 (meaning True or on), the logic will take the False path.
- Acts on all bits.

Examples:

Data type and value of *Is*	Is, converted to binary	Inverted	Result1
decimal -1	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111	0	True
hex 14A	0001 0100 1010	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1110 1011 0101	True
decimal 0	0	1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111 1111	False

1Bits are numbered from right to left, starting at 0.

Arguments:

Argument 0
Is
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Condition Block Example: This example reads the bits of nMask, and then inverts them. If any bit is 0 (zero, off) the logic will take the True path. Otherwise, the logic will take the False path.

Bit NOT?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*nMask*

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

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- Use NOT if the objective is to toggle the value between True and False.

See Also:
"Bit On?" on page 415
"Bit Off?" on page 413

Bit Off in Numeric Table Element?

Logical Condition

Function: To test the False status of a specific bit in an allowable value in a numeric table.

Typical Use: To test a bit used as a flag in an integer element in a numeric table.

Details:

- Evaluates the specified bit (*Bit*, Argument 2) in the value located in the specified table (*Of Table*, Argument 1) and index (*At Index*, Argument 0).
  - If the bit is 0 (zero, meaning False or off), the logic will take the True path.
  - If the bit is 1 (meaning True or on), the logic will take the False path.
- Note that *Bit* is a 32-bit integer because the top of the valid range is 63.

Examples:

Data type and value in *Of Table[At Index]*		Value converted to binary	Bit	Result1
decimal	195	1100 0011	2	True
hex	20	0010 0000	2	True
decimal	114	0111 0010	1	False
hex	14A	0001 0100 1010	1	False

1Bits are numbered from right to left, starting at 0.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
At Index	**Of Table**	**Bit**
Integer 32 Literal	Integer 32 Table	Integer 32 Literal
Integer 32 Variable	Integer 64 Table	Integer 32 Variable

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

Bit Off In Numeric Table Element?

Argument Name	Type	Name
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Integer 32 Table*	*NUMERIC_TABLE_1*
Bit	*Integer 32 Literal*	*15*

OptoScript Example: **IsBitOffInNumTableElement(***At Index, Of Table, Bit***)**

```
if (IsBitOffInNumTableElement(TABLE_INDEX, NUMERIC_TABLE_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, and so forth. For more information, see “Logical Commands” and “Using OptoScript” in the *PAC Control User’s Guide* (form 1700).

See Also: “Bit On in Numeric Table Element?” on page 414

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 the bit specified in *Bit* (Argument 1) in the value of *In* (Argument 0).
  - If the bit in *In* is 0 (zero, meaning False or off), the logic will take the True path.
  - If the bit in *In* is 1 (meaning True or on), the logic will take the False path.
- Note that *Bit* is a 32-bit integer because the top of the valid range is 63.

Examples:

Data type and value of *In*		In, converted to binary	*Bit*	Result1
decimal	195	1100 0011	2	True
hex	20	0010 0000	2	True
decimal	114	0111 0010	1	False
hex	14A	0001 0100 1010	1	False

1Bits are numbered from right to left, starting at 0.

Arguments:

Argument 0	Argument 1
In	**Bit**
Integer 32 Variable	Integer 32 Literal
Integer 64 Variable	Integer 32 Variable

Condition Block Example: In this example, if bit 15 of *nMask* is off, the logic will take the True path. Otherwise, the logic will take the False path.

Bit Off

Argument Name	Type	Name
In	*Integer 32 Variable*	*nMask*
Bit	*Integer 32 Literal*	*15*

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

```
if (IsBitOff(nMask, 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, and so forth. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

Notes:

- 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?" on page 415
"Bit AND?" on page 403
"Bit Test" on page 423

Bit On in Numeric Table Element?

Logical Condition

Function: To test the True status of a specific bit in an allowable value in a numeric table.

Typical Use: To test a bit used as a flag in an integer element in a numeric table.

Details:

- Evaluates the specified bit (*Bit*, Argument 2) in the value located in the specified table (*Of Table*, Argument 1) and index (*At Index*, Argument 0).
  – If the bit is 1 (meaning True or on), the logic will take the True path.
  – If the bit is 0 (zero, meaning False or off), the logic will take the False path.
- Note that *Bit* is a 32-bit integer because the top of the valid range is 63.

Examples:

Data type and value in ***Of Table[At Index]***		Value converted to binary	***Bit***	Result1
decimal	114	0111 0010	1	True
hex	14A	0001 0100 1010	1	True
decimal	195	1100 0011	2	False
hex	20	0010 0000	2	False

1Bits are numbered from right to left, starting at 0.

Arguments:

Argument 0
At Index
Integer 32 Literal
Integer 32 Variable

Argument 1
Of Table
Integer 32 Table
Integer 64 Table

Argument 2
Bit
Integer 32 Literal
Integer 32 Variable

Condition Block Example: This example evaluates to True if point 0 of I/O UNIT 1 is on; otherwise, it's False.

Bit On In Numeric Table Element?

Argument Name	Type	Name
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Integer 32 Table*	*NUMERIC_TABLE_1*
Bit	*Integer 32 Literal*	*0*

OptoScript Example:

`IsBitOnInNumTableElement``(`*At Index, Of Table, Bit*`)`

```
if (IsBitOnInNumTableElement(TABLE_INDEX, NUMERIC_TABLE_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, and so forth. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

See Also: "Bit Off in Numeric Table Element?" on page 412

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 the bit specified in *Bit* (Argument 1) in the value of *In* (Argument 0).
  - If the bit in *In* is 1 (meaning True or on), the logic will take the True path.
  - If the bit in *In* is 0 (zero, meaning False or off), the logic will take the False path.
- Note that *Bit* is a 32-bit integer because the top of the valid range is 63.

Examples:

Data type and value of *In*		In, converted to binary	*Bit*	Result1
decimal	114	0111 0010	1	True
hex	14A	0001 0100 1010	1	True
decimal	195	1100 0011	2	False
hex	20	0010 0000	2	False

1Bits are numbered from right to left, starting at 0.

Arguments:

Argument 0	Argument 1
In	**Bit**
Integer 32 Variable	Integer 32 Literal
Integer 64 Variable	Integer 32 Variable

Condition Block Example: In this example, if bit 0 of *nMask* is on, the logic will take the True path. Otherwise, the logic will take the False path.

Bit On?

Argument Name	Type	Name
In	*Integer 32 Variable*	*nMask*
Bit	*Integer 32 Literal*	*0*

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

```
if (IsBitOn(nMask, 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, and so forth. For more information, see “Logical Commands” and “Using OptoScript” in the *PAC Control User's Guide* (form 1700).

Notes:

- 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?” on page 413
“Bit AND?” on page 403
“Bit Test” on page 423

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 the value of Argument 0 and *With* (Argument 1) (that is, it combines all bits set to 1 in Argument 0 and *With*), and stores the result in *Put Result in* (Argument 2). You can also choose to store the result in a different parameter.
- Acts on all bits.

Examples:

Argument 0	*With*	*Put Result in*
0	0	0
0xF	0	0xF
0	0xF	0xF
0xF	0xF	0xF

Arguments:

Argument 0
[Value]
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 1
With
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
Put Result in
Digital Output
Integer 32 Variable
Integer 64 Variable

Action Block Example: This example sets bit 2 in a copy of Argument 0 and stores the result in *Put Result in* (Argument 2).

Bit OR

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*VALUE*
With	*Integer 32 Literal*	*4*
Put Result in	*Integer 32 Variable*	*RESULT*

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 the value to the same I/O unit:

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

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 0, make the value of Argument 0 and *Put Result in* (Argument 2) 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" on page 420
"OR" on page 465
"Bit XOR" on page 424
"XOR" on page 487

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 *Is* (Argument 0) and the value of Argument 1.
  - If any bit is set to 1 in either *Is* or Argument 1, the logic will take the True path.
  - If neither argument has a bit set to 1, the logic will take the False path.
- Acts on all bits.

Examples:

Is	Argument 1	Result
1	0	True
0	1	True
1	1	True
0	0	False

Arguments:

Argument 0	Argument 1
Is	**[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

Condition Block Example:

Bit OR?		
Argument Name	Type	Name
Is	*Integer 32 Variable*	*Fault_Bits_1*
(none)	*Integer 32 Variable*	*Fault_Bits_2*

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:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 415
"Bit Off?" on page 413
"OR?" on page 467

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 *Count* (Argument 1) is positive, bits rotate left. If it is negative, bits rotate right. If it is zero, no rotation occurs.
- Note that the types for *Count* are 32-bit integers, because an integer 32 provides enough range to handle either a 32-bit or a 64-bit shift.

Arguments:

Argument 0 **[Value]**	Argument 1 **Count**	Argument 2 **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		

Action Block Example: 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).

Bit Rotate

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Mask_Variable*
Count	*Integer 32 Literal*	*4*
Put Result in	*Integer 32 Variable*	*Result_Variable*

OptoScript Example: **BitRotate(***Argument 0, 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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. In OptoScript code it cannot be consumed by an I/O unit, however. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

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:

- To rotate bits in Argument 0, make the value of Argument 0 and *Move To* (Argument 2) the same.
- To get rid of all bits that move past either end, use Bit Shift.

See Also: "Bit Shift" on page 421

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 the value of Argument 0, and then moves the copy to *Put Result in* (Argument 2).
- Note that the types for *Bit to Set* (Argument 1) are 32-bit integers, because an integer 32 provides enough range to handle either a 32-bit or a 64-bit number.

Arguments:

Argument 0	Argument 1	Argument 2
[Value]	**Bit to Set**	**Put Result in**
Integer 32 Variable	Integer 32 Literal	Integer 32 Variable
Integer 64 Variable	Integer 32 Variable	Integer 64 Variable

Action Block Example: In this example, 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).

Bit Set		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Pump3_Ctrl_Bits*
Bit to Set	*Integer 32 Literal*	*15*
Put Result in	*Integer 32 Variable*	*Pump3_Ctrl_Bits*

OptoScript Example: **BitSet(***Argument 0, 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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. It cannot be consumed by an I/O unit, however. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

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:
- 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 0, make the value of Argument 0 and *Put Result in* (Argument 2) the same.
- To set several bits at once, use Bit OR.

See Also:
"Bit OR" on page 416
"Bit Test" on page 423
"Bit Clear" on page 405

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* (Argument 1) 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 PAC Control command, if *Count* 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 *Count* are 32-bit integers, because an integer 32 provides enough range to handle either a 32-bit or a 64-bit shift.

Arguments:

Argument 0	Argument 1	Argument 2
[Value]	**Count**	**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		

Action Block Example: 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).

Bit Shift

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Mask_Variable*
Count	*Integer 32 Literal*	*-8*
Put Result in	*Integer 32 Variable*	*Result_Variable*

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:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- To shift bits in Argument 0, make the value of Argument 0 and *Put Result in* (Argument 2) the same.
- To retain all bits that move past either end, use Bit Rotate.

See Also: "Bit Rotate" on page 419

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:

- If the bit is clear (0), False (0) is moved to *Put Result in* (Argument 2).
- If the bit is set (1), True (non-zero) is moved to *Put Result in*.
- The result can also be sent directly to a digital output.
- Note that *Bit to Test* (Argument 2) is a 32-bit integer because the top of the valid range is 63.

Arguments:

Argument 0	Argument 1	Argument 2
[Value]	**Bit to Test**	**Put Result in**
Integer 32 Variable	Integer 32 Literal	Digital Output
Integer 64 Variable	Integer 32 Variable	Integer 32 Variable

Action Block Example:

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

Bit Test		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Pump3_Ctrl_Bits*
Bit to Test	*Integer 32 Literal*	*15*
Put Result in	*Integer 32 Variable*	*Pump3_Ctrl_Bits*

OptoScript Example:

`BitTest(`*Argument 0, 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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

Notes:

- 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" on page 405
"Bit Set" on page 420
"Bit On?" on page 415

Bit XOR

Logical Action

Function: To perform a bitwise EXCLUSIVE OR (XOR) 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 the value of Argument 0 and *With* (Argument 1), and then stores the result in *Put Result in* (Argument 2).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- Acts on all bits. One value is the mask for selecting specific bits in the other value.

Examples:

Argument 0	*With*	*Put Result in*
0	1	1
1	0	1
0	0	0
1	1	0

Arguments:

Argument 0
[Value]
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 1
With
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
Put Result in
Digital Output
Integer 32 Variable
Integer 64 Variable

Action Block Example: 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.

Bit XOR

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Data*
With	*Integer 32 Literal*	*22*
Put Result in	*Integer 32 Variable*	*Data_New*

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:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 0, make the value of Argument 0 and *Put Result in* (Argument 2) the same.
- To toggle a bit, Bit XOR with 1. (Zero leaves the bit unchanged.)

See Also:

"XOR" on page 487
"Bit NOT" on page 408
"NOT" on page 455
"Bit XOR?" on page 426

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 *Is* (Argument 0) and the value of Argument 1.
  - If *Is* and Argument 1 are not equal, the logic will take the True path.
  - If *Is* and the value of Argument 1 *are* equal, the logic will take the False path.
- Acts on all bits.
- When used with integer data types, is functionally equivalent to Not Equal?

Examples:

Is	Argument 1	Result
0	1	True
1	0	True
0	0	False
1	1	False

Arguments:

Argument 0
Is
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

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

Condition Block Example:

Bit XOR?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*nMask_1*
(none)	*Integer 32 Variable*	*nMask_2*

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

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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:

"Bit AND?" on page 403
"Bit NOT" on page 408

"Bit OR?" on page 418
"Bit XOR" on page 424
"Equal?" on page 430

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:

- If the value of *Is* (Argument 0) is the same as the value at the specified table (*Of Table*, Argument 2) and index (*At Index*, Argument 1), the logic will take the True path.
- If *Is* is not the same as the value at the specified table and index, the logic will take the False path.

Examples:

Is	Value at *Of Table[At Index]*	Result
0.0	0.0	True
-98.765	-98.765	True
-32768	-32768	True
2222	2222	True
0.0001	0.0	False

Arguments:

Argument 0	Argument 1	Argument 2
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		

Condition Block Example:

Equal to Numeric Table Element?		
Argument Name	Type	Name
Is	*Float Variable*	*THIS_READING*
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Float Table*	*TABLE_OF_READINGS*

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:

- In OptoScript code, the `==` operator has many uses. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 438
"Less Than or Equal to Numeric Table Element?" on page 447

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:

- If *Is* (Argument 0) and *To* (Argument 1) are equal, the logic will take the True path.
- If *Is* and *To* are not equal, the logic will take the False path.

Examples:

Is	*To*	Result
-1	-1	True
22.22	22.22	True
-1	1	False
22.22	22.221	False

Arguments:

Argument 0	Argument 1
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

Condition Block Example:

Equal?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*BATCH_STEP*
To	*Integer 32 Literal*	*4*

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

```
if (BATCH_STEP == 4) then
```

Notes:

- In OptoScript code, the `==` operator has many uses. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 Than Numeric Table Element?" on page 436
"Not Equal to Numeric Table Element?" on page 457
"Less Than or Equal to Numeric Table Element?" on page 447
"Greater Than or Equal?" on page 440
"Less?" on page 451
"Within Limits?" on page 485

Flip Flop JK

Logical Action

Function: Control the Output variable according to the JK Flip Flop rules.

Details: Implements a standard JK Flip Flop.

Output Rules:

J	K	Q
0	0	Hold (no change)
0	1	0
1	0	1
1	1	Toggle

Arguments:

Argument 0
Set [J]
Integer 32 Variable
Integer 32 Literal

Argument 1
Reset [K]
Integer 32 Variable
Integer 32 Literal

Argument 2
Output [Q]
Integer 32 Variable
Integer 32 Variable

Action Block Example:

Flip Flop JK

Argument Name	Type	Name
Set [J]	*Integer 32 Variable*	*J_Flag*
Reset [K]	*Integer 32 Variable*	*K_Flag*
Output [Q]	*Integer 32 Variable*	*Q_Flag*

OptoScript Example:

FlipFlopJK(*Set [J], Reset [K], Output [Q]***)**

```
FlipFlopJK(J_FLAG, K_FLAG, Q_FLAG);
```

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

Float to Int32 Bits

Logical Action

Function: To move the internal bit pattern of a float into an integer 32.

Typical Use: To help parse or create binary data when communicating with other devices.

Details: This command is similar to Move 32 Bits, but is more flexible in OptoScript blocks. It can be used inside another expression, which can reduce the need for temporary variables compared to Move 32 Bits.

Arguments:

Argument 0	Argument 1
From	**To**
Float Literal	Integer 32 Variable
Float Variable	

Action Block Example:

Float to Int32 Bits		
Argument Name	Type	Name
From	*Float Variable*	*nFloat*
To	*Integer 32 Variable*	*nInt32*

OptoScript Example:

FloatToInt32Bits(*From*)

```
nInt32 = FloatToInt32Bits(nFloat);
```

This is a function command; it returns an integer 32 value. The returned value can be consumed by a variable (as in the example shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 0	Argument 1
High Bits From	**Put in**
Integer 64 Variable	Integer 32 Variable

Action Block Example:

Get High Bits of Integer 64		
Argument Name	Type	Name
High Bits From	*Integer 64 Variable*	*INPUT_BOARD_2*
Put in	*Integer 32 Variable*	*IN_BD2_HIGH*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

See Also:

"Get Low Bits of Integer 64" on page 435
"Make Integer 64" on page 453

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 0	Argument 1
Low Bits From	**Put in**
Integer 64 Variable	Integer 32 Variable

Action Block Example:

Get Low Bits of Integer 64		
Argument Name	Type	Name
Low Bits From	*Integer 64 Variable*	*INPUT_BOARD_2*
Put in	*Integer 32 Variable*	*IN_BD2_LOW*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also:

"Get High Bits of Integer 64" on page 434
"Make Integer 64" on page 453

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:
- If the value of *Is* (Argument 0) is greater than the value at the specified table (*Of Table*, Argument 2) and index (*At Index*, Argument 1), the logic will take the True path.
- If the value of *Is* is not greater than the value at the specified table and index, the logic will take the False path.

Examples:

Is	Value at Of Table[At Index]	Result
0.0001	0.0	True
1	0	True
22221	2222	True
0.0	0.0	False
-98.765	-98.765	False

Arguments:

Argument 0	Argument 1	Argument 2
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		

Condition Block Example:

Greater Than Numeric Table Element?

Argument Name	Type	Name
Is	*Float Variable*	*THIS_READING*
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Float Table*	*TABLE_OF_READINGS*

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:
- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 445
"Not Equal to Numeric Table Element?" on page 457
"Greater Than or Equal To Numeric Table Element?" on page 438
"Less Than or Equal to Numeric Table Element?" on page 447

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:
- If the value of *Is* (Argument 0) is greater than or equal to the value at the specified table (*Of Table,* Argument 2) and index (*At Index*, Argument 1), the logic will take the True path.
- If the value of *Is* is not greater than or equal to the value at the specified table and index, the logic will take the False path.

Examples:

Is	**Value at *Of Table[At Index]***	**Result**
0.0	0.0	True
0.0001	0.0	True
22221	2222	True
22.22	22.222	False
-32768	-32767	False

Arguments:

Argument 0
Is
Analog Input
Analog Output
Digital Input
Digital Output
Down Timer Variable
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable
Up Timer Variable

Argument 1
At Index
Integer 32 Literal
Integer 32 Variable

Argument 2
Of Table
Float Table
Integer 32 Table
Integer 64 Table

Condition Block Example:

Greater Than or Equal to Numeric Table Element?

Argument Name	Type	Name
Is	*Float Variable*	*THIS_READING*
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Float Table*	*TABLE_OF_READINGS*

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:
- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- To test for less than, use either Less Than Numeric Table Element? or the False exit.

Queue Errors: -12 = Invalid table index.

See Also: "Less Than Numeric Table Element?" on page 445
"Not Equal to Numeric Table Element?" on page 457
"Greater Than Numeric Table Element?" on page 436
"Less Than or Equal to Numeric Table Element?" on page 447

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:

- If *Is* (Argument 0) is greater than or equal to *To* (Argument 1), the logic will take the True path.
- If *Is* (Argument 0) is *not* greater than or equal to *To* (Argument 1), the logic will take the False path.

Examples:

Is	*To*	Result
0	0	True
1	0	True
22221	2222	True
-32768	-32767	False

Arguments:

Argument 0	Argument 1
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

Condition Block Example:

Greater Than or Equal?

Argument Name	Type	Name
Is	*Analog Input*	*ROOM_TEMP*
To	*Float Literal*	*78.5000*

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

```
if (ROOM_TEMP >= 78.5000) then
```

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 451
"Less Than or Equal to Numeric Table Element?" on page 447

"Not Equal to Numeric Table Element?" on page 457
"Within Limits?" on page 485

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:

- If *Is* (Argument 0) is greater than *Than* (Argument 1), the logic will take the True path.
- If *Is* is *not* greater than *Than*, the logic will take the False path.

Examples:

Is	*Than*	Result
-1	-3	True
22.221	22.220	True
0	0	False
-1	0	False

Arguments:

Argument 0 **Is**	Argument 1 **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

Condition Block Example:

Greater?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*CALCULATED_VALUE*
Than	*Integer 32 Literal*	*1000*

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

```
if (CALCULATED_VALUE > 1000) then
```

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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?" on page 451
"Not Equal to Numeric Table Element?" on page 457
"Greater Than or Equal?" on page 440

"Less Than or Equal to Numeric Table Element?" on page 447
"Within Limits?" on page 485

Int32 to Float Bits

Logical Action

Function: To move the internal bit pattern of an integer 32 into a float.

Typical Use: To help parse or create binary data when communicating with other devices.

Details: This command is similar to Move 32 Bits, but is more flexible in OptoScript blocks. It can be used inside another expression, which can reduce the need for temporary variables compared to Move 32 Bits.

Arguments:

Argument 0	Argument 1
From	**To**
Integer 32 Literal	Float Variable
Integer 32 Variable	

Action Block Example:

Int32 to Float Bits		
Argument Name	Type	Name
From	*Integer 32 Variable*	*nInt32*
To	*Float Variable*	*nFloat*

OptoScript Example:

Int32ToFloatBits(*From*)

```
nFloat = Int32ToFloatBits(nInt32);
```

This is a function command; it returns a float value. The returned value can be consumed by a variable (as in the example shown) or used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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:

- If the value of *Is* (Argument 0) is less than the value at the specified table (*Of Table*, Argument 2) and index (*At Index*, Argument 1), the logic will take the True path.
- If *Is* is greater than or equal to the value at the specified table and index, the logic will take the False path.

Examples:

Is	**Value at *Of Table[At Index]***	**Result**
-98.766	-98.765	True
-32768	-32767	True
0.0	0.0	False
0.0001	0.0	False
22221	2222	False

Arguments:

Argument 0 **Is**	Argument 1 **At Index**	Argument 2 **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		

Condition Block Example:

Less Than Numeric Table Element?

Argument Name	Type	Name
Is	*Float Variable*	*THIS_READING*
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Float Table*	*TABLE_OF_READINGS*

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:

- The example shown is only one of many ways to use the `<` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- To test for greater than or equal to, use either Greater Than or Equal To Numeric 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?" on page 438

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:

- If the value of *Is* (Argument 0) is less than or equal to the value at the specified table (*Of Table*, Argument 2) and index (*At Index*, Argument 1), the logic will take the True path.
- If *Is* is greater than the value in the specified table and index, the logic will take the False path.

Examples:

Is	Value at *Of Table[At Index]*	Result
0.0	0.0	True
22.22	22.222	True
-32768	-32767	True
0.0001	0.0	False
22221	2222	False

Arguments:

Argument 0 Is	Argument 1 At Index	Argument 2 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		

Condition Block Example:

Less Than or Equal to Numeric Table Element?		
Argument Name	Type	Name
Is	*Float Variable*	*THIS_READING*
At Index	*Integer 32 Variable*	*TABLE_INDEX*
Of Table	*Float Table*	*TABLE_OF_READINGS*

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:

- The example shown is only one of many ways to use the `<=` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- To test for greater than, use either Greater Than Numeric 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?" on page 436
"Not Equal to Numeric Table Element?" on page 457
"Equal to Numeric Table Element?" on page 428
"Greater Than or Equal To Numeric Table Element?" on page 438

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:

- If the value of *Is* (Argument 0) is less than or equal to the value of *To* (Argument 1), the logic will take the True path.
- If the value of *Is* is greater than *To*, the logic will take the False path.

Examples:

Is	*To*	Result
0	0	True
-1	0	True
-1	-3	False
22.221	22.220	False

Arguments:

Argument 0	Argument 1
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

Condition Block Example:

Less Than or Equal?		
Argument Name	Type	Name
Is	*Float Variable*	*TEMPERATURE*
To	*Float Literal*	*98.60*

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

```
if (TEMPERATURE <= 98.60) then
```

Notes:

- The example shown is only one of many ways to use the `<=` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 Than Numeric Table Element?" on page 436
"Not Equal to Numeric Table Element?" on page 457
"Greater Than or Equal?" on page 440

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:

- If *Is* (Argument 0) is less than *Than* (Argument 1), the logic will take the True path.
- If *Is* is greater than or equal to *Than*, the logic will take the False path.

Examples:

Is	*Than*	Result
-1	0	True
0	0	False
-1	-3	False
22.221	22.220	False

Arguments:

Argument 0 **Is**	Argument 1 **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

Condition Block Example:

Less		
Argument Name	Type	Name
Is	*Analog Input*	*TANK_LEVEL*
Than	*Float Variable*	*FILL_SETPOINT*

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

```
if (TANK_LEVEL < FILL_SETPOINT) then
```

Notes:

- The example shown is only one of many ways to use the `<` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 Than Numeric Table Element?" on page 436
"Not Equal to Numeric Table Element?" on page 457

"Equal?" on page 430
"Greater Than or Equal?" on page 440

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 *Put in* (Argument 2) is an I/O unit.

Arguments:

Argument 0	Argument 1	Argument 2
High Integer	**Low Integer**	**Put in**
Integer 32 Literal	Integer 32 Literal	Integer 64 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Make Integer 64		
Argument Name	Type	Name
High Integer	*Integer 32 Variable*	*IN_BD2_HIGH*
Low Integer	*Integer 32 Variable*	*IN_BD2_LOW*
Put in	*Integer 64 Variable*	*IN_BD2_STATUS*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. It cannot be consumed by an I/O unit, however. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

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 to get information from a program that doesn't directly support 64-bit integers, such as PAC Display and third-party products.

See Also:

"Get High Bits of Integer 64" on page 434
"Get Low Bits of Integer 64" on page 435

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 0 **From**	Argument 1 **To**
Float Literal	Float Variable
Float Variable	Integer 32 Variable
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Move 32 Bits		
Argument Name	Type	Name
From	*Integer 32 Variable*	*Source_Data*
To	*Float Variable*	*Float*

OptoScript Example:

`Move32Bits(`*From, To***`)`**

```
Move32Bits(Source_Data, Float);
```

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

Notes: For more information, see “Logical Commands” and “Using OptoScript” in the *PAC Control User’s Guide* (form 1700).

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 the value of Argument 0 and stores the result in *Put Result in* (Argument 1).
- If Argument 0 is False (0, off), the result will be a non-zero value (meaning True).
- If Argument 0 is True (non-zero, on), the result will be 0 (zero, meaning False).

NOTE: In programming logic, 0 represents False and any non-zero number represents True.

Examples:

Argument 0	*Put Result in*
0	1
-1	0
22	0

Arguments:

Argument 0	Argument 1
[Value]	**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	

Action Block Example:

NOT		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Current_State*
Put Result in	*Digital Output*	*DOUT1*

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

```
DOUT1 = not Current_State;
```

Notes:
- The example shown is only one of many ways to use the `not` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 0, make the value of Argument 0 and *Put Result in* (Argument 1) the same.
- To toggle all 32 or 64 bits of an integer, use Bit NOT.

See Also:

"Bit NOT" on page 408
"Test Greater" on page 472
"Test Less" on page 476
"Test Within Limits" on page 482

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:

- If the value of *Is* (Argument 0) is different than the value at the specified table (*Of Table*, Argument 2) and index (*At Index*, Argument 1), the logic will take the True path.
- If the value of *Is* is the same as the value at the specified table and index, the logic will take the False path.

Examples:

Is	Value at *Of Table[At Index]*	Result
0.0001	0.0	True
0.0	0.0	False
-98.765	-98.765	False
-32768	-32768	False

Arguments:

Argument 0
Is
Analog Input
Analog Output
Digital Input
Digital Output
Down Timer Variable
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable
Up Timer Variable

Argument 1
At Index
Integer 32 Literal
Integer 32 Variable

Argument 2
Of Table
Float Table
Integer 32 Table
Integer 64 Table

Condition Block Example:

Not Equal to Numeric Table Element?		
Argument Name	Type	Name
Is	*Float Variable*	*This_Reading*
At Index	*Integer 32 Variable*	*Table_Index*
Of Table	*Float Table*	*Table_of_Readings*

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:

- In OptoScript code, the `<>` operator can be used in several ways. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- To test for equality, use either Equal to Numeric 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?" on page 428
"Greater Than Numeric Table Element?" on page 436
"Greater Than or Equal To Numeric Table Element?" on page 438
"Less Than Numeric Table Element?" on page 445
"Less Than or Equal to Numeric Table Element?" on page 447

Not Equal?

Logical Condition

Function: To determine if two values are different.

Typical Use: To perform reverse logic.

Details:

- If *Is* (Argument 0) and *To* (Argument 1) are not the same, the logic will take the True path.
- If *Is* and *To* are the same, the logic will take the False path.

Examples:

Is	*To*	Result
0	65280	True
65280	22.2	True
0	0	False
22.22	22.22	False

Arguments:

Argument 0	**Argument 1**
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

Condition Block Example:

Not Equal?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*BATCH_STEP*
To	*Integer 32 Literal*	*4*

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

```
if (BATCH_STEP <> 4) then
```

Notes:

- In OptoScript code, the `<>` operator can be used in several ways. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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 Than Numeric Table Element?" on page 436
"Less?" on page 451
"Less Than or Equal to Numeric Table Element?" on page 447

"Greater Than or Equal?" on page 440
"Equal?" on page 430
"Within Limits?" on page 485

NOT?

Logical Condition

Function: To determine if a value is False (zero, off).

Typical Use: To perform False testing.

Details:

- If *Is* (Argument 0) is False (0, off), the logic will take the True path.
- If *Is* is True (non-zero, on), the logic will take the False path.

  NOTE: In programming logic, 0 (zero) represents False and any non-zero number represents True.
- Functionally equivalent to Variable False?

Examples:

Is	Result
0	True
-1	False
22	False

Arguments:

Argument 0
Is
Digital Input
Digital Output
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Condition Block Example:

NOT?		
Argument Name	Type	Name
Is	*Integer 32 Variable*	*CURRENT_STATE*

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

```
if (not Current_State) then
```

Notes:

- 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.
- The example shown is only one of many ways to use the `not` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

See Also:

"AND?" on page 399
"OR?" on page 467
"XOR?" on page 489
"Variable True?" on page 484
"Within Limits?" on page 485
"Greater Than Numeric Table Element?" on page 436
"Less?" on page 451

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 0	Argument 1	Argument 2
Element Index	**Of Integer Table**	**Bit To Clear**
Integer 32 Literal	Integer 32 Table	Integer 32 Literal
Integer 32 Variable	Integer 64 Table	Integer 32 Variable

Action Block Example:

Numeric Table Element Bit Clear		
Argument Name	Type	Name
Element Index	*Integer 32 Literal*	*4*
Of Integer Table	*Integer 32 Table*	*PUMP_CTRL_BITS*
Bit To Clear	*Integer 32 Literal*	*15*

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" on page 405
"Numeric Table Element Bit Set" on page 463
"Numeric Table Element Bit Test" on page 464

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 0	Argument 1	Argument 2
Element Index	**Of Integer Table**	**Bit to Set**
Integer 32 Literal	Integer 32 Table	Integer 32 Literal
Integer 32 Variable	Integer 64 Table	Integer 32 Variable

Action Block Example:

Numeric Table Element Bit Set		
Argument Name	Type	Name
Element Index	*Integer 32 Literal*	*4*
Of Integer Table	*Integer 32 Table*	*PUMP_CTRL_BITS*
Bit to Set	*Integer 32 Literal*	*15*

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" on page 420
"Numeric Table Element Bit Clear" on page 462
"Numeric Table Element Bit Test" on page 464

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 0	Argument 1	Argument 2	Argument 3
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

Action Block Example:

Numeric Table Element Bit Test		
Argument Name	Type	Name
Element Index	*Integer 32 Literal*	*4*
Of Integer Table	*Integer 32 Table*	*Pump_Ctrl_Bits*
Bit to Test	*Integer 32 Literal*	*15*
Put Result in	*Integer 32 Variable*	*Result*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

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" on page 463
"Numeric Table Element Bit Clear" on page 462

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 the value of Argument 0 and *With* (Argument 1), and then stores the result in *Put Result in* (Argument 2).
  - If *both* Argument 0 and *With* are True (non-zero, on), the result will be a non-zero value (meaning True).
  - If Argument 0 is True but *With* is False (0, off), the result will be a non-zero value (meaning True).
  - If Argument 0 is False but *With* is True, the result will be a non-zero value (meaning True).
  - If *both* Argument 0 and *With* are False (0, off), the result will be 0 (meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Argument 0	*With*	*Put Result in*
-1	-1	1
-1	0	1
0	-1	1
0	0	0

Arguments:

Argument 0	**Argument 1**	**Argument 2**
[Value]	**With**	**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	

Action Block Example:

OR		
Argument Name	Type	Name
(none)	*Digital Input*	*LIMIT_SWITCH1*
With	*Digital Output*	*LIMIT_SWITCH2*
Put Result in	*Digital Output*	*MOTOR1_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:

- The example shown is only one of many ways to use the `or` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- You should 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 PAC Control 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" on page 416

OR?

Logical Condition

Function: To determine if either or both of two values are True (non-zero, on).

Typical Use: To OR two values within an AND? type condition block.

Details:

- If *both* arguments are True (non-zero, on), the logic will take the True path.
- If *either* argument is True (non-zero, on), the logic will take the True path.
- If *both* arguments are False (0, off), the logic will take the False path.

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.

Examples:

Is	Argument 1	Result
-1	-1	True
-1	0	True
0	-1	True
0	0	False

Arguments:

Argument 0 **Is**	Argument 1 **[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

Condition Block Example:

OR?

Argument Name	Type	Name
Is	*Digital Input*	*LIMIT_SWITCH1*
(none)	*Digital Input*	*LIMIT_SWITCH2*

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:

- The example shown is only one of many ways to use the `or` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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" on page 455
"AND?" on page 399
"XOR?" on page 489

Set Variable False

Logical Action

Function: To move a 0 (zero, meaning False) value into a variable.

NOTE: In programming logic, 0 represents False and any non-zero number represents True.

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 0
[Value]
Float Variable
Integer 32 Variable

Action Block Example:

Set Variable False		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Flag_Hopper_Full*

OptoScript Example:

SetVariableFalse(*Argument 0*)

```
SetVariableFalse(Flag_Hopper_Full);
```

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

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- *Speed Tip:* This command is faster than Move for moving a zero to a variable.

See Also: "Set Variable True" on page 469

Set Variable True

Logical Action

Function: To move a 1 (one, meaning True) value into a variable.

NOTE: In programming logic, 0 represents False and any non-zero number represents True.

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 0
[Value]
Float Variable
Integer 32 Variable

Action Block Example:

Set Variable True		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*FLAG_JOB_DONE*

OptoScript Example:

SetVariableTrue(*Argument 0*)

```
SetVariableTrue(FLAG_JOB_DONE);
```

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

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- *Speed Tip:* This command is faster than Move for moving a +1 value to a variable.

See Also: "Set Variable False" on page 468

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 the value of Argument 0 is equal to the value of *With* (Argument 1), and stores the result in *Put Result in* (Argument 2).
  - If the values *are* the same, the result will be a non-zero value (meaning True).
  - If the values *are not* the same, the result will be 0 (meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Argument 0	*With*	*Put Result in*
0	0	1
22.22	22.22	1
-1	0	0
255	65280	0

Arguments:

Argument 0	Argument 1	Argument 2
[Value]	**With**	**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	

Action Block Example:

Test Equal

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*TOP_LEVEL*
With	*Integer 32 Literal*	*1000*
Put Result in	*Integer 32 Variable*	*FLAG_AT_THE_TOP*

OptoScript Example: For an OptoScript equivalent, see the Equal? command.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

- 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" on page 472
"Test Less" on page 476
"Test Greater or Equal" on page 474
"Test Less or Equal" on page 478
"Test Not Equal" on page 480

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 *Is* (Argument 0) is greater than *Greater Than* (Argument 1), and stores the result in *Put Result in* (Argument 2).
  - If *Is* is greater than *Greater Than*, the result will be a non-zero value (meaning True).
  - If *Is* is less than or equal to *Greater Than*, the result will be 0 (zero, meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Is	***Greater than***	***Put Result In***
-1	-3	1
22.221	22.220	1
0	0	0
-1	0	0

Arguments:

Argument 0	Argument 1	Argument 2
Is	**Greater than**	**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	

Action Block Example:

Test Greater		
Argument Name	Type	Name
Is	*Analog Input*	*TEMP*
Greater than	*Integer 32 Literal*	*1000*
Put Result in	*Integer 32 Variable*	*TEMP_COMPARISON*

OptoScript Example: For an OptoScript equivalent, see the Greater? command.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

- Consider using Test Greater or Equal instead.

See Also:

"Test Equal" on page 470
"Test Less" on page 476
"Test Greater or Equal" on page 474
"Test Less or Equal" on page 478
"Test Not Equal" on page 480

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 *Is* (Argument 0) is greater than or equal to the value of *> or =* (Argument 1), and stores the result in *Put Result in* (Argument 2).
  - If *Is* is greater than or equal to *> or =*, the result will be a non-zero value (meaning True).
  - If *Is* is less than *> or =*, the result will be 0 (zero, meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Is	*> or =*	*Put Result in*
-1	-3	1
22.221	22.220	1
0	0	0
-1	0	0

Arguments:

Argument 0	Argument 1	Argument 2
Is	**> or =**	**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	

Action Block Example:

Test Greater or Equal

Argument Name	Type	Name
Is	*Analog Input*	*ROOM_TEMP*
> or =	*Float Literal*	*78.5000*
Put Result in	*Integer 32 Variable*	*FLAG_ROOM_TEMP_OK*

OptoScript Example: For an OptoScript equivalent, see the Greater Than or Equal? command.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

- 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" on page 470
"Test Less" on page 476
"Test Greater" on page 472
"Test Less or Equal" on page 478
"Test Not Equal" on page 480

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 *Is* (Argument 0) is less than *Less than* (Argument 1), and stores the result in *Put Result in* (Argument 2).
  - If *Is* is less than *Less than*, the result will be a non-zero value (meaning True).
  - If *Is* is greater than or equal to *Less than*, the result will be 0 (zero, meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Is	*Less than*	*Put Result in*
-1	0	1
0	0	0
22.221	22.220	0
-1	-3	0

Arguments:

Argument 0 **Is**	Argument 1 **Less than**	Argument 2 **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	

Action Block Example:

Test Less

Argument Name	Type	Name
Is	*Analog Input*	*TANK_LEVEL*
Less than	*Integer 32 Variable*	*FULL_TANK_LEVEL*
Put Result in	*Digital Output*	*FLAG_TANK_FILL_VALVE*

OptoScript Example: For an OptoScript equivalent, see the Less? command.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

- Consider using Test Less or Equal instead, since exact matches of non-integer types are rare.

See Also:

"Test Greater" on page 472
"Test Equal" on page 470
"Test Greater or Equal" on page 474
"Test Less or Equal" on page 478
"Test Not Equal" on page 480

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 *Is* (Argument 0) is less than or equal to the value of < *or* = (Argument 1), and stores the result in *Put Result in* (Argument 2).
  - If *Is* is less than or equal to the value of < *or* =, the result will be a non-zero value (meaning True).
  - If *Is* is greater than > *or* =, the result will be a 0 (meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Is	**< *or* =**	***Put Result in***
0	0	1
-1	0	1
22.221	22.220	0
-1	-3	0

Arguments:

Argument 0	**Argument 1**	**Argument 2**
Is	**< or =**	**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	

Action Block Example:

Test Less or Equal		
Argument Name	Type	Name
Is	*Float Variable*	*TEMPERATURE*
< or =	*Float Literal*	*98.6*
Put Result in	*Integer 32 Variable*	*FLAG_TEMP_OK*

OptoScript Example: For an OptoScript equivalent, see the Less Than or Equal? command.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

- 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" on page 472
"Test Less" on page 476
"Test Greater or Equal" on page 474
"Test Equal" on page 470
"Test Not Equal" on page 480

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 *Is* (Argument 0) is different than *Not Equal to* (Argument 1), and stores the result in *Put Result in* (Argument 2).
  – If *Is* is not the same as *Not Equal to*, the result will be a non-zero value (meaning True).
  – If *Is* and *Not Equal to* are the same, the result will be 0 (meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Is	*Not Equal to*	*Put Result in*
-1	0	1
255	65280	1
22.22	22.22	0
0	0	0

Arguments:

Argument 0 **Is**	Argument 1 **Not Equal to**	Argument 2 **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	

Action Block Example:

Test Not Equal		
Argument Name	Type	Name
Is	*Integer 32 Variable*	*COUNTER_VALUE*
Not Equal to	*Integer 32 Literal*	*100*
Put Result in	*Integer 32 Variable*	*FLAG_NOT_DONE*

OptoScript Example: For an OptoScript equivalent, see the Not Equal? command.

Notes:

- For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

- 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 Equal" on page 470

"Test Greater" on page 472

"Test Greater or Equal" on page 474

"Test Less" on page 476

"Test Less or Equal" on page 478

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 0	Argument 1	Argument 2	Argument 3
Is	**>=**	**And <=**	**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			

Action Block Example:

Test Within Limits		
Argument Name	Type	Name
Is	*Float Variable*	*CURRENT_TEMP*
>=	*Float Variable*	*COLDEST_TEMP*
And <=	*Float Variable*	*HOTTEST_TEMP*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example: For an OptoScript equivalent, see the Within Limits? command.

See Also:

"Test Greater" on page 472
"Test Less" on page 476
"Test Greater or Equal" on page 474
"Test Less or Equal" on page 478
"Test Equal" on page 470
"Test Not Equal" on page 480

Variable False?

Logical Condition

Function: To determine if the specified variable is 0 (zero, meaning False).

Typical Use: To determine if further processing should take place.

Details: If the value of *Is* (Argument 0) is 0 (zero, meaning False), the logic will take the True path. If the value of *Is* is *not* 0, the logic will take the False path.

NOTE: In programming logic, 0 represents False and any non-zero number represents True.

Arguments:

Argument 0
Is
Float Variable
Integer 32 Variable
Integer 64 Variable

Condition Block Example:

Variable False?		
Argument Name	Type	Name
Is	*Integer 32 Variable*	*Pressure_Difference*

OptoScript Example:

IsVariableFalse(*Is*)

```
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, and so forth. For more information, see “Logical Commands” and “Using OptoScript” in the *PAC Control User’s Guide* (form 1700).

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?” on page 484

Variable True?

Logical Condition

Function: To determine if the specified variable is non-zero (meaning True)

Typical Use: To determine if further processing should take place.

Details: If the value of the variable is a non-zero number, the logic will take the True path.
If the value of the variable is 0 (zero, meaning False), the logic will take the False path.

NOTE: In programming logic, 0 represents False and any non-zero number represents True.

Arguments:

Argument 0
Is
Float Variable
Integer 32 Variable
Integer 64 Variable

Condition Block Example:

Variable True?		
Argument Name	Type	Name
Is	*Integer 32 Variable*	*Pressure_Difference*

OptoScript Example:

IsVariableTrue(*Is*)

```
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, and so forth.

A shorter way to achieve the same result in OptoScript code is to use the following:

```
if (Pressure_Difference) then
```

For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

See Also: "Variable False?" on page 483

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:

- If *Is* (Argument 0) is greater than or equal to > *or* = (Argument 1) **and** less than or equal to *And* <= (Argument 2), the logic will take the True path.
- If *Is* is less than > *or* =, the logic will take the False path.
- If *Is* is greater than *And* <=, the logic will take the False path.

Examples:

Is	>=	*And* <=	Result
0.0	0.0	100.0	True
-1.0	-45.0	45.0	True
-32768	0.0	100.0	False
75.1	68.0	72.0	False

Arguments:

Argument 0 **Is**	Argument 1 **> =**	Argument 2 **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		

Condition Block Example: This example evaluates True if *Current_Temp* is greater than or equal to *Coldest_Temp* and less than or equal to *Hottest_Temp*. It evaluates False, otherwise.

Within Limits?		
Argument Name	Type	Name
Is	*Float Variable*	*Current_Temp*
>=	*Float Variable*	*Coldest_Temp*
And <=	*Float Variable*	*Hottest_Temp*

OptoScript Example: **IsWithinLimits(*Is*, >=, *And* <=)**

```
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, and so forth. For more information, see “Logical Commands” and “Using OptoScript” in the *PAC Control User's Guide* (form 1700).

Notes:

- Use to replace two conditions: Less Than or Equal? and Greater Than or Equal?

See Also: "Less Than or Equal to Numeric Table Element?" on page 447
"Greater Than or Equal?" on page 440

XOR

Logical Action

Function: To perform a logical EXCLUSIVE OR (XOR) 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 the value of Argument 0 and *With* (Argument 1), and stores the result in *Put Result in* (Argument 2).
  - If Argument 0 is True (non-zero, on) but *With* is False (0), the result will be a non-zero value (meaning True).
  - If Argument 0 is False but *With* is True, the result will be a non-zero value (meaning True).
  - If both Argument 0 and *With* are True (non-zero, on), the result will be 0 (meaning False).
  - If both Argument 0 and *With* are False (0, off), the result will be 0 (meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- The result can be sent directly to a digital output if desired.

Examples:

Argument 0	*With*	*Put Result in*
1	0	1
22	0	1
-1	0	1
0	1	1
0	-1	1
1	1	0
-1	-1	0
22	-4	0
0	0	0

Arguments:

Argument 0
[Value]
Digital Input
Digital Output
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 1
With
Digital Input
Digital Output
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable
Integer 64 Literal
Integer 64 Variable

Argument 2
Put Result in
Digital Output
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

In this example, if SUPPLY_FAN is on, it will turn off and vice versa.

XOR		
Argument Name	Type	Name
(none)	*Digital Output*	*SUPPLY_FAN*
With	*Integer 32 Literal*	*1*
Put Result in	*Digital Output*	*SUPPLY_FAN*

OptoScript Example:

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

```
SUPPLY_FAN = SUPPLY_FAN xor 1;
```

Notes:

- The example shown is only one of many ways to use the `xor` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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" on page 424
"Not Equal to Numeric Table Element?" on page 457
"Turn On" on page 206
"Turn Off" on page 205
"On?" on page 194
"Off?" on page 192

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 0 and Argument 1 have different True/False states.
  - If one argument is True (non-zero, on) and the other is False (0), the logic will take the True path.
  - If both arguments are True (non-zero, on), the logic will take the False path.
  - If both arguments are False (0, off), the logic will take the False path.

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- Functionally equivalent to the Not Equal? condition when using allowable values.

Examples:

Is (Argument 0)	*Is* (Argument 1)	Result
1	0	True
22	0	True
-1	0	True
0	1	True
0	-1	True
1	1	False
-1	-1	False
22	-4	False
0	0	False

Arguments:

Argument 0	Argument 1
Is	**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

Condition Block Example:

XOR?

Argument Name	Type	Name
Is	*Integer 32 Variable*	*Limit_Switch1_Prev*
Is	*Digital Input*	*Limit_Switch1*

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:

- The example shown is only one of many ways to use the `xor` operator. For more information, see "Logical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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" on page 455
"AND?" on page 399
"OR?" on page 467

16: Mathematical Commands

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 *Of* (Argument 0), drops the minus sign if one exists, and stores the results in *Put Result in* (Argument 1).

Arguments:

Argument 0 **Of**	Argument 1 **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	

Action Block Example:

Absolute Value

Argument Name	Type	Name
Of	*Float Variable*	*Negative_Value*
Put Result in	*Float Variable*	*Positive_Value*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- To change a negative value to a positive value, make the values of *Of* and *Put Result in* the same.

See Also: "Complement" on page 500

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 PAC Control command adds the value of Argument 0 to the value of *Plus* (Argument 1), and stores the result in *Put Result in* (Argument 2).
- *Put Result in* (Argument 2) 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 0 **[Value]**	Argument 1 **Plus**	Argument 2 **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	

Action Block Example:

Add		
Argument Name	Type	Name
(none)	*Analog Input*	*Ingredient_1_Weight*
Plus	*Analog Input*	*Ingredient_2_Weight*
Put Result in	*Analog Output*	*Total_Weight*

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:

- For more information, see "Mathematical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

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

See Also:

"Increment Variable" on page 509
"Subtract" on page 523

Arccosine

Mathematical Action

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

Typical Use: To solve trigonometric calculations.

Details:

- Calculates the arccosine of *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).
- *Of* (which is 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 0 **Of**	Argument 1 **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	

Action Block Example:

Arccosine		
Argument Name	Type	Name
Of	*Float Variable*	*X*
Put Result in	*Float Variable*	*RADIANS*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 501
"Arcsine" on page 494
"Arctangent" on page 495

Arcsine

Mathematical Action

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

Typical Use: To solve trigonometric calculations.

Details:

- Calculates the arcsine of *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).
- *Of* (which is 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 0 **Of**	Argument 1 **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	

Action Block Example:

Arcsine		
Argument Name	Type	Name
Of	*Float Variable*	*X*
Put Result in	*Float Variable*	*RADIANS*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700)
- 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" on page 520
"Arccosine" on page 493
"Arctangent" on page 495

Arctangent

Mathematical Action

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

Typical Use: To solve trigonometric calculations.

Details:

- Calculates the arctangent of *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).
- *Of* (which is 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 0 **Of**	**Argument 1** **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	

Action Block Example:

Arctangent		
Argument Name	Type	Name
Of	*Float Variable*	*X*
Put Result in	*Float Variable*	*RADIANS*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 493
"Arcsine" on page 494
"Tangent" on page 524

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 0	Argument 1	Argument 2	Argument 3
High Limit	**Low Limit**	**Element Index**	**Of Float 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		

Action Block Example:

Clamp Float Table Element

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

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" on page 498
"Clamp Float Variable" on page 497
"Clamp Integer 32 Variable" on page 499

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 0	**Argument 1**	**Argument 2**
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	

Action Block Example:

Clamp Float Variable

Argument Name	Type	Name
High Limit	*Float Variable*	*Max_Flow_Rate*
Low Limit	*Float Variable*	*Low_Flow_Cutoff*
Float Variable	*Float Variable*	*Flow_Var*

OptoScript Example:

ClampFloatVariable(*High Limit, Low Limit, Float Variable***)**

```
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" on page 499
"Clamp Float Table Element" on page 496
"Clamp Integer 32 Table Element" on page 498

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 0 **High Limit**	Argument 1 **Low Limit**	Argument 2 **Element Index**	Argument 3 **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		

Action Block Example:

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

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" on page 496
"Clamp Integer 32 Variable" on page 499
"Clamp Float Variable" on page 497

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 0	Argument 1	Argument 2
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	

Action Block Example:

Clamp Integer 32 Variable

Argument Name	Type	Name
High Limit	*Float Variable*	*Max_Flow_Rate*
Low Limit	*Float Variable*	*Low_Flow_Cutoff*
Integer 32 Variable	*Integer 32 Variable*	*Flow_Var*

OptoScript Example:

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

```
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" on page 497
"Clamp Integer 32 Table Element" on page 498
"Clamp Float Table Element" on page 496

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.
NOTE: The command Absolute Value is better way to accomplish the same thing.

Details: Same as Multiplying by -1, but executes faster. Thus, -1 becomes 1, 1 becomes -1, and so forth.

Arguments:

Argument 0
[Value]
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Complement		
Argument Name	Type	Name
(none)	*Float Variable*	*Temperature_Difference*

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

```
Temperature_Difference = -Temperature_Difference
```

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- The complement of zero is zero.

See Also:

"Bit NOT" on page 408
"NOT" on page 455
"Absolute Value" on page 491

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 *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).
- *Of* (Argument 0) has a theoretical range of -infinity to +infinity, but is limited by the size of the argument you pass.
- The range of *Put Result in* (Argument 1) is -1.0 to 1.0, inclusive.

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 0	Argument 1
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	

Action Block Example:

Cosine		
Argument Name	Type	Name
Of	*Float Variable*	*RADIANS*
Put Result in	*Float Variable*	*COSINE*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 493
"Sine" on page 520
"Tangent" on page 524

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, and so forth.

Arguments:

Argument 0
[Value]
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Decrement Variable		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Num_Holes_Left_to_Punch*

OptoScript Example:

DecrementVariable(*Argument 0***)**

```
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 the *PAC Control User’s Guide* (form 1700).
- Executes faster than subtracting 1, both in standard commands and in OptoScript code.

See Also: “Increment Variable” on page 509

Divide

Mathematical Action

Function: To divide two numerical values.

Typical Use: To perform a standard division action.

Details:

- Divides the value of Argument 0 by *By* (Argument 1), and stores the result in *Put Result in* (Argument 2).
- *Put Result in* (Argument 2) 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 *By* (Argument 1) is 0, an error -15 (divide by zero) is added to the message queue.

Arguments:

Argument 0 [Value]	Argument 1 By	Argument 2 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	

Action Block Example:

Divide		
Argument Name	Type	Name
(none)	*Float Variable*	*Total_Distance*
By	*Float Literal*	*2.0*
Put Result in	*Float Variable*	*Half_Distance*

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

```
Half_Distance = Total_Distance / 2.0;
```

Notes:

- For more information, see "Mathematical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- Avoid divide-by-zero errors by checking *By* (Argument 1) **before** doing the division to be sure it does not equal zero. Use Variable True? (if it's True, the result will be a non-zero value) 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, and so forth.

Queue Errors: -15 = Divide by zero.

See Also:

"Modulo" on page 512
"Multiply" on page 513
"Bit Shift" on page 421

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 0
Put in
Float Variable

Action Block Example:

Generate Random Number		
Argument Name	Type	Name
Put in	*Float Variable*	*LOTTO_SEED*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 519

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 *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).

Arguments:

Argument 0 **Of**	Argument 1 **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	

Action Block Example:

Hyperbolic Cosine		
Argument Name	Type	Name
Of	*Float Literal*	*2.0*
Put Result in	*Float Variable*	*ANSWER*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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

See Also: "Hyperbolic Sine" on page 507
"Hyperbolic Tangent" on page 508

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 *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).

Arguments:

Argument 0 **Of**	Argument 1 **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	

Action Block Example:

Hyperbolic Sine		
Argument Name	Type	Name
Of	*Float Literal*	*2.0*
Put Result in	*Float Variable*	*ANSWER*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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

See Also:

"Hyperbolic Cosine" on page 506
"Hyperbolic Tangent" on page 508

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 *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).
- The result is a value ranging from -1.0 to 1.0.

Arguments:

Argument 0 **Of**	Argument 1 **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	

Action Block Example:

Hyperbolic Tangent		
Argument Name	Type	Name
Of	*Float Literal*	*2.0*
Put Result in	*Float Variable*	*ANSWER*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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

See Also:

"Hyperbolic Cosine" on page 506
"Hyperbolic Sine" on page 507

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, and so forth.

Arguments:

Argument 0
[Value]
Float Variable
Integer 32 Variable
Integer 64 Variable

Action Block Example:

Increment Variable		
Argument Name	Type	Name
(none)	*Integer 32 Variable*	*LOOP_COUNTER*

OptoScript Example:

IncrementVariable(*Argument 0***)**

```
IncrementVariable(LOOP_COUNTER);
```

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

Notes:

- See “Mathematical Commands” in the *PAC Control User’s Guide* (form 1700).
- Executes faster than adding 1.

See Also: “Decrement Variable” on page 503

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 0 **Compare**	**Argument 1** **With**	**Argument 2** **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	

Action Block Example:

Maximum

Argument Name	Type	Name
Compare	*Analog Input*	*Pressure_A*
With	*Analog Input*	*Pressure_B*
Put Maximum in	*Float Variable*	*Highest_Pressure*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also: "Minimum" on page 511

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 0 **Compare**	Argument 1 **With**	Argument 2 **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	

Action Block Example:

Minimum

Argument Name	Type	Name
Compare	*Analog Input*	*Pressure_A*
With	*Analog Input*	*Pressure_B*
Put Minimum in	*Float Variable*	*Lowest_Pressure*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also: "Maximum" on page 510

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 0 [Value]	Argument 1 By	Argument 2 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	

Action Block Example:

Modulo

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Num_Parts_Produced*
By	*Integer 32 Variable*	*Minutes_Elapsed*
Put Result in	*Integer 32 Variable*	*Productivity_Remainder*

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

```
Productivity_Remainder = Num_Parts_Produced % Minutes_Elapsed;
```

Notes:

- For more information, see "Mathematical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).

See Also:

"Divide" on page 504
"Multiply" on page 513

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 the value of Argument 0 and the value of *Times* (Argument 1), and stores the result in *Put Result in* (Argument 2).
- *Put Result in* 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 0 **[Value]**	Argument 1 **Times**	Argument 2 **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	

Action Block Example:

Multiply		
Argument Name	Type	Name
(none)	*Analog Input*	*Ingredient_1_Weight*
Times	*Float Variable*	*Temperature_Adjust*
Put Result in	*Analog Output*	*Corrected_Weight*

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

```
Corrected_Weight = Ingredient_1_Weight * Temperature_Adjust;
```

Notes:

- For more information, see "Mathematical Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- *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" on page 504
"Bit Shift" on page 421

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 *Of* (Argument 0), and stores the result in *Put Result in* (Argument 1).

Arguments:

Argument 0 **Of**	Argument 1 **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	

Action Block Example:

Natural Log		
Argument Name	Type	Name
Of	*Float Variable*	*Fermentation_Rate*
Put Result in	*Float Variable*	*Rate_Calculation*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: PAC Control 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:

$$\text{Log}_{BASE}(\text{number}) = \frac{\ln(\text{number})}{\ln(\text{base})}$$

For example:

$$\text{Log}_{10}(100) = \frac{\ln(100)}{\ln(10)} = 2$$

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

number	LOG_e	LOG_{10}
100	4.605170	2
1000	6.907755	3

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

-14 = Invalid number.

See Also: "Raise to Power" on page 517

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 *Exponent* (Argument 0).
- The constant e, the base of the natural system of logarithms, has a value of 2.7182818.

Arguments:

Argument 0 **Exponent**	Argument 1 **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	

Action Block Example:

Raise e to Power		
Argument Name	Type	Name
Exponent	*Analog Input*	*Gas_Pressure*
Put Result in	*Float Variable*	*Pressure_Calculation*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).

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

See Also:
"Natural Log" on page 514
"Raise to Power" on page 517

Raise to Power

Mathematical Action

Function: To raise a value to a specified power.

Typical Use: To solve exponentiation calculations.

Details:

- Raises *Raise* (Argument 0) to the power specified by *To the* (Argument 1), and stores the result in *Put Result in* (Argument 2).

Arguments:

Argument 0 **Raise**	Argument 1 **To the**	Argument 2 **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	

Action Block Example:

Raise to Power		
Argument Name	Type	Name
Raise	*Integer 32 Literal*	*10*
To the	*Integer 32 Literal*	*2*
Put Result in	*Integer 32 Variable*	*TEN_SQUARED*

OptoScript Example:

Power (*Raise, To the***)**

```
TEN_SQUARED = 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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 516
"Square Root" on page 522

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. Note that 1.50 rounds up to 2.0, and 1.49 rounds down to 1.0.

Arguments:

Argument 0 **[Value]**	Argument 1 **Put Result in**
Float Literal	Float Variable
Float Variable	Integer 32 Variable
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

Round

Argument Name	Type	Name
(none)	*Float Variable*	*Boiler_Avg_Temp*
Put Result in	*Float Variable*	*Boiler_Working_Temp*

OptoScript Example:

Round (*Argument 0*)

```
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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: Using Move (or an assignment in OptoScript code) to copy a float value to an integer variable will round automatically.

See Also: "Truncate" on page 525

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.

Action Block Example:

Seed Random Number
No arguments

OptoScript Example:

`SeedRandomNumber()`

```
SeedRandomNumber();
```

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

See Also: "Generate Random Number" on page 505

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 *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).
- *Of* has a theoretical range of -infinity to +infinity, but is limited by the type of variable used.
- The range of *Put Result in* is -1.0 to 1.0, inclusive.

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 0 **Of**	Argument 1 **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	

Action Block Example:

Sine		
Argument Name	Type	Name
Of	*Float Variable*	*Radians*
Put Result in	*Float Variable*	*SINE*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 494
"Cosine" on page 501
"Tangent" on page 524

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 *Of* (Argument 0) and stores the result in *Put Result in* (Argument 1).

Arguments:

Argument 0	Argument 1
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	

Action Block Example:

Square Root		
Argument Name	Type	Name
Of	*Integer 32 Variable*	*Area_of_Square*
Put Result in	*Integer 32 Variable*	*Height_of_Square*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 517
"Greater Than Numeric Table Element?" on page 436

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 the value of *Minus* (Argument 1) from the value of Argument 0, and stores the result in *Put Result in* (Argument 2).
- *Put Result in* 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 0 **[Value]**	Argument 1 **Minus**	Argument 2 **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	

Action Block Example:

Subtract

Argument Name	Type	Name
(none)	*Integer 32 Variable*	*Num_to_Produce*
Minus	*Integer 32 Variable*	*Num_Produced*
Put Result in	*Integer 32 Variable*	*Num_Left_to_Make*

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

```
Num_Left_to_Make = Num_to_Produce - Num_Produced;
```

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- In OptoScript code, the – operator has many uses. For more information on mathematical expressions in OptoScript code, see the *PAC Control User's Guide* (form 1700).

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

See Also:

"Decrement Variable" on page 503
"Add" on page 492

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 *Of* (Argument 0), and stores the result in *Put Result in* (Argument 1).
- 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 value for *Of* (Argument 0) before calling the Tangent command.
- Tangent is sin (angle) / cos (angle).

Arguments:

Argument 0 **Of**	Argument 1 **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	

Action Block Example:

Tangent

Argument Name	Type	Name
Of	*Float Variable*	*RADIANS*
Put Result in	*Float Variable*	*TANGENT*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Mathematical Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 495
"Cosine" on page 501
"Sine" on page 520

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 0	**Argument 1**
[Value]	**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

Action Block Example:

Truncate		
Argument Name	Type	Name
(none)	*Float Variable*	*Flow_Total_Raw*
Put Result in	*Integer 32 Variable*	*Flow_Total_Integer*

OptoScript Example:

`Truncate(`*Argument 0***`)`**

```
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" on page 518

17: Miscellaneous Commands

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.

Example:

Comment (Block)	
Start Chart	
Chart	Log_Data
Put Status In	nChartStatus
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)" on page 529

Comment (Opto Control Conversion Issue)

Miscellaneous Action or Condition

Function: A comment is inserted automatically when a command does not automatically convert from Opto Control to PAC Control. 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 (Opto Control Conversion Issue), and 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 0
[Value]
String Literal

Example:

Comment (Single Line)		
Argument Name	Type	Name
(none)	*String Literal*	*Remember to test this block*

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

```
// Remember to test this block
```

See Also: "Comment (Block)" on page 527

Flag Lock

Miscellaneous Action

Function: Flag Lock and "Flag Unlock" on page 532 use an integer 32 variable as a synchronization object. These commands allow a strategy to give a chart exclusive access to one or more objects (for example, a table, integer, or I/O unit) while the chart has the synchronization object locked.

Typical Use: To successfully perform multiple operations on one or more objects such that each operation can completely finish before another chart gains access to the objects.

Details:

- For a chart to store multiple values to an object—for example, a table—without interference from another chart, you associate a 32-bit integer variable (called a "flag") with the table. Then, you construct logic so that no chart writes to the table without first locking the associated flag. Using this kind of logic, while the flag is locked, no other chart can access the table.
- For *Flag* (Argument 0), specify the integer 32 variable to be used as the flag. If you try to use anything other than an integer 32 variable, a -29 (Wrong Object Type) status code is returned.
- For *Timeout* (Argument 1), use one of the following timeout values:

Value	Type	Description
-1	Blocking	• If the flag is *unlocked* when a blocking timeout is sent, the system locks the flag immediately. • If the flag is *locked* when the blocking timeout is sent, the system tries to acquire the flag until it succeeds (or until the chart requesting the blocking lock is stopped). Chart logic doesn't proceed to the next command until the Flag Lock command is able to acquire the flag. A blocking timeout always returns 0 (Success) status code.
Number of milliseconds to wait, from +1 (1 millisecond) to +2147483647 (about 25 days)	Timed	• If the flag is *unlocked* when a timed timeout is sent, the system locks the flag immediately and returns a 0 (Success) status code. • If the flag is *locked* when the timed timeout is sent, the system tries to acquire the flag until it succeeds (in which case it returns a 0 (Success) status code) or until the time specified for *Timeout* expires (in which case the system returns a -37 (Timeout on lock) status code). Chart logic doesn't proceed to the next command until either the Flag Lock command is able to acquire the flag or the time-out expires.
0	Non-blocking	With a non-blocking timeout, the system attempts to acquire the flag only once. • If the flag is *unlocked*, the system locks the flag immediately and returns a 0 status code. • If the flag is already *locked*, the system returns a -17 (Port or object is already locked) status code.

Arguments:

Argument 0	Argument 1	Argument 2
Flag	**Timeout**	**Put Result in**
Integer 32 Variable	Integer 32 Literal	Integer 32 Variable
	Integer 32 Variable	

Action Block Example:

Flag Lock		
Argument Name	Type	Name
Flag	*Integer 32 Variable*	*Flag_1*
Timeout	*Integer 32 Variable*	*nTimeout*
Put Result in	*Integer 32 Variable*	*nResult1*

OptoScript Example:

FlagLock(*Flag, Timeout***)**

```
nResult1 = FlagLock(Flag_1, nTimeout);
```

This is a function command; it returns a value of 0 (Success) or one of the status codes listed below.

Status Codes:

-8 = Invalid data. Verify that *Timeout* (Argument 1) is not a negative number other than -1. (The only acceptable negative value is -1, which is the value for a blocking timeout.)

-17 = Port or object is already locked.

-29 = Wrong object type.

-37 = Timeout on lock.

Notes:

Here are tips for working with flag locks:

- For best performance, you should hold a flag lock only as long as necessary. Remember—any other charts that need access to the flag are going to be held up until the flag is unlocked.
- If you use a Flag Lock in a chart, you must also include a Flag Unlock in the same chart.
- If you use a Flag Unlock to unlock a flag in a *different* chart, the Flag Unlock command must include the Force Unlock option (any non-zero value).
- Don't use flag variables for anything other than Flag Lock and Flag Unlock commands. To make it obvious that a variable is being used as a flag, assign it a meaningful name; for example, FlagForInterProcessLock.
- Don't set the value of the flag variable using any methods or commands other than Flag Lock and Flag Unlock.
- In systems without redundant controllers, you should use *initialize-on-run variables only*. Don't use persistent or initialize-on-download variables.
- In systems with redundant controllers only:
  - Configure flag variables to be persistent.
  - At the start of the strategy, initialize them using Flag Unlock with the Force unlock option (to prevent issues in case the strategy was previously interrupted).
  - To prevent the flag from staying locked, make sure to clear it (using the Flag Unlock command) later in the same chart.
- In PAC firmware R9.5c and lower, if a chart is stopped while it is holding a lock on a flag, the lock isn't released, preventing other charts from accessing the flag. This may also result in losing communication to the controller's Host task until the controller is rebooted.

  In PAC firmware R9.5d and higher, if you attempt to stop a chart, the chart will continue to run until the flag is unlocked.

See Also:

"Flag Unlock" on page 532

Flag Unlock

Miscellaneous Action

Function: Flag Unlock and "Flag Lock" on page 530 use an integer 32 variable as a synchronization object. These commands allow a strategy to give a chart exclusive access to one or more objects (for example, a table, integer, or I/O unit) while the chart has the synchronization object locked.

Typical Use: To successfully perform multiple operations on one or more objects such that each operation can completely finish with the objects before another chart gains access to the objects.

Details:

- For *Flag* (Argument 0), specify the integer 32 variable that is locked (by having been issued a Flag Lock command).
- Use *Force unlock* (Argument 1 with a non-zero value) to handle what could otherwise be an unrecoverable situation—such as when you're unable to unlock a flag by normal means. A non-zero value (True) forces the flag variable to be unlocked and returns a 0 (Success) status code.
- When the flag variable is locked, a Flag Unlock command (with a Force unlock parameter value of zero) will fail unless the chart that is asking for the unlock is the same chart that locked it. In other words, if Chart 1 locks a flag, only Chart 1 can unlock it (with a Force unlock parameter value of zero).

  EXCEPTION: When the Force unlock parameter is set to a non-zero value, any chart can unlock the flag.

Arguments:

Argument 0	Argument 1	Argument 2
Flag	**Force unlock**	**Put Result in**
Integer 32 Variable	Integer 32 Literal	Integer 32 Variable
	Integer 32 Variable	

Action Block Example:

Flag Unlock

Argument Name	Type	Name
Flag	*Integer 32 Variable*	*Flag_2*
Force unlock	*Integer 32 Variable*	*nForce_Unlock*
Put Result in	*Integer 32 Variable*	*nResult2*

OptoScript Example:

`FlagUnlock``(`*Flag, Force unlock*`)`

```
nResult2 = FlagUnlock(Flag_2, nForce_Unlock);
```

This is a function command; it returns a value of 0 (Success) or one of the status codes listed below.

Status Codes:

-25 = Port or object is not locked. Indicates the flag is not locked.

-29 = Wrong object type. Indicates the variable you are trying to unlock is not flag. (In other words, it contains a non-zero value that was *not* put there by a Flag Lock command.)

-103 = Wrong task. If the Force Unlock parameter value was zero, indicates that the chart requesting the Unlock is not the chart that set the Lock.

Notes:

- For best performance, you should hold a flag lock only as long as necessary. Remember—any other charts that need access to the flag are going to be held up until the flag is unlocked.
- If you use a Flag Lock in a chart, you must also include a Flag Unlock in the same chart.
- If you use a Flag Unlock to unlock a flag in a *different* chart, the Flag Unlock command must include the Force Unlock option (any non-zero value).
- Don't use flag variables for anything other than Flag Lock and Flag Unlock commands. To make it obvious that a variable is being used as a flag, assign it a meaningful name; for example, FlagForInterProcessLock.

- Don't set the value of the flag variable using any methods or commands other than Flag Lock and Flag Unlock.
- In systems without redundant controllers, you should use *initialize-on-run variables only*. Don’t use persistent or initialize-on-download variables.
- In systems with redundant controllers only:
  - Configure flag variables to be persistent.
  - At the start of the strategy, initialize them using Flag Unlock with the Force unlock option (to prevent issues in case the strategy was previously interrupted).
  - To prevent the flag from staying locked, make sure to clear it (using the Flag Unlock command) later in the same chart.

See Also: “Flag Lock” on page 530

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 0
Is
Float Variable

Condition Block Example:

Float Valid?		
Argument Name	Type	Name
Is	*Float Variable*	*Oil_Pressure*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: Analog points on an unplugged module return a value of NaN (not a number).

See Also:

"Move 32 Bits" on page 454
"Get I/O Unit Scratch Pad Float Element" on page 363
"Read Number from I/O Unit Memory Map" on page 343

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 transferred 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 0	Argument 1	Argument 2
Start Value	**Table**	**Starting Element**
Integer 32 Literal	Float Table	Integer 32 Literal
Integer 32 Variable	Integer 32 Table	Integer 32 Variable

Argument 2	Argument 3	Argument 4
Starting Element	**Number of Elements**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Generate Reverse CRC-16 on Table (32 bit)		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
Table	*Float Table*	*VALUES_TO_SEND*
Starting Element	*Integer 32 Literal*	*1*
Number of Elements	*Integer 32 Literal*	*31*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example:

`GenerateReverseCrc16OnTable32``(`*Start Value, Table, Starting Element, Number of Elements*`)`

```
RESULT = 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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 Numeric 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:

  4. Receive Numeric Table.
  5. Generate Reverse CRC-16 on Table (32 bit) on table elements 1–31.

6. Compare the calculated CRC against the value stored in element 0.

See Also: "Generate Forward CRC-16 on String" on page 673
"Generate Reverse CCITT on String" on page 674
"Generate Forward CCITT on String" on page 672

Get Length of Table

Miscellaneous 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 0	**Argument 1**
Table	**Put in**
Float Table	Float Variable
Integer 32 Table	Integer 32 Variable
Integer 64 Table	
Pointer Table	
String Table	

Action Block Example:

Get Length of Table		
Argument Name	Type	Name
Table	*Integer 32 Table*	*Config_Data*
Put in	*Integer 32 Variable*	*Config_Data_Size*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 Type From Name

Miscellaneous Action

Function: To find out the data type (string, floating point, and so forth) 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 *Name* (Argument 0) and places a bitmask in *Put In* (Argument 1), 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
00200000	PID
00400003	Mistic Digital
00400004	Mistic Analog
00402000*	Generic MMP Device
00402005	Mixed I/O Unit
00402006	Digital 64 I/O Unit
00402007	Mixed 64 I/O Unit
00800000	Integer 32
00800001	Integer 64
00800002	Float

Value in Hex	Data Type
00800003	Down Timer
00800004	Up Timer
00810000	Integer 32 Table
00810001	Integer 64 Table
00810002	Float Table
01000000	String
01010000	String Table
02000000	Chart
02000001	Subroutine
09000000	Communication Handle
20000000	Pointer
20010000	Pointer Table

* This value might represent a GRV-EPIC-PR1, R1, R1-B, or R2 I/O Unit data type.

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 0	**Argument 1**
Name	**Put in**
String Literal	Integer 32 Variable
String Variable	

Action Block Example:

Get Type From Name		
Argument Name	Type	Name
Name	*String Literal*	*Variable_Name*
Put in	*Integer 32 Variable*	*DATA_TYPE*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Note: The data type values returned by this command are set by Opto 22 and may be changed in future releases.

See Also: "Get Value From Name" on page 540

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 *Name* (Argument 0) and places it in the form of a string in *Put Result in* (Argument 1).
- The value of *Name* (Argument 0) can be of various types; it will automatically be converted into a string.
- The string variable in *Put Result in* (Argument 1) 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 arguments 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 1).

Arguments:

Argument 0	Argument 1	Argument 2
Name	**Put Result in**	**Put Status in**
String Literal	String Variable	Integer 32 Variable
String Variable		

Action Block Example:

Get Value From Name		
Argument Name	Type	Name
Name	*String Variable*	*Item_Count*
Put Result in	*String Variable*	*Production*
Put Status in	*Integer 32 Variable*	*Status*

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 used by an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "Miscellaneous Commands" in For more information, see the *PAC Control User's Guide* (form 1700).
- If you need to know the data type of *Name* (Argument 0), 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

-12 = Invalid table index. Check the index of the named table element and the size of that table.

-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 0 not found. Check the name and case.

See Also:

"Get Type From Name" on page 538
"Convert String to Float" on page 659
"Convert String to Integer 32" on page 661
"Convert String to Integer 64" on page 663

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:

- PAC Control automatically converts the type of *From* (Argument 0) to match that of *To* (Argument 1). 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 0	**Argument 1**
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	

Action Block Example:

Move		
Argument Name	Type	Name
From	*Digital Input*	*DIG1*
To	*Integer 32 Variable*	*DIG1_STATUS*

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" on page 682
"Move to Numeric Table Element" on page 547 and other Move to Table commands
"Move from Numeric Table Element" on page 544 and other Move from Table commands.

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:

- PAC Control automatically converts the type of *Of Table* (Argument 1) to match that of *To* (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.
  - *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.
- The valid range for the index is 0 (zero) to the table length minus 1 (size – 1).

Arguments:

Argument 0 **From Index**	Argument 1 **Of Table**	Argument 2 **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

Action Block Example:

Move from Numeric Table Element		
Argument Name	Type	Name
From Index	*Integer 32 Literal*	*0*
Of Table	*Float Table*	*LOOK_UP_TABLE*
To	*Analog Output*	*PRESS_OUT*

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" on page 545
"Move to Numeric Table Element" on page 547
"Shift Numeric Table Elements" on page 550

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.
- PAC Control automatically converts the type of *Of Table* (Argument 1) to match that of *Of Table* (Argument 3).

  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 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.
- Any value sent to an invalid index is discarded, and a -12 (Invalid table index value) status code is added to the message queue.
- The valid range for each index is 0 (zero) to the table length minus 1 (size – 1).

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
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

Action Block Example:

Move Numeric Table Element to Numeric Table

Argument Name	Type	Name
From Index	*Integer 32 Literal*	*17*
Of Table	*Integer 32 Table*	*I/O_STATUS_TABLE*
To Index	*Integer 32 Literal*	*27*
Of Table	*Integer 32 Table*	*I/O_STATUS_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" on page 547

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 width (for example, Integer 64 Tables can only be copied to other Integer 64 Tables) and must be different tables. They can be different sizes, but make sure *Length* (Argument 4) is not too long for either table. Also, one or both tables can be pointer tables. The pointers in those tables must point to variables of the same width, as described above. A null pointer will be skipped with a -69 (Invalid parameter) status code in the queue, and a pointer to something of mismatched size will give a -29 (Wrong object type) status code in the queue mentioned below.
- The valid range for each table index is 0 (zero) to the table length - 1 (size - 1).

Arguments:

Argument 0	Argument 1	Argument 2
From Table	**From Index**	**To Table**
Float Table	Integer 32 Literal	Float Table
Integer 32 Table	Integer 32 Variable	Integer 32 Table
Integer 64 Table		Integer 64 Table
Pointer Table		Pointer Table

Argument 3	Argument 4
To Index	**Length**
Integer 32 Literal	Integer 32 Literal
Integer 32 Variable	Integer 32 Variable

Action Block Example:

Move Numeric Table to Numeric Table		
Argument Name	Type	Name
From Table	*Integer 32 Table*	*Temp_Table*
From Index	*Integer 32 Literal*	*0*
To Table	*Integer 32 Table*	*Status_Table*
To Index	*Integer 32 Literal*	*16*
Length	*Integer 32 Literal*	*8*

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. *From Table* (Argument 0) and *To Table* (Argument 2) must have the same width.

-69 = Invalid parameter (null pointer) passed to command. Received if a null table object pointer is passed.

See Also: "Move to Numeric Table Element" on page 547

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:

- PAC Control automatically converts the type of *From* (Argument 0) to match that of *Of Table* (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.
  - *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.
- Any value sent to an invalid index is discarded, and a -12 (Invalid table index value) status code is added to the message queue.
- The valid range for each index is 0 (zero) to the table length minus 1 (size – 1).

Arguments:

Argument 0	Argument 1	Argument 2
From	**To 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		
Float Literal		
Float Variable		
Integer 32 Literal		
Integer 32 Variable		
Integer 64 Literal		
Integer 64 Variable		

Action Block Example:

Move to Numeric Table Element

Argument Name	Type	Name
From	*Integer 32 Literal*	*0*
To Index	*Integer 32 Literal*	*27*
Of Table	*Integer 32 Table*	*IO_STATUS_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" on page 544
"Move to Numeric Table Elements" on page 549

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:
- Any value sent to an invalid index is discarded, and a -12 (Invalid table index value) status code is added to the message queue.
- The valid range for each index is 0 (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 0	Argument 1	Argument 2	Argument 3
From	**Start Index**	**End Index**	**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			

Action Block Example:

Move to Numeric Table Elements

Argument Name	Type	Name
From	*Integer 32 Literal*	*0*
Start Index	*Integer 32 Literal*	*4*
End Index	*Integer 32 Literal*	*10*
Of Table	*Integer 32 Table*	*IO_STATUS_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" on page 544
"Move to Numeric Table Element" on page 547

Shift Numeric Table Elements

Miscellaneous 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 0	Argument 1
Shift Count	**Table**
Integer 32 Literal	Float Table
Integer 32 Variable	Integer 32 Table
	Integer 64 Table

Action Block Example:

Shift Numeric Table Elements		
Argument Name	Type	Name
Shift Count	*Integer 32 Literal*	*-5*
Table	*Float Table*	*MY_TABLE*

OptoScript Example:

ShiftNumTableElements(*Shift Count, Table*)

```
ShiftNumTableElements(-5, MY_TABLE);
```

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.
- If desired, use Move to Numeric Table Element after this command, for example, to fill vacated entries.

See Also:

"Move Numeric Table Element to Numeric Table" on page 545
"Move from Numeric Table Element" on page 544
"Move to Numeric Table Element" on page 547

Shift String Table Elements

Miscellaneous Action

Function: To shift string 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.
- Empty strings are written to entries left empty by shifting.

Arguments:

Argument 0	**Argument 1**
Shift Count	**Table**
Integer 32 Literal	String Table
Integer 32 Variable	

Action Block Example:

Shift String Table Elements		
Argument Name	Type	Name
Shift Count	*Integer 32 Literal*	*-5*
Table	*String Table*	*MY_TABLE*

OptoScript Example:

`ShiftStringTableElements(`*Shift Count, Table***`)`**

```
ShiftStringTableElements(-5, MY_TABLE);
```

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

Notes:

- Use Move from String Table Element before this command to capture values that will be shifted out of the table, if they need to be used.
- If desired, use Move to String Table Element after this command, for example, to fill vacated entries.

See Also:

"Move from String Table Element" on page 680
"Move to String Table Element" on page 683
"Move to String Table Elements" on page 684

18: PID - Ethernet Commands

Get PID Configuration Flags

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 PAC Manager or PAC Control, 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 0	Argument 1
PID Loop	**Configuration Flags**
PID Loop	Integer 32 Variable

Action Block Example:

Get PID Configuration Flags

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Configuration Flags	*Integer 32 Variable*	*PID_CONFIG_FLAGS*

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 SNAP PAC I/O brain's memory map. (See Details, above). The returned value can be consumed by a variable (as in the example shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Set PID Configuration Flags" on page 576

Get PID Current Input

PID—Ethernet Action

NOTE: This command is not for use with the SNAP-PID-V module.

Function: To read the current input value (also known as the process variable) of a PID whose input is determined by an analog point or a PID output (for cascaded PIDs).

NOTE: To get the input of a PID that was set from Host, use *"Get PID Input" on page 562*

Typical Use: To read the current value of a PID input.

Details:

- The input must be from an analog point or a PID output (for cascaded PIDs). The command returns zero (0) if the input is configured to be from Host.
- Get PID Current Input retrieves the value of the input right now, independent of scan time.
- The value read has the same engineering units as the specified PID input.

Arguments:

Argument 0	Argument 1
PID Loop	**Input**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Current Input		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Input	*Float Variable*	*PID_INPUT_VALUE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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:

"Disable Communication to PID Loop" on page 613
"Get PID Input" on page 562
"Set PID Input" on page 582

Get PID Current Setpoint

PID—Ethernet Action

NOTE: This command is not for use with SNAP-PID-V modules.

Function: To read the current setpoint value of a PID whose setpoint is determined by an analog point or a PID output (for cascaded PIDs).

NOTE: To get the setpoint of a PID that was set from Host, use Get PID Setpoint.

Typical Use: To read the current value of a PID setpoint.

Details:

- The setpoint must be from an analog point or a PID output (for cascaded PIDs). The command returns zero (0) if the setpoint is configured to be from Host.
- Get PID Current Setpoint retrieves the value of the setpoint right now, independent of scan time.
- The value read has the same engineering units as the specified PID setpoint.

Arguments:

Argument 0	Argument 1
PID Loop	**Setpoint**
PID Loop	Analog Output Float Variable Integer 32 Variable

Action Block Example:

Get PID Current Setpoint		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Heater_3*
Setpoint	*Float Variable*	*Pid_Setpoint_Value*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 618
"Get PID Setpoint" on page 572
"Set PID Setpoint" on page 592

Get PID Feed Forward

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 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 0	Argument 1
PID Loop	**Feed Forward**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Feed Forward		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Feed Forward	*Float Variable*	*PID_FEED_FORWARD*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Set PID Feed Forward" on page 577

Get PID Feed Forward Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 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 0	**Argument 1**
PID Loop	**Feed Fwd Gain**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Feed Forward Gain		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Feed Fwd Gain	*Float Variable*	*PID_FEED_FD_GAIN*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Set PID Feed Forward Gain" on page 578

Get PID Forced Output When Input Over Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Forced Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Forced Output When Input Over Range

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Forced Output	*Float Variable*	*PID_OUTPUT_OVER_RANGE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 560
"Set PID Forced Output When Input Over Range" on page 579

Get PID Forced Output When Input Under Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Forced Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Forced Output When Input Under Range		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Forced Output	*Float Variable*	*PID_OUTPUT_UNDER_RANGE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 559
"Set PID Forced Output When Input Under Range" on page 580

Get PID Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; it is not for use with the SNAP-PID-V module.

Function: Reads the gain value from the PID.

Typical Use: To store PID arguments 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 0	Argument 1
PID Loop	**Gain**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Gain		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Gain	*Float Variable*	*Zone08_Gain*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 581

Get PID Input

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; it is not for use with the SNAP-PID-V module.

Function: To read the input value (also known as the process variable) of a 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 point 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 0	Argument 1
PID Loop	**Input**
PID Loop	Analog Output Float Variable Integer 32 Variable

Action Block Example:

Get PID Input

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Input	*Float Variable*	*PID_INPUT_VALUE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 618
"Get PID Current Input" on page 555
"Set PID Input" on page 582

Get PID Input High Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**High Range**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Input High Range		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
High Range	*Float Variable*	*PID_High_Range*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Input Low Range" on page 564
"Set PID Input High Range" on page 583

Get PID Input Low Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Low Range**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Input Low Range		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Low Range	*Float Variable*	*PID_LOW_RANGE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Input High Range" on page 563
"Set PID Input Low Range" on page 584

Get PID Max Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 arguments 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 0	Argument 1
PID Loop	**Max Change**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Max Output Change		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Max Change	*Float Variable*	*PID_MAX_LIMIT*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Max Output Change" on page 565
"Set PID Scan Time" on page 591

Get PID Min Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 arguments 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 0	Argument 1
PID Loop	**Min Change**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Min Output Change		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Min Change	*Float Variable*	*PID_MIN_LIMIT*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Max Output Change" on page 565
"Set PID Min Output Change" on page 586
"Set PID Scan Time" on page 591

Get PID Mode

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 arguments 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 0	Argument 1
PID Loop	**Mode**
PID Loop	Integer 32 Variable

Action Block Example:

Get PID Mode

Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Mode	*Integer 32 Variable*	*ZONE08_MODE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

See Also: "Set PID Mode" on page 587

Get PID Output

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Output**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Output

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Output	*Analog Output*	*TPO_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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 618
"Set PID Output" on page 588

Get PID Output High Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**High Clamp**
PID Loop	Analog Output Float Variable Integer 32 Variable

Action Block Example:

Get PID Output High Clamp

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
High Clamp	*Float Variable*	*PID_HIGH_CLAMP*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Output Low Clamp" on page 570
"Set PID Output High Clamp" on page 589

Get PID Output Low Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Low Clamp**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Output Low Clamp

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Low Clamp	*Float Variable*	*PID_LOW_CLAMP*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Output High Clamp" on page 569
"Set PID Output Low Clamp" on page 590

Get PID Scan Time

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 arguments 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 0	Argument 1
PID Loop	**Scan Time (sec)**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Scan Time		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Scan Time (sec)	*Float Variable*	*Zone08_Scan_Time*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- To store the result, always use a float variable.

See Also: "Set PID Scan Time" on page 591

Get PID Setpoint

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; it is not for use with the SNAP-PID-V module.

Function: To read the setpoint value of a PID.

Typical Uses:

- To verify that the setpoint of the PID is as expected.
- 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 point 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 0	Argument 1
PID Loop	**Setpoint**
PID Loop	Analog Output Float Variable Integer 32 Variable

Action Block Example:

Get PID Setpoint		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Heater_3*
Setpoint	*Float Variable*	*Pid_Setpoint_Value*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 618
"Get PID Current Setpoint" on page 556
"Set PID Setpoint" on page 592

Get PID Status Flags

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 bitmask 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 0	Argument 1
PID Loop	**Status Flags**
PID Loop	Integer 32 Variable

Action Block Example:

Get PID Status Flags		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Status Flags	*Integer 32 Variable*	*PID_STATUS_FLAGS*

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 SNAP PAC 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "PID Commands" in the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Configuration Flags" on page 553
"Set PID Configuration Flags" on page 576

Get PID Tune Derivative

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 arguments 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 0	**Argument 1**
PID Loop	**Tune Derivative**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Tune Derivative		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Tune Derivative	*Float Variable*	*Zone08_Derivative*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- To store the result, always use a float variable.

See Also: "Set PID Tune Derivative" on page 593

Get PID Tune Integral

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 arguments 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 0	Argument 1
PID Loop	**Tune Integral**
PID Loop	Analog Output
	Float Variable
	Integer 32 Variable

Action Block Example:

Get PID Tune Integral		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Tune Integral	*Float Variable*	*Zone08_Integral*

OptoScript Example:

GetPidTuneIntegral(*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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- To store the result, always use a float variable.

See Also: "Set PID Tune Integral" on page 594

Set PID Configuration Flags

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 PAC Manager or PAC Control.
- 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 0	Argument 1
PID Loop	**Configuration Flags**
PID Loop	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Configuration Flags		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Configuration Flags	*Integer 32 Variable*	*PID_CONFIG_FLAGS*

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 the *PAC Control User's Guide* (form 1700).

Dependencies: Communication to the PID must be enabled for this command to take effect.

See Also:

"Enable Communication to PID Loop" on page 618
"Get PID Configuration Flags" on page 553

Set PID Feed Forward

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 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 0	Argument 1
PID Loop	**Feed Forward**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Feed Forward		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Feed Forward	*Float Variable*	*PID_FEED_FORWARD*

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 the *PAC Control User’s Guide* (form 1700).
- 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” on page 618
“Get PID Feed Forward” on page 557

Set PID Feed Forward Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 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 0	Argument 1
PID Loop	**Feed Fwd Gain**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Feed Forward Gain

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Feed Fwd Gain	*Float Variable*	*PID_FEED_FD_GAIN*

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 the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Feed Forward Gain" on page 558

Set PID Forced Output When Input Over Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Forced Output**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Forced Output When Input Over Range		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Forced Output	*Float Variable*	*PID_OUTPUT_OVER_RANGE*

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 559
"Set PID Forced Output When Input Under Range" on page 580

Set PID Forced Output When Input Under Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Forced Output**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Forced Output When Input Under Range		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Forced Output	*Float Variable*	*PID_OUTPUT_UNDER_RANGE*

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 the *PAC Control User’s Guide* (form 1700).
- 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” on page 560
“Set PID Forced Output When Input Over Range” on page 579

Set PID Gain

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0 **PID Loop**	Argument 1 **Gain**
PID Loop	Analog Input Analog Output Float Literal Float Variable Integer 32 Literal Integer 32 Variable

Action Block Example:

Set PID Gain		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Gain	*Float Variable*	*Zone08_Gain*

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 618
"Get PID Gain" on page 561

Set PID Input

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0 **PID Loop**	Argument 1 **Input**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Input		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Input	*Float Variable*	*PID_INPUT_VALUE*

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 the *PAC Control User’s Guide* (form 1700).
- 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” on page 618
“Get PID Input” on page 562

Set PID Input High Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**High Range**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Input High Range		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
High Range	*Float Variable*	*PID_High_Range*

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 the *PAC Control User's Guide* (form 1700).

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" on page 563
"Set PID Input Low Range" on page 584

Set PID Input Low Range

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Low Range**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Input Low Range

Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Low Range	*Float Variable*	*PID_LOW_RANGE*

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 the *PAC Control User's Guide* (form 1700).

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" on page 564
"Set PID Input High Range" on page 583

Set PID Max Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Max Change**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Max Output Change		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Max Change	*Float Variable*	*PID_MAX_LIMIT*

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 the *PAC Control User’s Guide* (form 1700).

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” on page 618
“Get PID Max Output Change” on page 565
“Set PID Scan Time” on page 591

Set PID Min Output Change

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Min Change**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Min Output Change		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Min Change	*Float Variable*	*PID_MIN_LIMIT*

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 the *PAC Control User’s Guide* (form 1700).

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” on page 618
“Get PID Max Output Change” on page 565
“Set PID Scan Time” on page 591

Set PID Mode

PID—Ethernet Action

NOTE: This command is used for PID loops on PAC brains in PAC Control; it is not for use with the SNAP-PID-V module.

For mistic PID loops, see *"20: PID - Mistic" on page 631*.

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:

- Use these values to set auto and manual modes: auto = 0, manual = 1.
  - In auto mode, the PID output functions normally.
  - In manual mode, the PID output register (which is part of the PID loop structure) is not updated by the PID calculation. However, the PID loop will continue to update the analog output point from the PID output register at the configured PID scan rate.

    To use two PID loops for the same analog output point, configure the output for both loops to be "Host" (PAC Control > Configure > I/O > PID Loops). Then, use logic to read the appropriate PID output, and then move it to the analog point.
- To change the PID output value while in manual mode, use Set PID Output; if using Debug mode, PAC Manager, or PAC Display, write to the PID output instead of to the analog point.

Arguments:

Argument 0	Argument 1
PID Loop	**Mode**
PID Loop	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Mode		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Mode	*Integer 32 Variable*	*ZONE08_MODE*

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 the *PAC Control User's Guide* (form 1700).

See Also:

"Get PID Mode" on page 567
"Set PID Output" on page 588

Set PID Output

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Output**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Output		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Output	*Analog Output*	*TPO_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 the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Output" on page 568
"Set PID Mode" on page 587
"Get PID Mode" on page 567

Set PID Output High Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**High Clamp**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Output High Clamp		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
High Clamp	*Float Variable*	*PID_HIGH_CLAMP*

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 the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Output High Clamp" on page 569
"Set PID Output Low Clamp" on page 590

Set PID Output Low Clamp

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0 **PID Loop**	Argument 1 **Low Clamp**
PID Loop	Analog Input Analog Output Float Literal Float Variable Integer 32 Literal Integer 32 Variable

Action Block Example:

Set PID Output Low Clamp		
Argument Name	Type	Name
PID Loop	*PID Loop*	*HEATER_3*
Low Clamp	*Float Variable*	*PID_LOW_CLAMP*

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 the *PAC Control User's Guide* (form 1700).

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" on page 618
"Get PID Output Low Clamp" on page 570
"Set PID Output High Clamp" on page 589

Set PID Scan Time

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Scan Time (sec)**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Scan Time

Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Scan Time (sec)	*Float Variable*	*Zone08_Scan_Time*

OptoScript Example:

SetPidScanTime(*PID Loop, Scan Time (sec)***)**

```
SetPidScanTime(Extruder_Zone08, Zone08_Scan_Time);
```

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

Notes:

- See "PID Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 618
"Get PID Scan Time" on page 571,

Set PID Setpoint

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	Argument 1
PID Loop	**Setpoint**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Setpoint		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Heater_3*
Setpoint	*Float Variable*	*Pid_Setpoint_Value*

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 the *PAC Control User’s Guide* (form 1700).
- 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” on page 618
“Get PID Setpoint” on page 572

Set PID Tune Derivative

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0 **PID Loop**	Argument 1 **Tune Derivative**
PID Loop	Analog Input Analog Output Float Literal Float Variable Integer 32 Literal Integer 32 Variable

Action Block Example:

Set PID Tune Derivative		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Tune Derivative	*Float Variable*	*Zone08_Derivative*

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 the *PAC Control User’s Guide* (form 1700).
- 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.

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” on page 618
“Get PID Tune Derivative” on page 574

Set PID Tune Integral

PID—Ethernet Action

NOTE: This command is used for PID loops in PAC Control; 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 0	**Argument 1**
PID Loop	**Tune Integral**
PID Loop	Analog Input
	Analog Output
	Float Literal
	Float Variable
	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Set PID Tune Integral		
Argument Name	Type	Name
PID Loop	*PID Loop*	*Extruder_Zone08*
Tune Integral	*Float Variable*	*Zone08_Integral*

OptoScript Example:

SetPidTuneIntegral(*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 the *PAC Control User's Guide* (form 1700).
- 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.

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" on page 618
"Get PID Tune Integral" on page 575

19: Pointers Commands

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 0
Pointer
Pointer Variable

Action Block Example:

Clear Pointer		
Argument Name	Type	Name
Pointer	*Pointer Variable*	*IO_Pointer*

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" on page 599
"Clear Pointer Table Element" on page 596

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

Action Block Example:

Clear Pointer Table Element		
Argument Name	Type	Name
Index	*Integer 32 Literal*	*17*
Of Table	*Pointer Table*	*IO_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" on page 602

Get Pointer From Name

Pointers Action

Function: To assign an object to a pointer variable 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 variable is a float pointer, the variable name must be for a float variable.
- The variable name is case sensitive.
- If a string variable is used in *Name* (Argument 0), the command assigns a pointer to the object whose name is contained within the string variable (that is, the contents of the string variable), not to the string variable itself.

  The type of the pointer must match the type of the object whose name is contained within the string variable.

  In the example

  ```
  My_String = "My_Integer" ;
  GetPointerFromName(My_String, pInteger);
  ```

  The variable `My_String` is a string variable containing the name of an integer variable (`My_Integer`) that must exist in the strategy. This command then moves the pointer for `My_Integer` into the pointer variable pInteger.

Arguments:

Argument 0	Argument 1
Name	**Pointer**
String Literal	Pointer Variable
String Variable	

Action Block Example:

Get Pointer From Name		
Argument Name	Type	Name
Name	*String Literal*	*"My_Integer"*
Pointer	*Pointer Variable*	*pInteger*

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 the *PAC Control User's Guide* (form 1700).

See Also: "Move to Pointer" on page 599

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

Action Block Example:

Move From Pointer Table Element		
Argument Name	Type	Name
Index	*Integer 32 Variable*	*CURRENT_INDEX*
Of Table	*Pointer Table*	*IO_POINTERS*
To Pointer	*Pointer Variable*	*TANK_SWITCH_POINTER*

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: -29 = Wrong object type. Most likely caused by moving a pointer table element to a pointer of the wrong type.

-69 = Invalid parameter (null pointer) passed to driver. Use Move to Pointer Table Element to initialize the table entry.

See Also: "Move to Pointer" on page 599
"Move to Pointer Table Element" on page 602,

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.

For Arguments, see next page.

Arguments:

Argument 0	**Argument 1**
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	
E1	
E2	
Event/Reaction Group*	
Float Table	
Float Variable	
G4A8R, G4RAX*	
G4D16R*	
G4D32RS*	
G4EB2	
Generic OptoMMP Device	
GRV-EPIC-PR1	
GRV-EPIC-PR2	
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	
Integer 32 Table	
Integer 32 Variable	
Integer 64 Table	
Integer 64 Variable	
Mistic PID Loop*	
PID Loop	
Pointer Variable	
SNAP-B3000-ENET, SNAP-ENET-RTC**	
SNAP-BRS*	
SNAP-ENET-D64**	
SNAP-ENET-S64**	
SNAP-PAC-EB1	
SNAP-PAC-EB2	
SNAP-PAC-R1	
SNAP-PAC-R1-B	
SNAP-PAC-R2	
SNAP-PAC-SB1	
SNAP-PAC-SB2	
SNAP-UP1-ADS**	
SNAP-UP1-D64**	
SNAP-UP1-M64**	
String Table	
String Variable	
Up Timer Variable	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Move To Pointer		
Argument Name	Type	Name
Object	*Digital Output*	*PUMP_VALVE*
Pointer	*Pointer Variable*	*IO_POINTER*

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" on page 595
"Pointer Equal to Null?" on page 605

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.

For Arguments, see next page.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
Object	**Index**	**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		
E1		
E2		
Event/Reaction Group*		
Float Table		
Float Variable		
G4A8R, G4RAX*		
G4D16R*		
G4D32RS*		
G4EB2		
Generic OptoMMP Device		
GRV-EPIC-PR1		
GRV-EPIC-PR2		
GRV-R7-MM1001-10		
GRV-R7-MM2001-10		
GRV-R7-I1VAPM-3		
Integer 32 Table		
Integer 32 Variable		
Integer 64 Table		
Integer 64 Variable		
Mistic PID Loop*		
PID Loop		
Pointer Variable		
SNAP-B3000-ENET, SNAP-ENET-RTC**		
SNAP-BRS*		
SNAP-ENET-D64**		
SNAP-ENET-S64**		
SNAP-PAC-EB1		
SNAP-PAC-EB2		
SNAP-PAC-R1		
SNAP-PAC-R1-B		
SNAP-PAC-R2		
SNAP-PAC-SB1		
SNAP-PAC-SB2		
SNAP-UP1-ADS**		
SNAP-UP1-D64**		
SNAP-UP1-M64**		
String Table		
String Variable		
Up Timer Variable		

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Move to Pointer Table Element		
Argument Name	Type	Name
Object	*Integer 32 Variable*	*Valve_One*
Index	*Integer 32 Variable*	*Current_Index*
Of Table	*Pointer Table*	*Digital_Outputs*

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" on page 598
"Pointer Table Element Equal to Null?" on page 606

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 0
Pointer
Pointer Variable

Condition Block Example:

Pointer Equal to NULL?		
Argument Name	Type	Name
Pointer	*Pointer Variable*	*IO_Pointer*

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 the *PAC Control User's Guide* (form 1700).
- If you try to perform an operation on a NULL pointer, an error will be posted in the message queue.

See Also:

"Clear Pointer" on page 595
"Move to Pointer" on page 599

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

Condition Block Example:

Pointer Table Element Equal to NULL?		
Argument Name	Type	Name
Index	*Integer 32 Variable*	*Current_Index*
Of Table	*Pointer Table*	*IO_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 the *PAC Control User's Guide* (form 1700).
- 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" on page 596
"Move to Pointer Table Element" on page 602

20: Simulation Commands

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.

Condition Block Example:

Communication to All I/O Points Enabled?
No arguments

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 609
"Enable Communication to All I/O Points" on page 615
"Disable Communication to Point" on page 614
"I/O Point Communication Enabled?" on page 620

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.

Condition Block Example:

Communication to All I/O Units Enabled?
No arguments

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 610
"Enable Communication to All I/O Units" on page 616
"Disable Communication to I/O Unit" on page 611
"I/O Unit Communication Enabled?" on page 621

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.

Action Block Example:

Disable Communication to All I/O Points
No arguments

OptoScript Example:

`DisableCommunicationToAllIoPoints()`

```
DisableCommunicationToAllIoPoints();
```

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

See Also: "Enable Communication to All I/O Points" on page 615

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 PAC Control can still communicate to the I/O units, since it ignores the disabled flag.

Arguments: None.

Action Block Example:

Disable Communication to All I/O Units
No arguments

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" on page 616

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 0
[Value]
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Disable Communication to I/O Unit		
Argument Name	Type	Name
(none)	*GRV-EPIC-PR1*	*Vapor_Extraction*

OptoScript Example:

DisableCommunicationToIoUnit(*Argument 0*)

```
DisableCommunicationToIoUnit(Vapor_Extraction);
```

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

Notes:

- Communication to I/O units is normally disabled using PAC Control.
- 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 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 I/O Unit as Binary Value 64: 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, and so forth.)

  Repeat 1 through 9.

See Also:

"Enable Communication to I/O Unit" on page 617

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 PAC R-series controller, the PID runs on the I/O side, not the control side. While communication to the PID is disabled, any PAC Control 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 0
[Value]
PID Loop

Action Block Example:

Disable Communication to PID Loop		
Argument Name	Type	Name
(none)	*PID Loop*	*HEATER_3*

OptoScript Example:

DisableCommunicationToPidLoop(*Argument 0***)**

```
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" on page 618
"Set PID Mode" on page 587

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 0
[Value]
Analog Input
Analog Output
Digital Input
Digital Output

Action Block Example:

Disable Communication to Point		
Argument Name	Type	Name
(none)	*Analog Input*	*TANK_LEVEL*

OptoScript Example:

DisableCommunicationToPoint(*Argument 0*)

```
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" on page 619
"I/O Point Communication Enabled?" on page 620

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.

Action Block Example:

Enable Communication to All I/O Points
No arguments

OptoScript Example:

`EnableCommunicationToAllIoPoints()`

```
EnableCommunicationToAllIoPoints();
```

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

See Also: "Disable Communication to All I/O Points" on page 609
"I/O Point Communication Enabled?" on page 620

Enable Communication to All I/O Units

Simulation Action

Function: Attempts to bring the I/O Units back online.

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: Sends a test message (Powerup Clear command) to the brains. If the test message is successful it will enable communication and configure the I/O if necessary.

Arguments: None.

Action Block Example:

Enable Communication to All I/O Units
No arguments

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" on page 610

Enable Communication to I/O Unit

Simulation Action

Function: Attempts to bring the I/O Unit back online.

Typical Use: To re-establish communication between the control engine and the I/O unit after it was automatically or manually disabled.

Details: Sends a test message (Powerup Clear command) to the brain. If the test message is successful, it will enable communication and configure the I/O if necessary.

Arguments:

Argument 0	Argument 0 (continued)
[Value]	SNAP-B3000-ENET, SNAP-ENET-RTC**
B100*	SNAP-BRS*
B200*	SNAP-ENET-D64**
B3000 (Analog)*	SNAP-ENET-S64**
B3000 (Digital)*	SNAP-PAC-EB1
E1	SNAP-PAC-EB2
E2	SNAP-PAC-R1
G4A8R, G4RAX*	SNAP-PAC-R1-B
G4D16R*	SNAP-PAC-R2
G4D32RS*	SNAP-PAC-SB1
G4EB2	SNAP-PAC-SB2
Generic OptoMMP Device	SNAP-UP1-ADS**
GRV-EPIC-PR1	SNAP-UP1-D64**
GRV-EPIC-PR2	SNAP-UP1-M64**
GRV-R7-MM1001-10	
GRV-R7-MM2001-10	
GRV-R7-I1VAPM-3	

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

Enable Communication to I/O Unit		
Argument Name	Type	Name
(none)	*GRV-EPIC-PR1*	*Vapor_Extraction*

OptoScript Example:

`EnableCommunicationToIoUnit(`*Argument 0*`)`

```
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" on page 611

Enable Communication to PID Loop

Simulation Action

NOTE: This command is used for PID loops in PAC Control; 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 PAC R-series controller, the PID runs on the I/O side, not the control side. While communication to the PID is enabled, any PAC Control command that refers to it by name will have full access.

Arguments:

Argument 0
[Value]
PID Loop

Action Block Example:

Enable Communication to PID Loop		
Argument Name	Type	Name
(none)	*PID Loop*	*HEATER_3*

OptoScript Example:

EnableCommunicationToPidLoop(*Argument 0*)

```
EnableCommunicationToPidLoop(HEATER_3);
```

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

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 0
[Value]
Analog Input
Analog Output
Digital Input
Digital Output

Action Block Example:

Enable Communication to Point

Argument Name	Type	Name
(none)	*Analog Input*	*TANK_LEVEL*

OptoScript Example:

EnableCommunicationToPoint(*Argument 0***)**

```
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" on page 614
"I/O Point Communication Enabled?" on page 620

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: If communication is enabled, the logic will take the True path.
If communication is *not* enabled, the logic will take the False path.

Arguments:

Argument 0
I/O Point
Analog Input
Analog Output
Digital Input
Digital Output

Condition Block Example:

I/O Point Communication Enabled?		
Argument Name	Type	Name
I/O Point	*Analog Input*	*PUMP_3_STATUS*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

See Also:

"Enable Communication to Point" on page 619
"Disable Communication to Point" on page 614
"I/O Unit Communication Enabled?" on page 621

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: If communication is enabled, the logic will take the True path.
If communication is *not* enabled, the logic will take the False path.

Arguments:

Argument 0
I/O Unit
B100*
B200*
B3000 (Analog)*
B3000 (Digital)*
E1
E2
G4A8R, G4RAX*
G4D16R*
G4D32RS*
G4EB2
Generic OptoMMP Device
GRV-EPIC-PR1
GRV-EPIC-PR2
GRV-R7-MM1001-10
GRV-R7-MM2001-10
GRV-R7-I1VAPM-3
SNAP-B3000-ENET, SNAP-ENET-RTC**
SNAP-BRS*
SNAP-ENET-D64**
SNAP-ENET-S64**
SNAP-PAC-EB1
SNAP-PAC-EB2
SNAP-PAC-R1
SNAP-PAC-R1-B
SNAP-PAC-R2
SNAP-PAC-SB1
SNAP-PAC-SB2
SNAP-UP1-ADS**
SNAP-UP1-D64**
SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).
** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Condition Block Example:

I/O Unit Communication Enabled?		
Argument Name	Type	Name
I/O Unit	*SNAP-PAC-EB1*	*PUMP_HOUSE*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

See Also: "Enable Communication to I/O Unit" on page 617
"Disable Communication to I/O Unit" on page 611
"I/O Point Communication Enabled?" on page 620
"I/O Unit Ready?" on page 309

IVAL Set Analog Filtered Value

Simulation Action

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

Function: Used with a *mistic* I/O unit, this command writes to the internal value (IVAL) of a filtered analog input. Filtering is activated by the command, Set Analog Filter Weight.

Typical Use: Simulation, testing, and certification where communication to the I/O point or I/O unit is disabled.

Details:
- The strategy 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 a real field signal.
- When analog input filtering is implemented with *mistic* brains, the analog input value is not affected. This command allows the filtered analog input value to be read.

Arguments:

Argument 0	Argument 1
To	**On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set Analog Filter Value		
Argument Name	Type	Name
To	*Float Literal*	*5.63*
On Point	*Analog Input*	*PROCESS_PH*

OptoScript Example:

IvalSetAnalogFilteredValue(*To, On Point***)**

```
IvalSetAnalogFilteredValue(5.63, PROCESS_PH);
```

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

Notes:
- For an explanation of the use of IVALs (internal values) and XVALs (external values), see the *PAC Control User's Guide* (form 1700).
- This command does not set the IVAL of the filter *weight*.

See Also:

"Disable Communication to Point" on page 614
"Set Analog Filter Weight" on page 47
"Disable Communication to I/O Unit" on page 611
"Disable Communication to All I/O Units" on page 610

IVAL Set Analog Maximum Value

Simulation Action

Function: Writes to the internal value (IVAL) of the maximum value register of an analog input.

Typical Use: Simulation, testing, and certification where communication to the I/O point or I/O unit is disabled.

Details: The strategy 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 a real field signal.

Arguments:

Argument 0 **To**	Argument 1 **On Point**
Float Literal	Analog Input
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set Analog Max Value

Argument Name	Type	Name
To	*Float Literal*	*5.63*
On Point	*Analog Input*	*PROCESS_PH*

OptoScript Example:

`IvalSetAnalogMaxValue(`*To, On Point***`)`**

```
IvalSetAnalogMaxValue(5.63, PROCESS_PH);
```

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

Notes: For an explanation of the use of IVALs (internal values) and XVALs (external values), see the *PAC Control User's Guide* (form 1700).

"Disable Communication to All I/O Units" on page 610
"Disable Communication to I/O Unit" on page 611
"Disable Communication to Point" on page 614

IVAL Set Analog Minimum Value

Simulation Action

Function: Writes to the internal value (IVAL) of the minimum value register of an analog input.

Typical Use: Simulation, testing, and certification where communication to the I/O point or I/O unit is disabled.

Details: The strategy 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 a real field signal.

Arguments:

Argument 0 **To**	Argument 1 **On Point**
Float Literal Float Variable Integer 32 Literal Integer 32 Variable	Analog Input

Action Block Example:

IVAL Set Analog Min Value

Argument Name	Type	Name
To	*Float Literal*	*5.63*
On Point	*Analog Input*	*PROCESS_PH*

OptoScript Example:

`IvalSetAnalogMinValue(`*To, On Point***`)`**

```
IvalSetAnalogMinValue(5.63, PROCESS_PH);
```

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

Notes: For an explanation of the use of IVALs (internal values) and XVALs (external values), see the *PAC Control User's Guide* (form 1700).

See Also:

"Disable Communication to All I/O Points" on page 609
"Disable Communication to All I/O Units" on page 610
"Disable Communication to I/O Unit" on page 611

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 0 **To**	Argument 1 **On Point**
Float Literal	Analog Input
Float Variable	Analog Output
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set Analog Point

Argument Name	Type	Name
To	*Float Literal*	*5.63*
On Point	*Analog Input*	*PROCESS_PH*

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" on page 610
"Disable Communication to I/O Unit" on page 611

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 a counter is 0 to 4,294,967,295 counts.
- Valid range for a quadrature counter is -2,147,483,647 to 2,147,483,648 counts.

Arguments:

Argument 0	Argument 1
To	**On Point**
Integer 32 Literal	Counter
Integer 32 Variable	Quadrature Counter
Integer 64 Literal	
Integer 64 Variable	

Action Block Example:

IVAL Set Counter		
Argument Name	Type	Name
To	*Integer 32 Literal*	*2484*
On Point	*Counter*	*PROCESS_FLOW_TOTAL*

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" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Set Frequency

Simulation Action

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 0	Argument 1
To	**On Point**
Integer 32 Literal	Frequency
Integer 32 Variable	

Action Block Example:

IVAL Set Frequency		
Argument Name	Type	Name
To	*Integer 32 Literal*	*400*
On Point	*Frequency*	*Process_Flow_Rate*

OptoScript Example:

IvalSetFrequency(*To, On Point***)**

```
IvalSetFrequency(400, Process_Flow_Rate);
```

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

Dependencies: Available on GRV-EPIC processors, SNAP-PAC-R1 and -R1-B controllers, and on SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also: "Disable Communication to All I/O Units" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Set I/O Unit from MOMO Masks

Simulation Action

Function: Writes to the internal value (IVAL) of multiple 4-channel digital input and output points for the specified I/O unit, simultaneously with a single command.

Typical Use: Simulation, testing, and certification of a selected group of 4-channel digital inputs and outputs 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 input and output points.
- To turn a point on, set the respective bit in *On Mask* (Argument 0)—which is the on bitmask—to a value of 1, and set the same bit of *Off Mask* (Argument 1) to a value of 0.
- To turn a point off, set the respective bit in *Off Mask* (Argument 1)—which is the off bitmask—to a value of 1, and set the same bit of *On Mask* to a value of 0.
- To leave a point unaffected, set its bits to a value of 0 in both *On Mask* and *Off Mask*.
- The least significant bit corresponds to point zero.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
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	E1
Integer 64 Variable	Integer 64 Variable	G4A8R, G4RAX*
		G4D16R*
		G4D32RS*
		G4EB2
		GRV-EPIC-PR1
		GRV-EPIC-PR2
		GRV-R7-MM1001-10
		GRV-R7-MM2001-10
		GRV-R7-I1VAPM-3
		SNAP-B3000-ENET, SNAP-ENET-RTC**
		SNAP-BRS*
		SNAP-ENET-D64**
		SNAP-ENET-S64**
		SNAP-PAC-EB1
		SNAP-PAC-EB2
		SNAP-PAC-R1
		SNAP-PAC-R1-B
		SNAP-PAC-R2
		SNAP-PAC-SB1
		SNAP-PAC-SB2
		SNAP-UP1-ADS**
		SNAP-UP1-D64**
		SNAP-UP1-M64**

* Available only in PAC Control Professional when *mistic* products are enabled (File > Strategy Options > Legacy tab > Mistic I/O units and commands).

** Available only when Legacy products are enabled (File > Strategy Options > Legacy tab > Ethernet, Ultimate, and Simple I/O units).

Action Block Example:

IVAL Set I/O Unit from MOMO Masks		
Argument Name	Type	Name
On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
On I/O Unit	*SNAP-PAC-R1*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 610
"Disable Communication to I/O Unit" on page 611

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 0	Argument 1
To	**On Point**
Integer 32 Literal	Digital Input
Integer 32 Variable	

Action Block Example:

IVAL Set Off-Latch		
Argument Name	Type	Name
To	*Integer 32 Literal*	*-1*
On Point	*Digital Input*	*Process_Stop_Button*

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" on page 610
"Disable Communication to I/O Unit" on page 611

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.

Arguments:

Argument 0	Argument 1
To	**On Point**
Float Literal	Off Pulse
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set Off-Pulse		
Argument Name	Type	Name
To	*Integer 32 Literal*	*150000*
On Point	*Off Pulse*	*TIME_PULSE_INPUT*

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" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Set Off-Totalizer

Simulation Action

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 0	Argument 1
To	**On Point**
Float Literal	Off Totalizer
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set Off-Totalizer

Argument Name	Type	Name
To	*Integer 32 Literal*	*36000000*
On Point	*Totalizer Off*	*PUMP_OFF_TIME*

OptoScript Example:

IvalSetOffTotalizer(*To, On Point***)**

```
IvalSetOffTotalizer(36000000, PUMP_OFF_TIME);
```

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

Dependencies: Available on SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also:

"Get Off-Time Totalizer" on page 183
"Disable Communication to All I/O Units" on page 610
"Disable Communication to I/O Unit" on page 611

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 0	Argument 1
To	**On Point**
Integer 32 Literal	Digital Input
Integer 32 Variable	

Action Block Example:

IVAL Set On-Latch

Argument Name	Type	Name
To	*Integer 32 Literal*	*0*
On Point	*Digital Input*	*Process_Start_Button*

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" on page 610
"Disable Communication to I/O Unit" on page 611

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 0	Argument 1
To	**On Point**
Float Literal	On Pulse
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set On-Pulse		
Argument Name	Type	Name
To	*Integer 32 Literal*	*133300*
On Point	*On Pulse*	*TIME_PULSE_INPUT*

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" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Set On-Totalizer

Simulation Action

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 0	Argument 1
To	**On Point**
Float Literal	On Totalizer
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set On-Totalizer		
Argument Name	Type	Name
To	*Integer 32 Literal*	*72000000*
On Point	*On Totalizer*	*PUMP_ON_TIME*

OptoScript Example:

IvalSetOnTotalizer(*To, On Point***)**

```
IvalSetOnTotalizer(72000000, PUMP_ON_TIME);
```

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

Dependencies: Available on SNAP PAC R-series controllers, and on SNAP PAC EB- and SB-series brains with firmware 8.2 or later.

See Also:
"Get On-Time Totalizer" on page 187
"Disable Communication to All I/O Units" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Set Period

Simulation Action

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 0	Argument 1
To	**On Point**
Float Literal	Period
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set Period

Argument Name	Type	Name
To	*Float Literal*	*5.63*
On Point	*Period*	*Pump_On_Time*

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.

Dependencies: Available on GRV-EPIC processors, SNAP-PAC-R1 and -R1-B controllers, and on SNAP-PAC-EB1 and SNAP-PAC-SB1 brains with firmware 8.1 or later.

See Also: "Get Period" on page 188
"Disable Communication to All I/O Units" on page 610
"Disable Communication to I/O Unit" on page 611

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 0	Argument 1
To	**On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set TPO Percent

Argument Name	Type	Name
To	*Float Literal*	*43.66*
On Point	*TPO*	*ZONE_3_HEATER*

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" on page 610
"Disable Communication to I/O Unit" on page 611

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 0	Argument 1
To	**On Point**
Float Literal	TPO
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example:

IVAL Set TPO Period		
Argument Name	Type	Name
To	*Float Literal*	*1.00*
On Point	*TPO*	*ZONE_3_HEATER*

OptoScript Example:

`IvalSetTpoPeriod(`*To, 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" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Turn Off

Simulation Action

Function: Writes to the internal value (IVAL) of a digital point.

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 0
[Value]
Digital Input
Digital Output

Action Block Example:

IVAL Turn Off		
Argument Name	Type	Name
(none)	*Digital Input*	*Process_Start_Button*

OptoScript Example:

IvalTurnOff(*Argument 0*)

```
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" on page 610
"Disable Communication to I/O Unit" on page 611

IVAL Turn On

Simulation Action

Function: Writes to the internal value (IVAL) of a digital point.

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 0
[Value]
Digital Input
Digital Output

Action Block Example:

IVAL Turn On

Argument Name	Type	Name
(none)	*Digital Input*	*PROCESS_START_BUTTON*

OptoScript Example:

IvalTurnOn(*Argument 0***)**

```
IvalTurnOn(PROCESS_START_BUTTON);
```

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

Notes: Turns On the IVAL for the specified point.

PID Loop Communication Enabled?

Simulation Condition

NOTE: This command is used for PID loops in PAC Control; 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:

- If communication is enabled, the logic will take the True path.
  If communication is *not* enabled, the logic will take the False path.
- Because the PID runs on the I/O unit, not in the control engine, any PAC Control command referring to a PID loop by name will not affect the PID while communication to it is disabled. Even on a SNAP PAC R-series controller, 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 0
PID Loop
PID Loop

Condition Block Example:

PID Loop Communication Enabled?		
Argument Name	Type	Name
PID Loop	*PID Loop*	*FACTORY_HEAT_2BA*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

See Also:

"Enable Communication to PID Loop" on page 618

"I/O Point Communication Enabled?" on page 620

21: String Commands

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:

- The character is represented by an ASCII value. (See the ASCII table in the *PAC Control User's Guide*, form 1700.) For example, a space is a character 32, a "1" is a character 49, and an exclamation point (!) is a character 33.
- 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.
- Quotes (" ") are used in OptoScript code, but not in standard PAC Control code.

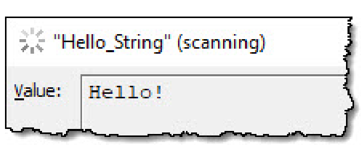
Arguments:

Argument 0	**Argument 1**
Append	**To**
Float Literal	String Variable
Float Variable	
Integer 32 Literal	
Integer 32 Variable	

Action Block Example: The following example appends the ASCII code 33 (`!`, an exclamation mark) to a string; for example, `Hello` would become `Hello!`

Append Character to String		
Argument Name	Type	Name
Append	*Integer 32 Literal*	*33*
To	*String Variable*	*Hello_String*

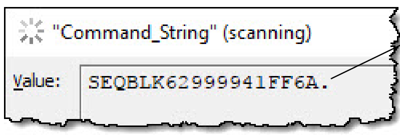
Example Result:



The following example appends an ETX (ASCII character 3) to a string. An ETX or some other terminating character may be required when sending commands to serial devices, such as bar code printers, scales, or single-loop controllers.

Append Character to String		
Argument Name	Type	Name
Append	*Integer 32 Literal*	*3*
To	*String Variable*	*Command_String*

Example Result:



In this example, the ETX appears as a period (.) at the end of the command.

OptoScript Example:

In OptoScript, this function is built-in. Instead of using a command, you use the += and = operators and a variable or string. (In OptoScript, the string must be in quotes.)

Alternatively, the operator can be followed by the Chr keyword and an ASCII code in parentheses (to convert an ASCII code to its text equivalent).

The first example above can be written in OptoScript in either of these ways:

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

The OptoScript code for the second example is:

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

Notes:

- For more information, see "String Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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" on page 645

Append String to String

String Action

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

Typical Use: To build strings.

Details:

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

Arguments:

Argument 0	Argument 1
Append	**To**
String Literal	String Variable
String Variable	

Action Block Example: The following example appends the string `world` to a string. For example, `Hello` would become `Hello world` (note the space before the "w" in " world").

Append String to String		
Argument Name	Type	Name
Append	*String Literal*	*world*
To	*String Variable*	*Hello_String*

OptoScript Example: In OptoScript, this function is built-in. Instead of using a command, you use the += and = operators and a variable or string. (In OptoScript, the string must be in quotes.)

The example above can be written in OptoScript like this:

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

Notes:

- For more information, see "String Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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" on page 643

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" in the *PAC Control User's Guide* (form 1700). 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.
- Quotes (" ") are used in OptoScript code, but not in standard PAC Control code.
- The result returned (*Put Result in*, Argument 2) indicates the relationship between the two strings:

-1 =	*Compare* string (Argument 0) is less than *With* string (Argument 1)
0 =	Strings are equal
1 =	*Compare* string is greater than *With* string

Examples:

Compare	*With*	*Put Result in*	Relationship
"abcDEF"	"abcDEF"	0	Strings equal
"abcDEF"	"abcdef"	-1	*Compare* string is less
"abcDEF"	"abcdEF"	-1	*Compare* string is less
"abcDEF"	"abcDEFG"	-1	*Compare* string is less
"abcDEF"	"ABCDEF"	1	*Compare* string is greater
"abcDEF"	"AbcDEF"	1	*Compare* string is greater
"abcDEF"	"abcDE"	1	*Compare* string is greater
"abcDEF"	"aBcDEF"	1	*Compare* string is greater
"abcDEF"	"9abcDEF"	1	*Compare* string is greater
"abcDEF"	"DEFabc"	1	*Compare* string is greater

Arguments:

Argument 0	Argument 1	Argument 2
Compare	**With**	**Put Result in**
String Literal	String Literal	Integer 32 Variable
String Variable	String Variable	

Action Block Example:

Compare Strings		
Argument Name	Type	Name
Compare	*String Variable*	*Search_Name*
With	*String Variable*	*Current_Name*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

`CompareStrings(`*Compare, With***`)`**

```
nResult = 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "String Commands" in the *PAC Control User's Guide* (form 1700).

See Also: "Test Equal Strings" on page 697

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:

- *Length* (Argument 1) specifies the final length of the resulting string, including the decimal point. Leading spaces (character 32) are added if required.
- *Decimals* (Argument 2) 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 0	Argument 1	Argument 2	Argument 3
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			

Action Block 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

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

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:

- For more information, see "String Commands" and "Using OptoScript" in the *PAC Control User's Guide* (form 1700).
- 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" on page 659
"Convert Number to String" on page 657
"Convert Number to String Field" on page 658

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 PAC Control 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.

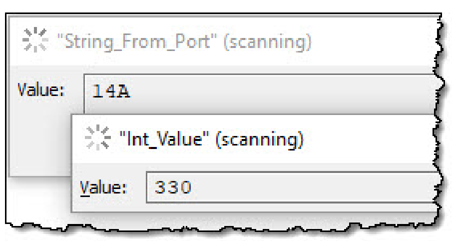
Arguments:

Argument 0	**Argument 1**
Convert	**Put Result in**
String Literal	Float Variable
String Variable	Integer 32 Variable

Action Block Example:

Convert Hex String to Number		
Argument Name	Type	Name
Convert	*String Variable*	*String_From_Port*
Put Result in	*Integer 32 Variable*	*Int_Value*

Example Result:



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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "String Commands" in the *PAC Control User's Guide* (form 1700).
- If the hex string contains an IEEE float, you must use Convert IEEE Hex String to Number.

See Also:

"Convert Number to Hex String" on page 656
"Convert String to Float" on page 659
"Convert String to Integer 32" on page 661
"Convert IEEE Hex String to Number" on page 651

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 PAC Control 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 0	Argument 1
Convert	**Put Result in**
String Literal	Float Variable
String Variable	Integer 32 Variable

Action Block 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

Argument Name	Type	Name
Convert	*String Variable*	*STRING_FROM_PORT*
Put Result in	*Float Variable*	*MY_FLOAT_VALUE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "String Commands" in the *PAC Control User's Guide* (form 1700).

See Also: "Convert Hex String to Number" on page 650

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

Action Block Example:

Convert Integer 32 to IP Address String		
Argument Name	Type	Name
Convert	*Integer 32 Variable*	*IP_Integer*
Put Result in	*String Variable*	*IP_String*

OptoScript Example:

Int32ToIpAddressString(*Convert, Put Result in*)

```
Int32ToIpAddressString(IP_Integer, IP_String);
```

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

Notes: See "String Commands" in the *PAC Control User's Guide* (form 1700).

See Also: "Convert IP Address String to Integer 32" on page 653

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 PAC Control code.

Arguments:

Argument 0	Argument 1
Convert	**Put Result in**
String Literal	Integer 32 Variable
String Variable	

Action Block Example:

Convert IP Address String to Integer 32		
Argument Name	Type	Name
Convert	*String Variable*	*IP_String*
Put Result in	*Integer 32 Variable*	*IP_Integer*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User’s Guide* (form 1700).

Notes: See “String Commands” in the *PAC Control User’s Guide* (form 1700).

Error Code: -45 = String is empty.

-46 = Invalid string.

See Also: “Convert Integer 32 to IP Address String” on page 652

or used in an item such as a Condition block, an OptoScript block, or a mathematical expression

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:

- *Length* (Argument 1) 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 *Convert* (Argument 1) to 8, and use a float variable or literal. If less than eight characters are used, asterisks appear in *Put Result in* (Argument 2), 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 0	**Argument 1**	**Argument 2**
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		

Action Block 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

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

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 648
"Convert Number to Hex String" on page 656
"Convert Number to String" on page 657
"Convert Number to String Field" on page 658

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, and then converted to a hex string.

Arguments:

Argument 0	**Argument 1**
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	

Action Block 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		
Argument Name	Type	Name
Convert	*Integer 32 Variable*	*My_Address*
Put Result in	*String Variable*	*Address_as_Hex*

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 the *PAC Control User's Guide* (form 1700).
- 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 Float to String" on page 648
"Convert Number to String" on page 657
"Convert Number to String Field" on page 658

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: 12n becomes 12n—note no change for integers.
- Floats will have an exponential format.

Arguments:

Argument 0	Argument 1
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	

Action Block 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 1.234000e+01; if MY_VALUE is the integer value 1234, the string becomes 1234):

Convert Number to String		
Argument Name	Type	Name
Convert	*Float Variable*	*My_Value*
Put Result in	*String Variable*	*Value_as_String*

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 661
"Convert Float to String" on page 648

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 *Length* (Argument 1).
- 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 PAC Control code. They are used here for clarity only):

  2n6 becomes " 2n6"—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).

  2n678 becomes 2n678—The six-digit specified length is ignored.

Arguments:

Argument 0	Argument 1	Argument 2
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		

Action Block Example:

Convert Number to String Field		
Argument Name	Type	Name
Convert	*Integer 32 Variable*	*Value*
Length	*Integer 32 Literal*	*6*
Put Result in	*String Variable*	*Value_as_String*

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 the *PAC Control User's Guide* (form 1700).
- 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 Float to String" on page 648
"Convert Number to String" on page 657
"Convert Number to Hex String" on page 656

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 PAC Control 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, "-" (hyphen), 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
"-12.345"	-12.345
"1.23e01"	12.3

Arguments:

Argument 0	Argument 1
Convert	**Put Result in**
String Literal	Float Variable
String Variable	

Action Block Example:

Convert String to Float		
Argument Name	Type	Name
Convert	*String Variable*	*String_from_Port*

Put Result in	*Float Variable*	*Float_Value*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: See "String Commands" in the *PAC Control User's Guide* (form 1700).

See Also:
"Convert Float to String" on page 648
"Convert String to Integer 32" on page 661

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 PAC Control 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 0	Argument 1
Convert	**Put Result in**
String Literal	Integer 32 Variable
String Variable	

Action Block Example:

Convert String to Integer 32		
Argument Name	Type	Name
Convert	*String Variable*	*String_from_Port*
Put Result in	*Integer 32 Variable*	*Int_Value*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "String Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 659
"Convert Number to String" on page 657

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 PAC Control 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 0
Convert
String Literal
String Variable

Argument 1
Put Result in
Integer 64 Variable

Action Block Example:

Convert String to Integer 64		
Argument Name	Type	Name
Convert	*String Variable*	*String_from_Port*
Put Result in	*Integer 64 Variable*	*Int_Value*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "String Commands" in the *PAC Control User's Guide* (form 1700).
- Avoid alpha characters. Use characters 0 to 9.

See Also:

"Convert String to Float" on page 659
"Convert Number to String" on page 657

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, and so forth.

Arguments:
Argument 0
Convert
String Variable

Action Block Example:

Convert String to Lower Case		
Argument Name	Type	Name
Convert	*String Variable*	*IO_COMMAND*

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" on page 666

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, and so forth.

Arguments:

Argument 0
Convert
String Variable

Action Block Example:

Convert String to Upper Case		
Argument Name	Type	Name
Convert	*String Variable*	*IO_COMMAND*

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" on page 665

Find Character in String

String Action

Function: Locate a character within a string. Note that when using the command in an Action Block, you must use the character's ASCII code for the *Find* argument (Argument 0).

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.
- *Put Result in* (Argument 3) 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 minus one.

Arguments:

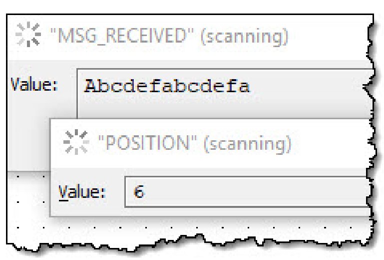
Argument 0	Argument 1	Argument 2	Argument 3
Find	**Start at Index**	**Of String**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	String Variable	

Action Block Example:

Find Character in String

Argument Name	Type	Name
Find	*Integer 32 Literal*	*97*
Start at Index	*Integer 32 Literal*	*0*
Of String	*String Variable*	*MSG_RECEIVED*
Put Result in	*Integer 32 Variable*	*POSITION*

Example Result:



In this example, the first instance of *Find* (ASCII 97, which is the letter 'a') was found in position 6.
Note that position 0 was skipped, because the search is based on ASCII which is case-sensitive.

OptoScript Example:

`FindCharacterInString``(`*Find, Start at Index, Of String*`)`

```
POSITION = FindCharacterInString(97, 0, MSG_RECEIVED);
```

Because the OptoScript compiler can automatically translate a string character, you can also use the following example to search for ASCII code 97 (the letter **a**). Note that the character must be enclosed in straight single quotes.

POSITION = FindCharacterInString('a', 0, MSG_RECEIVED);

This is a function command; it returns the position at which the character is located in the string.

Notes:

- When using an Action Block instruction, you must enter the character's ASCII code; for example, the letter **a** (as shown in the Action Block example) is ASCII code **97**.
- The first position in the string is referred to as position 0.

- When looking for multiple instances of the same character in the string, use the same variable for *Start at Index* (Argument 1) and *Put Result in* (Argument 3):

  ```
  POSITION = FindCharacterInString(97, POSITION, MSG_RECEIVED);
  ```

  Then, increment the variable after each find so the same character won't be found repeatedly.

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" on page 669

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 PAC Control code.
- The search is case-sensitive.
- The search begins at the location specified so that multiple occurrences of the same substring can be found.
- *Put Result in* (Argument 3) 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 minus one, 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 0	Argument 1	Argument 2	Argument 3
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	

Action Block Example:

Find Substring in String		
Argument Name	Type	Name
Find	*String Literal*	*SHIFT*
Start at Index	*Integer 32 Variable*	*INDEX*
Of String	*String Variable*	*MSG_RECEIVED*
Put Result in	*Integer 32 Variable*	*POSITION*

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 *Put Result in* (Argument 3).

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" on page 667

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.
- *Start Value* (Argument 0) 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), *Start Value* must be the result of the calculation on the previous character(s).
- *On String* (Argument 1) can contain as little as one character.

Arguments:

Argument 0	Argument 1	Argument 2
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Generate Checksum on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
On String	*String Variable*	*MSG_TO_SEND*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example:

`GenerateChecksumOnString``(`*Start Value, On String*`)`

```
RESULT = 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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** *Length* **(Argument 1) 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" on page 702

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.
- *Start Value* (Argument 0) 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), *Start Value* must be the result of the calculation on the previous character(s).
- *On String* (Argument 1) can contain as little as one character.

Arguments:

Argument 0	Argument 1	Argument 2
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Generate Forward CCITT on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
On String	*String Variable*	*MSG_TO_SEND*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example:

`GenerateForwardCcittOnString``(`*Start Value, On String*`)`

```
RESULT = 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: The forward CCITT can be appended to the string by using the following commands:

1. Convert Number to Formatted Hex String with *Length* (Argument 1) 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: *Put Result in* (Argument 2) will contain the Forward CCITT that was calculated.

See Also:

"Generate Reverse CCITT on String" on page 674
"Generate Forward CRC-16 on String" on page 673
"Generate Reverse CRC-16 on Table (32 bit)" on page 535

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.
- *Start Value* (Argument 0) 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), *Start Value* must be the result of the calculation on the previous character(s).
- *On String* (Argument 1) can contain as little as one character.

Arguments:

Argument 0	Argument 1	Argument 2
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Generate Forward CRC-16 on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
On String	*String Variable*	*MSG_TO_SEND*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example:

`GenerateForwardCrc16OnString``(`*Start Value, On String*`)`

```
RESULT = 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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** *Length* **(Argument 1) 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" on page 675
"Generate Forward CCITT on String" on page 672
"Generate Reverse CRC-16 on Table (32 bit)" on page 535

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.
- *Start Value* (Argument 0) 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), *Start Value* must be the result of the calculation on the previous character(s).
- *On String* (Argument 1) can contain as little as one character.

Arguments:

Argument 0	Argument 1	Argument 2
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Generate Reverse CCITT on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
On String	*String Variable*	*MSG_TO_SEND*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example:

`GenerateReverseCcittOnString``(`*Start Value, On String*`)`

```
RESULT = 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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 *Length* (Argument 1) 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" on page 672
"Generate Reverse CRC-16 on String" on page 675
"Generate Reverse CRC-16 on Table (32 bit)" on page 535

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.
- *Start Value* (Argument 0) 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), *Start Value* must be the result of the calculation on the previous character(s).
- *On String* (Argument 1) can contain as little as one character.

Arguments:

Argument 0	Argument 1	Argument 2
Start Value	**On String**	**Put Result in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Generate Reverse CRC-16 on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
On String	*String Variable*	*MSG_TO_SEND*
Put Result in	*Integer 32 Variable*	*RESULT*

OptoScript Example:

`GenerateReverseCrc16OnString``(`*Start Value, On String*`)`

```
RESULT = 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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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** *Length* **(Argument 1) 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" on page 673
"Generate Reverse CCITT on String" on page 674
"Generate Reverse CRC-16 on Table (32 bit)" on page 535

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 PAC Control code.
- Valid range for *Index* (Argument 1) is 0 (zero) to the string length minus one.
- A negative result (-12) indicates an error in the value of *Index*.

Arguments:

Argument 0	Argument 1	Argument 2
From String	**Index**	**Put Result in**
String Literal	Integer 32 Literal	Float Variable
String Variable	Integer 32 Variable	Integer 32 Variable

Action Block Example: The following example gets the decimal ASCII value for a character in the string "ABC".
If *Index* (Argument 1) is 0 (zero), the returned value will be 65 (the decimal ASCII value for "A").

Get Nth Character

Argument Name	Type	Name
From String	*String Literal*	*ABC*
Index	*Integer 32 Variable*	*INDEX*
Put Result in	*Integer 32 Variable*	*ASCII_VALUE*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "String Commands" in the *PAC Control User's Guide* (form 1700).
- 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" on page 678
"Append Character to String" on page 643
"Get String Length" on page 677

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 PAC Control code.
- An empty string has a length of zero.
- The length of a string contained in a string variable is not the same as the width of the string variable. Width is the maximum string length the string variable can hold and is set in the PAC Control 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 five.
- A string with width 10 containing "Hello " has a length of six (five for "Hello" plus one for the trailing space).

Arguments:

Argument 0	**Argument 1**
Of String	**Put Result in**
String Literal	Float Variable
String Variable	Integer 32 Variable

Action Block 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

Argument Name	Type	Name
Of String	*String Literal*	*MY_STRING*
Put Result in	*Integer 32 Variable*	*STRING_LEN*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- See "String Commands" in the *PAC Control User's Guide* (form 1700).
- Use before Get Nth Character to stay within the string length.

See Also: "Get Nth Character" on page 676

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 PAC Control code.
- Valid range for *Start At Index* (Argument 1) is 0 (zero) to the string length minus one. If it is less than 0 or longer than *From String* (Argument 0), a null string is copied to the substring.
- If the combination of *Start At Index* (Argument 1) and *Num. Characters* (Argument 2) extends 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 0	Argument 1	Argument 2	Argument 3
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	

Action Block Example: The following example gets a single day from the string `MONTUEWEDTHUFRI`.

Get Substring

Argument Name	Type	Name
From String	*String Literal*	*MONTUEWEDTHUFRI*
Start at Index	*Integer 32 Variable*	*INDEX*
Num. Characters	*Integer 32 Literal*	*3*
Put Result in	*String Variable*	*STRING*

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 the *PAC Control User’s Guide* (form 1700).
- 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 *Start At Index* (Argument 1) above.

  See “Move from String Table Element” on page 680 for a similar example.

See Also: "Get Nth Character" on page 676

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 PAC Control code.
- Valid range for *From Index* (Argument 0) is 0 (zero) to the table length minus 1 (that is, size – 1).
- If the string moved from the table is longer than the string variable width of *To* (Argument 2), it is truncated to fit.

Arguments:

Argument 0	Argument 1	Argument 2
From Index	**Of Table**	**To**
Integer 32 Literal	String Table	String Variable
Integer 32 Variable		

Action Block 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		
Argument Name	Type	Name
From Index	*Integer 32 Variable*	*INDEX*
Of Table	*String Table*	*STRING_TABLE*
To	*String Variable*	*STRING*

The results of this command are:

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 the *PAC Control User’s Guide* (form 1700).
- 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 Element or Move to String Table Elements 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" on page 683
"String Equal to String Table Element?" on page 694
"Get Substring" on page 678
"Get Length of Table" on page 537

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 PAC Control 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 0	Argument 1
Move String	**To**
String Literal	String Variable
String Variable	

Action Block Example:

The following example initializes a string variable to `Hello`:

Move String		
Argument Name	Type	Name
Move String	*String Literal*	*Hello*
To	*String Variable*	*HELLO_STRING*

The following example clears a string variable:

Move String		
Argument Name	Type	Name
Move String	*String Literal*	
To	*String Variable*	*MY_STRING*

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 645
"Copy Time to String" on page 712

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 PAC Control code.
- Valid range for *To Index* (Argument 1) is 0 (zero) to the table length minus 1 (that is, size – 1).
- Strings with a length greater than the width of the table will be truncated to fit.

Arguments:

Argument 0	Argument 1	Argument 2
From	**To Index**	**Of Table**
String Literal	Integer 32 Literal	String Table
String Variable	Integer 32 Variable	

Action Block Example:

Move to String Table Element		
Argument Name	Type	Name
From	*String Literal*	*MON*
To Index	*Integer 32 Variable*	*INDEX*
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 the *PAC Control User's Guide* (form 1700).
- 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" on page 680
"Get Length of Table" on page 537
"Move to String Table Elements" on page 684

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 PAC Control code.
- Valid range for *Start Index* (Argument 1) is 0 (zero) to the table length minus 1 (that is, 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 0	Argument 1	Argument 2	Argument 3
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	

Action Block Example:

Move to String Table Element		
Argument Name	Type	Name
From	*String Literal*	*MON*
Start Index	*Integer 32 Literal*	*0*
End Index	*Integer 32 Literal*	*6*
Of Table	*String Table*	*DAYS*

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 680
"Get Length of Table" on page 537
"Move to String Table Element" on page 683

Pack Float into String

String Action

Function: Packs the data bytes of a float value into a string.

NOTE: This command is designed to create HART argument strings, but it can also be used to pack data bytes of a float value into a single string packed with other types of data.

Typical Use: To pack float value data into a string along with other types of data to be sent in a single send rather a series of sends.

To unpack data from a string that has been packed with various types of data, see "Unpack String" on page 700.

Details: The arguments are as follows:

- *From Value*: Value to pack into string.
- *To String*: Destination string.
- *Start Index*: Index where the value should be placed in the string.
  - If you specify the index, (and the string is long enough), it will pack the data at the requested location. The string must already contain enough characters; the packed data replaces whatever characters are already there.
  - If you specify -1 as the index, the value is packed and appended to the end of the string.
- *Width*: The number of bytes the value should occupy. Currently, 4 is the only valid value.
- *Endian Type*: The following Endian type examples use 0x12345678 as the source value:
  - 0 - Big Endian (0x12 0x34 0x56 0x78)
  - 1 - Little Endian (0x78 0x56 0x34 0x12)
  - 2 - Modbus Big Endian (0x12 0x34 0x56 0x78)
  - 3 - Modbus Little-Endian (0x56 0x78 0x12 0x34)
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0	Argument 1	Argument 2
From Value	**To String**	**Start Index**
Float Literal	String Variable	Integer 32 Literal
Float Variable		Integer 32 Variable

Argument 3	Argument 4	Argument 5
Width	**Endian Type**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Pack Float into String		
Argument Name	Type	Name
From	*Float Variable*	*3.14159*
To String	*String Variable*	*sPack*
Start Index	*Integer 32 Variable*	*1*
Width	*Integer 32 Variable*	*4*
Endian Type	*Integer 32 Variable*	*1*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

PackFloatIntoString(*From Value, To String, Start Index, Width, Endian Type***)**

```
Result = PackFloatIntoString(3.14159, sPack, 1, 4, 1);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: To unpack data from a string that has been packed with various types of data, see "Unpack String" on page 700.

Status Codes:

0 = success

-3 = Invalid length

-6 = Invalid data field)

-8 = Invalid data)

-13 = Overflow)

-23 = String too short)

See Also:

"Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691
"Unpack String" on page 700

Pack Integer 32 into String

String Action

Function: Packs the data bytes of a 32-bit integer value into a string.

NOTE: This command is designed to create HART argument strings, but it can also be used to pack data bytes of a 32-bit integer value into a single string packed with other types of data.

Typical Use: To pack 32-bit integer value data into a string along with other types of data to be sent in a single send rather a series of sends.

Details: The arguments are as follows:

- *From Value*: Value to pack into string.
- *To String*: Destination string.
- *Start Index*: Index where the value should be placed in the string.
  - If you specify the index, (and the string is long enough), it will pack the data at the requested location. The string must already contain enough characters; the packed data replaces whatever characters are already there.
  - If you specify -1 as the index, the value is packed and appended to the end of the string.
- *Width*: Number of bytes the value should occupy; 1 – 4 are valid.
- *Endian Type*: The following Endian type examples use 0x12345678 as the source value:
  - 0 - Big Endian (0x12 0x34 0x56 0x78)
  - 1 - Little Endian (0x78 0x56 0x34 0x12)
  - 2 - Modbus Big Endian (0x12 0x34 0x56 0x78)
  - 3 - Modbus Little-Endian (0x56 0x78 0x12 0x34)
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0	Argument 1	Argument 2
From Value	**To String**	**Start Index**
Integer 32 Literal	String Variable	Integer 32 Literal
Integer 32 Variable		Integer 32 Variable

Argument 3	Argument 4	Argument 5
Width	**Endian Type**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Pack Integer 32 into String

Argument Name	Type	Name
From Value	*Integer 32 Variable*	*3*
To String	*String Variable*	*sPack*
Start Index	*Integer 32 Variable*	*1*
Width	*Integer 32 Variable*	*4*
Endian Type	*Integer 32 Variable*	*1*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

PackInt32IntoString(*From Value, To String, Start Index, Width, Endian Type***)**

```
Result = PackInt32IntoString(3, sPack, 1, 4, 1);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: To unpack data from a string that has been packed with various types of data, see "Unpack String" on page 700.

Status Codes: 0 = success

-3 = Invalid length.

-6 = Invalid data field.

-8 = Invalid data.

-13 = Overflow.

-23 = String too short.

See Also:

"Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691
"Unpack String" on page 700

Pack Integer 64 into String

String Action

Function: Packs the data bytes of a 64-bit integer value into a string.

NOTE: This command is designed to create HART argument strings, but it can also be used to pack data bytes of a 64-bit integer value into a single string packed with other types of data.

Typical Use: To pack 64-bit integer value data into a string along with other types of data to be sent in a single send rather a series of sends.

Details: The arguments are:

- *From Value*: Value to pack into string.
- *To String*: Destination string. If you specify the index, (and the string is long enough), it will pack the data at the requested location. The string must already contain enough characters; the packed data replaces whatever characters are already there.
- *Start Index*: Index where the value should be placed in the string.
  - If you specify the index, (and the string is long enough), it will pack the data at the requested location. The string must already contain enough characters; the packed data replaces whatever characters are already there.
  - If you specify -1 as the index, the value is packed and appended to the end of the string.
- *Width*: The number of bytes the value should occupy, (1-8 are valid).
- *Endian Type*: The following Endian type examples use 0x12345678 as the source value:
  - 0 - Big Endian (0x12 0x34 0x56 0x78)
  - 1 - Little Endian (0x78 0x56 0x34 0x12)
  - 2 - Modbus Big Endian (0x12 0x34 0x56 0x78)
  - 3 - Modbus Little-Endian (0x56 0x78 0x12 0x34)
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0	Argument 1	Argument 2
From Value	**To String**	**Start Index**
Integer 64 Literal	String Variable	Integer 32 Literal
Integer 64 Variable		Integer 32 Variable

Argument 3	Argument 4	Argument 5
Width	**Endian Type**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Pack Integer 64 into String		
Argument Name	Type	Name
From Value	*Integer 64 Variable*	*3*
To String	*String Variable*	*sPack*
Start Index	*Integer 32 Variable*	*1*
Width	*Integer 32 Variable*	*4*
Endian Type	*Integer 32 Variable*	*1*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

`PackInt64IntoString(`*From Value, To String, Start Index, Width, Endian Type***`)`**

```
Result = PackInt64IntoString(3, sPack, 1, 4, 1);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

To unpack data from a string that has been packed with various types of data, see "Unpack String" on page 700.

Status Codes:

0 = success

-3 = Invalid length.

-6 = Invalid data field.

-8 = Invalid data.

-13 = Overflow.

-23 = String too short.

See Also:

"Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack String into String" on page 691
"Unpack String" on page 700

Pack String into String

String Action

Function: Packs a string value into another string.

NOTE: This command is designed to create HART argument strings, but it can also be used to pack a string into a single string along with other types of data.

Typical Use: To pack a string into another string that is packed with other types of data to be sent in a single send rather a series of sends.

Details: The arguments are:

- *From Value*: Value to pack into string.
- *To String*: Destination string.
- *Start Index*: Index where the value should be placed in the string.
  - If you specify the index, (and the string is long enough), it will pack the data at the requested location. The string must already contain enough characters; the packed data replaces whatever characters are already there.
  - If you specify -1 as the index, the value is packed and appended to the end of the string.
- *Width*: The number of bytes the string should occupy.
- *Data Type*: String data type.
  - 5 – HART-encoded date. 24 bits, MS byte = day, followed by 1-byte month and 1-byte year = current year minus 1900. You give it a date between Jan 1, 1900 and Dec 31, 2155 and it does the conversion for you. The same goes for time: Give it hours, minutes, seconds, and so forth, and it does the math.
  - 6 – HART time (32 bits - Time elapsed since midnight, in 1/32 mSec increments)
  - 7 – Packed ASCII string (6-bit chars packed into 8-bit string, reduces size by 25%)
  - 8 – ASCII string (8-bit characters, simply copies source to destination at specified index and width)
  - 9 – Hex as two-character ASCII (for example, FF, 3E, and so forth.)
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
From Value	**To String**	**Start Index**
String Literal	String Variable	Integer 32 Literal
String Variable		Integer 32 Variable

Argument 3	**Argument 4**	**Argument 5**
Width	**Data Type**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Pack String into String		
Argument Name	Type	Name
From Value	*String Literal*	*3*
To String	*String Variable*	*sPack*
Start Index	*Integer 32 Variable*	*1*
Width	*Integer 32 Variable*	*4*
Data Type	*Integer 32 Variable*	*1*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

`PackStringIntoString(`*From Value, To String, Start Index, Width, Data Type***`)`**

```
Result = PackStringIntoString(3, sPack, 1, 4, 5);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: To unpack data from a string that has been packed with various types of data, see "Unpack String" on page 700.

Status Codes:

0 = success

-3 = Invalid length.

-6 = Invalid data field.

-8 = Invalid data.

-13 = Overflow.

-23 = String too short.

See Also:

"Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Unpack String" on page 700

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 minus one.
- Valid range for the character is 0–255.

Arguments:

Argument 0	Argument 1	Argument 2	Argument 3
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

Action Block Example:

Set Nth Character		
Argument Name	Type	Name
To	*Integer 32 Literal*	*62*
In String	*String Variable*	*MSG_RECEIVED*
At Index	*Integer 32 Variable*	*POSITION*
Put Status In	*Integer 32 Variable*	*STATUS*

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 the *PAC Control User's Guide* (form 1700).
- 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" on page 667
"Get Nth Character" on page 676

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 the value of *Is* (Argument 0) is equal to the value defined by *At Index* (Argument 1) and *Of Table* (Argument 2).
  - If the strings are exactly the same, the logic will take the True path.
  - If the strings are *not* exactly the same, the logic will take the False path.
- This test is case-sensitive. For example, a "T" does not equal a "t."
- Only an exact match on all characters (including leading or trailing spaces) will return a True.
- Quotes (" ") are used in OptoScript code, but not in standard PAC Control code.
- A valid range for *At Index* (Argument 1) is 0 (zero) to the table length (that is, the table size).
- Functionally equivalent to the Test Equal Strings action.

Examples:

Is	Value at *Of Table[At Index]*	Result
"OPTO"	"OPTO"	True
"22"	"22"	True
"OPTO"	"Opto"	False
"2 2"	"22"	False

Arguments:

Argument 0	Argument 1	Argument 2
Is	**At Index**	**Of Table**
String Literal	Integer 32 Literal	String Table
String Variable	Integer 32 Variable	

Condition Block 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.

String Equal to String Table Element?		
Argument Name	Type	Name
Is	*String Variable with Barcode*	*NEW_BARCODE*
At Index	*Integer 32 Variable*	*Loop_Index*
Of Table	*String Table*	*Current_Products*

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 the *PAC Control User's Guide* (form 1700).
- The example shown is only one way to use the `==` operator. For more information on using comparison operators and strings in OptoScript code, see the *PAC Control User's Guide* (form 1700).

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

See Also: "Test Equal Strings" on page 697
"String Equal?" on page 696

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 *Is* (Argument 0) and *To* (Argument 1) are equal.
  - If the strings are exactly the same, the logic will take the True path.
  - If the strings are *not* exactly the same, the logic will take the False path.
- This test is case-sensitive. For example, a "T" does not equal a "t."
- Only an exact match on all characters (including leading or trailing spaces) will return a True.
- Quotes (" ") are used in OptoScript code, but not in standard PAC Control code.
- Functionally equivalent to the Test Equal Strings action.

Examples:

Is	To	Result
"OPTO"	"OPTO"	True
"22"	"22"	True
"OPTO"	"Opto"	False
"2 2"	"22"	False

Arguments:

Argument 0	Argument 1
Is	**To**
String Literal	String Literal
String Variable	String Variable

Condition Block Example:

String Equal?		
Argument Name	Type	Name
Is	*String Variable*	*NEW_ENTRY*
To	*String Variable*	*PASSWORD*

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 the *PAC Control User's Guide* (form 1700).
- The example shown is only one way to use the `==` operator. For more information on using comparison operators and strings in OptoScript code, see the *PAC Control User's Guide* (form 1700).
- Use String Equal to String Table Element? to compare with strings in a table.

See Also:

"Test Equal Strings" on page 697
"String Equal to String Table Element?" on page 694

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 *Compare* (Argument 0) and *With* (Argument 1) are equal, and stores the result in *Put Result in* (Argument 2).
  - If the strings are exactly the same, the result will be a non-zero value (meaning True).
  - If the strings are *not* exactly the same, the result will be 0 (zero, meaning False).

  NOTE: In programming logic, 0 represents False and any non-zero number represents True.
- This test is case-sensitive. For example, a "T" does not equal a "t."
- Only an exact match on all characters (including leading or trailing spaces) will return a True.
- Quotes (" ") are used in OptoScript code, but not in standard PAC Control code.
- The result can be sent directly to a digital output if desired.
- This action is functionally equivalent to the String Equal? condition.

Examples:

Compare	With	Put Result in
OPTO	OPTO	1
22	22	1
OPTO	Opto	0
2 2	22	0

Arguments:

Argument 0	Argument 1	Argument 2
Compare	**With**	**Put Result in**
String Literal	String Literal	Digital Output
String Variable	String Variable	Float Variable
		Integer 32 Variable

Action Block Example: The following example compares a password variable to a string constant. The resulting value in IS_AUTHORIZED could be used at several points in the program to determine if the user has sufficient authorization.

Test Equal Strings		
Argument Name	Type	Name
Compare	*String Variable*	*Password*
With	*String Literal*	*LISA*
Put Result in	*Integer 32 Variable*	*IS_AUTHORIZED*

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		
Argument Name	Type	Name
Compare	*String Variable*	*BARCODE*
With	*String Variable*	*BARCODE_FROM_LIST*
Put Result in	*Integer 32 Variable*	*IS_IN_LIST*

OptoScript Example: For an OptoScript equivalent, see the String Equal? command.

Notes:

- See "String Commands" in the *PAC Control User's Guide* (form 1700).
- Use String Equal to String Table Element? to compare with strings in a table.

See Also: "Compare Strings" on page 646
"String Equal?" on page 696
"String Equal to String Table Element?" on page 694

Trim String

String Action

Function: Trims spaces at the ends of a text string.

Typical Use: Use this command to trim the beginning or the ending spaces of a text string, or both.

Details: For *Option* (Argument 1), the possible values are:

1 = Trims leading spaces only

2 = Trims trailing spaces only

3 = Trims both leading and trailing spaces

Arguments:

Argument 0	Argument 1
String	**Option**
String Variable	Integer 32 Literal
	Integer 32 Variable

Action Block Example:

Trim String		
Argument Name	Type	Name
String	*String Table*	*MyTextString*
Option	*Integer 32 Variable*	*3*

OptoScript Example:

TrimString(*String, Option***)**

```
TrimString( MyTextString, 3 );
```

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

Notes:

- If result = 0, the spaces at the ends of the string were trimmed successfully.
- If result < 0, there was an error executing this command.

See Also:

"Append Character to String" on page 643
"Append String to String" on page 645

Unpack String

String Action

Function: Unpacks various types of data from a string that has been packed.

NOTE: This command is designed to unpack HART argument strings, but it can also be used to unpack various other types of data.

Typical Use: To unpack data back to their original types from a string that was packed with multiple data types.

Details: The arguments are:

- *From String*: String to unpack.
- *Start Index*: Index where the value should be retrieved from the string.
- *Width*: The number of bytes the value occupies in the string. When unpacking data from a string, some types imply the number of bytes. For those types, the width will be ignored.

  Examples:
  - Date – 3 bytes
  - Time – 4 bytes
  - Single-precision floats, (the only type we currently support) - 4 bytes
- *To Value*: Destination variable.
- *Data Type*: Use one of the following types:

  0 – Auto (type is determined by the destination variable)

  1 – Integer 32 Variable

  2 – Integer 64 Variable

  3 – Float 32

  4 – (Reserved for future use)

  5 – HART-Encoded Date. For *From String* (Argument 0), enter the date using numeric format, for example "3/3/14". It will be converted automatically to the HART format.

  6 – HART-Encoded Time. For *From String* (Argument 0), enter the time military time, for example, "14:27:32". It will be converted automatically to the HART format.

  7 – Packed ASCII Data (6-bit chars packed into 8-bit string, reduces size by 25%)

  8 – String (8-bit chars, simply copies source to destination at specified index and width)

  9 – ASCII Hex (as two-char ASCII, e.g. FF, 3E, and so forth.)
- *Endian Type*: The following Endian type examples use 0x12345678 as the source value:
  - 0 - Big Endian (0x12 0x34 0x56 0x78)
  - 1 - Little Endian (0x78 0x56 0x34 0x12)
  - 2 - Modbus Big Endian (0x12 0x34 0x56 0x78)
  - 3 - Modbus Little-Endian (0x56 0x78 0x12 0x34)
- *Put Result in*: Indicates success or failure of the operation.

Arguments:

Argument 0 **From String**	Argument 1 **Start Index**	Argument 2 **Width**	Argument 3 **To Value**
String Variable	Integer 32 Literal	Integer 32 Literal	Variable
	Integer 32 Variable	Integer 32 Variable	Integer 32 Variable
			Integer 64 Variable
			String Variable

Argument 4	Argument 5	Argument 6
Data Type	**Endian Type**	**Put Result in**
Integer 32 Literal	Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	Integer 32 Variable	

Action Block Example:

Unpack String		
Argument Name	Type	Name
From String	*String Variable*	*sString*
Start Index	*Integer 32 Variable*	*1*
Width	*Integer 32 Variable*	*4*
To Value	*Integer 32 Variable*	*nValue*
Data Type	*Integer 32 Variable*	*0*
Endian Type	*Integer 32 Variable*	*0*
Put Result in	*Integer 32 Variable*	*Result*

OptoScript Example:

UnpackString(*From String, Start Index, Width, To Value, Data Type, Endian Type***)**

```
Result = UnpackString(sString, 1, 4, nValue, 0, 0);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Status Codes:

0 = success

-3 = Invalid length.

-6 = Invalid data field.

-8 = Invalid data.

-12 = Invalid index.

-13 = Overflow.

-23 = String too short.

-29 = Wrong object type.

-45 = String is empty.

See Also:

"Get HART Unique Address" on page 37
"Receive HART Response" on page 43
"Send/Receive HART Command" on page 45
"Send/Receive HART Command" on page 45
"Pack Float into String" on page 685
"Pack Integer 32 into String" on page 687
"Pack Integer 64 into String" on page 689
"Pack String into String" on page 691

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 0	Argument 1	Argument 2
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Verify Checksum on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*0*
On String	*String Variable*	*RESPONSE_MSG*
Put Status In	*Integer 32 Variable*	*CKSUM_STATUS*

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" on page 670

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 0	Argument 1	Argument 2
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Verify Forward CCITT on String

Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*-1*
On String	*String Variable*	*RESPONSE_MSG*
Put Status In	*Integer 32 Variable*	*CRC_STATUS*

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" on page 705
"Generate Forward CCITT on String" on page 672

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 0	Argument 1	Argument 2
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Verify Forward CRC-16 on String

Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*-1*
On String	*String Variable*	*RESPONSE_VSS*
Put Status In	*Integer 32 Variable*	*CRC_STATUS*

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" on page 706
"Generate Forward CRC-16 on String" on page 673

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 0	Argument 1	Argument 2
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Verify Reverse CCITT on String		
Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*-1*
On String	*String Variable*	*RESPONSE_MSG*
Put Status In	*Integer 32 Variable*	*CRC_STATUS*

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" on page 703
"Generate Reverse CCITT on String" on page 674

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 0	Argument 1	Argument 2
Start Value	**On String**	**Put Status in**
Integer 32 Literal	String Literal	Integer 32 Variable
Integer 32 Variable	String Variable	

Action Block Example:

Verify Reverse CRC-16 on String

Argument Name	Type	Name
Start Value	*Integer 32 Literal*	*-1*
On String	*String Variable*	*RESPONSE_MSG*
Put Status In	*Integer 32 Variable*	*CRC_STATUS*

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" on page 704
"Generate Reverse CRC-16 on String" on page 675

22: Time/Date Commands

Convert Date & Time to NTP Timestamp

Time/Date Action

Function: To take the date and time from an Integer 32 table and convert it to a 64-bit Network Time Protocol (NTP) time stamp, without applying any offset for the time zone in which the controller is located.

NOTE: If you are using a SoftPAC controller, we recommend that you allow Windows to handle the clock, even though this can cause this command to not work as expected. See also the notes for "Set Time Zone Configuration" on page 733 and "Synchronize Clock SNTP" on page 736.

Typical Use: Comparing two times using a simple numerical standard. Useful for security logs or any application requiring the NTP standard.

Details:

- The table must have at least eight elements which are used as follows: 0 = month, 1 = day of the month, 2 = year, 3 = day of the week, 4 = hours, 5 = minutes, 6 = seconds, 7 = milliseconds. (Day of week can be set to zero.)
- Valid dates are January 1, 2001 through December 31, 2135. You must determine that the values passed are valid.

Arguments:

Argument 0	Argument 1	Argument 2
Date & Time	**NTP Timestamp**	**Put Result in**
Integer 32 Table	Integer 64 Variable	Integer 32 Variable

Action Block Example:

Convert Date & Time to NTP Timestamp		
Argument Name	Type	Name
Date & Time	*Integer 32 Table*	*DateTimeIntTbl*
NTP Timestamp	*Integer 64 Variable*	*Current_NTP_Timestamp*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

`DateTimeToNtpTimestamp``(`*Date&Time, NTP Timestamp*`)`

```
nResult = DateTimeToNtpTimestamp(DateTimeIntTbl, Current_NTP_Timestamp);
```

This is a function command; it returns the NTP timestamp. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- If result = 0, the date and time were retrieved successfully.
- If result < 0, there was an error executing this command.

Status Codes: 0 = Success

-3 = Buffer overrun or invalid length. The table must consist of at least eight elements. (See Details above.)

-69 = Null object passed to command.

See Also: "Convert NTP Timestamp to Date & Time" on page 709

Convert NTP Timestamp to Date & Time

Time/Date Action

Function: To take a Network Time Protocol (NTP) time stamp, convert it to the date and time without applying any offset for the time zone in which the controller is located, and then place the date and time in an Integer 32 table.

NOTE: If you are using a SoftPAC controller, we recommend that you allow Windows to handle the clock, even though this can cause this command to not work as expected. See also the notes for *"Set Time Zone Configuration" on page 733* *and* *"Synchronize Clock SNTP" on page 736**.*

Typical Use: Security logs or any other use requiring the NTP standard

Details:

- The target table must have at least eight elements, which are used as follows: 0 = month, 1 = day of the month, 2 = year, 3 = day of the week, 4 = hours, 5 = minutes, 6 = seconds, 7 = milliseconds.
- Valid dates are January 1, 2001 through December 31, 2135.

Arguments:

Argument 0	Argument 1	Argument 2
Date & Time	**NTP Timestamp**	**Put Result in**
Integer 32 Table	Integer 64 Variable	Integer 32 Variable

Action Block Example:

Convert NTP Timestamp to Date & Time		
Argument Name	Type	Name
Date & Time	*Integer 32 Table*	*DateTimeIntTbl*
NTP Timestamp	*Integer 64 Variable*	*Current_NTP_Timestamp*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

`NtpTimeStampToDateTime` **(***Date&Time, NTP Timestamp***)**

```
nResult = NtpTimestampToDateTime(DateTimeIntTbl, Current_NTP_Timestamp);
```

This is a function command; it returns the date and time. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- If result = 0, the date and time were retrieved successfully.
- If result < 0, there was an error executing this command.

Status Codes: 0 = Success

-3 = Buffer overrun or invalid length. The table must consist of at least eight elements. (See Details above.)

-69 = Null object passed to command.

See Also: "Convert Date & Time to NTP Timestamp" on page 707

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 a PAC Control program.

Details:

- If the current date is March 1, 2002, this action would use the string "01/03/2002" as *To* (Argument 0).
- The destination string should have a minimum width of ten.

Arguments:

Argument 0
To
String Variable

Action Block Example:

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

OptoScript Example:

DateToStringDDMMYYYY(*To*)

```
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.
- If the destination string is too short, *none* of the source string is copied.

Queue Error: -44 = String too short.

See Also:

"Copy Date to String (MM/DD/YYYY)" on page 711
"Copy Time to String" on page 712
"Set Date" on page 726
"Set Time" on page 732

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 a PAC Control program.

Details:

- If the current date is March 1, 2002, this action would use the string "03/01/2002" as *To* (Argument 0).
- The destination string should have a minimum width of ten.

Arguments:

Argument 0
To
String Variable

Action Block Example:

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

OptoScript Example:

DateToStringMMDDYYYY(*To*)

```
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)" on page 710
"Copy Time to String" on page 712
"Set Date" on page 726
"Set Time" on page 732

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 a PAC Control 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 use the string "14:35:00" as *To* (Argument 0).
- The destination string should have a minimum width of eight.

Arguments:

Argument 0
To
String Variable

Action Block Example:

Copy Time to String		
Argument Name	Type	Name
To	*String Variable*	*TIME_STRING*

OptoScript Example:

`TimeToString(`*`To`*`)`

```
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)" on page 711
"Copy Date to String (MM/DD/YYYY)" on page 711
"Set Date" on page 726
"Set Time" on page 732

Get Date & Time

Time/Date Action

Function: To read the date and time from a control engine's real-time clock/calendar atomically (in a single transaction) and put each element an integer table.

Typical Use: This command assures that the date and time are retrieved on the same date.

Details:

- Reading the date and time separately could result in the following situation:

  At 11:59:59 PM, a system operator reads the date—for example, April 1, 2011. Next, the operator reads the time, which now reads 00:00:02. The operator then stores the date and time, which reads as April 1, 2011 at 00:00:02. However, the actual date is now April 2, so the timestamp is a complete day behind. This error can also occur if you read the time first; reading atomically (in a single transaction) ensures you get a snapshot of the exact date and time you asked for.
- The table must have at least eight elements which are used as follows: 0 = month, 1 = day of the month, 2 = year, 3 = day of the week, 4 = hours, 5 = minutes, 6 = seconds, 7 = milliseconds.

Arguments:

Argument 0	Argument 1
Table	**Put Result in**
Integer 32 Table	Integer 32 Variable

Action Block Example:

Get Date & Time

Argument Name	Type	Name
Table	*Integer 32 Table*	*DateTimeStrTbl*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

GetDateTime(*Table*)

```
nResult = GetDateTime(DateTimeStrTbl);
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- If result = 0, the date and time were retrieved successfully.
- If result < 0, there was an error executing this command.

Status Codes:

0 = Success

-3 = Buffer overrun or invalid length. The table must consist of at least eight elements. (See Details above.)

-29 = Wrong object type. Must be an Integer 32 Table.

See Also:

"Copy Date to String (DD/MM/YYYY)" on page 710
"Copy Date to String (MM/DD/YYYY)" on page 711
"Copy Time to String" on page 712
"Set Date" on page 726
"Set Time" on page 732

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 a PAC Control 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 use the value 2 as *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Day		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*Day_of_Month*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.

See Also:

"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control 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:

  0 = Sunday
  1 = Monday
  2 = Tuesday
  3 = Wednesday
  4 = Thursday
  5 = Friday
  6 = Saturday
- If the current day is a Wednesday, this action would put a `3` in *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Day of Week		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*Day_of_Week*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.

See Also:

"Get Day" on page 714
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control 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 use the value 14 for *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Hours		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*HOURS*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.

See Also:

"Get Day" on page 714
"Get Day of Week" on page 715
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 0
Put in
Integer 32 Variable

Action Block Example:

Get Julian Day		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*Todays_Julian_Day*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also: "Copy Date to String (MM/DD/YYYY)" on page 711

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 a PAC Control 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 use the value 35 for *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Minutes		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*MINUTES*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.

See Also:

"Get Day" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 use the value 3 for *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Month		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*MONTH*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.

See Also:

"Get Day" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Seconds" on page 720
"Get Year" on page 725
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control 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 use the value 26 for *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Seconds		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*SECONDS*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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.
- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.

See Also:

"Get Seconds Since Midnight" on page 721
"Get Day" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Year" on page 725
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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.

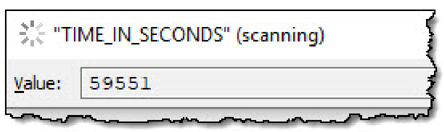
Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Seconds Since Midnight		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*TIME_IN_SECONDS*

Example Results:



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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

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" on page 720

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 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get System Time		
Argument Name	Type	Name
Put In	*Integer 32 Variable*	*TIME_IN_SECONDS*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

See Also: "Get Seconds Since Midnight" on page 721

Get Time Zone Description

Time/Date Action

Function: Get one of the controller's currently-set time zone descriptions as a string.

NOTE: If you are using a SoftPAC controller, we recommend that you allow Windows to handle the clock, even though this can cause this command to not work as expected. See also the notes for *"Set Time Zone Configuration" on page 733* and *"Synchronize Clock SNTP" on page 736*.

Typical Use: Display the name of the controller's time zone(s).

Details:

- *Configuration* (Argument 0) is an integer that specifies which time zone description should be returned. Valid values for *Configuration* are:

  0 - Currently-active time zone
  1 - Controller's first time zone
  2 - Controller's second time zone
- *Description* (Argument 1) is a pointer to a string where the description should be placed
- If everything is OK, a string containing the specified time zone description is stored in *Put Result in* (Argument 2), and the code for success (0) is returned. Otherwise, the string is emptied, and an error code is returned.

Arguments:

Argument 0	Argument 1	Argument 2
Configuration	**Description**	**Put Result in**
Integer 32 Literal	String Variable	Integer 32 Variable
Integer 32 Variable		

Action Block Example:

Get Time Zone Description		
Argument Name	Type	Name
Configuration	*Integer 32 Literal*	*0*
Description	*String Variable*	*StrDescription*
Put Result in	*Integer 32 Variable*	*Time_Zone_Description*

OptoScript Example:

GetTimeZoneDescription(*Configuration, Description*)

```
Time_Zone_Description = GetTimeZoneDescription(0, StrDescription);
```

This is a function command; it returns the specified time zone description. The returned value can be consumed by a variable (as in the example shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes: If the destination string is too short, only as much as will fit in the string is copied, and a -23 error code is returned.

Status Codes:

0 = Success

-6 = Invalid data field. Something other than 0,1, or 2 was used in *Configuration* (Argument 0).

-23 = String too short.

See Also:

"Set Time Zone Configuration" on page 733
"Get Minutes" on page 718
"Synchronize Clock SNTP" on page 736

Get Time Zone Offset

Time/Date Action

Function: Get the offset, in seconds from UTC (Coordinated Universal Time), of one of the controller's currently-set time zones.

NOTE: If you are using a SoftPAC controller, we recommend that you allow Windows to handle the clock, even though this can cause this command to not work as expected. See also the notes for “Set Time Zone Configuration” on page 733 and “Synchronize Clock SNTP” on page 736.

Typical Use: Convert between local time and UTC.

Details:

- *Configuration* (Argument 0) is an integer that specifies which time zone offset should be returned. Valid values for *Configuration* are:
- 0 - Currently-active time zone
  1 - Controller's first time zone
  2 - Controller's second time zone
- If everything is OK, the signed offset (in seconds) of the specified time zone from UTC is returned.

Arguments:

Argument 0	Argument 1
Configuration	**Put Result in**
Integer 32 Literal	Integer 32 Variable
Integer 32 Variable	

Action Block Example:

Get Time Zone Offset		
Argument Name	Type	Name
Configuration	*Integer 32 Literal*	*0*
Put Result in	*Integer 32 Variable*	*OFFSET*

OptoScript Example:

GetTimeZoneOffset(*Configuration*)

```
OFFSET = GetTimeZoneOffset(0);
```

This is a function command; it returns the signed offset, in seconds, of the specified time zone from UTC. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Status Codes: -1 = Error

See Also:

“Set Time Zone Configuration” on page 733
“Get Time Zone Description” on page 723
“Synchronize Clock SNTP” on page 736

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 a PAC Control 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 use the value 2002 for *Put In* (Argument 0).

Arguments:

Argument 0
Put in
Float Variable
Integer 32 Variable

Action Block Example:

Get Year		
Argument Name	Type	Name
Put in	*Integer 32 Variable*	*YEAR*

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- If you need to read more than one element of time from the controller (for example, seconds, minutes, and hours), use "Get Date & Time" on page 713. It's easier and more accurate than getting individual elements separately.
- 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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Set Day" on page 727
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control program.

Details:

- Uses the standard
- If the desired date to set is March 1, 2002, *To* (Argument 0) 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 0
To
String Literal
String Variable

Action Block Example:

Set Date		
Argument Name	Type	Name
To	*String Variable*	*US_DATE_STRING*

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 PAC Control Debug mode or in PAC Terminal.
- 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)" on page 710
"Copy Date to String (MM/DD/YYYY)" on page 711
"Copy Time to String" on page 712

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 a PAC Control program.

Details:
- *To* (Argument 0) can be an integer or a float, although an integer is preferred.
- If the desired day of the month is March 2, 2017, *To* should be 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 0
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Set Day		
Argument Name	Type	Name
To	*Integer 32 Variable*	*DAY_OF_MONTH*

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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Hours" on page 728
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control program.

Details:

- *To* (Argument 0) 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 is 2 p.m. (14:00:00), *To* should be 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 0
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Set Hours		
Argument Name	Type	Name
To	*Integer 32 Variable*	*HOURS*

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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Day" on page 727
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control program.

Detail:

- *To* (Argument 0) 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 is 2:35 p.m. (14:35:00), *To* should be 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 0
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Set Minutes		
Argument Name	Type	Name
To	*Integer 32 Variable*	*MINUTES*

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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Hours" on page 728
"Set Day" on page 727
"Set Month" on page 730
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control program.

Details:

- *To* (Argument 0) can be an integer or a float, although an integer is preferred.
- If the desired month is March, *To* should be 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 0
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Set Month		
Argument Name	Type	Name
To	*Integer 32 Variable*	*MONTH*

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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Hours" on page 728
"Set Day" on page 727
"Set Minutes" on page 729
"Set Seconds" on page 731
"Set Year" on page 735

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 a PAC Control program.

Details:

- *To* (Argument 0) 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 is 2:35:26 p.m., then *To* should be 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 0
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Set Seconds		
Argument Name	Type	Name
To	*Integer 32 Variable*	*SECONDS*

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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Hours" on page 728
"Set Day" on page 727
"Set Minutes" on page 729
"Set Month" on page 730
"Set Year" on page 735

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 a PAC Control program.

Details:

- *From* (Argument 0) 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 is 2:35:00 p.m., *From* should be 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 0
To
String Literal
String Variable

Action Block Example:

Set Time		
Argument Name	Type	Name
To	*String Variable*	*TIME_STRING*

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)" on page 710
"Copy Date to String (MM/DD/YYYY)" on page 711
"Copy Time to String" on page 712
"Set Date" on page 726

Set Time Zone Configuration

Time/Date Action

Function: To set the time zone information for the controller to your local time.

NOTE: If you are using a SoftPAC controller, this command is not recommended because it can contend with the Windows built-in clock synchronization functionality. See Notes below.

Typical Use: To tell the controller the local time zone and when it takes effect, and (if applicable) when Daylight Savings Time takes effect.

Details:

- A string containing time zone data is passed. The existing time zone in the controller is replaced with a time zone code (indicating its deviation from UTC), and its effective month, week, day of week, and time.
- When Synchronize Clock SNTP is also used, use this command first.
- The syntax is:

  *With Daylight Savings Time:*TZ1,Mo1,Day1,Ord1,Time1/TZ2,Mo2,Day2,Ord2,Time2
  Example:`PST1,11,0,1,0200/PDT,3,0,2,0200`
  *Without Daylight Savings Time:*TZ1,Mo1,Day1,Ord1,Time1 [1]
  Example:`PST1,11,0,1,0200`

  where:

  – TZ1 is a 3-5 letter abbreviation of the desired time zone 1
    (For time zone abbreviations, see "C: Time Zone Abbreviations" on page 841.)
  – Mo1 is the month the time zone begins, (1-12)
  – Day1 is the day of the week the time zone begins, (0-6 = Su-Sa)
  – Ord1 is the ordinal of the day the time zone begins, (values must be from 1 to 5, with 5 meaning the last occurrence of the specified day within the month).
  – Time1 is the time of day the time zone begins.
  – (Optional) The second set of time zone data is used to change back and forth between the first and second time zones (for example, from Standard to Daylight Savings Time).

  For example, `PST1,11,0,1,0200` means:
  PST1 Pacific Standard Time
  11November
  0Sunday
  1 1st Occurrence (in this case, 1st Sunday of a possible 5 Sundays in a month)
  0200 Start time = 2:00AM

  The order of the time zones is not important, but the data for each zone must be in the specified order.
- If the command is successful, the passed time zone data is written as specified, and the code for success (0) is returned. Otherwise, a single time zone of UTC is written, and an Invalid String error (-46) is returned.
- This command sets the time zone in the controller's battery-backed RAM, so it stays set unless the time zone is deliberately changed or cleared, or the battery dies.

Arguments:

Argument 0	Argument 1
Configuration	**Put Result in**
String Literal	Integer 32 Variable
String Variable	

Action Block Example:

Set Time Zone Configuration

[1] These arguments are required and must be valid values (even if you aren't using Daylight Savings Time).

Argument Name	Type	Name
Configuration	*String Literal*	*StrTimeZone*
Put Result in	*Integer 32 Variable*	*STATUS*

OptoScript Example:

`SetTimeZoneConfiguration(`*Configuration*`)`

```
STATUS = SetTimeZoneConfiguration(StrTimeZone);
```

Example to set times for Standard and Daylight Saving Time:

STATUS = SetTimeZoneConfiguration("PST1,11,0,1,0200/PDT,3,0,2,0200");

To clear settings that were set previously, pass an empty string:

```
STATUS = SetTimeZoneConfiguration("");
```

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 used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Put this command before Synchronize Clock SNTP when you use it. Otherwise, Synchronize Clock SNTP will set the controller to Coordinated Universal Time.
- If you are using a SoftPAC controller, this command is not recommended.

  Attempting to use this command without disabling the Windows time handling logic can cause your computer's clock to malfunction. However, if you understand the risks, you can choose to let SoftPAC handle synchronizing the PC's clock by using the Synchronize Clock SNTP command. (This is considered an advanced topic that is not covered in this document.)

Status Codes:

0 = Success

-6 = Invalid data field.

-46 = Invalid string.

-606 Redundant Mode Bad. This can be caused by attempting to use this command on a backup redundant controller. This command is only allowed on the active controller.

See Also:

"Get Minutes" on page 718
"Get Time Zone Description" on page 723
"Synchronize Clock SNTP" on page 736

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 a PAC Control program.

Details:

- *To* (Argument 0) 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 0
To
Float Literal
Float Variable
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Set Year		
Argument Name	Type	Name
To	*Integer 32 Variable*	*YEAR*

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" on page 714
"Get Day of Week" on page 715
"Get Hours" on page 716
"Get Minutes" on page 718
"Get Month" on page 719
"Get Seconds" on page 720
"Get Year" on page 725
"Set Hours" on page 728
"Set Day" on page 727
"Set Minutes" on page 729
"Set Month" on page 730
"Set Seconds" on page 731

Synchronize Clock SNTP

Time/Date Action

Function: Synchronize the controller's clock with an external Network Time Protocol (NTP) server.

NOTE: If you are using a SoftPAC controller, this command is not recommended because it can contend with the Windows built-in clock synchronization functionality. See Notes below.

Typical Use: To make sure the controller clock is accurate whenever the command is applied.

Details:

- *Server URL* (Argument 1) is the URL of the time server you want to use, such as UDP port 123. For example, `udp:time.nist.gov:123`.
- If everything is OK, the date and time are both set and a 0 (zero—the code for success) is stored in *Put Result in* (Argument 2). If there is an error, a negative-number status code (listed below) is returned.

Arguments:

Argument 0 **Timeout (Seconds)**	Argument 1 **Server URL**	Argument 2 **Put Result in**
Float Literal	String Literal	Integer 32 Variable
Float Variable	String Variable	
Integer 32 Literal		
Integer 32 Variable		

Action Block Example:

Synchronize Clock SNTP		
Argument Name	Type	Name
Timeout (Seconds)	*Integer 32 Literal*	*nTimeout*
Server URL	*String Literal*	*StrServerURL*
Put Result in	*Integer 32 Variable*	*nResult*

OptoScript Example:

SynchronizeClockSNTP(*Timeout (Seconds), Server URL***)**

```
nResult = SynchronizeClockSNTP(nTimeout, StrServerURL);
```

This is a function command; it sets the date and time with the NTP server and returns the status code for success (0) or one of the other status codes listed below. The returned value can be consumed by a variable (as shown) or used in an item such as a Condition block, an OptoScript block, or a mathematical expression. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- Normally syncs within one minute.
- Uses SNTP (Simple NTP) to sync the clock.
- If you are using a SoftPAC controller, this command is not recommended. Attempting to use this command without disabling the Windows time handling logic can cause your computer's clock to malfunction. However, if you understand the risks, you can choose to let SoftPAC handle synchronizing the PC's clock by using the Synchronize Clock SNTP command. This is considered an advanced topic that is not covered in this document.

Status Codes:

0 = Success

-17 = Port or object is already locked.

-39 = Receive timeout.

-46 = Invalid string.

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

-49 = No more connections are available. Maximum number of connections already in use.

-50 = Open connection timeout. Could not establish connection within the timeout period. Make sure that the controller is configured with a DNS and gateway for the network.

-58 = Character not found.

-443 = Could not receive on socket. The NTP server hasn't responded within the time configured in *Timeout (Seconds)* (Argument 0). Try increasing the timeout value or try a different Server URL (in case there's a problem with the time server pool itself).

-454 = Unable to connect to DNS server. Check the DNS and gateway configuration.

See Also: "Set Time Zone Configuration" on page 733
"Get Minutes" on page 718
"Get Time Zone Description" on page 723

23: Timing Commands

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, forward/reverse) time.

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

Arguments:

Argument 0
Timer
Down Timer Variable
Up Timer Variable

Action Block Example:

Continue Timer

Argument Name	Type	Name
Timer	*Down Timer Variable*	*OVEN_TIMER*

OptoScript Example:

`ContinueTimer`(*Timer*)

```
ContinueTimer(OVEN_TIMER);
```

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

Note:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- If the timer is running, this command has no effect.

See Also:

"Start Off-Pulse" on page 202
"Stop Timer" on page 748
"Pause Timer" on page 744
"Set Down Timer Preset Value" on page 745
"Set Up Timer Target Value" on page 746

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 0
[Value]
Integer 32 Literal
Integer 32 Variable

Action Block Example:

Delay (mSec)		
Argument Name	Type	Name
(none)	*Integer 32 Literal*	*1*

OptoScript Example:

DelayMsec(*Argument 0***)**

```
DelayMsec(1);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- For readability, use Delay (Sec) for delays longer than 10 seconds.
- When high accuracy is needed, reduce the number of charts running concurrently.
- If you use a delay of zero, PAC Control will ignore the delay command.

Queue Errors: -8 = Value less than zero.

See Also:

"Delay (Sec)" on page 741
"Start Off-Pulse" on page 202
"Stop Timer" on page 748
"Pause Timer" on page 744
"Continue Timer" on page 739

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 0
[Value]
Float Literal
Float Variable

Action Block Example:

Delay (Sec)		
Argument Name	Type	Name
(none)	*Float Literal*	*10.525*

OptoScript Example:

DelaySec(*Argument 0***)**

```
DelaySec(10.525);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- Use Delay (mSec) for delays shorter than 10 seconds.
- When high accuracy is needed, reduce the number of charts running concurrently.
- If you use a delay of zero, PAC Control will ignore the delay command.

Queue Errors: -8 = Value less than zero.

See Also: "Delay (mSec)" on page 740

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 0
Down Timer
Down Timer Variable

Condition Block Example:

Down Timer Expired?		
Argument Name	Type	Name
Down Timer	*Down Timer Variable*	*OVEN_TIMER*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) for more information on using timer commands.

See Also:

"Start Off-Pulse" on page 202
"Stop Timer" on page 748
"Continue Timer" on page 739
"Pause Timer" on page 744
"Set Down Timer Preset Value" on page 745
"Delay (Sec)" on page 741
"Delay (mSec)" on page 740

Get & Restart Timer

Timing Action

Function: Retrieves a timer variable and restarts it.

Typical Use: By combining a read and start into one transaction, this command provides more accurate sequential timing.

Details: If you do a separate read and start, there could be a task switch between the time you read the timer and the time you restart it resulting in a delay in which no timing would be performed. With this command, the timer is instantly restarted as soon as the current time is retrieved.

Arguments:

Argument 0	Argument 1
Timer	**Put In**
Down Timer Variable	Float Variable
Up Timer Variable	Integer 32 Variable

Action Block Example:

Get & Restart Timer		
Argument Name	Type	Name
Timer	*Down Timer Variable*	*Down_Timer*
Put In	*Integer 32 Variable*	*nResult*

OptoScript Example:

GetRestartTimer(*Timer*)

```
nResult = GetRestartTimer(Down_Timer);
```

This is a function command; it returns the value of the timer before it was restarted.

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) for more information on using timers.

See Also:

"Delay (Sec)" on page 741
"Start Off-Pulse" on page 202
"Stop Timer" on page 748
"Pause Timer" on page 744
"Continue Timer" on page 739

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 0
Timer
Down Timer Variable
Up Timer Variable

Action Block Example:

Pause Timer		
Argument Name	Type	Name
Timer	*Down Timer Variable*	*OVEN_TIMER*

OptoScript Example:

PauseTimer(*Timer***)**

```
PauseTimer(OVEN_TIMER);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) for more information on using timers.

See Also:

"Start Off-Pulse" on page 202
"Stop Timer" on page 748
"Continue Timer" on page 739
"Set Down Timer Preset Value" on page 745
"Set Up Timer Target Value" on page 746

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 0* must be a positive number in seconds.

Arguments:

Argument 0	Argument 1
Target Value	**Down Timer**
Float Literal	Down Timer Variable
Float Variable	

Action Block Example:

Set Down Timer Preset Value

Argument Name	Type	Name
Target Value	*Float Literal*	*60.0*
Down Timer	*Down Timer Variable*	*OVEN_TIMER*

OptoScript Example:

SetDownTimerPreset(*Target Value, Down Timer***)**

```
SetDownTimerPreset(60.0, OVEN_TIMER);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) 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" on page 202
"Stop Timer" on page 748
"Continue Timer" on page 739
"Pause Timer" on page 744
"Down Timer Expired?" on page 742

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 0	Argument 1
Target Value	**Up Timer**
Float Literal	Up Timer Variable
Float Variable	

Action Block Example:

Set Up Timer Target Value		
Argument Name	Type	Name
Target Value	*Float Literal*	*60.0*
Up Timer	*Up Timer Variable*	*OVEN_TIMER*

OptoScript Example:

SetUpTimerTarget(*Target Value, Up Timer***)**

```
SetUpTimerTarget(60.0, OVEN_TIMER);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) 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" on page 202
"Stop Timer" on page 748
"Continue Timer" on page 739
"Pause Timer" on page 744
"Up Timer Target Time Reached?" on page 750

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 0
Timer
Down Timer Variable
Up Timer Variable

Action Block Example:

Start Timer		
Argument Name	Type	Name
Timer	*Down Timer Variable*	*Oven_Timer*

OptoScript Example:

StartTimer(*Timer*)

```
StartTimer(Oven_Timer);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) 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" on page 748
"Continue Timer" on page 739
"Pause Timer" on page 744
"Set Down Timer Preset Value" on page 745
"Set Up Timer Target Value" on page 746

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 0
Timer
Down Timer Variable
Up Timer Variable

Action Block Example:

Stop Timer		
Argument Name	Type	Name
Timer	*Down Timer Variable*	*OVEN_TIMER*

OptoScript Example:

StopTimer(*Timer*)

```
StopTimer(OVEN_TIMER);
```

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

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) for more information on timers.

See Also:

"Start Off-Pulse" on page 202
"Continue Timer" on page 739
"Pause Timer" on page 744
"Set Down Timer Preset Value" on page 745
"Set Up Timer Target Value" on page 746

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: If the specified timer has reached its target value, the logic will take the True path.
If the specified timer has *not* reached its target value, the logic will take the False path.

Arguments:

Argument 0
Is
Down Timer Variable
Up Timer Variable

Condition Block Example:

Timer Expired?		
Argument Name	Type	Name
Is	*Down Timer Variable*	*EGG_TIMER*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:

- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) 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" on page 746
"Set Down Timer Preset Value" on page 745
"Start Off-Pulse" on page 202
"Up Timer Target Time Reached?" on page 750
"Down Timer Expired?" on page 742

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 0
Up Timer
Up Timer Variable

Condition Block Example:

Up Timer Target Time Reached?		
Argument Name	Type	Name
Up Timer	*Up Timer Variable*	*OVEN_TIMER*

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, and so forth. For more information, see the *PAC Control User's Guide* (form 1700).

Notes:
- A timer stores elapsed time in units of seconds (with a resolution in milliseconds). Up timers count up from zero; down timers start from a value you set, and then and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15} seconds.
- See "Timing Commands" in the *PAC Control User's Guide* (form 1700) for more information on using timers.

See Also:
"Start Off-Pulse" on page 202
"Stop Timer" on page 748
"Continue Timer" on page 739
"Pause Timer" on page 744
"Set Up Timer Target Value" on page 746

A: High-Speed Digital Function Support

Use the following table to determine whether a brain supports high-speed digital functions and the commands used with those functions. High-speed functions include high-speed counting, quadrature counting, on-pulse and off-pulse measurement, and frequency and period measurement.

Part number	High-Speed Digital Function Support?
***mistic* bricks**	
B100	yes
E1	yes
G4D16R	yes
G4D32RS	no
serial B3000 *mistic*	
B3000	yes
SNAP-BRS	no
EIO	
SNAP-B3000-ENET	yes
SNAP-ENET-D64	no
SNAP-ENET-RTC	yes
SNAP-ENET-S64	no
UIO	
SNAP-UP1-ADS	yes
SNAP-UP1-D64	no
SNAP-UP1-M64	no

Part number	High-Speed Digital Function Support?
GRV-EPIC	
GRV-EPIC-PR1	yes
SNAP PAC R-series	
SNAP-PAC-R1	yes
SNAP-PAC-R1-B	yes
SNAP-PAC-R2	no
SNAP PAC EB-series	
SNAP-PAC-EB1	yes
SNAP-PAC-EB2	no
SNAP PAC SB-series	
SNAP-PAC-SB1	yes
SNAP-PAC-SB2	no

B: Table Index Offset Examples

This section provides information for using the *Points per Module* argument in the following commands:

"IVAL Move Numeric Table to I/O Unit Ex" on page 313

"Move I/O Unit to Numeric Table Ex" on page 317

"Move Numeric Table to I/O Unit Ex" on page 322

For examples of table index offsets, see "Table Index Offsets" below.

For the length of the table required for values of Points per Module, see "Length of Table Required" on page 756.

TABLE INDEX OFFSETS

The following tables show the table index offsets for values received and written from high-density analog modules. Each table shows offsets for a different Points per Module value.

Module Position	Channel of Module	Index Offset
0	1	0
1	1	1
2	1	2
3	1	3
…	…	…
15	1	15

Module Position	Channel of Module	Index Offset
0	1	0
0	2	1
1	1	2
1	2	3
2	1	4
2	2	5
3	1	6
3	2	7
4	1	8

Module Position	Channel of Module	Index Offset
4	2	9
…	…	…
15	1	30
15	2	31

Module Position	Channel of Module	Index Offset
0	1	0
0	2	1
0	3	2
0	4	3
1	1	4
1	2	5
1	3	6
1	4	7
2	1	8
2	2	9
2	3	10
2	4	11
…	…	…
15	1	60
15	2	61
15	3	62
15	4	63

Module Position	Channel of Module	Index Offset
0	1	0
0	2	1
0	3	2
0	4	3
0	5	4
0	6	5
0	7	6
0	8	7
1	1	8
1	2	9
1	3	10
1	4	11
…	…	…
15	4	124
15	5	125
15	6	126
15	7	127

Module Position	Channel of Module	Index Offset
0	1	0
0	2	1
0	3	2
0	4	3
0	5	4
0	6	5
0	7	6
0	8	7
0	9	8
0	10	9
0	11	10
0	12	11
0	13	12
0	14	13
0	15	14
0	16	15
0	17	16
0	18	17
0	19	18
0	20	19
0	21	20
0	22	21
0	23	22
0	24	23
0	25	24
0	26	25
0	27	26
0	28	27
0	29	28
0	30	29
0	31	30
0	32	31
…	…	…
15	1	480
15	2	481
15	3	482
15	4	483
15	5	484
15	6	485
15	7	486
15	8	487

Module Position	Channel of Module	Index Offset
15	9	488
15	10	489
15	11	490
15	12	491
15	13	492
15	14	493
15	15	494
15	16	495
15	17	496
15	18	497
15	19	498
15	20	499
15	21	500
15	22	501
15	23	502
15	24	503
15	25	504
15	26	505
15	27	506
15	28	507
15	29	508
15	30	509
15	31	510
15	32	511

LENGTH OF TABLE REQUIRED

This table shows the length of the table required for different PPM (points per module) values.

PPM	Number of Elements from Starting Index Required
1	16
2	32
3	48
4	64
5	80
6	96
7	112
8	128
9	144
10	160
11	176

PPM	Number of Elements from Starting Index Required
12	192
13	208
14	224
15	240
16	256
17	272
18	288
19	304
20	320
21	336
22	352
23	368
24	384
25	400
26	416
27	432
28	448
29	464
30	480
31	496
32	512

C: Deprecated Commands

Deprecated commands are commands that are functional; however, they've been replaced by newer commands for developing new strategies.

Deprecated commands are visible in the drop-down list in PAC Control's Add Instruction dialog box, but the command's name is appended with the text: [DEPRECATED]. This gives you a clear indication that you might want to select the deprecated command's replacement.

This chapter provides documentation for deprecated commands.

Clear Receive Buffer

Obsolete. Use "Clear Communication Receive Buffer" on page 77.

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" on page 629* *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 0	Argument 1	Argument 2
On Mask	**Off Mask**	**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

Argument Name	Type	Name
On Mask	*Integer 32 Variable*	*PUMPS_ON_MASK*
Off Mask	*Integer 32 Literal*	*0*
On I/O Unit	*B3000 (Digital)*	*PUMP_CTRL*

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" on page 610
"Disable Communication to I/O Unit" on page 611

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" on page 629* *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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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		
Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
Digital-64 I/O Unit	*SNAP-UP1-D64*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 610
"Disable Communication to I/O Unit" on page 611
"IVAL Set Mixed I/O Unit from MOMO Masks" on page 766,
"IVAL Set Mixed 64 I/O Unit from MOMO Masks" on page 764

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" on page 629* *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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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		
Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
Mixed 64 I/O Unit	*SNAP-UP1-M64*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 610
"Disable Communication to I/O Unit" on page 611
"IVAL Set Mixed I/O Unit from MOMO Masks" on page 766
"IVAL Set Digital-64 I/O Unit from MOMO Masks" on page 762

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" on page 629* *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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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		
Argument Name	Type	Name
Must On Mask	*Integer 32 Variable*	*PUMPS_ON_MASK*
Must Off Mask	*Integer 32 Literal*	*0xB001A020*
Mixed I/O Unit	*SNAP-UP1-ADS*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 610
"Disable Communication to I/O Unit" on page 611
"IVAL Set Digital-64 I/O Unit from MOMO Masks" on page 762
"IVAL Set Mixed 64 I/O Unit from MOMO Masks" on page 764

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" on page 629 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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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		
Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
Simple 64 I/O Unit	*SNAP-ENET-S64*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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” on page 610
“Disable Communication to I/O Unit” on page 611
“IVAL Set Digital-64 I/O Unit from MOMO Masks” on page 762
“IVAL Set Mixed 64 I/O Unit from MOMO Masks” on page 764

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" on page 329* *instead.*

Function: **For 16-channel** ***mistic*** **digital I/O units only.** 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 uses bits 0 through 15, because *mistic* digital I/O units have 16 points.
- 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.
- To turn on a point, set the respective bit in the 32-bit data field of *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 32-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
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		
Argument Name	Type	Name
Must On Mask	*Integer 32 Variable*	*PUMPS_ON_MASK*
Must Off Mask	*Integer 32 Literal*	*3840*
Digital I/O Unit	*B3000 (Digital)*	*PUMP_CTRL*

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 Bitmask	Binary	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
	Hex	0				0				F				0			
Must-off Bitmask	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 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" on page 329 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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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		
Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
Digital-64 I/O Unit	*SNAP-UP1-D64*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 301

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" on page 329 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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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		
Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
Mixed 64 I/O Unit	*SNAP-UP1-M64*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 301

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" on page 329* *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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 32-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	**Argument 1**	**Argument 2**
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

Argument Name	Type	Name
Must On Mask	*Integer 32 Variable*	*PUMPS_ON_MASK*
Must Off Mask	*Integer 32 Literal*	*0xB001A020*
Mixed I/O Unit	*SNAP-UP1-ADS*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 301
"Set I/O Unit from MOMO Masks" on page 329

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" on page 329 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 *Must On Mask* (Argument 0) to a value of 1.
- To turn off a point, set the respective bit in the 64-bit data field of *Must Off Mask* (Argument 1) to a value of 1.
- To leave a point unaffected, set its bits to a value of 0 in both *Must On Mask* and *Must Off Mask*. 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) are written.

Arguments:

Argument 0	Argument 1	Argument 2
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

Argument Name	Type	Name
Must On Mask	*Integer 64 Literal*	*0x060003C0000000C2*
Must Off Mask	*Integer 64 Literal*	*0xB0F240010308A020*
Simple 64 I/O Unit	*SNAP-ENET-S64*	*PUMP_CTRL_UNIT*

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 Bitmask	Binary	0	0	0	0	0	1	1	0	> > >	1	1	0	0	0	0	1	0
	Hex	0				6				> > >	C				2			
Must-off Bitmask	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" on page 301

Index

H

S